**WEEK-2-PLSQL**

**Answers for all the exercises are merged into a single program.**

-- Control Structures

-- Scenario 1: Apply a 1% discount to loan interest rates for customers above 60 years old

BEGIN

FOR rec IN (SELECT CustomerID, DOB FROM Customers) LOOP

IF EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM rec.DOB) > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

-- Scenario 2: Set IsVIP to TRUE for customers with a balance over $10,000

BEGIN

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

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

-- Scenario 3: Print reminders for loans due in the next 30 days

BEGIN

FOR rec IN (SELECT LoanID, CustomerID, EndDate FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + INTERVAL '30' DAY) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID || ' for Customer ID ' || rec.CustomerID || ' is due on ' || rec.EndDate);

END LOOP;

END;

/

-- Error Handling

-- Scenario 1: SafeTransferFunds

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

BEGIN

-- Check balance before transfer

IF (SELECT Balance FROM Accounts WHERE AccountID = p\_from\_account) < p\_amount THEN

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

ELSE

-- Transfer funds

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;

END IF;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred during fund transfer: ' || SQLERRM);

ROLLBACK;

END;

/

-- Scenario 2: UpdateSalary

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_increase\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID does not exist');

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

-- Scenario 3: AddNewCustomer

CREATE OR REPLACE PROCEDURE AddNewCustomer (

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

END;

/

-- Stored Procedures

-- Scenario 1: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01);

COMMIT;

END;

/

-- Scenario 2: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

END;

/

-- Scenario 3: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

BEGIN

IF (SELECT Balance FROM Accounts WHERE AccountID = p\_from\_account) < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient balance for transfer');

ELSE

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;

END IF;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

-- Functions

-- Scenario 1: CalculateAge

CREATE OR REPLACE FUNCTION CalculateAge (p\_dob DATE) RETURN NUMBER IS

BEGIN

RETURN EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM p\_dob);

END;

/

-- Scenario 2: CalculateMonthlyInstallment

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_duration\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_total\_months NUMBER;

BEGIN

v\_monthly\_rate := p\_interest\_rate / 12 / 100;

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

RETURN p\_loan\_amount \* (v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_total\_months)) / (POWER(1 + v\_monthly\_rate, v\_total\_months) - 1);

END;

/

-- Scenario 3: HasSufficientBalance

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;

/

-- 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 OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, TransactionDate, Amount, TransactionType)

VALUES (:NEW.TransactionID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType);

END;

/

-- Scenario 3: CheckTransactionRules

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > (SELECT Balance FROM Accounts WHERE AccountID = :NEW.AccountID) THEN

RAISE\_APPLICATION\_ERROR(-20004, 'Withdrawal amount exceeds balance');

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20005, 'Deposit amount must be positive');

END IF;

END IF;

END;

/

-- Cursors

-- Scenario 1: GenerateMonthlyStatements

DECLARE

CURSOR c\_transactions IS

SELECT DISTINCT CustomerID FROM Transactions

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

AND EXTRACT(YEAR FROM TransactionDate) = EXTRACT(YEAR FROM SYSDATE);

v\_customer\_id Customers.CustomerID%TYPE;

BEGIN

FOR rec IN c\_transactions LOOP

DBMS\_OUTPUT.PUT\_LINE('Generating statement for Customer ID: ' || rec.CustomerID);

-- Add more detailed logic here to generate statements

END LOOP;

END;

/

-- Scenario 2: ApplyAnnualFee

DECLARE

CURSOR c\_accounts IS

SELECT AccountID, Balance FROM Accounts;

BEGIN

FOR rec IN c\_accounts LOOP

UPDATE Accounts

SET Balance = Balance - 50 -- Annual maintenance fee

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

END;

/

-- Scenario 3: UpdateLoanInterestRates

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate FROM Loans;

BEGIN

FOR rec IN c\_loans LOOP

UPDATE Loans

SET InterestRate = InterestRate \* 1.02 -- Applying new policy, increase by 2%

WHERE LoanID = rec.LoanID;

END LOOP;

COMMIT;

END;

/

-- Packages

-- Scenario 1: CustomerManagement

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddNewCustomer(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_dob IN DATE, p\_balance IN NUMBER);

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_dob IN DATE);

FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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

COMMIT;

END;

PROCEDURE UpdateCustomerDetails(p\_customer\_id IN NUMBER, p\_name IN VARCHAR2, p\_dob IN DATE) IS

BEGIN

UPDATE Customers

SET Name = p\_name, DOB = p\_dob, LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

COMMIT;

END;

FUNCTION GetCustomerBalance(p\_customer\_id IN NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END;

END;

/

-- Scenario 2: EmployeeManagement

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id IN NUMBER, p\_salary IN NUMBER);

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER) RETURN NUMBER;

END;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2) IS

BEGIN

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

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

COMMIT;

END;

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

BEGIN

UPDATE Employees

SET Salary = p\_salary

WHERE EmployeeID = p\_employee\_id;

COMMIT;

END;

FUNCTION CalculateAnnualSalary(p\_employee\_id IN NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

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

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

END;

/

-- Scenario 3: AccountOperations

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_account\_type IN VARCHAR2, p\_balance IN NUMBER);

PROCEDURE CloseAccount(p\_account\_id IN NUMBER);

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

END;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_account\_type IN VARCHAR2, p\_balance IN NUMBER) IS

BEGIN

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

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

COMMIT;

END;

PROCEDURE CloseAccount(p\_account\_id IN NUMBER) IS

BEGIN

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

COMMIT;

END;

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

v\_total\_balance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total\_balance

FROM Accounts WHERE CustomerID = p\_customer\_id;

RETURN v\_total\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END;

END;

/