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.

Code:

```
DECLARE
  CURSOR c_customers IS
    SELECT c.CustomerID, l.LoanID, l.InterestRate
    FROM Customers c
    JOIN Loans l ON c.CustomerID = l.CustomerID
    WHERE TRUNC(MONTHS_BETWEEN(SYSDATE, c.DOB) / 12) > 60;
BEGIN
  FOR rec IN c_customers LOOP
    UPDATE Loans
    SET InterestRate = rec.InterestRate - 1
    WHERE LoanID = rec.LoanID;
  END LOOP;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error occurred: ' || SQLERRM);
    ROLLBACK;
END;
/
```

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.

Code:

```
ALTER TABLE Customers ADD IsVIP NUMBER(1);
BEGIN
  UPDATE Customers
  SET IsVIP = CASE WHEN balance > 10000 THEN 1 ELSE 0 END;
  COMMIT;
END;
/
```

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.

Code:

```
SET SERVEROUTPUT ON;

DECLARE
  CURSOR c_loans_due IS
    SELECT c.name, l.loanid, l.enddate
    FROM customers c
    JOIN loans l ON c.customerid = l.customerid
    WHERE l.enddate BETWEEN SYSDATE AND SYSDATE + 30;
BEGIN
  FOR rec IN c_loans_due LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder for ' || rec.name || ': Loan ' || rec.loanid || ' is due on ' ||
TO_CHAR(rec.enddate, 'YYYY-MM-DD'));
  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.

Code:

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds (
    from_account_id IN NUMBER,
    to_account_id IN NUMBER,
    amount IN NUMBER
)
IS
  insufficient_funds EXCEPTION;
  funds_balance NUMBER;
  error_msg VARCHAR2(4000);

BEGIN
  -- Check balance of the from_account with locking
  SELECT balance INTO funds_balance
  FROM accounts
  WHERE accountid = from_account_id
  FOR UPDATE;

  -- Raise exception if insufficient funds
  IF funds_balance < amount THEN
```

```
    RAISE insufficient_funds;
  END IF;

  -- Performing transfer
  UPDATE accounts
  SET balance = balance - amount
  WHERE accountid = from_account_id;

  UPDATE accounts
  SET balance = balance + amount
  WHERE accountid = to_account_id;

  COMMIT;

EXCEPTION
  WHEN insufficient_funds THEN
    error_msg := 'Insufficient funds for transfer';
    BEGIN
      INSERT INTO error_logs (message, log_time)
      VALUES (error_msg, SYSDATE);
    EXCEPTION
      WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error logging insufficient funds: ' || SQLERRM);
    END;
  WHEN OTHERS THEN
    error_msg := SQLERRM;
    BEGIN
      INSERT INTO error_logs (message, log_time)
      VALUES (error_msg, SYSDATE);
    EXCEPTION
      WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error logging general error: ' || SQLERRM);
    END;
  ROLLBACK;
END;
/
```

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.

Code:

```
CREATE OR REPLACE PROCEDURE UpdateSalary (
   employee_id IN NUMBER,
```

```
    percentage IN NUMBER
)
IS
    employee_not_found EXCEPTION;
    v_count NUMBER;
    error_msg VARCHAR2(4000);
    error_code NUMBER;
BEGIN
    -- Check if employee exists
    SELECT COUNT(*) INTO v_count
    FROM Employees
    WHERE employeeid = employee_id;


    IF v_count = 0 THEN
        RAISE employee_not_found;
    ELSE
        -- Update the employee's salary
        UPDATE Employees
        SET salary = salary * (1 + percentage / 100)
        WHERE employeeid = employee_id;

    END IF;

    COMMIT;

EXCEPTION
    WHEN employee_not_found THEN
        INSERT INTO error_logs (log_time, message)
        VALUES (SYSDATE, 'Employee ID not found');
        COMMIT;
    WHEN OTHERS THEN
        error_msg := SQLERRM;
        error_code := SQLCODE;
        INSERT INTO error_logs (log_time, message, error_code)
        VALUES (SYSDATE, error_msg, error_code);
        COMMIT;
END UpdateSalary;
```

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.

Code:

```
CREATE OR REPLACE PROCEDURE AddNewCustomer (
   customer_id IN NUMBER,
   customer_name IN VARCHAR2,
   dob IN DATE,
   balance IN NUMBER
)
IS
   customer_exists EXCEPTION;
   v_count NUMBER;
   error_msg VARCHAR2(4000);
   error_code NUMBER;
BEGIN
   -- Check if customer already exists
   SELECT COUNT(*) INTO v_count
   FROM Customers
   WHERE customerid = customer_id;

   IF v_count > 0 THEN
      RAISE customer_exists;
   ELSE
      INSERT INTO Customers (customerid, name, dob, balance, lastmodified, isvip)
      VALUES (customer_id, customer_name, dob, balance, SYSDATE, 0);
   END IF;

   COMMIT;

EXCEPTION
   WHEN customer_exists THEN
      error_msg := 'Customer ID already exists';
      error_code := SQLCODE;
      INSERT INTO error_logs (log_time, message, error_code)
      VALUES (SYSDATE, error_msg, error_code);
   WHEN OTHERS THEN
      error_msg := SQLERRM;
      error_code := SQLCODE;
      INSERT INTO error_logs (log_time, message, error_code)
      VALUES (SYSDATE, error_msg, error_code);
   ROLLBACK;
END AddNewCustomer;
```

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.

Code:

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest
AS
  update_interest_rate CONSTANT NUMBER := 0.01;
BEGIN
  UPDATE Accounts
  SET Balance = Balance * (1 + update_interest_rate),
     LastModified = SYSDATE
  WHERE AccountType = 'Savings'
  AND LastModified < TRUNC(SYSDATE);
END;
/
```

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.

Code:

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(dept VARCHAR2, bonus_percent
NUMBER)
AS
  error_msg VARCHAR2(4000);
BEGIN
  IF bonus_percent <= 0 THEN
    INSERT INTO error_logs (log_time, message, error_code)
    VALUES (SYSTIMESTAMP, 'Bonus percentage must be greater than zero', -20001);
    RETURN;
  END IF;

  BEGIN
    UPDATE Employees
    SET Salary = Salary * (1 + bonus_percent / 100)
    WHERE Department = dept
    AND Salary > 0;

    IF SQL%ROWCOUNT = 0 THEN
      INSERT INTO error_logs (log_time, message, error_code)
      VALUES (SYSTIMESTAMP, 'No employees found in department ' || dept, -20002);
```

```
    END IF;
  EXCEPTION
    WHEN OTHERS THEN
      error_msg := 'An unexpected error occurred: ' || SQLERRM;
      INSERT INTO error_logs (log_time, message, error_code)
      VALUES (SYSTIMESTAMP, error_msg, -20003);
  END;
END;
/
```

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.

Code:

```
CREATE OR REPLACE PROCEDURE TransferFunds(src_acc_id NUMBER, target_acc_id NUMBER,
p_amount NUMBER)
AS
  src_balance NUMBER;
BEGIN
  -- Begin a transaction
  BEGIN
    -- Check source account balance
    SELECT Balance INTO src_balance
    FROM Accounts
    WHERE AccountID = src_acc_id;

    IF src_balance < p_amount THEN
      RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds');
    END IF;

    -- Update account balance
    UPDATE Accounts
    SET Balance = Balance - p_amount,
        LastModified = SYSDATE
    WHERE AccountID = src_acc_id;

    UPDATE Accounts
    SET Balance = Balance + p_amount,
        LastModified = SYSDATE
    WHERE AccountID = target_acc_id;

    -- Insert transaction records
```

```
    INSERT INTO Transactions (TRANSACTIONID, AccountID, TransactionDate, Amount,
TransactionType)
    VALUES (trans_id_seq.NEXTVAL, src_acc_id, SYSDATE, -p_amount, 'TransOut');

    INSERT INTO Transactions (TRANSACTIONID, AccountID, TransactionDate, Amount,
TransactionType)
    VALUES (trans_id_seq.NEXTVAL, target_acc_id, SYSDATE, p_amount, 'TransIn');

    COMMIT;
  EXCEPTION
    WHEN OTHERS THEN
      ROLLBACK;
      RAISE;
  END;
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.

Code :

```
CREATE OR REPLACE FUNCTION CalculateAge (
  cust_id IN NUMBER
) RETURN NUMBER
IS
  age NUMBER;
  dob DATE;
BEGIN
  SELECT DOB INTO dob FROM Customers WHERE CustomerID = cust_id;
  age := FLOOR(MONTHS_BETWEEN(SYSDATE, dob) / 12);
  RETURN age;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN NULL;
  WHEN OTHERS THEN
    RETURN NULL;
END;
/
```

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.

Code:

```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (
  loan_amount IN NUMBER,
  annual_interest_rate IN NUMBER,
  loan_duration_yrs IN NUMBER
) RETURN NUMBER
IS
  monthly_interest_rate NUMBER;
  no_of_payments NUMBER;
  monthly_installment NUMBER;
BEGIN
  IF loan_amount <= 0 OR annual_interest_rate < 0 OR loan_duration_yrs <= 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Invalid input parameters');
  END IF;

  monthly_interest_rate := annual_interest_rate / 12 / 100;
  no_of_payments := loan_duration_yrs * 12;

  IF monthly_interest_rate = 0 THEN
    monthly_installment := loan_amount / no_of_payments;
  ELSE
    monthly_installment := loan_amount * monthly_interest_rate /
      (1 - POWER(1 + monthly_interest_rate, -no_of_payments));
  END IF;

  RETURN monthly_installment;
EXCEPTION
  WHEN OTHERS THEN
    RAISE;
END;
/
```

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.

Code:

```
CREATE OR REPLACE FUNCTION HasSufficientBalance(
  p_account_id NUMBER,
  p_amount NUMBER
) RETURN NUMBER
```

```
IS
  v_balance NUMBER;
BEGIN
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_account_id;

  IF v_balance >= p_amount THEN
    RETURN 1; -- TRUE
  ELSE
    RETURN 0; -- FALSE
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN 0; -- FALSE
  WHEN OTHERS THEN
    RETURN 0; -- FALSE
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.

Code:

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON Customers
FOR EACH ROW
BEGIN
   :NEW.LastModified := SYSDATE;
END UpdateCustomerLastModified;
/
```

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.

Code:

```
CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
  INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType,
Action, LogTime)
```

```
  VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount,
:NEW.TransactionType, 'INSERT', CURRENT_TIMESTAMP);
END;
/
```

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.

Code:

```
CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
  current_balance NUMBER;
BEGIN
  -- Check if the transaction is a withdrawal
  IF :NEW.TransactionType = 'WITHDRAWAL' THEN
    -- Retrieve the current balance of the account
    SELECT Balance INTO current_balance
    FROM Accounts
    WHERE AccountID = :NEW.AccountID
    FOR UPDATE;

    -- Ensure the withdrawal amount does not exceed the current balance
    IF :NEW.Amount > current_balance THEN
      RAISE_APPLICATION_ERROR(-20001, 'Withdrawal amount exceeds current balance.');
    END IF;
  ELSIF :NEW.TransactionType = 'DEPOSIT' THEN
    -- Ensure the deposit amount is positive
    IF :NEW.Amount <= 0 THEN
      RAISE_APPLICATION_ERROR(-20002, 'Deposit amount must be positive.');
    END IF;
  END IF;
END CheckTransactionRules;
/
```

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.

Code:

```
SET SERVEROUTPUT ON;
DECLARE
  CURSOR c_transactions IS
    SELECT DISTINCT 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 EXTRACT(MONTH FROM t.TransactionDate) = EXTRACT(MONTH FROM SYSDATE)
     AND EXTRACT(YEAR FROM t.TransactionDate) = EXTRACT(YEAR FROM SYSDATE);

  v_customerID Customers.CustomerID%TYPE;
  v_name Customers.Name%TYPE;
  v_transactionDate Transactions.TransactionDate%TYPE;
  v_amount Transactions.Amount%TYPE;
  v_transactionType Transactions.TransactionType%TYPE;

BEGIN
  OPEN c_transactions;

  LOOP
    FETCH c_transactions INTO v_customerID, v_name, v_transactionDate, v_amount,
v_transactionType;
    EXIT WHEN c_transactions%NOTFOUND;

    -- Print the statement (for demonstration purposes, using DBMS_OUTPUT)
    DBMS_OUTPUT.PUT_LINE('Customer ID: ' || v_customerID);
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_name);
    DBMS_OUTPUT.PUT_LINE('Transaction Date: ' || TO_CHAR(v_transactionDate, 'YYYY-MM-
DD'));
    DBMS_OUTPUT.PUT_LINE('Amount: ' || v_amount);
    DBMS_OUTPUT.PUT_LINE('Transaction Type: ' || v_transactionType);
    DBMS_OUTPUT.PUT_LINE('-------------------------');
  END LOOP;

  CLOSE c_transactions;
END;
/
```

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.

Code:

```
SET SERVEROUTPUT ON;
DECLARE
```

```
 CURSOR accounts_cursor IS
  SELECT AccountID, Balance
  FROM Accounts;

 v_acc_id Accounts.AccountID%TYPE;
 v_balance Accounts.Balance%TYPE;
 v_annual_fee NUMBER := 10; -- Example of annual fee
BEGIN
 OPEN accounts_cursor;
 LOOP
  FETCH accounts_cursor INTO v_acc_id, v_balance;
  EXIT WHEN accounts_cursor%NOTFOUND;

  UPDATE Accounts
  SET Balance = Balance - v_annual_fee
  WHERE AccountID = v_acc_id;
 END LOOP;
 CLOSE accounts_cursor;
END;
/
```

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.

Code:

```
SET SERVEROUTPUT ON;

DECLARE
 CURSOR c_loans IS
  SELECT LoanID, InterestRate
  FROM Loans
  FOR UPDATE; -- clause for locking

 v_loanID Loans.LoanID%TYPE;
 currentInterestRate Loans.InterestRate%TYPE;
 newInterestRate Loans.InterestRate%TYPE;

 -- Example function to calculate new interest rate
 FUNCTION calculate_new_interest_rate(p_current_rate NUMBER) RETURN NUMBER IS
 BEGIN
  RETURN p_current_rate + 0.01; -- Example increase interest rate by 1%
 END;

BEGIN
 OPEN c_loans;
```

```
  LOOP
    FETCH c_loans INTO v_loanID, currentInterestRate;
    EXIT WHEN c_loans%NOTFOUND;

    newInterestRate := calculate_new_interest_rate(currentInterestRate);

    UPDATE Loans
    SET InterestRate = newInterestRate
    WHERE CURRENT OF c_loans; -- Use WHERE CURRENT OF for efficiency

    -- Optional logging
    DBMS_OUTPUT.PUT_LINE('Updated LoanID: ' || v_loanID || ' to new interest rate: ' ||
newInterestRate);
  END LOOP;

  CLOSE c_loans;
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.

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'));

Code:

Scenario 1:

```
--1. Create the Package Specification:
CREATE OR REPLACE PACKAGE CustomerManagement AS
   PROCEDURE AddNewCustomer(p_CustomerID NUMBER, p_Name VARCHAR2, p_DOB DATE, p_Balance
NUMBER);
   PROCEDURE UpdateCustomerDetails(p_CustomerID NUMBER, p_Name VARCHAR2, p_Balance
NUMBER);
   FUNCTION GetCustomerBalance(p_CustomerID NUMBER) RETURN NUMBER;
END CustomerManagement;
/

--2. Create the Package Body:
CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

   PROCEDURE AddNewCustomer(p_CustomerID NUMBER, p_Name VARCHAR2, p_DOB DATE, p_Balance
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('An unexpected error occurred: ' || SQLERRM);
  END AddNewCustomer;

  PROCEDURE UpdateCustomerDetails(p_CustomerID NUMBER, p_Name VARCHAR2, p_Balance
NUMBER) IS
  BEGIN
    UPDATE Customers
    SET Name = p_Name, Balance = p_Balance, LastModified = SYSDATE
    WHERE CustomerID = p_CustomerID;
    IF SQL%ROWCOUNT = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Error: Customer ID ' || p_CustomerID || ' not found.');
    END IF;
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
  END UpdateCustomerDetails;

  FUNCTION GetCustomerBalance(p_CustomerID 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 || ' not found.');
      RETURN NULL;
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
      RETURN NULL;
  END GetCustomerBalance;

END CustomerManagement;
/
```

Scenario 2:

```
CREATE OR REPLACE PACKAGE EmployeeManagement AS

  PROCEDURE HireEmployee(p_name VARCHAR2, p_position VARCHAR2, p_salary NUMBER, p_dept
VARCHAR2, p_hiredate DATE);

  PROCEDURE UpdateEmployee(employee_id NUMBER, p_name VARCHAR2, p_position VARCHAR2,
p_salary NUMBER, p_dept VARCHAR2);

  FUNCTION CalculateAnnualSalary(employee_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/


CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

  PROCEDURE HireEmployee(p_name VARCHAR2, p_position VARCHAR2, p_salary NUMBER, p_dept
VARCHAR2, p_hiredate DATE) IS

  BEGIN

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

    VALUES (Employees_Seq.NEXTVAL, p_name, p_position, p_salary, p_dept, p_hiredate);

  EXCEPTION

    WHEN DUP_VAL_ON_INDEX THEN

      RAISE_APPLICATION_ERROR(-20001, 'Employee ID already exists.');

    WHEN OTHERS THEN

      RAISE; -- Re-raise the exception for detailed handling

  END;


  PROCEDURE UpdateEmployee(employee_id NUMBER, p_name VARCHAR2, p_position VARCHAR2,
p_salary NUMBER, p_dept VARCHAR2) IS

  BEGIN

    UPDATE Employees

    SET Name = p_name, Position = p_position, Salary = p_salary, Department = p_dept

    WHERE EmployeeID = employee_id;

  END;
```

```
  FUNCTION CalculateAnnualSalary(employee_id NUMBER) RETURN NUMBER IS
    v_salary NUMBER;
  BEGIN
    SELECT Salary INTO v_salary
    FROM Employees
    WHERE EmployeeID = employee_id;
    RETURN v_salary * 12;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN -1; -- Or handle the exception appropriately
  END;

END EmployeeManagement;
/
```

Scenario 3:

```
CREATE OR REPLACE PACKAGE BODY AccountOperations AS
  PROCEDURE OpenAccount(cust_id NUMBER, p_acc_type VARCHAR2) IS
  BEGIN
    INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)
    VALUES (Accounts_Seq.NEXTVAL, cust_id, p_acc_type, 0, SYSDATE);
  EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
      RAISE_APPLICATION_ERROR(-20001, 'Account ID already exists.');
    WHEN OTHERS THEN
      RAISE;
  END;

  PROCEDURE CloseAccount(p_acc_id NUMBER) IS
```

```
  BEGIN
    DELETE FROM Accounts WHERE AccountID = p_acc_id;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR(-20002, 'Account not found.');
    WHEN OTHERS THEN
      RAISE;
  END;

  FUNCTION GetCustomerTotalBalance(cust_id NUMBER) RETURN NUMBER IS
    v_total_balance NUMBER := 0;
  BEGIN
    SELECT SUM(Balance) INTO v_total_balance
    FROM Accounts
    WHERE CustomerID = cust_id;
    RETURN v_total_balance;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RETURN 0; -- Or handle as needed
    WHEN OTHERS THEN
      RAISE;
  END;
END AccountOperations;
```