**PLSQL EXERCISES**

**Exercise 1: Control Structures**

-- =============================================

-- 1. SCHEMA CREATION

-- =============================================

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Add column for VIP status

ALTER TABLE Customers ADD IsVIP CHAR(1);

-- =============================================

-- 2. SAMPLE DATA INSERTION

-- =============================================

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

VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 10000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 15000, 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, 6, SYSDATE, SYSDATE + 15); -- Due soon

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

VALUES (2, 2, 7000, 7, SYSDATE, SYSDATE + 60); -- Not due soon

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'));

-- =============================================

-- 3. SCENARIO 1: Apply 1% Interest Discount for Customers > 60

-- =============================================

BEGIN

FOR rec IN (

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

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

) LOOP

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

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to Loan ID ' || rec.LoanID || ' for customer over 60.');

END IF;

END LOOP;

END;

/

-- =============================================

-- 4. SCENARIO 2: Set VIP Status for Balance > 10000

-- =============================================

BEGIN

FOR rec IN (SELECT CustomerID, Balance 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;

/

-- =============================================

-- 5. SCENARIO 3: Remind Customers with Loan Due in 30 Days

-- =============================================

BEGIN

FOR rec IN (

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

FROM Loans l

JOIN Customers c ON c.CustomerID = l.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

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

' for ' || rec.Name ||

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

END LOOP;

END;

/

-- =============================================

-- 6. CHECK RESULTS

-- =============================================

SELECT \* FROM Customers;

SELECT \* FROM Loans;

**Output:**



**Exercise 2: Error Handling**

-- Procedure 1: SafeTransferFunds

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_from\_balance NUMBER;

v\_txn\_id NUMBER;

BEGIN

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

IF v\_from\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts SET Balance = Balance + p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_to\_account\_id;

SELECT NVL(MAX(TransactionID), 0) + 1 INTO v\_txn\_id FROM Transactions;

INSERT INTO Transactions VALUES (v\_txn\_id, p\_from\_account\_id, SYSDATE, p\_amount, 'Transfer Out');

INSERT INTO Transactions VALUES (v\_txn\_id + 1, p\_to\_account\_id, SYSDATE, p\_amount, 'Transfer In');

COMMIT;

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

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

-- Procedure 2: UpdateSalary

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_emp\_id IN NUMBER,

p\_percent IN NUMBER

) AS

v\_old\_salary NUMBER;

BEGIN

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

UPDATE Employees

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

WHERE EmployeeID = p\_emp\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated from ' || v\_old\_salary ||

' to ' || (v\_old\_salary + v\_old\_salary \* p\_percent / 100));

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

WHEN OTHERS THEN

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

END;

/

-- Procedure 3: AddNewCustomer

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_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, IsVIP)

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

COMMIT;

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

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END;

/

-- Test SafeTransferFunds

BEGIN

SafeTransferFunds(1, 2, 200);

END;

/

-- Test UpdateSalary

BEGIN

UpdateSalary(1, 10);

END;

/

-- Test UpdateSalary for invalid ID

BEGIN

UpdateSalary(99, 10);

END;

/

-- Test AddNewCustomer

BEGIN

AddNewCustomer(3, 'Kiran Rao', TO\_DATE('1992-10-10', 'YYYY-MM-DD'), 3000);

END;

/

-- Try duplicate customer

BEGIN

AddNewCustomer(3, 'Another Kiran', TO\_DATE('1990-01-01', 'YYYY-MM-DD'), 5000);

END;

/

-- Check data

SELECT \* FROM Customers;

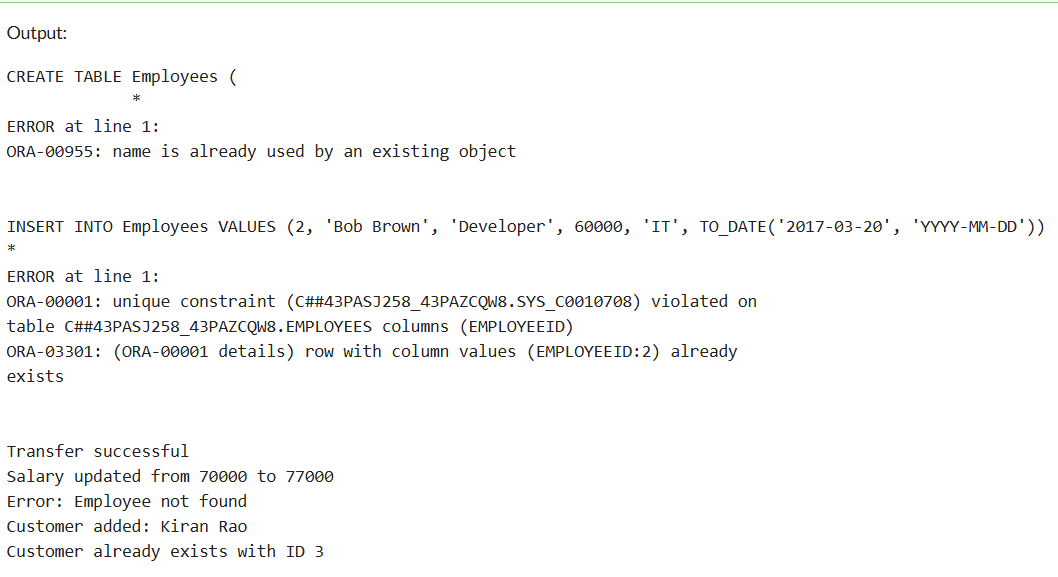
SELECT \* FROM Employees;

SELECT \* FROM Accounts;

SELECT \* FROM Transactions;

/

**Output:**





**Exercise 3: Stored Procedures**

-- DROP IF EXISTS (for re-run safety)

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Transactions CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Loans CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS';

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,

IsVIP CHAR(1)

);

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(20),

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, 'N');

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

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'));

/

-- ENABLE OUTPUT

SET SERVEROUTPUT ON;

-- PROCEDURE 1: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance + (acc.Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest added for Account ' || acc.AccountID ||

'. New Balance: ' || TO\_CHAR(acc.Balance \* 1.01));

END LOOP;

COMMIT;

END;

/

-- PROCEDURE 2: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

FOR emp IN (SELECT EmployeeID, Salary FROM Employees WHERE Department = p\_department) LOOP

UPDATE Employees

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

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Updated salary for Employee ' || emp.EmployeeID ||

' to ' || TO\_CHAR(emp.Salary + emp.Salary \* p\_bonus\_percent / 100));

END LOOP;

COMMIT;

END;

/

-- PROCEDURE 3: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

v\_txn\_id 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, LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount, LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

SELECT NVL(MAX(TransactionID), 0) + 1 INTO v\_txn\_id FROM Transactions;

INSERT INTO Transactions VALUES (v\_txn\_id, p\_from\_account, SYSDATE, p\_amount, 'Transfer Out');

INSERT INTO Transactions VALUES (v\_txn\_id + 1, p\_to\_account, SYSDATE, p\_amount, 'Transfer In');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred ₹' || p\_amount ||

' from Account ' || p\_from\_account ||

' to Account ' || p\_to\_account);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

-- ===== TEST PROCEDURES =====

BEGIN

ProcessMonthlyInterest;

END;

/

BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

BEGIN

TransferFunds(1, 2, 100);

END;

/

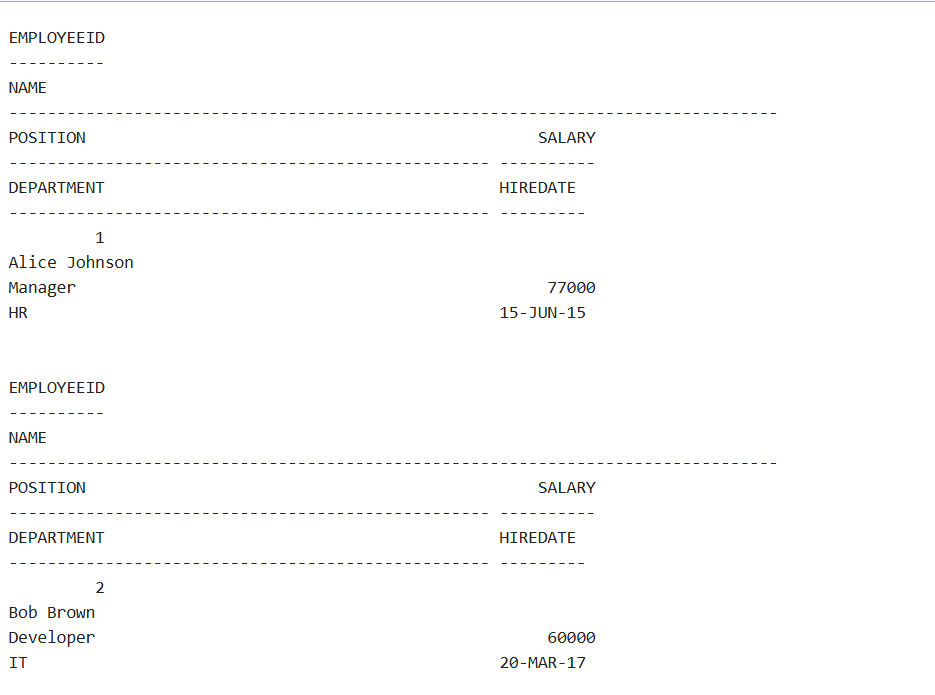
-- ===== CHECK RESULTS =====

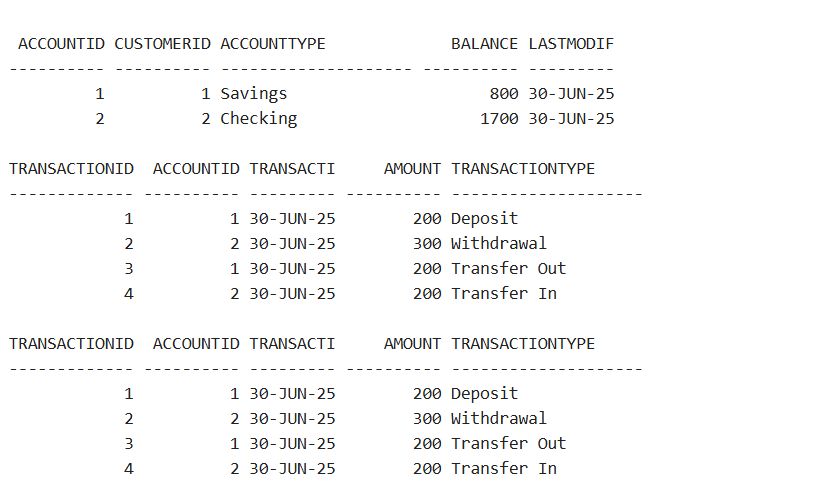
SELECT \* FROM Accounts;

SELECT \* FROM Employees;

SELECT \* FROM Transactions;

**Output:**





**Exercise 4: Functions**

-- Enable output

SET SERVEROUTPUT ON;

-- Drop functions if they already exist

BEGIN

EXECUTE IMMEDIATE 'DROP FUNCTION CalculateAge';

EXECUTE IMMEDIATE 'DROP FUNCTION CalculateMonthlyInstallment';

EXECUTE IMMEDIATE 'DROP FUNCTION HasSufficientBalance';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Function 1: CalculateAge (returns age in years 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;

/

-- Function 2: CalculateMonthlyInstallment

-- Formula used: EMI = P \* R \* (1 + R)^N / ((1 + R)^N - 1)

-- Where:

-- P = loan amount, R = monthly rate (annual / 12 / 100), N = months

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_amount NUMBER,

p\_annual\_rate NUMBER,

p\_duration\_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\_duration\_years \* 12;

IF v\_monthly\_rate = 0 THEN

v\_emi := p\_amount / v\_months;

ELSE

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

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

END IF;

RETURN ROUND(v\_emi, 2);

END;

/

-- Function 3: HasSufficientBalance (check if account has enough funds)

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;

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

-- ====== Test Section ======

-- Test CalculateAge

DECLARE

v\_age NUMBER;

BEGIN

v\_age := CalculateAge(TO\_DATE('1990-07-20', 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Age for DOB 1990-07-20: ' || v\_age);

END;

/

-- Test CalculateMonthlyInstallment

DECLARE

v\_emi NUMBER;

BEGIN

v\_emi := CalculateMonthlyInstallment(500000, 8.5, 5); -- loan amount, rate, years

DBMS\_OUTPUT.PUT\_LINE('Monthly EMI for 5L @ 8.5% for 5 yrs: ₹' || v\_emi);

END;

/

-- Test HasSufficientBalance

DECLARE

v\_ok BOOLEAN;

BEGIN

v\_ok := HasSufficientBalance(1, 500); -- AccountID 1, check for ₹500

IF v\_ok THEN

DBMS\_OUTPUT.PUT\_LINE('Account 1 has sufficient balance for ₹500.');

ELSE

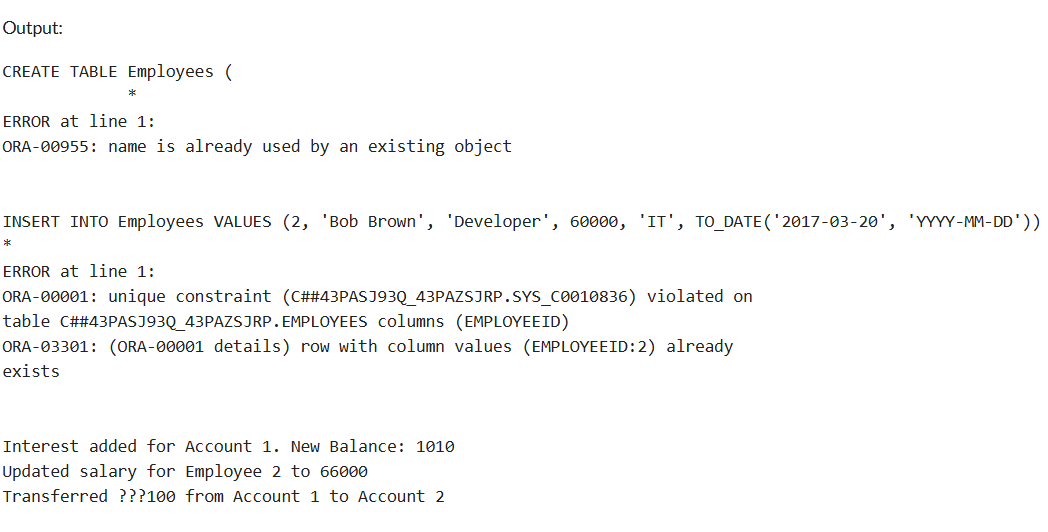
DBMS\_OUTPUT.PUT\_LINE('Account 1 does NOT have sufficient balance for ₹500.');

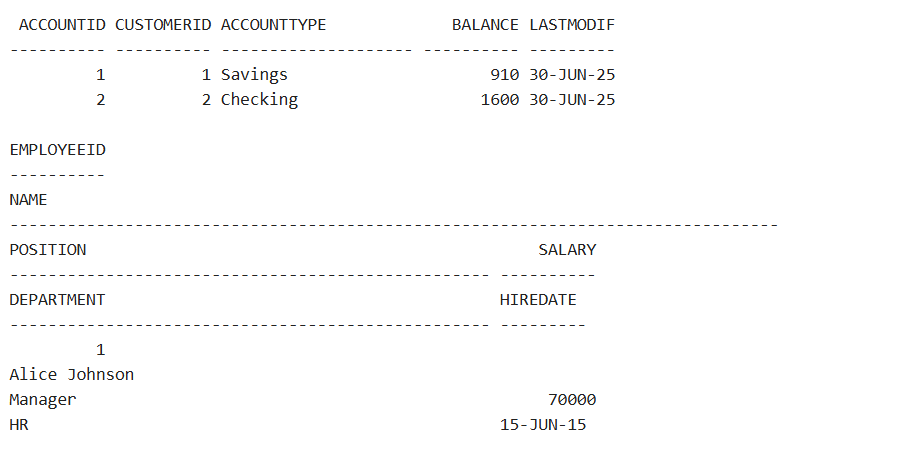
END IF;

END;

/

**Output:**





**Exercise 5: Triggers**

-- DROP IF EXISTS for safety

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE AuditLog';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

-- 1. Create AuditLog Table

CREATE TABLE AuditLog (

LogID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

LogDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(20)

);

/

-- 2. Trigger: UpdateCustomerLastModified

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

-- 3. Trigger: LogTransaction

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, AccountID, LogDate, Amount, TransactionType)

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

END;

/

-- 4. Trigger: CheckTransactionRules

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

-- Get account 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, 'Withdrawal amount exceeds account balance');

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;

/

-- ====== Test Section ======

SET SERVEROUTPUT ON;

-- 1. Test update to Customers

UPDATE Customers SET Balance = Balance + 100 WHERE CustomerID = 1;

SELECT CustomerID, Name, Balance, TO\_CHAR(LastModified, 'DD-MON-YYYY') AS LastModified FROM Customers;

-- 2. Valid deposit

INSERT INTO Transactions VALUES (3, 1, SYSDATE, 500, 'Deposit');

-- 3. Valid withdrawal

INSERT INTO Transactions VALUES (4, 1, SYSDATE, 100, 'Withdrawal');

-- 4. Invalid deposit (negative)

BEGIN

INSERT INTO Transactions VALUES (5, 1, SYSDATE, -300, 'Deposit');

EXCEPTION

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

END;

/

-- 5. Invalid withdrawal (more than balance)

BEGIN

INSERT INTO Transactions VALUES (6, 1, SYSDATE, 100000, 'Withdrawal');

EXCEPTION

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

END;

/

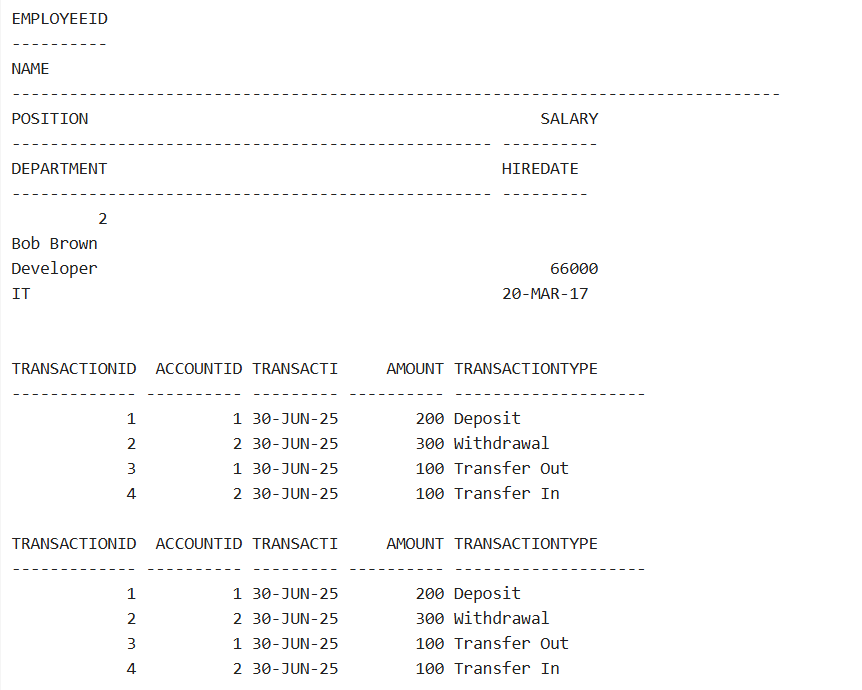
-- 6. Check Audit Log

SELECT \* FROM AuditLog;

-- 7. Check Transactions Table

SELECT \* FROM Transactions;

**Output:**



**Exercise 6: Cursors**

-- Enable output

SET SERVEROUTPUT ON;

-- 1. GenerateMonthlyStatements: Print transactions for current month

DECLARE

CURSOR txn\_cursor IS

SELECT c.CustomerID, c.Name, t.TransactionID, t.TransactionDate, t.Amount, t.TransactionType

FROM Customers c

JOIN Accounts a ON c.CustomerID = a.CustomerID

JOIN Transactions t ON a.AccountID = t.AccountID

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

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

ORDER BY c.CustomerID;

v\_cust\_id Customers.CustomerID%TYPE;

v\_name Customers.Name%TYPE;

v\_txn\_id Transactions.TransactionID%TYPE;

v\_txn\_date Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_type Transactions.TransactionType%TYPE;

v\_last\_id NUMBER := NULL;

BEGIN

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

OPEN txn\_cursor;

LOOP

FETCH txn\_cursor INTO v\_cust\_id, v\_name, v\_txn\_id, v\_txn\_date, v\_amount, v\_type;

EXIT WHEN txn\_cursor%NOTFOUND;

IF v\_last\_id IS NULL OR v\_last\_id != v\_cust\_id THEN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Customer ID: ' || v\_cust\_id || ' | Name: ' || v\_name);

DBMS\_OUTPUT.PUT\_LINE('Transactions:');

v\_last\_id := v\_cust\_id;

END IF;

DBMS\_OUTPUT.PUT\_LINE(' #' || v\_txn\_id || ' | ' || v\_txn\_date || ' | ' || v\_type || ' | ₹' || v\_amount);

END LOOP;

CLOSE txn\_cursor;

END;

/

-- 2. ApplyAnnualFee: Deduct ₹100 from all accounts

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance FROM Accounts;

v\_id Accounts.AccountID%TYPE;

v\_bal Accounts.Balance%TYPE;

v\_new\_bal NUMBER;

BEGIN

FOR acc IN acc\_cursor LOOP

v\_id := acc.AccountID;

v\_bal := acc.Balance;

IF v\_bal >= 100 THEN

v\_new\_bal := v\_bal - 100;

UPDATE Accounts SET Balance = v\_new\_bal, LastModified = SYSDATE WHERE AccountID = v\_id;

DBMS\_OUTPUT.PUT\_LINE('Annual fee of ₹100 deducted from Account ' || v\_id ||

'. New Balance: ₹' || v\_new\_bal);

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account ' || v\_id || ' has insufficient balance for fee deduction.');

END IF;

END LOOP;

COMMIT;

END;

/

-- 3. UpdateLoanInterestRates: Increase rate by 0.5% if below 6%

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate FROM Loans;

v\_id Loans.LoanID%TYPE;

v\_rate Loans.InterestRate%TYPE;

BEGIN

FOR loan IN loan\_cursor LOOP

v\_id := loan.LoanID;

v\_rate := loan.InterestRate;

IF v\_rate < 6 THEN

UPDATE Loans SET InterestRate = v\_rate + 0.5 WHERE LoanID = v\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated Loan ' || v\_id || ' interest rate from ' || v\_rate || ' to ' || (v\_rate + 0.5));

ELSE

DBMS\_OUTPUT.PUT\_LINE('Loan ' || v\_id || ' already has high interest rate: ' || v\_rate);

END IF;

END LOOP;

COMMIT;

END;

/

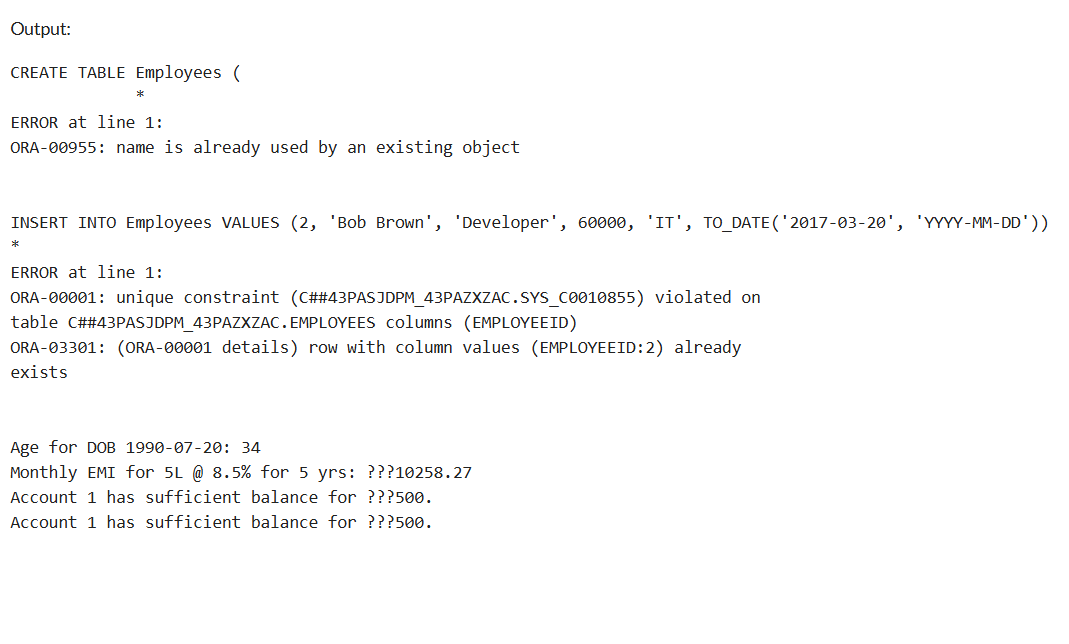
-- Check updated data

SELECT \* FROM Loans;

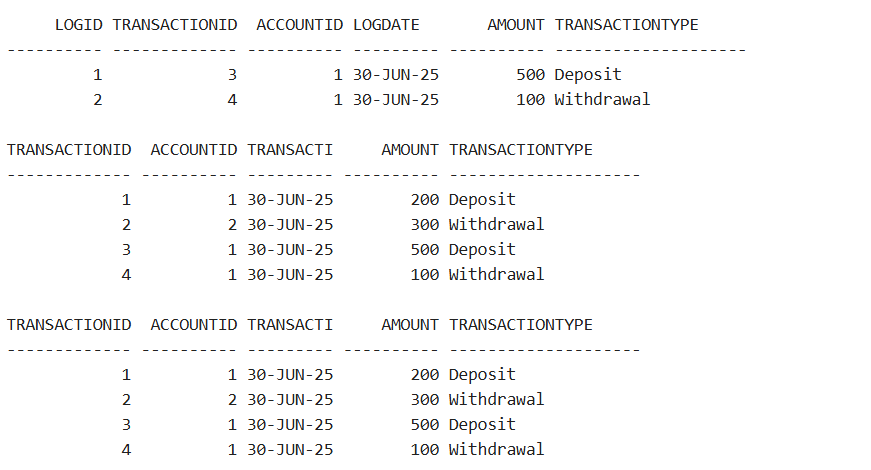
SELECT \* FROM Accounts;

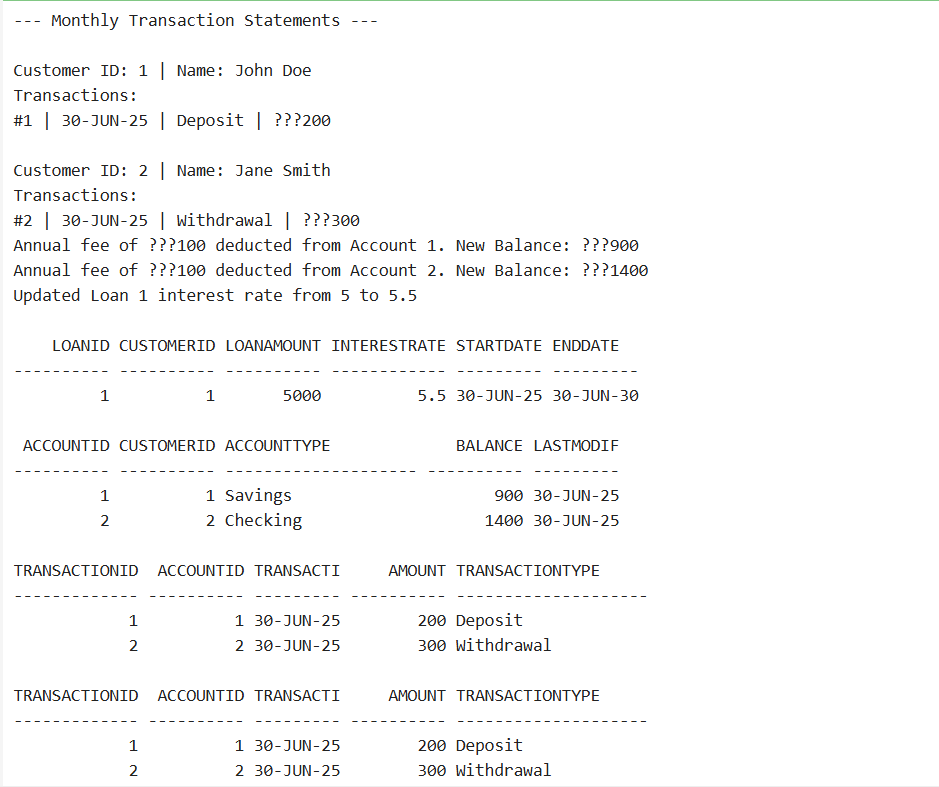
SELECT \* FROM Transactions;

**Output:**









**Exercise 7: Packages**

-- Enable output

SET SERVEROUTPUT ON;

-- 1. CustomerManagement Package

-- Package Specification

CREATE OR REPLACE PACKAGE CustomerManagement AS

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;

/

-- Package Body

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

BEGIN

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

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('Error: Customer with 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 ID ' || p\_id || ' not found.');

ELSE

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

END;

END CustomerManagement;

/

-- 2. EmployeeManagement Package

-- Specification

CREATE OR REPLACE PACKAGE EmployeeManagement AS

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

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

FUNCTION AnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

-- Body

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

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

BEGIN

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

DBMS\_OUTPUT.PUT\_LINE('Hired employee: ' || p\_name);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

ELSE

DBMS\_OUTPUT.PUT\_LINE('Updated salary for Employee ID ' || p\_id);

END IF;

END;

FUNCTION AnnualSalary(p\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT Salary \* 12 INTO v\_salary FROM Employees WHERE EmployeeID = p\_id;

RETURN v\_salary;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

RETURN NULL;

END;

END EmployeeManagement;

/

-- 3. AccountOperations Package

-- Specification

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

PROCEDURE CloseAccount(p\_acc\_id NUMBER);

FUNCTION TotalCustomerBalance(p\_cust\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

-- Body

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_acc\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (p\_acc\_id, p\_cust\_id, p\_type, p\_balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Account opened: ID ' || p\_acc\_id);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

END;

PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS

BEGIN

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

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account ID ' || p\_acc\_id || ' closed.');

END IF;

END;

FUNCTION TotalCustomerBalance(p\_cust\_id NUMBER) RETURN NUMBER IS

v\_total NUMBER := 0;

BEGIN

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

RETURN NVL(v\_total, 0);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

END AccountOperations;

/

-- ====== TEST SECTION ======

BEGIN

CustomerManagement.AddCustomer(5, 'Hari Priya', TO\_DATE('1994-02-18', 'YYYY-MM-DD'), 2500);

CustomerManagement.UpdateCustomer(5, 'Hari P', 3000);

DBMS\_OUTPUT.PUT\_LINE('Balance: ₹' || CustomerManagement.GetCustomerBalance(5));

END;

/

BEGIN

EmployeeManagement.HireEmployee(5, 'Meena Roy', 'Tester', 40000, 'QA', TO\_DATE('2020-01-01', 'YYYY-MM-DD'));

EmployeeManagement.UpdateEmployee(5, 45000);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ₹' || EmployeeManagement.AnnualSalary(5));

END;

/

BEGIN

AccountOperations.OpenAccount(5, 5, 'Savings', 5000);

DBMS\_OUTPUT.PUT\_LINE('Total Balance for Customer 5: ₹' || AccountOperations.TotalCustomerBalance(5));

AccountOperations.CloseAccount(5);

END;

/

-- View updated data

SELECT \* FROM Customers WHERE CustomerID = 5;

SELECT \* FROM Employees WHERE EmployeeID = 5;

SELECT \* FROM Accounts WHERE CustomerID = 5;

**Output:**



