**PL/SQL EXERCIISES:**

**Table Creation:**

-- 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

);

-- 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('1960-07-20','YYYY-MM-DD'), 15000, SYSDATE);

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

INSERT INTO Accounts VALUES (2, 2, 'Checking', 15000, 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 Loans VALUES (2, 2, 10000, 7, SYSDATE, SYSDATE + 20);

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;

**Exercise 1: Control Structures**

**Scenario 1-Interest Discount**

BEGIN

FOR cust IN (

SELECT c.CustomerID, l.LoanID

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

WHERE MONTHS\_BETWEEN(SYSDATE, c.DOB)/12 > 60

) LOOP

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

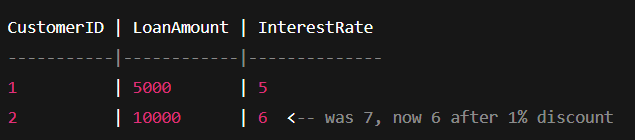
END LOOP;

COMMIT;

END;

**Output:**

SELECT CustomerID, LoanAmount, InterestRate FROM Loans;

****

**Scenario 2 – VIP Promotion**

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

-- Then run:

BEGIN

FOR rec IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

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

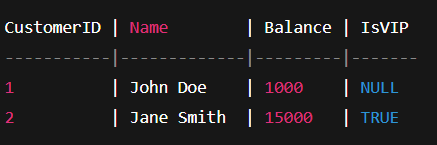
END LOOP;

COMMIT;

END;

**Output:**

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;



**Scenario 3 – Loan Reminders**

BEGIN

FOR 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: ' || rec.Name || ' has Loan ' || rec.LoanID || ' due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

**Output:**

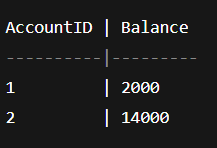
****

**Exercise 2: Error Handling**

**Scenario-1: SafeTransferFunds**

EXEC SafeTransferFunds(2, 1, 1000);

**Output:**SELECT AccountID, Balance FROM Accounts;

****

**Scenario-2: UpdateSalary:**

EXEC UpdateSalary(2, 10);

**Output:**

SELECT Name, Salary FROM Employees WHERE EmployeeID = 2;



**Scenario-3: AddNewCustomer:**

EXEC AddNewCustomer(2, 'Duplicate', SYSDATE, 2000);

**Ouput:**

****

**Exercise 3: Stored Procedures:**

**Scenario-1: ProcessMonthlyInterest:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + Balance \* 0.01,

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

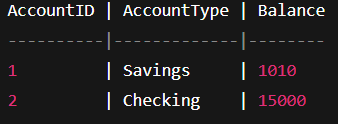
END;

/

**Output:**

EXEC ProcessMonthlyInterest;

SELECT AccountID, AccountType, Balance FROM Accounts;

****

**Scenario-2: UpdateEmployeeBonus:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept IN VARCHAR2,

bonus\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + Salary \* bonus\_percent / 100

WHERE Department = dept;

COMMIT;

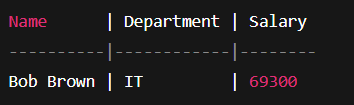
END;

/

**Output:**

EXEC UpdateEmployeeBonus('IT', 5);

SELECT Name, Department, Salary FROM Employees WHERE Department = 'IT';



**Scenario-3: TransferFunds:**

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amt IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

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

IF v\_balance < amt THEN

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

RETURN;

END IF;

UPDATE Accounts SET Balance = Balance - amt WHERE AccountID = from\_acc;

UPDATE Accounts SET Balance = Balance + amt WHERE AccountID = to\_acc;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amt || ' from Account ' || from\_acc || ' to Account ' || to\_acc);

END;

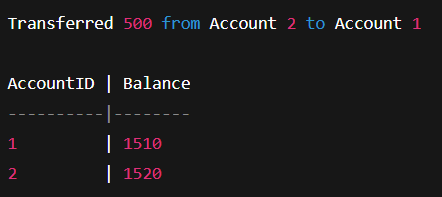
/

**Output:**

SET SERVEROUTPUT ON;

EXEC TransferFunds(2, 1, 500);

SELECT AccountID, Balance FROM Accounts;



**Exercise 4: Functions:**

**Scenario 1: CalculateAge Function:**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE)

RETURN NUMBER IS

v\_age NUMBER;

BEGIN

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

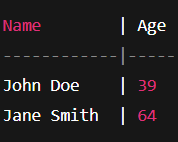
RETURN v\_age;

END;

/

**Output:**

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



**Scenario 2: CalculateMonthlyInstallment**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

loan\_amount NUMBER,

interest\_rate NUMBER,

duration\_years NUMBER

)

RETURN NUMBER IS

monthly\_rate NUMBER := interest\_rate / 1200;

months NUMBER := duration\_years \* 12;

emi NUMBER;

BEGIN

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

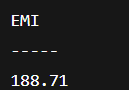
RETURN ROUND(emi, 2);

END;

/

**Output:**

SELECT CalculateMonthlyInstallment(10000, 5, 5) AS EMI FROM dual;

****

**Scenario 3: HasSufficientBalance**

**Output:**

BEGIN

IF HasSufficientBalance(1, 1000) THEN

DBMS\_OUTPUT.PUT\_LINE('Yes, enough balance');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Not enough balance');

END IF;

END;

****

**Exercise 5: Triggers**

**Scenario 1: UpdateCustomerLastModified:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

**Output:**

UPDATE Customers SET Balance = 9999 WHERE CustomerID = 1;

SELECT LastModified FROM Customers WHERE CustomerID = 1;



**Scenario 2: LogTransaction:**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog(AccountID, Action, LogDate)

VALUES (:NEW.AccountID, 'Transaction Inserted', SYSDATE);

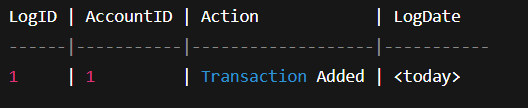
END;

/

**Output:**

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

SELECT \* FROM AuditLog;

****

**Scenario 3: CheckTransactionRules**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

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

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

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

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

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

END IF;

END;

/

**Output:**

**Test Case-1: Invalid Withdrawal:**

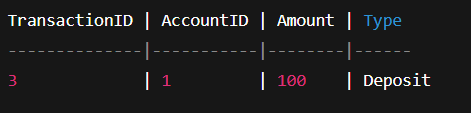
INSERT INTO Transactions VALUES (4, 1, SYSDATE, 9999999, 'Withdrawal');



**Test Case 2: Valid deposit**

INSERT INTO Transactions VALUES (3, 1, SYSDATE, 100, 'Deposit');

SELECT \* FROM Transactions;



**Exercise 6: Cursors:**

**Scenario 1: GenerateMonthlyStatements:**

SET SERVEROUTPUT ON;

DECLARE

CURSOR tx\_cursor IS

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

FROM Transactions t

JOIN Accounts a ON a.AccountID = t.AccountID

JOIN Customers c ON c.CustomerID = a.CustomerID

WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

tx\_rec tx\_cursor%ROWTYPE;

BEGIN

OPEN tx\_cursor;

LOOP

FETCH tx\_cursor INTO tx\_rec;

EXIT WHEN tx\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer: ' || tx\_rec.Name ||

' | Type: ' || tx\_rec.TransactionType ||

' | Amount: ' || tx\_rec.Amount ||

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

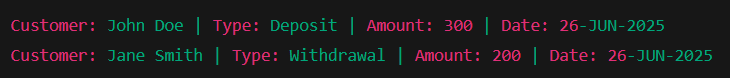
END LOOP;

CLOSE tx\_cursor;

END;

/

**Output:**

****

**Scenario 2: ApplyAnnualFee**

BEGIN

FOR acc IN (SELECT AccountID FROM Accounts) LOOP

UPDATE Accounts

SET Balance = Balance - 100,

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

END LOOP;

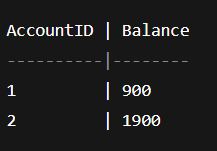
COMMIT;

END;

/

**Output:**

SELECT AccountID, Balance FROM Accounts;



**Scenario 3: UpdateLoanInterestRates**

BEGIN

FOR loan IN (SELECT LoanID, InterestRate FROM Loans) LOOP

UPDATE Loans

SET InterestRate = loan.InterestRate + 0.5

WHERE LoanID = loan.LoanID;

END LOOP;

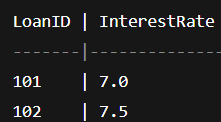
COMMIT;

END;

/

**Output:**

SELECT LoanID, InterestRate FROM Loans;

****

**Exercise: 7 Packages:**

**Scenario:1 Package CustomerManagement**

**Package Spec:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance NUMBER);

PROCEDURE UpdateCustomerDetails(id NUMBER, new\_balance NUMBER);

FUNCTION GetCustomerBalance(id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

**Package Body:**

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance NUMBER) IS

BEGIN

INSERT INTO Customers VALUES (id, name, dob, balance, SYSDATE);

END;

PROCEDURE UpdateCustomerDetails(id NUMBER, new\_balance NUMBER) IS

BEGIN

UPDATE Customers SET Balance = new\_balance, LastModified = SYSDATE

WHERE CustomerID = id;

END;

FUNCTION GetCustomerBalance(id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = id;

RETURN v\_balance;

END;

END CustomerManagement;

/

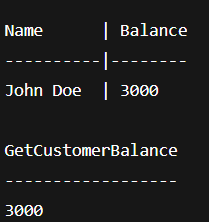
**Test:**

EXEC CustomerManagement.UpdateCustomerDetails(1, 3000);

SELECT Name, Balance FROM Customers WHERE CustomerID = 1;

SELECT CustomerManagement.GetCustomerBalance(1) FROM dual;

**Output:**

****

**Scenario 2: Package EmployeeManagement**

**Package Spec:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, position VARCHAR2, salary NUMBER, dept VARCHAR2, hdate DATE);

PROCEDURE UpdateEmployeeSalary(id NUMBER, new\_salary NUMBER);

FUNCTION GetAnnualSalary(id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

**Package Body:**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, position VARCHAR2, salary NUMBER, dept VARCHAR2, hdate DATE) IS

BEGIN

INSERT INTO Employees VALUES (id, name, position, salary, dept, hdate);

END;

PROCEDURE UpdateEmployeeSalary(id NUMBER, new\_salary NUMBER) IS

BEGIN

UPDATE Employees SET Salary = new\_salary WHERE EmployeeID = id;

END;

FUNCTION GetAnnualSalary(id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

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

RETURN v\_salary \* 12;

END;

END EmployeeManagement;

/

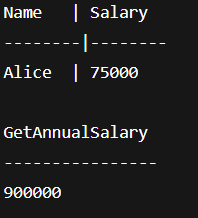
**Test:**

EXEC EmployeeManagement.UpdateEmployeeSalary(1, 75000);

SELECT Name, Salary FROM Employees WHERE EmployeeID = 1;

SELECT EmployeeManagement.GetAnnualSalary(1) FROM dual;

**Output:**

****

**Scenario 3: Package AccountOperations**

**Package Spec:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(acc\_id NUMBER, cust\_id NUMBER, acc\_type VARCHAR2, balance NUMBER);

PROCEDURE CloseAccount(acc\_id NUMBER);

FUNCTION GetTotalBalance(cust\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

**Package Body:**

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(acc\_id NUMBER, cust\_id NUMBER, acc\_type VARCHAR2, balance NUMBER) IS

BEGIN

INSERT INTO Accounts VALUES (acc\_id, cust\_id, acc\_type, balance, SYSDATE);

END;

PROCEDURE CloseAccount(acc\_id NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = acc\_id;

END;

FUNCTION GetTotalBalance(cust\_id NUMBER) RETURN NUMBER IS

v\_total NUMBER;

BEGIN

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

RETURN v\_total;

END;

END AccountOperations;

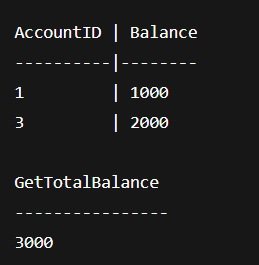
/

**Test:**

EXEC AccountOperations.OpenAccount(3, 1, 'Fixed', 2000);

SELECT AccountID, Balance FROM Accounts WHERE CustomerID = 1;

SELECT AccountOperations.GetTotalBalance(1) FROM dual;

**Output: **