**Table Creation:**

*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'));*

**Exercise 1: Control Structures**

**Scenario 1:**

**Query:**

DECLARE

CURSOR customer\_cursor IS

SELECT CustomerID, DOB

FROM Customers;

v\_age INT;

BEGIN

FOR customer\_rec IN customer\_cursor LOOP

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, customer\_rec.DOB) / 12);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = GREATEST(InterestRate - 1, 1)

WHERE CustomerID = customer\_rec.CustomerID

AND InterestRate > 1;

END IF;

END LOOP;

COMMIT;

END;

**Scenario 2:**

**Query:**

ALTER TABLE Customers ADD IsVIP NUMBER(1) DEFAULT 0;

BEGIN

UPDATE Customers

SET IsVIP = 1

WHERE Balance > 10000;

COMMIT;

END;

SELECT CustomerID, Name, Balance, IsVIP

FROM Customers

WHERE IsVIP = 1;

**Scenario 3:**

**Query:**

BEGIN

FOR loan\_rec IN (

SELECT C.Name AS CustomerName, C.CustomerID, 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: Customer ' || loan\_rec.CustomerName ||

' (CustomerID: ' || loan\_rec.CustomerID ||

') has a loan (LoanID: ' || loan\_rec.LoanID ||

') due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD') || '.');

END LOOP;

END;

**Exercise 2: Error Handling**

**Scenario 1:**

**Query:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER) AS

v\_from\_balance NUMBER;

v\_to\_balance NUMBER;

insufficient\_funds EXCEPTION;

PRAGMA EXCEPTION\_INIT(insufficient\_funds, -20001);

BEGIN

SAVEPOINT transfer\_start;

IF p\_amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Transfer amount must be greater than zero.');

END IF;

SELECT Balance INTO v\_from\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id;

IF v\_from\_balance < p\_amount THEN

RAISE insufficient\_funds;

END IF;

SELECT Balance INTO v\_to\_balance

FROM Accounts

WHERE AccountID = p\_to\_account\_id;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully.');

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK TO transfer\_start;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds for the transfer.');

WHEN OTHERS THEN

ROLLBACK TO transfer\_start;

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

END SafeTransferFunds;

**To get the output:**

**Query:**

BEGIN

SafeTransferFunds(1, 2, 100);

END;

**Scenario 2:**

**Query:**

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

v\_current\_salary NUMBER;

employee\_not\_found EXCEPTION;

PRAGMA EXCEPTION\_INIT(employee\_not\_found, -20001);

BEGIN

IF p\_percentage <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Percentage must be greater than zero.');

END IF;

BEGIN

SELECT Salary INTO v\_current\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for EmployeeID: ' || p\_employee\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: EmployeeID ' || p\_employee\_id || ' does not exist.');

WHEN OTHERS THEN

ROLLBACK;

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

END;

END UpdateSalary;

**To get the output:**

**Query:**

BEGIN

UpdateSalary(1, 10);

END;

**Scenario 3:**

**Query:**

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER,

p\_last\_modified IN DATE

) AS

duplicate\_id EXCEPTION;

PRAGMA EXCEPTION\_INIT(duplicate\_id, -1);

BEGIN

BEGIN

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

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, p\_last\_modified);

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Customer added successfully with CustomerID: ' || p\_customer\_id);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

ROLLBACK;

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

WHEN OTHERS THEN

ROLLBACK;

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

END;

END AddNewCustomer;

**To get the Output:**

**Query:**

BEGIN

AddNewCustomer(1, 'Alice Johnson', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

END;

**Exercise 3: Stored Procedures**

**Scenario 1:**

**Query:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END ProcessMonthlyInterest;

**To display the output:**

**Query:**

BEGIN

ProcessMonthlyInterest;

END;

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings';

**Scenario 2:**

**Query:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary \* (1 + p\_bonus\_percentage / 100)

WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salaries updated with a bonus of ' || p\_bonus\_percentage || '% for department: ' || p\_department);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END UpdateEmployeeBonus;

**To display the output:**

**Query:**

BEGIN

UpdateEmployeeBonus('HR', 5);

END;

SELECT EmployeeID, Name, Salary

FROM Employees

WHERE Department = 'HR';

**Scenario 3:**

**Query:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account\_id IN NUMBER,

p\_target\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_source\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account\_id;

IF v\_source\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_source\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_target\_account\_id;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Successfully transferred ' || p\_amount || ' from account ' || p\_source\_account\_id || ' to account ' || p\_target\_account\_id);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END TransferFunds;

**To display the output:**

**Query:**

BEGIN

TransferFunds(1, 2, 100);

END;

SELECT AccountID, Balance

FROM Accounts

WHERE AccountID IN (1, 2);

**Exercise 4: Functions**

**Scenario 1:**

**Query**:

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 CalculateAge;

**Function Calling:**

**Query:**

SELECT CustomerID, Name, DOB, CalculateAge(DOB) AS Age

FROM Customers;

**Scenario 2:**

**Query:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount NUMBER,

p\_annual\_interest\_rate NUMBER,

p\_duration\_years NUMBER

) RETURN NUMBER

IS

v\_monthly\_interest\_rate NUMBER;

v\_total\_payments NUMBER;

v\_monthly\_installment NUMBER;

BEGIN

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

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

v\_monthly\_installment := p\_loan\_amount \* v\_monthly\_interest\_rate \* POWER(1 + v\_monthly\_interest\_rate, v\_total\_payments) / (POWER(1 + v\_monthly\_interest\_rate, v\_total\_payments) - 1);

RETURN v\_monthly\_installment;

END CalculateMonthlyInstallment;

**Function Calling:**

**Query:**

DECLARE

v\_monthly\_installment NUMBER;

BEGIN

v\_monthly\_installment := CalculateMonthlyInstallment(5000, 5, 5);

DBMS\_OUTPUT.PUT\_LINE('Monthly Installment: ' || v\_monthly\_installment);

END;

**Scenario 3:**

**Query:**

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;

IF v\_balance >= p\_amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'An error occurred while checking the balance.');

END HasSufficientBalance;

**Function Calling:**

**Query:**

DECLARE

v\_result BOOLEAN;

BEGIN

v\_result := HasSufficientBalance(1, 500);

IF v\_result THEN

DBMS\_OUTPUT.PUT\_LINE('Account 1 has sufficient balance.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account 1 does not have sufficient balance .');

END IF;

v\_result := HasSufficientBalance(2, 2000);

IF v\_result THEN

DBMS\_OUTPUT.PUT\_LINE('Account 2 has sufficient balance.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Account 2 does not have sufficient balance.');

END IF;

END;

**Exercise 5: Triggers**

**Scenario 1:**

**Query:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

**Updating name:**

**Query:**

UPDATE Customers

SET Name = 'Meenakshi Sharma'

WHERE CustomerID =1;

**To display after update:**

**Query:**

SELECT CustomerID, Name, LastModified

FROM Customers

WHERE CustomerID = 1;

**Scenario 2:**

**Query:**

CREATE TABLE AuditLog (

LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

TransactionID NUMBER,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

ActionDate DATE,

ActionType VARCHAR2(10)

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (

TransactionID,

AccountID,

TransactionDate,

Amount,

TransactionType,

ActionDate,

ActionType

)

VALUES (

:NEW.TransactionID,

:NEW.AccountID,

:NEW.TransactionDate,

:NEW.Amount,

:NEW.TransactionType,

SYSDATE,

'INSERT'

);

END;

**Inserting Values:**

**Query:**

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (3, 1, SYSDATE, 150, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (4, 2, SYSDATE, 250, 'Withdrawal');

**Checking records in audit table:**

**Query:**

SELECT \* FROM AuditLog;

**Scenario 3:**

**Query:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

current\_balance NUMBER;

BEGIN

SELECT Balance INTO current\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF :NEW.TransactionType = 'Withdrawal' THEN

IF :NEW.Amount > current\_balance THEN

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

END IF;

ELSIF :NEW.TransactionType = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

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

END IF;

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'Invalid transaction type.');

END IF;

END;

**Checking with new data in transaction table:**

**Query:**

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (4, 1, SYSDATE, -100, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (6, 1, SYSDATE, 50, 'Withdrawal');

**Exercise 6: Cursors**

**Scenario 1:**

**Query:**

DECLARE

CURSOR transaction\_cursor IS

SELECT t.TransactionID, t.AccountID, t.TransactionDate, t.Amount, t.TransactionType, a.CustomerID

FROM Transactions t

JOIN Accounts a ON t.AccountID = a.AccountID

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

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

transaction\_record transaction\_cursor%ROWTYPE;

current\_customer\_id Accounts.CustomerID%TYPE := NULL;

BEGIN

OPEN transaction\_cursor;

FETCH transaction\_cursor INTO transaction\_record;

WHILE transaction\_cursor%FOUND LOOP

IF current\_customer\_id IS NULL OR current\_customer\_id != transaction\_record.CustomerID THEN

IF current\_customer\_id IS NOT NULL THEN

DBMS\_OUTPUT.PUT\_LINE('End of statement for Customer ID: ' || current\_customer\_id);

END IF;

DBMS\_OUTPUT.PUT\_LINE('Statement for Customer ID: ' || transaction\_record.CustomerID);

current\_customer\_id := transaction\_record.CustomerID;

END IF;

DBMS\_OUTPUT.PUT\_LINE('Transaction ID: ' || transaction\_record.TransactionID);

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || transaction\_record.AccountID);

DBMS\_OUTPUT.PUT\_LINE('Date: ' || TO\_CHAR(transaction\_record.TransactionDate, 'DD-MON-YYYY'));

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || transaction\_record.Amount);

DBMS\_OUTPUT.PUT\_LINE('Type: ' || transaction\_record.TransactionType);

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

FETCH transaction\_cursor INTO transaction\_record;

END LOOP;

CLOSE transaction\_cursor;

DBMS\_OUTPUT.PUT\_LINE('End of statement for Customer ID: ' || current\_customer\_id);

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

IF transaction\_cursor%ISOPEN THEN

CLOSE transaction\_cursor;

END IF;

END;

**Scenario 2:**

**Query:**

DECLARE

CURSOR account\_cursor IS

SELECT AccountID, Balance

FROM Accounts;

account\_record account\_cursor%ROWTYPE;

annual\_fee NUMBER := 50;

BEGIN

OPEN account\_cursor;

FETCH account\_cursor INTO account\_record;

WHILE account\_cursor%FOUND LOOP

]

UPDATE Accounts

SET Balance = Balance - annual\_fee

WHERE AccountID = account\_record.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Applied annual fee to Account ID: ' || account\_record.AccountID);

DBMS\_OUTPUT.PUT\_LINE('New Balance: ' || (account\_record.Balance - annual\_fee));

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

FETCH account\_cursor INTO account\_record;

END LOOP;

CLOSE account\_cursor;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

IF account\_cursor%ISOPEN THEN

CLOSE account\_cursor;

END IF;

END;

**Scenario 3:**

**Query:**

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, InterestRate

FROM Loans;

loan\_record loan\_cursor%ROWTYPE;

new\_interest\_rate NUMBER := 6;

BEGIN

OPEN loan\_cursor;

FETCH loan\_cursor INTO loan\_record;

WHILE loan\_cursor%FOUND LOOP

UPDATE Loans

SET InterestRate = new\_interest\_rate

WHERE LoanID = loan\_record.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Updated interest rate for Loan ID: ' || loan\_record.LoanID);

DBMS\_OUTPUT.PUT\_LINE('Old Interest Rate: ' || loan\_record.InterestRate);

DBMS\_OUTPUT.PUT\_LINE('New Interest Rate: ' || new\_interest\_rate);

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

FETCH loan\_cursor INTO loan\_record;

END LOOP;

CLOSE loan\_cursor;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

IF loan\_cursor%ISOPEN THEN

CLOSE loan\_cursor;

END IF;

END;

**Exercise 7: Packages**

**Scenario 1:**

**Query:**

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer (

p\_CustomerID IN NUMBER,

p\_Name IN VARCHAR2,

p\_DOB IN DATE,

p\_Balance IN NUMBER

);

PROCEDURE UpdateCustomerDetails (

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:**

**Query:**

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer (

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);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

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

WHEN OTHERS THEN

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

END AddCustomer;

PROCEDURE UpdateCustomerDetails (

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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID ' || p\_CustomerID || ' does not exist.');

WHEN OTHERS THEN

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

END UpdateCustomerDetails;

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;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID ' || p\_CustomerID || ' does not exist.');

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

**To get the output:**

**Query:**

BEGIN

CustomerManagement.UpdateCustomerDetails(1, 'Meenakshi Sharma', TO\_DATE('2003-08-28', 'YYYY-MM-DD'), 1200);

END;

DECLARE

v\_Balance NUMBER;

BEGIN

v\_Balance := CustomerManagement.GetCustomerBalance(1);

DBMS\_OUTPUT.PUT\_LINE('Customer Balance: ' || v\_Balance);

END;

**Scenario 2:**

**Query:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2,

p\_HireDate IN DATE

);

PROCEDURE UpdateEmployeeDetails (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2

);

FUNCTION CalculateAnnualSalary (

p\_EmployeeID IN NUMBER

) RETURN NUMBER;

END EmployeeManagement;

**Package body:**

**Query:**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee (

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

INSERT INTO Employees (

EmployeeID, Name, Position, Salary, Department, HireDate

) VALUES (

p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate

);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_EmployeeID || ' already exists.');

WHEN OTHERS THEN

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

END HireEmployee;

PROCEDURE UpdateEmployeeDetails (

p\_EmployeeID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Position IN VARCHAR2,

p\_Salary IN NUMBER,

p\_Department IN VARCHAR2

) IS

BEGIN

UPDATE Employees

SET Name = p\_Name,

Position = p\_Position,

Salary = p\_Salary,

Department = p\_Department,

HireDate = SYSDATE

WHERE EmployeeID = p\_EmployeeID;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_EmployeeID || ' does not exist.');

WHEN OTHERS THEN

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

END UpdateEmployeeDetails;

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; -- Assuming monthly salary

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_EmployeeID || ' does not exist.');

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END CalculateAnnualSalary;

END EmployeeManagement;

**Inserting and modifying the input:**

**Query:**

BEGIN

EmployeeManagement.HireEmployee(

p\_EmployeeID => 3,

p\_Name => 'Madhu Singh',

p\_Position => 'Analyst',

p\_Salary => 5000,

p\_Department => 'Finance',

p\_HireDate => TO\_DATE('2024-08-07', 'YYYY-MM-DD')

);

END;

BEGIN

EmployeeManagement.UpdateEmployeeDetails(

p\_EmployeeID => 3,

p\_Name => 'Ishika',

p\_Position => 'Senior Analyst',

p\_Salary => 5500,

p\_Department => 'Finance'

);

END;

DECLARE

v\_AnnualSalary NUMBER;

BEGIN

v\_AnnualSalary := EmployeeManagement.CalculateAnnualSalary(p\_EmployeeID => 3);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || v\_AnnualSalary);

END;

**Scenario 3:**

**Query:**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount (

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_Balance IN NUMBER

);

PROCEDURE CloseAccount (

p\_AccountID IN NUMBER

);

FUNCTION GetTotalBalance (

p\_CustomerID IN NUMBER

) RETURN NUMBER;

END AccountOperations;

**Package Body:**

**Query:**

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount (

p\_AccountID IN NUMBER,

p\_CustomerID IN NUMBER,

p\_AccountType IN VARCHAR2,

p\_Balance IN NUMBER

) IS

BEGIN

INSERT INTO Accounts (

AccountID, CustomerID, AccountType, Balance, LastModified

) VALUES (

p\_AccountID, p\_CustomerID, p\_AccountType, p\_Balance, SYSDATE

);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account ID ' || p\_AccountID || ' already exists.');

WHEN OTHERS THEN

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

END OpenAccount;

PROCEDURE CloseAccount (

p\_AccountID IN NUMBER

) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_AccountID;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Account ID ' || p\_AccountID || ' does not exist.');

WHEN OTHERS THEN

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

END CloseAccount;

FUNCTION GetTotalBalance (

p\_CustomerID IN NUMBER

) RETURN NUMBER IS

v\_TotalBalance NUMBER;

BEGIN

SELECT NVL(SUM(Balance), 0)

INTO v\_TotalBalance

FROM Accounts

WHERE CustomerID = p\_CustomerID;

RETURN v\_TotalBalance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No accounts found for Customer ID ' || p\_CustomerID);

RETURN 0;

WHEN OTHERS THEN

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

RETURN 0;

END GetTotalBalance;

END AccountOperations;

Inserting and displaying the output:

Query:

BEGIN

AccountOperations.OpenAccount(

p\_AccountID => 3,

p\_CustomerID => 1,

p\_AccountType => 'Savings',

p\_Balance => 2000

);

END;

BEGIN

AccountOperations.CloseAccount(p\_AccountID => 3);

END;

DECLARE

v\_TotalBalance NUMBER;

BEGIN

v\_TotalBalance := AccountOperations.GetTotalBalance(p\_CustomerID => 1);

DBMS\_OUTPUT.PUT\_LINE('Total Balance for Customer ID 1: ' || v\_TotalBalance);

END;