**PAKHI SHARMA**

**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* + **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**Scenario 1**

BEGIN

FOR r IN (SELECT \* FROM Customers c JOIN Loans l ON c.CustomerID = l.CustomerID) LOOP

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

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

END IF;

END LOOP;

END;

**Scenario 2**

BEGIN

FOR r IN (SELECT \* FROM Customers) LOOP

IF r.Balance > 10000 THEN

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

END IF;

END LOOP;

END;

**Scenario 3**

BEGIN

FOR r IN (SELECT \* FROM Loans WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

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

END LOOP;

END;

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

* + **Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**Scenario 2:** Manage errors when updating employee salaries.

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**Scenario 3:** Ensure data integrity when adding a new customer.

* + **Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

**Scenario 1**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(p\_FromAcc NUMBER, p\_ToAcc NUMBER, p\_Amount NUMBER) IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_FromAcc;

IF v\_Balance < p\_Amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_Amount WHERE AccountID = p\_FromAcc;

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_ToAcc;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

**Scenario 2**

CREATE OR REPLACE PROCEDURE UpdateSalary(p\_EmpID NUMBER, p\_Percent NUMBER) IS

BEGIN

UPDATE Employees SET Salary = Salary + (Salary \* p\_Percent / 100)

WHERE EmployeeID = p\_EmpID;

IF SQL%ROWCOUNT = 0 THEN

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

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

END;

**Scenario 3**

CREATE OR REPLACE PROCEDURE AddNewCustomer(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;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ID already exists.');

WHEN OTHERS THEN

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

END;

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + **Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**Scenario 1**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END;

**Scenario 2**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p\_Dept VARCHAR2, p\_BonusPct NUMBER) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_Dept;

COMMIT;

END;

**Scenario 3**

CREATE OR REPLACE PROCEDURE TransferFunds(p\_From NUMBER, p\_To NUMBER, p\_Amount NUMBER) IS

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 funds');

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;

END;

**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

* + **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

**Scenario 2:** The bank needs to compute the monthly installment for a loan.

* + **Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

* + **Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

**Scenario 1**

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

v\_Age NUMBER;

BEGIN

v\_Age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

RETURN v\_Age;

END;

**Scenario 2**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_LoanAmount NUMBER,

p\_InterestRate NUMBER,

p\_Years NUMBER

) RETURN NUMBER IS

v\_MonthlyRate NUMBER := p\_InterestRate / 12 / 100;

v\_Months NUMBER := p\_Years \* 12;

v\_Installment NUMBER;

BEGIN

v\_Installment := p\_LoanAmount \* v\_MonthlyRate / (1 - POWER(1 + v\_MonthlyRate, -v\_Months));

RETURN v\_Installment;

END;

**Scenario 3**

CREATE OR REPLACE FUNCTION HasSufficientBalance(p\_AccID NUMBER, p\_Amount NUMBER) RETURN BOOLEAN IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_AccID;

RETURN v\_Balance >= p\_Amount;

END;

**Exercise 5: Triggers**

**Scenario 1:** Automatically update the last modified date when a customer's record is updated.

* + **Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

**Scenario 2:** Maintain an audit log for all transactions.

* + **Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

**Scenario 3:** Enforce business rules on deposits and withdrawals.

* + **Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

**Scenario 1**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

**Scenario 2**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog(TransactionID, LogDate, Message)

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

END;

**Scenario 3**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_Balance NUMBER;

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

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

IF :NEW.Amount > v\_Balance THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal exceeds balance');

END IF;

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

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

END IF;

END;

**Exercise 6: Cursors**

**Scenario 1:** Generate monthly statements for all customers.

* + **Question:** Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

**Scenario 2:** Apply annual fee to all accounts.

* + **Question:** Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

**Scenario 3:** Update the interest rate for all loans based on a new policy.

* + **Question:** Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy.

**Scenario 1**

DECLARE

CURSOR c\_trans IS

SELECT \* FROM Transactions WHERE TRUNC(TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

v\_trans c\_trans%ROWTYPE;

BEGIN

OPEN c\_trans;

LOOP

FETCH c\_trans INTO v\_trans;

EXIT WHEN c\_trans%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Customer statement for Account: ' || v\_trans.AccountID || ', Amount: ' || v\_trans.Amount);

END LOOP;

CLOSE c\_trans;

END;

**Scenario 2**

DECLARE

CURSOR c\_accounts IS SELECT AccountID, Balance FROM Accounts;

v\_account c\_accounts%ROWTYPE;

BEGIN

OPEN c\_accounts;

LOOP

FETCH c\_accounts INTO v\_account;

EXIT WHEN c\_accounts%NOTFOUND;

UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = v\_account.AccountID;

END LOOP;

CLOSE c\_accounts;

COMMIT;

END;

**Scenario 3**

DECLARE

CURSOR c\_loans IS SELECT LoanID, InterestRate FROM Loans;

v\_loan c\_loans%ROWTYPE;

BEGIN

OPEN c\_loans;

LOOP

FETCH c\_loans INTO v\_loan;

EXIT WHEN c\_loans%NOTFOUND;

UPDATE Loans SET InterestRate = v\_loan.InterestRate + 0.5 WHERE LoanID = v\_loan.LoanID;

END LOOP;

CLOSE c\_loans;

COMMIT;

END;

**Exercise 7: Packages**

**Scenario 1:** Group all customer-related procedures and functions into a package.

* + **Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

**Scenario 2:** Create a package to manage employee data.

* + **Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

**Scenario 3:** Group all account-related operations into a package.

* + **Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

**Scenario 1**

CREATE OR REPLACE PACKAGE CustomerManagement IS

PROCEDURE AddCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_DOB DATE, p\_Balance NUMBER);

PROCEDURE UpdateCustomer(p\_ID NUMBER, p\_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 VALUES(p\_ID, p\_Name, p\_DOB, p\_Balance, SYSDATE, 'FALSE');

END;

PROCEDURE UpdateCustomer(p\_ID NUMBER, p\_Balance NUMBER) IS

BEGIN

UPDATE Customers SET Balance = p\_Balance WHERE CustomerID = p\_ID;

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;

**Scenario 2**

CREATE OR REPLACE PACKAGE EmployeeManagement IS

PROCEDURE HireEmployee(p\_ID NUMBER, p\_Name VARCHAR2, p\_Pos VARCHAR2, p\_Sal NUMBER, p\_Dept VARCHAR2, p\_Date DATE);

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

FUNCTION CalcAnnualSalary(p\_ID NUMBER) RETURN NUMBER;

END EmployeeManagement;

CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

PROCEDURE HireEmployee(p\_ID NUMBER, p\_Name VARCHAR2, p\_Pos VARCHAR2, p\_Sal NUMBER, p\_Dept VARCHAR2, p\_Date DATE) IS

BEGIN

INSERT INTO Employees VALUES(p\_ID, p\_Name, p\_Pos, p\_Sal, p\_Dept, p\_Date);

END;

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

BEGIN

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

END;

FUNCTION CalcAnnualSalary(p\_ID NUMBER) RETURN NUMBER IS

v\_Sal NUMBER;

BEGIN

SELECT Salary INTO v\_Sal FROM Employees WHERE EmployeeID = p\_ID;

RETURN v\_Sal \* 12;

END;

END EmployeeManagement;

**Scenario 3**

CREATE OR REPLACE PACKAGE AccountOperations IS

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 IS

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

BEGIN

INSERT INTO Accounts 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 v\_Total;

END;

END AccountOperations;

**Schema to be Created**

*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*

*);*

**Example Scripts for Sample Data Insertion**

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

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

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

*VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, 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, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));*

*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'));*

**SCHEMA CREATION**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE,

IsVIP VARCHAR2(5)

);

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

);

CREATE TABLE AuditLog (

LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

LogDate DATE,

Message VARCHAR2(255)

);