**6418561- Atchaya V**

Week 1

# PL/SQL Exercises and Queries

# Exercise 1: Control Structures

## Scenario 1: Interest Discount for Customers Above 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 MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12 > 60 THEN  
 UPDATE Loans  
 SET InterestRate = rec.InterestRate - 1  
 WHERE LoanID = rec.LoanID;  
 END IF;  
 END LOOP;  
END;

## Scenario 2: Set IsVIP Flag for High Balance Customers

ALTER TABLE Customers ADD IsVIP CHAR(1);  
BEGIN  
 FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP  
 IF rec.Balance > 10000 THEN  
 UPDATE Customers  
 SET IsVIP = 'Y'  
 WHERE CustomerID = rec.CustomerID;  
 END IF;  
 END LOOP;  
END;

## Scenario 3: Loan Due Reminders

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 ' || rec.Name || ' is due on ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));  
 END LOOP;  
END;

# Exercise 2: Error Handling

## Scenario 1: SafeTransferFunds Procedure

CREATE OR REPLACE PROCEDURE SafeTransferFunds(  
 p\_FromAccountID IN NUMBER,  
 p\_ToAccountID IN NUMBER,  
 p\_Amount IN NUMBER  
) IS  
 v\_FromBalance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_FromBalance FROM Accounts WHERE AccountID = p\_FromAccountID FOR UPDATE;  
 IF v\_FromBalance < p\_Amount THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds.');  
 END IF;  
 UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_FromAccountID;  
 UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_ToAccountID;  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);  
END;

## Scenario 2: UpdateSalary Procedure

CREATE OR REPLACE PROCEDURE UpdateSalary(  
 p\_EmployeeID IN NUMBER,  
 p\_Percent IN NUMBER  
) IS  
BEGIN  
 UPDATE Employees  
 SET Salary = Salary + (Salary \* p\_Percent / 100)  
 WHERE EmployeeID = p\_EmployeeID;  
 IF SQL%ROWCOUNT = 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID not found.');  
 END IF;  
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error updating salary: ' || SQLERRM);  
 ROLLBACK;  
END;

## Scenario 3: AddNewCustomer Procedure

CREATE OR REPLACE PROCEDURE AddNewCustomer(  
 p\_CustomerID IN NUMBER,  
 p\_Name IN VARCHAR2,  
 p\_DOB IN DATE,  
 p\_Balance IN NUMBER  
) IS  
BEGIN  
 INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)  
 VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);  
 COMMIT;  
EXCEPTION  
 WHEN DUP\_VAL\_ON\_INDEX THEN  
 DBMS\_OUTPUT.PUT\_LINE('Customer already exists.');  
 WHEN OTHERS THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
 ROLLBACK;  
END;

# Exercise 3: Stored Procedures

## Scenario 1: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
 UPDATE Accounts  
 SET Balance = Balance + (Balance \* 0.01)  
 WHERE AccountType = 'Savings';  
END;

## Scenario 2: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(  
 p\_Department IN VARCHAR2,  
 p\_BonusPercent IN NUMBER  
) IS  
BEGIN  
 UPDATE Employees  
 SET Salary = Salary + (Salary \* p\_BonusPercent / 100)  
 WHERE Department = p\_Department;  
END;

## Scenario 3: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds(  
 p\_From IN NUMBER,  
 p\_To IN NUMBER,  
 p\_Amount IN NUMBER  
) IS  
 v\_Balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_From FOR UPDATE;  
 IF v\_Balance < p\_Amount THEN  
 RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient Balance');  
 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;  
EXCEPTION  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
END;

# Exercise 4: Functions

## Scenario 1: CalculateAge

CREATE OR REPLACE FUNCTION CalculateAge(p\_DOB DATE) RETURN NUMBER IS  
BEGIN  
 RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);  
END;

## Scenario 2: CalculateMonthlyInstallment

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(  
 p\_Amount IN NUMBER,  
 p\_Rate IN NUMBER,  
 p\_Years IN NUMBER  
) RETURN NUMBER IS  
 v\_MonthlyRate NUMBER := p\_Rate / (12 \* 100);  
 v\_Months NUMBER := p\_Years \* 12;  
BEGIN  
 RETURN (p\_Amount \* v\_MonthlyRate) / (1 - POWER(1 + v\_MonthlyRate, -v\_Months));  
END;

# Scenario 3: HasSufficientBalance

CREATE OR REPLACE FUNCTION HasSufficientBalance(  
 p\_AccountID IN NUMBER,  
 p\_Amount IN NUMBER  
) RETURN BOOLEAN IS  
 v\_Balance NUMBER;  
BEGIN  
 SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_AccountID;  
 RETURN v\_Balance >= p\_Amount;  
EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN  
 RETURN FALSE;  
END;

**Exercise 5: Triggers**

**Scenario 1: UpdateCustomerLastModified**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

# Scenario 2: LogTransaction

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

LogDate DATE,

Message VARCHAR2(255)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, LogDate, Message)

VALUES (:NEW.TransactionID, SYSDATE, 'Transaction logged.');

END;

# 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, 'Withdrawal exceeds balance');

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

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

END IF;

END;

**Exercise 6: Cursors**

# Scenario 1: GenerateMonthlyStatements

DECLARE

CURSOR txn\_cursor IS

SELECT AccountID, Amount, TransactionType, TransactionDate

FROM Transactions

WHERE TRUNC(TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

BEGIN

FOR txn IN txn\_cursor LOOP

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

', ' || txn.TransactionType ||

', Amount: ' || txn.Amount ||

', Date: ' || txn.TransactionDate);

END LOOP;

END;

# Scenario 2: ApplyAnnualFee

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID FROM Accounts;

BEGIN

FOR acc IN acc\_cursor LOOP

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

# Scenario 3: UpdateLoanInterestRates

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR loan IN loan\_cursor LOOP

UPDATE Loans

SET InterestRate = loan.InterestRate + 0.5

WHERE LoanID = loan.LoanID;

END LOOP;

COMMIT;

END;

**Exercise 7: Packages**

# Scenario 1: CustomerManagement Package

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);

FUNCTION GetBalance(p\_ID NUMBER) RETURN NUMBER;

END CustomerManagement;

/

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)

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

END;

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

BEGIN

UPDATE Customers SET Name = p\_Name, LastModified = SYSDATE WHERE CustomerID = p\_ID;

END;

FUNCTION GetBalance(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;

# Scenario 2: EmployeeManagement Package

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_ID NUMBER, p\_Name VARCHAR2, p\_Pos VARCHAR2, p\_Salary NUMBER, p\_Dept VARCHAR2);

PROCEDURE UpdateEmployee(p\_ID NUMBER, p\_Salary NUMBER);

FUNCTION AnnualSalary(p\_ID NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_ID NUMBER, p\_Name VARCHAR2, p\_Pos VARCHAR2, p\_Salary NUMBER, p\_Dept VARCHAR2) IS

BEGIN

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

VALUES (p\_ID, p\_Name, p\_Pos, p\_Salary, p\_Dept, SYSDATE);

END;

PROCEDURE UpdateEmployee(p\_ID NUMBER, p\_Salary NUMBER) IS

BEGIN

UPDATE Employees SET Salary = p\_Salary WHERE EmployeeID = p\_ID;

END;

FUNCTION AnnualSalary(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;

# Scenario 3: AccountOperations Package

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_ID NUMBER, p\_CustID NUMBER, p\_Type VARCHAR2, p\_Balance NUMBER);

PROCEDURE CloseAccount(p\_ID NUMBER);

FUNCTION GetTotalBalance(p\_CustID NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_ID NUMBER, p\_CustID NUMBER, p\_Type VARCHAR2, p\_Balance NUMBER) IS

BEGIN

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

VALUES (p\_ID, p\_CustID, p\_Type, p\_Balance, SYSDATE);

END;

PROCEDURE CloseAccount(p\_ID NUMBER) IS

BEGIN

DELETE FROM Accounts WHERE AccountID = p\_ID;

END;

FUNCTION GetTotalBalance(p\_CustID NUMBER) RETURN NUMBER IS

v\_Total NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_Total FROM Accounts WHERE CustomerID = p\_CustID;

RETURN NVL(v\_Total, 0);

END;

END AccountOperations;