**CREATION OF THE TABLE AND INSERTION OF VALUES :**

create database cts;use cts;CREATE TABLE Customers ( CustomerID INT PRIMARY KEY AUTO\_INCREMENT, Name VARCHAR(100), DOB DATE, Balance DECIMAL(10, 2), LastModified TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP);CREATE TABLE Accounts ( AccountID INT PRIMARY KEY AUTO\_INCREMENT, CustomerID INT, AccountType VARCHAR(20), Balance DECIMAL(10, 2), LastModified TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP, FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID));CREATE TABLE Transactions ( TransactionID INT PRIMARY KEY AUTO\_INCREMENT, AccountID INT, TransactionDate DATE, Amount DECIMAL(10, 2), TransactionType VARCHAR(10), FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID));CREATE TABLE Loans ( LoanID INT PRIMARY KEY AUTO\_INCREMENT, CustomerID INT, LoanAmount DECIMAL(10, 2), InterestRate DECIMAL(5, 2), StartDate DATE, EndDate DATE, FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID));CREATE TABLE Employees ( EmployeeID INT PRIMARY KEY AUTO\_INCREMENT, Name VARCHAR(100), Position VARCHAR(50), Salary DECIMAL(10, 2), Department VARCHAR(50), HireDate DATE);

INSERT INTO Customers (Name, DOB, Balance, LastModified)VALUES ('John Doe', '1985-05-15', 1000, NOW());INSERT INTO Customers (Name, DOB, Balance, LastModified)VALUES ('Jane Smith', '1990-07-20', 1500, NOW());INSERT INTO Accounts (CustomerID, AccountType, Balance, LastModified)VALUES (1, 'Savings', 1000, NOW());INSERT INTO Accounts (CustomerID, AccountType, Balance, LastModified)VALUES (2, 'Checking', 1500, NOW());INSERT INTO Transactions (AccountID, TransactionDate, Amount, TransactionType)VALUES (1, CURDATE(), 200, 'Deposit');INSERT INTO Transactions (AccountID, TransactionDate, Amount, TransactionType)VALUES (2, CURDATE(), 300, 'Withdrawal');INSERT INTO Loans (CustomerID, LoanAmount, InterestRate, StartDate, EndDate)VALUES (1, 5000, 5, CURDATE(), DATE\_ADD(CURDATE(), INTERVAL 60 MONTH));INSERT INTO Employees (Name, Position, Salary, Department, HireDate)VALUES ('Alice Johnson', 'Manager', 70000, 'HR', '2015-06-15');INSERT INTO Employees (Name, Position, Salary, Department, HireDate)VALUES ('Bob Brown', 'Developer', 60000, 'IT', '2017-03-20');

**Exercise 1: Control Structures**

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

use cts;ALTER TABLE Customers ADD IsVIP BOOLEAN DEFAULT FALSE;

DELIMITER //CREATE PROCEDURE ApplyDiscountToSeniorCustomers()BEGIN DECLARE done INT DEFAULT FALSE; DECLARE customerID INT; DECLARE age INT; DECLARE cursor1 CURSOR FOR SELECT CustomerID, YEAR(CURDATE()) - YEAR(DOB) AS age FROM Customers; DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE; OPEN cursor1; read\_loop: LOOP FETCH cursor1 INTO customerID, age; IF done THEN LEAVE read\_loop; END IF; IF age > 60 THEN UPDATE Loans SET InterestRate = InterestRate - 1 WHERE CustomerID = customerID; END IF; END LOOP; CLOSE cursor1;END //DELIMITER ;

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

DELIMITER //

CREATE PROCEDURE PromoteVIPCustomers()BEGIN DECLARE done INT DEFAULT FALSE; DECLARE customerID INT; DECLARE balance DECIMAL(10, 2); DECLARE cursor1 CURSOR FOR SELECT CustomerID, Balance FROM Customers; DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE; OPEN cursor1; read\_loop: LOOP FETCH cursor1 INTO customerID, balance; IF done THEN LEAVE read\_loop; END IF; IF balance > 10000 THEN UPDATE Customers SET IsVIP = TRUE WHERE CustomerID = customerID; END IF; END LOOP; CLOSE cursor1;END //DELIMITER ;

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

DELIMITER //

CREATE PROCEDURE SendLoanDueReminders()BEGIN DECLARE done INT DEFAULT FALSE; DECLARE loanID INT; DECLARE customerID INT; DECLARE customerName VARCHAR(100); DECLARE endDate DATE; DECLARE cursor1 CURSOR FOR SELECT LoanID, CustomerID, EndDate FROM Loans WHERE EndDate BETWEEN CURDATE() AND DATE\_ADD(CURDATE(), INTERVAL 30 DAY); DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE; OPEN cursor1; read\_loop: LOOP FETCH cursor1 INTO loanID, customerID, endDate; IF done THEN LEAVE read\_loop; END IF; SELECT Name INTO customerName FROM Customers WHERE CustomerID = customerID; SELECT CONCAT('Reminder: Customer ', customerName, ' has a loan due on ', endDate) AS ReminderMessage; END LOOP; CLOSE cursor1;END //DELIMITER ;CALL ApplyDiscountToSeniorCustomers();CALL PromoteVIPCustomers();CALL SendLoanDueReminders();select \* from Customers;

**Exercise 2: Error Handling**

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

use cts;

DELIMITER //CREATE PROCEDURE SafeTransferFunds(IN fromAccountID INT, IN toAccountID INT, IN transferAmount DECIMAL(10, 2))BEGIN DECLARE insufficientFundsCondition CONDITION FOR SQLSTATE '45000'; DECLARE EXIT HANDLER FOR insufficientFundsCondition BEGIN -- Log the error message INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (CONCAT('Insufficient funds for transfer from account ', fromAccountID), NOW()); ROLLBACK; END; DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN -- Log the general error message INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (CONCAT('Error during transfer from account ', fromAccountID, ' to account ', toAccountID), NOW()); ROLLBACK; END; START TRANSACTION;

-- Check if there are sufficient funds IF (SELECT Balance FROM Accounts WHERE AccountID = fromAccountID) < transferAmount THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient funds'; END IF; -- Deduct from the source account UPDATE Accounts SET Balance = Balance - transferAmount WHERE AccountID = fromAccountID; -- Add to the destination account UPDATE Accounts SET Balance = Balance + transferAmount WHERE AccountID = toAccountID; COMMIT;END //DELIMITER ;

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

DELIMITER //CREATE PROCEDURE UpdateSalary(IN employeeID INT, IN percentageIncrease DECIMAL(5, 2))BEGIN DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN -- Log the error message INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (CONCAT('Error updating salary for employee ', employeeID), NOW()); END; -- Update the salary UPDATE Employees SET Salary = Salary + (Salary \* (percentageIncrease / 100)) WHERE EmployeeID = employeeID; IF ROW\_COUNT() = 0 THEN -- Employee ID does not exist, handle the exception INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (CONCAT('Employee ID ', employeeID, ' does not exist'), NOW()); END IF;END //DELIMITER ;

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

DELIMITER //CREATE PROCEDURE AddNewCustomer(IN customerID INT, IN customerName VARCHAR(100), IN customerDOB DATE, IN initialBalance DECIMAL(10, 2))BEGIN DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN -- Log the error message INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (CONCAT('Error adding new customer with ID ', customerID), NOW()); END; -- Insert the new customer INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified, IsVIP) VALUES (customerID, customerName, customerDOB, initialBalance, NOW(), FALSE); IF ROW\_COUNT() = 0 THEN -- Customer ID already exists, handle the exception INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (CONCAT('Customer ID ', customerID, ' already exists'), NOW()); END IF;END //DELIMITER ;CALL SafeTransferFunds(1, 2, 500);CALL UpdateSalary(1, 10);CALL AddNewCustomer(3, 'Michael Scott', '1964-03-15', 2000);CREATE TABLE ErrorLog ( ErrorID INT PRIMARY KEY AUTO\_INCREMENT, ErrorMessage VARCHAR(255), ErrorDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP);select \* from ErrorLog;select \* from Customers;

**Exercise 3: Stored Procedures:**

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

use cts;DELIMITER //CREATE PROCEDURE ProcessMonthlyInterest()BEGIN DECLARE done INT DEFAULT FALSE; DECLARE accountID INT; DECLARE balance DECIMAL(10, 2); DECLARE interestRate DECIMAL(5, 2) DEFAULT 0.01; DECLARE cursor1 CURSOR FOR SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings'; DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE; OPEN cursor1; read\_loop: LOOP FETCH cursor1 INTO accountID, balance; IF done THEN LEAVE read\_loop; END IF; -- Update the balance with interest UPDATE Accounts SET Balance = Balance + (Balance \* interestRate) WHERE AccountID = accountID; END LOOP; CLOSE cursor1;END //DELIMITER ;

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

DELIMITER //CREATE PROCEDURE UpdateEmployeeBonus(IN departmentName VARCHAR(50), IN bonusPercentage DECIMAL(5, 2))BEGIN DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN -- Log the error message INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (CONCAT('Error updating bonus for department ', departmentName), NOW()); END; -- Update the salary with the bonus UPDATE Employees SET Salary = Salary + (Salary \* (bonusPercentage / 100)) WHERE Department = departmentName;END //DELIMITER ;

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

DELIMITER //CREATE PROCEDURE TransferFunds(IN fromAccountID INT, IN toAccountID INT, IN transferAmount DECIMAL(10, 2))BEGIN DECLARE insufficientFundsCondition CONDITION FOR SQLSTATE '45000'; DECLARE EXIT HANDLER FOR insufficientFundsCondition BEGIN -- Log the error message INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (CONCAT('Insufficient funds for transfer from account ', fromAccountID), NOW()); ROLLBACK; END; DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN -- Log the general error message INSERT INTO ErrorLog (ErrorMessage, ErrorDate) VALUES (CONCAT('Error during transfer from account ', fromAccountID, ' to account ', toAccountID), NOW()); ROLLBACK; END; START TRANSACTION; -- Check if there are sufficient funds IF (SELECT Balance FROM Accounts WHERE AccountID = fromAccountID) < transferAmount THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient funds'; END IF; -- Deduct from the source account UPDATE Accounts SET Balance = Balance - transferAmount WHERE AccountID = fromAccountID; -- Add to the destination account UPDATE Accounts SET Balance = Balance + transferAmount WHERE AccountID = toAccountID; COMMIT;END //DELIMITER ;CALL ProcessMonthlyInterest();CALL UpdateEmployeeBonus('IT', 5);CALL TransferFunds(1, 2, 200);select \* from Employees;

**Exercise 4: Functions**

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

use cts;DELIMITER //CREATE FUNCTION CalculateAge(dob DATE) RETURNS INTDETERMINISTICBEGIN DECLARE age INT; SET age = YEAR(CURDATE()) - YEAR(dob); IF MONTH(CURDATE()) < MONTH(dob) OR (MONTH(CURDATE()) = MONTH(dob) AND DAY(CURDATE()) < DAY(dob)) THEN SET age = age - 1; END IF; RETURN age;END //DELIMITER ;

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

DELIMITER //CREATE FUNCTION CalculateMonthlyInstallment(loanAmount DECIMAL(10, 2), annualInterestRate DECIMAL(5, 2), loanDurationYears INT) RETURNS DECIMAL(10, 2)DETERMINISTICBEGIN DECLARE monthlyInterestRate DECIMAL(5, 4); DECLARE numberOfPayments INT; DECLARE monthlyInstallment DECIMAL(10, 2); SET monthlyInterestRate = annualInterestRate / 12 / 100; SET numberOfPayments = loanDurationYears \* 12; IF monthlyInterestRate > 0 THEN SET monthlyInstallment = loanAmount \* monthlyInterestRate / (1 - POW(1 + monthlyInterestRate, -numberOfPayments)); ELSE SET monthlyInstallment = loanAmount / numberOfPayments; END IF; RETURN monthlyInstallment;END //DELIMITER ;

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

DELIMITER //CREATE FUNCTION hasSufficientBal(accountID INT, amount DECIMAL(10, 2)) RETURNS BOOLEANREADS SQL DATABEGIN DECLARE balance DECIMAL(10, 2); -- Ensure only one row is selected by using LIMIT 1 SELECT Balance INTO balance FROM Accounts WHERE AccountID = accountID LIMIT 1; IF balance IS NULL THEN -- Handle the case where no account is found RETURN FALSE; ELSEIF balance >= amount THEN RETURN TRUE; ELSE RETURN FALSE; END IF;END //DELIMITER ;SELECT CalculateAge('1985-05-15');SELECT CalculateMonthlyInstallment(10000, 5, 3);SELECT hasSufficientBal(1, 500);

**Exercise 5: Triggers**

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

use cts;DELIMITER //CREATE TRIGGER UpdateCustomerLastModifiedBEFORE UPDATE ON CustomersFOR EACH ROWBEGIN SET NEW.LastModified = NOW();END //DELIMITER ;CREATE TABLE AuditLog ( AuditID INT AUTO\_INCREMENT PRIMARY KEY, TransactionID INT, AccountID INT, TransactionDate DATETIME, Amount DECIMAL(10, 2), TransactionType VARCHAR(10), ChangeDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP);

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

DELIMITER //CREATE TRIGGER LogTransactionAFTER INSERT ON TransactionsFOR EACH ROWBEGIN INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (NEW.TransactionID, NEW.AccountID, NEW.TransactionDate, NEW.Amount, NEW.TransactionType);END //DELIMITER ;

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

DELIMITER //CREATE TRIGGER CheckTransactionRulesBEFORE INSERT ON TransactionsFOR EACH ROWBEGIN DECLARE accountBalance DECIMAL(10, 2); -- Check for withdrawals exceeding balance IF NEW.TransactionType = 'Withdrawal' THEN -- Get the current balance of the account SELECT Balance INTO accountBalance FROM Accounts WHERE AccountID = NEW.AccountID; -- If there is insufficient balance, raise an error IF accountBalance < NEW.Amount THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient funds for withdrawal'; END IF; END IF; -- Check for deposits with a non-positive amount IF NEW.TransactionType = 'Deposit' AND NEW.Amount <= 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Deposit amount must be positive'; END IF;END //DELIMITER ;SHOW TRIGGERS;INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)VALUES (1, 1, NOW(), 1500, 'Withdrawal'); -- an error due to insufficient fundsINSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)VALUES (2, 1, NOW(), -100, 'Deposit'); -- an error due to negative deposit amountDESCRIBE Transactions;INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)VALUES (3, 1, '2024-08-07 17:03:18', 500, 'Deposit');

**Exercise 6: Cursors**

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

CURSOR customer\_cursor IS

SELECT DISTINCT CustomerID

FROM Transactions

WHERE MONTH(TransactionDate) = MONTH(SYSDATE)

AND YEAR(TransactionDate) = YEAR(SYSDATE);

v\_customerID Customers.CustomerID%TYPE;

v\_transactionDate Transactions.TransactionDate%TYPE;

v\_amount Transactions.Amount%TYPE;

v\_transactionType Transactions.TransactionType%TYPE;

BEGIN

OPEN customer\_cursor;

LOOP

FETCH customer\_cursor INTO v\_customerID;

EXIT WHEN customer\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Statement for Customer ID: ' || v\_customerID);

DBMS\_OUTPUT.PUT\_LINE('Date\t\t\tAmount\tType');

DBMS\_OUTPUT.PUT\_LINE('-------------------------------------');

FOR trans IN (SELECT TransactionDate, Amount, TransactionType

FROM Transactions

WHERE CustomerID = v\_customerID

AND MONTH(TransactionDate) = MONTH(SYSDATE)

AND YEAR(TransactionDate) = YEAR(SYSDATE)) LOOP

DBMS\_OUTPUT.PUT\_LINE(trans.TransactionDate || '\t' || trans.Amount || '\t' || trans.TransactionType);

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('-------------------------------------');

END LOOP;

CLOSE customer\_cursor;

END;

/

### **Scenario 2: Apply Annual Fee to All Accounts**

DECLARE

CURSOR account\_cursor IS

SELECT AccountID, Balance

FROM Accounts;

v\_accountID Accounts.AccountID%TYPE;

v\_balance Accounts.Balance%TYPE;

v\_annualFee DECIMAL(10, 2) := 50.00; -- Example annual fee

BEGIN

OPEN account\_cursor;

LOOP

FETCH account\_cursor INTO v\_accountID, v\_balance;

EXIT WHEN account\_cursor%NOTFOUND;

-- Deduct annual fee from balance

UPDATE Accounts

SET Balance = v\_balance - v\_annualFee

WHERE AccountID = v\_accountID;

-- Optionally, print the result

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || v\_accountID || ' - New Balance: ' || (v\_balance - v\_annualFee));

END LOOP;

CLOSE account\_cursor;

END;

/

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

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loanID Loans.LoanID%TYPE;

v\_oldInterestRate Loans.InterestRate%TYPE;

v\_newInterestRate DECIMAL(5, 2); -- Example new interest rate based on policy

BEGIN

OPEN loan\_cursor;

LOOP

FETCH loan\_cursor INTO v\_loanID, v\_oldInterestRate;

EXIT WHEN loan\_cursor%NOTFOUND;

-- Compute the new interest rate based on the policy

v\_newInterestRate := v\_oldInterestRate + 0.5; -- Example: increase by 0.5%

-- Update the interest rate

UPDATE Loans

SET InterestRate = v\_newInterestRate

WHERE LoanID = v\_loanID;

-- Optionally, print the result

DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || v\_loanID || ' - Old Rate: ' || v\_oldInterestRate || ' - New Rate: ' || v\_newInterestRate);

END LOOP;

CLOSE loan\_cursor;

END;

/

**Exercise 7: Packages**

**Scenario 1: Customer Management**  
  
**Package Specification for CustomerManagement**

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

PROCEDURE UpdateCustomer(p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE, p\_Balance IN NUMBER);

FUNCTION GetCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER;

END CustomerManagement;

/  
  
**Package Body for CustomerManagement**CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_Name IN VARCHAR2, p\_DOB IN DATE, p\_Balance IN NUMBER) IS

BEGIN

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

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

END AddCustomer;

PROCEDURE UpdateCustomer(p\_CustomerID IN NUMBER, p\_Name IN VARCHAR2, p\_DOB IN DATE, p\_Balance IN NUMBER) IS

BEGIN

UPDATE Customers

SET Name = p\_Name,

DOB = p\_DOB,

Balance = p\_Balance,

LastModified = SYSDATE

WHERE CustomerID = p\_CustomerID;

END UpdateCustomer;

FUNCTION GetCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Customers

WHERE CustomerID = p\_CustomerID;

RETURN v\_Balance;

END GetCustomerBalance;

END CustomerManagement;

/

**Scenario 2: Create a package to manage employee data.**  
  
**Package Specification for EmployeeManagement**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2, p\_HireDate IN DATE);

PROCEDURE UpdateEmployee(p\_EmployeeID IN NUMBER, p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2, p\_HireDate IN DATE);

FUNCTION CalculateAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

**Package Body for EmployeeManagement**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2, p\_HireDate IN DATE) IS

BEGIN

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

VALUES (p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate);

END HireEmployee;

PROCEDURE UpdateEmployee(p\_EmployeeID IN NUMBER, p\_Name IN VARCHAR2, p\_Position IN VARCHAR2, p\_Salary IN NUMBER, p\_Department IN VARCHAR2, p\_HireDate IN DATE) IS

BEGIN

UPDATE EmployeesSET Name = p\_Name,

Position = p\_Position,

Salary = p\_Salary,

Department = p\_Department,

HireDate = p\_HireDate

WHERE EmployeeID = p\_EmployeeID;

END UpdateEmployee;

FUNCTION CalculateAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

SELECT Salary INTO v\_Salary

FROM Employees

WHERE EmployeeID = p\_EmployeeID;

RETURN v\_Salary \* 12;

END CalculateAnnualSalary;

END EmployeeManagement;

/

**Scenario 3: Group all account-related operations into a package.**  
  
**Package Specification for AccountOperations**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_CustomerID IN NUMBER, p\_AccountType IN VARCHAR2, p\_Balance IN NUMBER);

PROCEDURE CloseAccount(p\_AccountID IN NUMBER);

FUNCTION GetTotalCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER;

END AccountOperations;

/  
  
**Package Body for AccountOperations**CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_CustomerID IN NUMBER, p\_AccountType IN VARCHAR2, p\_Balance IN NUMBER) IS

BEGIN

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

VALUES (p\_CustomerID, p\_AccountType, p\_Balance, SYSDATE);

END OpenAccount;

PROCEDURE CloseAccount(p\_AccountID IN NUMBER) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_AccountID;

END CloseAccount;

FUNCTION GetTotalCustomerBalance(p\_CustomerID IN NUMBER) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN NVL(v\_TotalBalance, 0);

END GetTotalCustomerBalance;

END AccountOperations;

/