PL/SQL

Superset ID : 6393540

Name : Antony Praveen E

E-mail : antonypraveen.2205009@srec.ac.in

Mandatory Questions:

1) Exercise 1: Control Structures

Solution:

Scenario 1 – Discount for Senior Citizens

```
BEGIN
  FOR c IN (SELECT * FROM Loans L JOIN Customers C ON L.CustomerID =
C.CustomerID) LOOP
    IF MONTHS_BETWEEN(SYSDATE, c.DOB) / 12 > 60 THEN
      UPDATE Loans
      SET InterestRate = InterestRate - 1
      WHERE LoanID = c.LoanID;
    END IF;
END LOOP;
END;
```

Scenario 2 – Promote to VIP

```
BEGIN
  FOR c IN (SELECT * FROM Customers) LOOP
    IF c.Balance > 10000 THEN
      UPDATE Customers
      SET IsVIP = 'Y'
      WHERE CustomerID = c.CustomerID;
    DBMS_OUTPUT.PUT_LINE('Customer ' || c.Name || ' promoted to VIP.');
    END IF;
  END LOOP;
END;
```

Scenario 3 – Loan Due Reminders

```
BEGIN
  FOR l IN (SELECT * FROM Loans WHERE EndDate <= SYSDATE + 30) LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan ' || l.LoanID || ' is due soon for Customer ID ' || l.CustomerID);
  END LOOP;
END;
/
```

2) Exercise 3: Stored Procedures

Solution:

Scenario 1 – Monthly Interest for Savings Accounts

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  UPDATE Accounts
  SET Balance = Balance + (Balance * 0.01)
  WHERE AccountType = 'Savings';
  COMMIT;
END;
/
```

Scenario 2 – Employee Bonus

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(p_dept VARCHAR2, p_bonus NUMBER) IS
BEGIN
  UPDATE Employees
  SET Salary = Salary + (Salary * p_bonus / 100)
  WHERE Department = p_dept;
  COMMIT;
END;
/
```

Scenario 3 – Transfer Between Accounts

```
CREATE OR REPLACE PROCEDURE TransferFunds(p_from NUMBER, p_to NUMBER,
p_amt NUMBER) IS
  v_balance NUMBER;
BEGIN
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_from;
  IF v_balance >= p_amt THEN
    UPDATE Accounts SET Balance = Balance - p_amt WHERE AccountID = p_from;
    UPDATE Accounts SET Balance = Balance + p_amt WHERE AccountID = p_to;
    COMMIT;
  ELSE
    DBMS_OUTPUT.PUT_LINE('Insufficient balance.');
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('One of the accounts does not exist.');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    ROLLBACK;
END;
/
```

Other Questions:

3) Exercise 2: Error Handling

Solution:

Scenario 1 – Safe Fund Transfer

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds(p_from NUMBER, p_to
NUMBER, p_amount NUMBER) IS
BEGIN
  UPDATE Accounts SET Balance = Balance - p_amount WHERE AccountID = p_from;
  UPDATE Accounts SET Balance = Balance + p_amount WHERE AccountID = p_to;
```

```
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || SQLERRM);
END;
/
```

Scenario 2 – Update Salary with Error Handling

```
CREATE OR REPLACE PROCEDURE UpdateSalary(p_empid NUMBER, p_percent
NUMBER) IS
BEGIN
  UPDATE Employees
  SET Salary = Salary + (Salary * p_percent / 100)
  WHERE EmployeeID = p_empid;
  IF SQL%ROWCOUNT = 0 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Employee ID not found.');
  END IF;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

Scenario 3 – Add New Customer with Duplicate Check

```
CREATE OR REPLACE PROCEDURE AddNewCustomer(p_id NUMBER, p_name
VARCHAR2, p_dob DATE, p_balance NUMBER) IS
BEGIN
  INSERT INTO Customers(CustomerID, Name, DOB, Balance, LastModified)
  VALUES (p_id, p_name, p_dob, p_balance, SYSDATE);
```

```
  COMMIT;
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.PUT_LINE('Error: Customer with ID already exists.');
END;
/
```

4) Exercise 4: Functions

Solution:

Scenario 1 – Calculate Age

```
CREATE OR REPLACE FUNCTION CalculateAge(p_dob DATE) RETURN NUMBER IS
BEGIN
  RETURN FLOOR(MONTHS_BETWEEN(SYSDATE, p_dob) / 12);
END;
/
```

Scenario 2 – Monthly Installment

```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(
  p_amount NUMBER,
  p_rate NUMBER,
  p_years NUMBER
) RETURN NUMBER IS
  v_monthly NUMBER;
  r NUMBER := p_rate / (12 * 100);
n NUMBER := p_years * 12;
BEGIN
  v_monthly := p_amount * r / (1 - POWER(1 + r, -n));
  RETURN ROUND(v_monthly, 2);
END;
/
```

Scenario 3 – Check Sufficient Balance

```
CREATE OR REPLACE FUNCTION HasSufficientBalance(
  p_accid NUMBER,
  p_amt NUMBER
) RETURN BOOLEAN IS
  v_balance NUMBER;
BEGIN
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = p_accid;
  RETURN v_balance >= p_amt;
END;
/
```

5) Exercise 5: Triggers

Solution:

Scenario 1 – Update LastModified

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

Scenario 2 – Log Transaction

```
CREATE TABLE AuditLog (
  LogID NUMBER GENERATED ALWAYS AS IDENTITY,
  TransactionID NUMBER,
  LogDate DATE DEFAULT SYSDATE
);
```

```
CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
  INSERT INTO AuditLog(TransactionID) VALUES (:NEW.TransactionID);
END;
/
```

Scenario 3 – Enforce Rules

```
CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
  v_balance NUMBER;
BEGIN
  SELECT Balance INTO v_balance FROM Accounts WHERE AccountID = :NEW.AccountID;
  IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v_balance THEN
  RAISE_APPLICATION_ERROR(-20001, 'Withdrawal exceeds balance');
  ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN
  RAISE_APPLICATION_ERROR(-20002, 'Deposit must be positive');
  END IF;
END;
/
```

6) Exercise 6: Cursors

Solution:

Scenario 1 – Monthly Statements

```
DECLARE
  CURSOR c_trans IS
    SELECT * FROM Transactions WHERE TransactionDate >= TRUNC(SYSDATE, 'MM');
BEGIN
```

```
  FOR t IN c_trans LOOP
    DBMS_OUTPUT.PUT_LINE('Account ' || t.AccountID || ' | ' || t.TransactionType || ' | ' ||
t.Amount);
  END LOOP;
END;
/
```

Scenario 2 – Apply Annual Fee

```
DECLARE
  CURSOR c_acc IS SELECT AccountID, Balance FROM Accounts;
BEGIN
  FOR a IN c_acc LOOP
    UPDATE Accounts
    SET Balance = Balance - 100
  WHERE AccountID = a.AccountID;
  END LOOP;
  COMMIT;
END;
/
```

Scenario 3 – Update Loan Interest

```
DECLARE
  CURSOR c_loans IS SELECT LoanID, InterestRate FROM Loans;
BEGIN
  FOR l IN c_loans LOOP
    UPDATE Loans
    SET InterestRate = InterestRate + 0.5
    WHERE LoanID = l.LoanID;
  END LOOP;
COMMIT;
END;
/
```

7) Exercise 7: Packages

Solution:

Scenario 1 – CustomerManagement Package

-- SPEC

```
CREATE OR REPLACE PACKAGE CustomerManagement AS
  PROCEDURE AddCustomer(p_id NUMBER, p_name VARCHAR2, p_dob DATE,
p_balance NUMBER);
  PROCEDURE UpdateCustomer(p_id NUMBER, p_balance NUMBER);
  FUNCTION GetBalance(p_id NUMBER) RETURN NUMBER;
END CustomerManagement;
/
```

-- BODY

```
CREATE OR REPLACE PACKAGE BODY CustomerManagement AS
  PROCEDURE AddCustomer(p_id NUMBER, p_name VARCHAR2, p_dob DATE,
p_balance NUMBER) IS
  BEGIN
    INSERT INTO Customers VALUES (p_id, p_name, p_dob, p_balance, SYSDATE);
  END;
  PROCEDURE UpdateCustomer(p_id NUMBER, p_balance NUMBER) IS
  BEGIN
    UPDATE Customers SET Balance = p_balance, LastModified = SYSDATE WHERE
CustomerID = p_id;
  END;
  FUNCTION GetBalance(p_id NUMBER) RETURN NUMBER IS
    v_bal NUMBER;
  BEGIN
    SELECT Balance INTO v_bal FROM Customers WHERE CustomerID = p_id;
    RETURN v_bal;
  END;
END CustomerManagement;
/
```

Scenario 2 – EmployeeManagement Package

-- SPEC

```
CREATE OR REPLACE PACKAGE EmployeeManagement AS
  PROCEDURE HireEmployee(p_id NUMBER, p_name VARCHAR2, p_pos VARCHAR2, p_sal NUMBER, p_dept VARCHAR2);
  PROCEDURE UpdateEmployee(p_id NUMBER, p_sal NUMBER);
FUNCTION AnnualSalary(p_id NUMBER) RETURN NUMBER;
END EmployeeManagement;
/
```

-- BODY

```
CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS
  PROCEDURE HireEmployee(p_id NUMBER, p_name VARCHAR2, p_pos VARCHAR2, p_sal NUMBER, p_dept VARCHAR2) IS
  BEGIN
    INSERT INTO Employees VALUES (p_id, p_name, p_pos, p_sal, p_dept, SYSDATE);
  END;
  PROCEDURE UpdateEmployee(p_id NUMBER, p_sal NUMBER) IS
  BEGIN
    UPDATE Employees SET Salary = p_sal WHERE EmployeeID = p_id;
  END;
  FUNCTION AnnualSalary(p_id NUMBER) RETURN NUMBER IS
    v_sal NUMBER;
  BEGIN
    SELECT Salary INTO v_sal FROM Employees WHERE EmployeeID = p_id;
    RETURN v_sal * 12;
  END;
END EmployeeManagement;
/
```

Scenario 3 – AccountOperations Package

-- SPEC

```
CREATE OR REPLACE PACKAGE AccountOperations AS
  PROCEDURE OpenAccount(p_accid NUMBER, p_custid NUMBER, p_type VARCHAR2, p_bal NUMBER);
  PROCEDURE CloseAccount(p_accid NUMBER);
  FUNCTION TotalCustomerBalance(p_custid NUMBER) RETURN NUMBER;
END AccountOperations;
/
```

-- BODY

```
CREATE OR REPLACE PACKAGE BODY AccountOperations AS
  PROCEDURE OpenAccount(p_accid NUMBER, p_custid NUMBER, p_type VARCHAR2, p_bal NUMBER) IS
  BEGIN
    INSERT INTO Accounts VALUES (p_accid, p_custid, p_type, p_bal, SYSDATE);
  END;
  PROCEDURE CloseAccount(p_accid NUMBER) IS
  BEGIN
    DELETE FROM Accounts WHERE AccountID = p_accid;
  END;
  FUNCTION TotalCustomerBalance(p_custid NUMBER) RETURN NUMBER IS
    v_total NUMBER;
  BEGIN
    SELECT SUM(Balance) INTO v_total FROM Accounts WHERE CustomerID = p_custid;
    RETURN v_total;
  END;
END AccountOperations;
/
```