Table Creation:

```
-- Customer Table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    name VARCHAR(100),
    DOB DATE,
    Balance DECIMAL(10, 2),
    LastModified DATE
);
-- Accounts Table
CREATE TABLE Accounts (
    AccountID INT PRIMARY KEY,
    CustomerID INT,
    AccountType VARCHAR(20),
    Balance DECIMAL(10, 2),
    LastModified DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
-- Transactions Table
CREATE TABLE Transactions (
    TransactionID INT PRIMARY KEY,
    AccountID INT,
    TransactionDate DATE,
    Amount DECIMAL(10, 2),
    TransactionType VARCHAR(10),
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);
-- Loans Table
CREATE TABLE Loans (
    LoanID INT PRIMARY KEY,
    CustomerID INT,
    LoanAmount DECIMAL(10, 2),
    InterestRate DECIMAL(5, 2),
    StartDate DATE,
    EndDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
-- Employees Table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    Position VARCHAR(50),
    Salary DECIMAL(10, 2),
    Department VARCHAR(50),
    HireDate DATE
);
```

Value Insertion:

```
-- Customer Table
INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
VALUES
    (1, 'John Doe', '1985-05-15', 1000.00, NOW()),
    (2, 'Jane Smith', '1990-07-20', 1500.00, NOW()),
    (3, 'Emily Davis', '1982-11-30', 2000.00, NOW()),
    (4, 'Michael Brown', '1975-02-22', 2500.00, NOW()),
    (5, 'Sarah Wilson', '1995-09-14', 1200.00, NOW());
-- Accounts Table
INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)
VALUES
    (1, 1, 'Savings', 1000.00, NOW()),
    (2, 2, 'Checking', 1500.00, NOW()),
    (3, 3, 'Savings', 2000.00, NOW()),
    (4, 4, 'Checking', 2500.00, NOW()),
    (5, 5, 'Savings', 1200.00, NOW());
-- Transactions Table
INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount,
TransactionType)
VALUES
    (1, 1, NOW(), 200.00, 'Deposit'),
    (2, 2, NOW(), 300.00, 'Withdrawal'),
    (3, 3, NOW(), 150.00, 'Deposit'),
    (4, 4, NOW(), 500.00, 'Deposit'),
    (5, 5, NOW(), 100.00, 'Withdrawal');
-- Loans Table
INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)
VALUES
    (6, 1, 5000.00, 5.00, NOW(), DATE_ADD(NOW(), INTERVAL 60 MONTH)),
    (7, 2, 7500.00, 4.50, NOW(), DATE_ADD(NOW(), INTERVAL 36 MONTH)),
    (8, 3, 3000.00, 6.00, NOW(), DATE_ADD(NOW(), INTERVAL 48 MONTH)),
    (9, 4, 10000.00, 4.75, NOW(), DATE_ADD(NOW(), INTERVAL 72 MONTH)),
    (10, 5, 2000.00, 5.25, NOW(), DATE_ADD(NOW(), INTERVAL 24 MONTH));

-- Employees Table
INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)
VALUES
    (1, 'Alice Johnson', 'Manager', 70000.00, 'HR', '2015-06-15'),
    (2, 'Bob Brown', 'Developer', 60000.00, 'IT', '2017-03-20'),
    (3, 'Charlie Clark', 'Analyst', 55000.00, 'Finance', '2018-08-25'),
    (4, 'Diana Evans', 'Designer', 65000.00, 'Marketing', '2019-10-30'),
    (5, 'Ethan Green', 'Technician', 50000.00, 'IT', '2020-01-12');
```

Exercise 1: Control Structures

-- Scenario 01

```
DECLARE
    v_age NUMBER;
    v_discount NUMBER := 1; -- 1% discount
BEGIN
    FOR rec IN (SELECT CustomerID, DOB FROM Customers) LOOP
        -- Calculate age
        v_age := FLOOR(MONTHS_BETWEEN(SYSDATE, rec.DOB) / 12);

        IF v_age > 60 THEN
            -- Update the loan interest rate with a 1% discount
            UPDATE Loans
            SET InterestRate = InterestRate - v_discount
            WHERE CustomerID = rec.CustomerID;
        END IF;
    END LOOP;
    COMMIT;
END;
/
```

-- Scenario 02:

```
ALTER TABLE Customers ADD ( IsVIP BOOLEAN DEFAULT FALSE );

BEGIN
    FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP
        IF rec.Balance > 10000 THEN
            -- Set the VIP flag to TRUE for customers with a balance over $10,000
            UPDATE Customers
            SET IsVIP = TRUE
            WHERE CustomerID = rec.CustomerID;
        END IF;
    END LOOP;
    COMMIT;
END;
/
```

-- Scenario 03:

```
DECLARE
    v_current_date DATE := SYSDATE;
```

```
BEGIN
   FOR rec IN (SELECT CustomerID, LoanID, EndDate
               FROM Loans
               WHERE EndDate BETWEEN v_current_date AND v_current_date + 30) LOOP
      -- Print a reminder message
      DBMS_OUTPUT.PUT_LINE('Reminder: Loan ID ' || rec.LoanID || ' for Customer ID ' ||
rec.CustomerID || ' is due on ' || rec.EndDate);
   END LOOP;
END;
/
```

Exercise 2: Error Handling

-- Create table for the error message

```
CREATE TABLE ErrorLog (
   ErrorID SERIAL PRIMARY KEY,
   ErrorMessage TEXT,
   ErrorDate DATETIME
);
```

-- Scenario 01:

```
DELIMITER //

CREATE PROCEDURE SafeTransferFunds(
   IN from_account_id INT,
   IN to_account_id INT,
   IN transfer_amount DECIMAL(10, 2)
)
BEGIN
   DECLARE v_from_balance DECIMAL(10, 2);
   DECLARE v_to_balance DECIMAL(10, 2);
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      -- Rollback in case of an error
      ROLLBACK;
      -- Log error message
      INSERT INTO ErrorLog (ErrorMessage, ErrorDate)
      VALUES ('Error during fund transfer.', NOW());
   END;

   START TRANSACTION;

   -- Get current balances
   SELECT Balance INTO v_from_balance FROM Accounts WHERE AccountID = from_account_id
FOR UPDATE;
```

```
    SELECT Balance INTO v_to_balance FROM Accounts WHERE AccountID = to_account_id FOR
UPDATE;

    -- Check if sufficient funds are available
    IF v_from_balance < transfer_amount THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient funds';
    ELSE
        -- Perform the transfer
        UPDATE Accounts SET Balance = Balance - transfer_amount WHERE AccountID =
from_account_id;
        UPDATE Accounts SET Balance = Balance + transfer_amount WHERE AccountID =
to_account_id;
    END IF;

    COMMIT;
END //

DELIMITER ;
```

-- Scenario 02:

```
DELIMITER //

CREATE PROCEDURE UpdateSalary(
    IN employee_id INT,
    IN percentage_increase DECIMAL(5, 2)
)
BEGIN
    DECLARE v_current_salary DECIMAL(10, 2);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Log error message
        INSERT INTO ErrorLog (ErrorMessage, ErrorDate)
        VALUES ('Error updating salary for employee ID ' || employee_id, NOW());
    END;

    -- Get current salary
    SELECT Salary INTO v_current_salary FROM Employees WHERE EmployeeID = employee_id;

    -- Check if employee exists
    IF v_current_salary IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Employee ID does not exist';
    ELSE
        -- Update salary
        UPDATE Employees
        SET Salary = Salary + (Salary * percentage_increase / 100)
        WHERE EmployeeID = employee_id;
    END IF;
END //

DELIMITER ;
```

-- Scenario 03:

```
DELIMITER //

CREATE PROCEDURE AddNewCustomer(
    IN customer_id INT,
    IN customer_name VARCHAR(100),
    IN dob DATE,
    IN balance DECIMAL(10, 2)
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Log error message
        INSERT INTO ErrorLog (ErrorMessage, ErrorDate)
        VALUES ('Error adding customer with ID ' || customer_id, NOW());
    END;

    -- Check if customer ID already exists
    IF EXISTS (SELECT 1 FROM Customers WHERE CustomerID = customer_id) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Customer ID already exists';
    ELSE
        -- Insert new customer
        INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
        VALUES (customer_id, customer_name, dob, balance, NOW());
    END IF;
END //

DELIMITER ;
```

Exercise 3: Stored Procedures

-- Scenario 01:

```
DELIMITER //

CREATE PROCEDURE ProcessMonthlyInterest()
BEGIN
    DECLARE v_interest_rate DECIMAL(5,2) DEFAULT 1.00; -- 1% interest rate

    -- Update balances for all savings accounts
    UPDATE Accounts
    SET Balance = Balance * (1 + v_interest_rate / 100)
    WHERE AccountType = 'Savings';
```

```
   COMMIT;
END //

DELIMITER ;
```

-- Scenario 02:

```
DELIMITER //

CREATE PROCEDURE UpdateEmployeeBonus(
   IN department_name VARCHAR(50),
   IN bonus_percentage DECIMAL(5,2)
)
BEGIN
   -- Update salaries by adding the bonus percentage
   UPDATE Employees
   SET Salary = Salary + (Salary * bonus_percentage / 100)
   WHERE Department = department_name;

   COMMIT;
END //

DELIMITER ;
```

-- Scenario 03:

```
DELIMITER //

CREATE PROCEDURE TransferFunds(
   IN from_account_id INT,
   IN to_account_id INT,
   IN transfer_amount DECIMAL(10,2)
)
BEGIN
   DECLARE v_from_balance DECIMAL(10,2);
   DECLARE v_to_balance DECIMAL(10,2);

   -- Get current balances
   SELECT Balance INTO v_from_balance FROM Accounts WHERE AccountID = from_account_id
FOR UPDATE;
   SELECT Balance INTO v_to_balance FROM Accounts WHERE AccountID = to_account_id FOR
UPDATE;

   -- Check if sufficient funds are available
   IF v_from_balance < transfer_amount THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient funds';
   ELSE
      -- Perform the transfer
```

```
        UPDATE Accounts SET Balance = Balance - transfer_amount WHERE AccountID =
from_account_id;
        UPDATE Accounts SET Balance = Balance + transfer_amount WHERE AccountID =
to_account_id;
    END IF;

    COMMIT;
END //

DELIMITER ;
```

Exercise 4: Functions

-- Scenario 01:

```
DELIMITER //

CREATE FUNCTION CalculateAge(dob DATE)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE v_age INT;
    SET v_age = TIMESTAMPDIFF(YEAR, dob, CURDATE());
    RETURN v_age;
END //

DELIMITER ;
```

-- Scenario 02:

```
DELIMITER //

CREATE FUNCTION CalculateMonthlyInstallment(
    loan_amount DECIMAL(10,2),
    annual_interest_rate DECIMAL(5,2),
    loan_duration_years INT
)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE v_monthly_interest_rate DECIMAL(5,2);
    DECLARE v_total_installments INT;
    DECLARE v_monthly_installment DECIMAL(10,2);

    SET v_monthly_interest_rate = annual_interest_rate / 12 / 100;
    SET v_total_installments = loan_duration_years * 12;

    SET v_monthly_installment = loan_amount * (v_monthly_interest_rate * POWER(1 +
v_monthly_interest_rate, v_total_installments)) /
```

```
                                (POWER(1 + v_monthly_interest_rate, v_total_installments) - 1);

    RETURN v_monthly_installment;
END //

DELIMITER ;
```

-- Scenario 03:

```
DELIMITER //

CREATE FUNCTION HasSufficientBalance(
    account_id INT,
    required_amount DECIMAL(10,2)
)
RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    DECLARE v_balance DECIMAL(10,2);
    DECLARE v_result BOOLEAN;

    -- Get the current balance of the account
    SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = account_id;

    -- Check if the balance is sufficient
    IF v_balance IS NULL THEN
        RETURN FALSE; -- Account does not exist
    ELSE
        SET v_result = (v_balance >= required_amount);
        RETURN v_result;
    END IF;
END //

DELIMITER ;
```

Exercise 5: Triggers

-- Scenario 01:

```
DELIMITER //

CREATE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON Customers
FOR EACH ROW
BEGIN
    SET NEW.LastModified = CURDATE();
END //

DELIMITER ;
```

-- Scenario 02:

```
-- Make sure we have the AuditLog table created
CREATE TABLE AuditLog (
   AuditID INT AUTO_INCREMENT PRIMARY KEY,
   TransactionID INT,
   AccountID INT,
   TransactionDate DATE,
   Amount DECIMAL(10,2),
   TransactionType VARCHAR(10),
   AuditDate DATETIME
);

DELIMITER //

CREATE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
   INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType,
AuditDate)
   VALUES (NEW.TransactionID, NEW.AccountID, NEW.TransactionDate, NEW.Amount,
NEW.TransactionType, NOW());
END //

DELIMITER ;
```

-- Scenario 03:

```
DELIMITER //

CREATE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
BEGIN
   DECLARE v_current_balance DECIMAL(10,2);

   -- Get the current balance of the account
   SELECT Balance INTO v_current_balance FROM Accounts WHERE AccountID = NEW.AccountID
FOR UPDATE;

   -- Check rules based on the transaction type
   IF NEW.TransactionType = 'Withdrawal' THEN
      IF v_current_balance IS NULL THEN
         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Account does not exist';
      ELSEIF v_current_balance < NEW.Amount THEN
         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient funds for withdrawal';
      END IF;
```

```
    ELSEIF NEW.TransactionType = 'Deposit' THEN
        IF NEW.Amount <= 0 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Deposit amount must be positive';
        END IF;
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid transaction type';
    END IF;
END //

DELIMITER ;
```

Exercise 6: Cursors

-- Scenario 01:

```
DECLARE
    CURSOR c_transactions IS
        SELECT c.CustomerID, c.Name, t.TransactionDate, t.Amount, t.TransactionType
        FROM Customers c
        JOIN Accounts a ON c.CustomerID = a.CustomerID
        JOIN Transactions t ON a.AccountID = t.AccountID
        WHERE t.TransactionDate >= TRUNC(SYSDATE, 'MM')
          AND t.TransactionDate < ADD_MONTHS(TRUNC(SYSDATE, 'MM'), 1);

    v_customer_id Customers.CustomerID%TYPE;
    v_name Customers.Name%TYPE;
    v_transaction_date Transactions.TransactionDate%TYPE;
    v_amount Transactions.Amount%TYPE;
    v_transaction_type Transactions.TransactionType%TYPE;
BEGIN
    OPEN c_transactions;
    LOOP
        FETCH c_transactions INTO v_customer_id, v_name, v_transaction_date, v_amount,
v_transaction_type;
        EXIT WHEN c_transactions%NOTFOUND;

        -- Print statement for each transaction (Replace with actual print logic)
        DBMS_OUTPUT.PUT_LINE('Customer: ' || v_name);
        DBMS_OUTPUT.PUT_LINE('Date: ' || v_transaction_date);
        DBMS_OUTPUT.PUT_LINE('Amount: ' || v_amount);
        DBMS_OUTPUT.PUT_LINE('Type: ' || v_transaction_type);
        DBMS_OUTPUT.PUT_LINE('---------------------------------');
    END LOOP;
    CLOSE c_transactions;
END;
```

-- Scenario 02:

```
DECLARE
    CURSOR c_accounts IS
```

```
        SELECT AccountID, Balance
        FROM Accounts;

    v_account_id Accounts.AccountID%TYPE;
    v_balance Accounts.Balance%TYPE;
    v_fee DECIMAL(10,2) := 50.00; -- Annual maintenance fee

BEGIN
    OPEN c_accounts;
    LOOP
        FETCH c_accounts INTO v_account_id, v_balance;
        EXIT WHEN c_accounts%NOTFOUND;

        -- Deduct the annual fee
        UPDATE Accounts
        SET Balance = v_balance - v_fee
        WHERE AccountID = v_account_id;

        -- Optionally log or print the update
        DBMS_OUTPUT.PUT_LINE('Account ID: ' || v_account_id || ' - Fee Applied: ' || v_fee);
    END LOOP;
    CLOSE c_accounts;

    COMMIT;
END;
```

-- Scenario 03:

```
DECLARE
    CURSOR c_loans IS
        SELECT LoanID, InterestRate
        FROM Loans;

    v_loan_id Loans.LoanID%TYPE;
    v_current_rate Loans.InterestRate%TYPE;
    v_new_rate DECIMAL(5,2);

BEGIN
    OPEN c_loans;
    LOOP
        FETCH c_loans INTO v_loan_id, v_current_rate;
        EXIT WHEN c_loans%NOTFOUND;

        -- Apply new interest rate policy (for example, increase by 0.5%)
        v_new_rate := v_current_rate + 0.50;

        -- Update the loan interest rate
        UPDATE Loans
        SET InterestRate = v_new_rate
        WHERE LoanID = v_loan_id;

        -- Optionally log or print the update
        DBMS_OUTPUT.PUT_LINE('Loan ID: ' || v_loan_id || ' - New Interest Rate: ' || v_new_rate);
    END LOOP;
    CLOSE c_loans;

    COMMIT;
END;
```

Exercise 7: Packages

-- Scenario 01:

-- (CustomerManagement.pks) Package Specification

```
CREATE OR REPLACE PACKAGE CustomerManagement AS
    PROCEDURE AddNewCustomer(
        p_CustomerID INT,
        p_Name VARCHAR2,
        p_DOB DATE,
        p_Balance DECIMAL,
        p_LastModified DATE
    );

    PROCEDURE UpdateCustomerDetails(
        p_CustomerID INT,
        p_Name VARCHAR2,
        p_Balance DECIMAL,
        p_LastModified DATE
    );

    FUNCTION GetCustomerBalance(p_CustomerID INT) RETURN DECIMAL;
END CustomerManagement;
/
```

-- (CustomerManagement.pkb) Package Body

```
CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

    PROCEDURE AddNewCustomer(
        p_CustomerID INT,
        p_Name VARCHAR2,
        p_DOB DATE,
        p_Balance DECIMAL,
        p_LastModified DATE
    ) IS
    BEGIN
        INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
        VALUES (p_CustomerID, p_Name, p_DOB, p_Balance, p_LastModified);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Customer with ID ' || p_CustomerID || ' already exists.');
    END AddNewCustomer;

    PROCEDURE UpdateCustomerDetails(
        p_CustomerID INT,
        p_Name VARCHAR2,
        p_Balance DECIMAL,
        p_LastModified DATE
    ) IS
    BEGIN
        UPDATE Customers
        SET Name = p_Name,
            Balance = p_Balance,
            LastModified = p_LastModified
        WHERE CustomerID = p_CustomerID;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Customer with ID ' || p_CustomerID || ' does not exist.');
    END UpdateCustomerDetails;

    FUNCTION GetCustomerBalance(p_CustomerID INT) RETURN DECIMAL IS
        v_Balance DECIMAL(10,2);
    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('Customer with ID ' || p_CustomerID || ' does not exist.');
            RETURN 0;
    END GetCustomerBalance;

END CustomerManagement;
/
```

-- Scenario 02:
-- (EmployeeManagement.pks) Package Specification

```
CREATE OR REPLACE PACKAGE EmployeeManagement AS
    PROCEDURE HireNewEmployee(
        p_EmployeeID INT,
        p_Name VARCHAR2,
        p_Position VARCHAR2,
        p_Salary DECIMAL,
        p_Department VARCHAR2,
        p_HireDate DATE
    );

    PROCEDURE UpdateEmployeeDetails(
        p_EmployeeID INT,
        p_Name VARCHAR2,
        p_Salary DECIMAL,
        p_Department VARCHAR2
    );

    FUNCTION CalculateAnnualSalary(p_EmployeeID INT) RETURN DECIMAL;
END EmployeeManagement;
/
```

-- (EmployeeManagement.pkb) Package Body

```
CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

    PROCEDURE HireNewEmployee(
        p_EmployeeID INT,
        p_Name VARCHAR2,
        p_Position VARCHAR2,
        p_Salary DECIMAL,
        p_Department VARCHAR2,
        p_HireDate 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('Employee with ID ' || p_EmployeeID || ' already exists.');
    END HireNewEmployee;

    PROCEDURE UpdateEmployeeDetails(
        p_EmployeeID INT,
        p_Name VARCHAR2,
        p_Salary DECIMAL,
        p_Department VARCHAR2
    ) IS
    BEGIN
        UPDATE Employees
        SET Name = p_Name,
            Salary = p_Salary,
            Department = p_Department
        WHERE EmployeeID = p_EmployeeID;
    EXCEPTION
```

```
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Employee with ID ' || p_EmployeeID || ' does not exist.');
    END UpdateEmployeeDetails;

    FUNCTION CalculateAnnualSalary(p_EmployeeID INT) RETURN DECIMAL IS
        v_Salary DECIMAL(10,2);
    BEGIN
        SELECT Salary INTO v_Salary
        FROM Employees
        WHERE EmployeeID = p_EmployeeID;
        RETURN v_Salary * 12;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Employee with ID ' || p_EmployeeID || ' does not exist.');
            RETURN 0;
    END CalculateAnnualSalary;

END EmployeeManagement;
/
```

-- Scenario 03:
-- (AccountOperations.pks) Package Specification

```
CREATE OR REPLACE PACKAGE AccountOperations AS
    PROCEDURE OpenNewAccount(
        p_AccountID INT,
        p_CustomerID INT,
        p_AccountType VARCHAR2,
        p_Balance DECIMAL,
        p_LastModified DATE
    );

    PROCEDURE CloseAccount(
        p_AccountID INT
    );

    FUNCTION GetTotalBalance(p_CustomerID INT) RETURN DECIMAL;
END AccountOperations;
/
```

-- (AccountOperations.pkb) Package Body

```
CREATE OR REPLACE PACKAGE BODY AccountOperations AS
    PROCEDURE OpenNewAccount(
        p_AccountID INT,
        p_CustomerID INT,
        p_AccountType VARCHAR2,
        p_Balance DECIMAL,
        p_LastModified DATE
    ) IS
    BEGIN
        INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)
        VALUES (p_AccountID, p_CustomerID, p_AccountType, p_Balance, p_LastModified);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Account with ID ' || p_AccountID || ' already exists.');
    END OpenNewAccount;

    PROCEDURE CloseAccount(
        p_AccountID INT
    ) IS
    BEGIN
        DELETE FROM Accounts
        WHERE AccountID = p_AccountID;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Account with ID ' || p_AccountID || ' does not exist.');
    END CloseAccount;

    FUNCTION GetTotalBalance(p_CustomerID INT) RETURN DECIMAL IS
        v_total_balance DECIMAL(10,2);
    BEGIN
        SELECT SUM(Balance) INTO v_total_balance
        FROM Accounts
        WHERE CustomerID = p_CustomerID;

        IF v_total_balance IS NULL THEN
            RETURN 0;
        ELSE
            RETURN v_total_balance;
        END IF;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Customer with ID ' || p_CustomerID || ' does not exist.');
            RETURN 0;
    END GetTotalBalance;

END AccountOperations;
/
```