PL/SQL Exercise Solution

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:

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

  Output:

```
SQL> BEGIN
       FOR rec IN (SELECT * FROM Customers c JOIN Loans l ON c.CustomerID = l.CustomerID) LOOP
         IF MONTHS_BETWEEN(SYSDATE, rec.DOB) / 12 > 60 THEN
           UPDATE Loans...
Show more...



PL/SQL procedure successfully completed.

Elapsed: 00:00:00.086
```

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 BOOLEAN;
BEGIN
 FOR rec IN (SELECT * FROM Customers) LOOP
  IF rec.Balance > 10000 THEN
   UPDATE Customers SET IsVIP = TRUE WHERE CustomerID = rec.CustomerID;
  END IF;
 END LOOP;
END;
/
```

  Output:

```
SQL> BEGIN
       FOR rec IN (SELECT * FROM Customers) LOOP
         IF rec.Balance > 10000 THEN
           UPDATE Customers SET IsVIP = TRUE WHERE CustomerID = rec.CustomerID;...
Show more...


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```

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:

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

  Output:

```
SQL> BEGIN
       FOR rec IN (
         SELECT * FROM Loans WHERE EndDate <= SYSDATE + 30
       ) LOOP...
Show more...


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.086
```

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_acc NUMBER, to_acc NUMBER, amt NUMBER
) AS
  insufficient_funds EXCEPTION;
  bal NUMBER;
BEGIN
  SELECT Balance INTO bal FROM Accounts WHERE AccountID = from_acc;
  IF bal < amt THEN
    RAISE insufficient_funds;
  END IF;
  UPDATE Accounts SET Balance = Balance - amt WHERE AccountID = from_acc;
  UPDATE Accounts SET Balance = Balance + amt WHERE AccountID = to_acc;
  COMMIT;
EXCEPTION
  WHEN insufficient_funds THEN
    DBMS_OUTPUT.PUT_LINE('Error: Insufficient funds.');
    ROLLBACK;
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || SQLERRM);
    ROLLBACK;
END;
/
```

Output:

```
SQL> CREATE OR REPLACE PROCEDURE SafeTransferFunds(
       from_acc NUMBER, to_acc NUMBER, amt NUMBER
     ) AS
       insufficient_funds EXCEPTION;...
Show more...



Procedure SAFETRANSFERFUNDS compiled

Elapsed: 00:00:00.025
```

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(emp_id NUMBER, pct NUMBER) AS
BEGIN
```

```
 UPDATE Employees SET Salary = Salary + (Salary * pct / 100) WHERE EmployeeID =
emp_id;
 IF SQL%NOTFOUND THEN
  DBMS_OUTPUT.PUT_LINE('Error: Employee not found.');
 END IF;
END;
/
```

Output:

```
SQL> CREATE OR REPLACE PROCEDURE UpdateSalary(emp_id NUMBER, pct NUMBER) AS
     BEGIN
       UPDATE Employees SET Salary = Salary + (Salary * pct / 100) WHERE EmployeeID = emp_id;
       IF SQL%NOTFOUND THEN...
Show more...



Procedure UPDATESALARY compiled

Elapsed: 00:00:00.019
```

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(
 id NUMBER, name VARCHAR2, dob DATE, bal NUMBER
) AS
BEGIN
 INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
 VALUES (id, name, dob, bal, SYSDATE);
EXCEPTION
 WHEN DUP_VAL_ON_INDEX THEN
  DBMS_OUTPUT.PUT_LINE('Error: Customer with ID already exists.');
END;
/
```

  Output:

```
SQL> CREATE OR REPLACE PROCEDURE AddNewCustomer(
       id NUMBER, name VARCHAR2, dob DATE, bal NUMBER
     ) AS
     BEGIN...
Show more...
```

```
Procedure ADDNEWCUSTOMER compiled

Elapsed: 00:00:00.020
```

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
BEGIN
 UPDATE Accounts
 SET Balance = Balance + (Balance * 0.01)
 WHERE AccountType = 'Savings';
END;
/
```

  Output:

```
SQL> CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS
     BEGIN
       UPDATE Accounts
       SET Balance = Balance + (Balance * 0.01)...
Show more...
```

```
Procedure PROCESSMONTHLYINTEREST compiled

Elapsed: 00:00:00.018
```

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_pct
NUMBER) AS
BEGIN
 UPDATE Employees
 SET Salary = Salary + (Salary * bonus_pct / 100)
 WHERE Department = dept;
END;
/
```

Output:

```
SQL> CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(dept VARCHAR2, bonus_pct NUMBER) AS
     BEGIN
       UPDATE Employees
       SET Salary = Salary + (Salary * bonus_pct / 100)...
Show more...



Procedure UPDATEEMPLOYEEBONUS compiled

Elapsed: 00:00:00.012
```

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 NUMBER, dest NUMBER, amt
NUMBER) AS
 src_bal NUMBER;
BEGIN
 SELECT Balance INTO src_bal FROM Accounts WHERE AccountID = src;
 IF src_bal >= amt THEN
  UPDATE Accounts SET Balance = Balance - amt WHERE AccountID = src;
  UPDATE Accounts SET Balance = Balance + amt WHERE AccountID = dest;
 ELSE
  DBMS_OUTPUT.PUT_LINE('Insufficient balance');
 END IF;
END;
/
```

Output:

```
SQL> CREATE OR REPLACE PROCEDURE TransferFunds(src NUMBER, dest NUMBER, amt NUMBER) AS
       src_bal NUMBER;
     BEGIN
       SELECT Balance INTO src_bal FROM Accounts WHERE AccountID = src;...
Show more...



Procedure TRANSFERFUNDS compiled

Elapsed: 00:00:00.015
```

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(dob DATE) RETURN NUMBER IS
BEGIN
 RETURN FLOOR(MONTHS_BETWEEN(SYSDATE, dob) / 12);
END;
/
```

  Output:

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



Function CALCULATEAGE compiled

Elapsed: 00:00:00.012
```

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(
 principal NUMBER, rate NUMBER, years NUMBER
) RETURN NUMBER IS
 r NUMBER := rate / (12 * 100);
 n NUMBER := years * 12;
```

```
BEGIN
 RETURN (principal * r) / (1 - POWER(1 + r, -n));
END;
/
```

Output:

```
SQL> CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(
       principal NUMBER, rate NUMBER, years NUMBER
     ) RETURN NUMBER IS
       r NUMBER := rate / (12 * 100);...
Show more...



Function CALCULATEMONTHLYINSTALLMENT compiled

Elapsed: 00:00:00.016
```

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(acc_id NUMBER, amt NUMBER)
RETURN BOOLEAN IS
 bal NUMBER;
BEGIN
 SELECT Balance INTO bal FROM Accounts WHERE AccountID = acc_id;
 RETURN bal >= amt;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
  RETURN FALSE;
END;
/
```

  Output:

```
SQL> CREATE OR REPLACE FUNCTION HasSufficientBalance(acc_id NUMBER, amt NUMBER) RETURN BOOLEAN IS
       bal NUMBER;
     BEGIN
       SELECT Balance INTO bal FROM Accounts WHERE AccountID = acc_id;...
Show more...



Function HASSUFFICIENTBALANCE compiled

Elapsed: 00:00:00.018
```

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;
/
```

  Output:

```
SQL> CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
     BEFORE UPDATE ON Customers
     FOR EACH ROW
     BEGIN...
Show more...



Trigger UPDATECUSTOMERLASTMODIFIED compiled

Elapsed: 00:00:00.013
```

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 TABLE AuditLog (
  LogID NUMBER GENERATED BY DEFAULT AS IDENTITY,
  TransactionID NUMBER,
  ActionDate DATE
);
CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
  INSERT INTO AuditLog (TransactionID, ActionDate)
  VALUES (:NEW.TransactionID, SYSDATE);
END;
/
```

  Output:

```
SQL> CREATE OR REPLACE TRIGGER LogTransaction
     AFTER INSERT ON Transactions
     FOR EACH ROW
     BEGIN...
Show more...



Trigger LOGTRANSACTION compiled

Elapsed: 00:00:00.014
```

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
 bal NUMBER;
BEGIN
 SELECT Balance INTO bal FROM Accounts WHERE AccountID = :NEW.AccountID;
 IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > bal 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;
/
```

  Output:

```
SQL> CREATE OR REPLACE TRIGGER CheckTransactionRules
     BEFORE INSERT ON Transactions
     FOR EACH ROW
     DECLARE...
Show more...



Trigger CHECKTRANSACTIONRULES compiled

Elapsed: 00:00:00.014
```

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:

```
DECLARE
  CURSOR txn_cur IS
    SELECT * FROM Transactions WHERE TransactionDate BETWEEN TRUNC(SYSDATE,
'MM') AND LAST_DAY(SYSDATE);
BEGIN
  FOR rec IN txn_cur LOOP
    DBMS_OUTPUT.PUT_LINE('Customer ID: ' || rec.AccountID || ' | Amount: ' ||
rec.Amount || ' | Type: ' || rec.TransactionType);
  END LOOP;
END;
/
```

  Output:

```
SQL> DECLARE
       CURSOR txn_cur IS
         SELECT * FROM Transactions WHERE TransactionDate BETWEEN TRUNC(SYSDATE, 'MM') AND LAST_DAY(SYSDATE);
     BEGIN...
Show more...


Customer ID: 1 | Amount: 200 | Type: Deposit
Customer ID: 2 | Amount: 300 | Type: Withdrawal


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.014
```

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:

```
DECLARE
  CURSOR acc_cur IS SELECT * FROM Accounts;
BEGIN
  FOR rec IN acc_cur LOOP
    UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = rec.AccountID;
  END LOOP;
END;
/
```

  Output:

```
SQL> DECLARE
       CURSOR acc_cur IS SELECT * FROM Accounts;
     BEGIN
       FOR rec IN acc_cur LOOP...
Show more...



PL/SQL procedure successfully completed.

Elapsed: 00:00:00.018
```

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:

```
DECLARE
 CURSOR loan_cur IS SELECT * FROM Loans;
BEGIN
 FOR rec IN loan_cur LOOP
  UPDATE Loans SET InterestRate = InterestRate + 0.5 WHERE LoanID = rec.LoanID;
 END LOOP;
END;
/
```

  Output:

```
SQL> DECLARE
       CURSOR loan_cur IS SELECT * FROM Loans;
     BEGIN
       FOR rec IN loan_cur LOOP...
Show more...



PL/SQL procedure successfully completed.

Elapsed: 00:00:00.011
```

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.

  Code:

```
CREATE OR REPLACE PACKAGE CustomerManagement AS
```

```
 PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance
NUMBER);
 PROCEDURE UpdateCustomerDetails(id NUMBER, name VARCHAR2);
 FUNCTION GetCustomerBalance(id NUMBER) RETURN NUMBER;
END CustomerManagement;
/
CREATE OR REPLACE PACKAGE BODY CustomerManagement AS
 PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance NUMBER)
IS
 BEGIN
  INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified, IsVIP)
  VALUES (id, name, dob, balance, SYSDATE, FALSE);
 END;

 PROCEDURE UpdateCustomerDetails(id NUMBER, name VARCHAR2) IS
 BEGIN
  UPDATE Customers SET Name = name WHERE CustomerID = id;
 END;

 FUNCTION GetCustomerBalance(id NUMBER) RETURN NUMBER IS
  bal NUMBER;
 BEGIN
  SELECT Balance INTO bal FROM Customers WHERE CustomerID = id;
  RETURN bal;
 END;
END CustomerManagement;
/
```

Output:

```
SQL> CREATE OR REPLACE PACKAGE CustomerManagement AS
       PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance NUMBER);
       PROCEDURE UpdateCustomerDetails(id NUMBER, name VARCHAR2);
       FUNCTION GetCustomerBalance(id NUMBER) RETURN NUMBER;...
Show more...


Package CUSTOMERMANAGEMENT compiled

Elapsed: 00:00:00.014
```

```
SQL> CREATE OR REPLACE PACKAGE BODY CustomerManagement AS
       PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, balance NUMBER) IS
       BEGIN
         INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified, IsVIP)...
Show more...


Package Body CUSTOMERMANAGEMENT compiled

Elapsed: 00:00:00.015
```

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.

Code:

```
CREATE OR REPLACE PACKAGE EmployeeManagement AS
 PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, pos VARCHAR2, sal
NUMBER, dept VARCHAR2);
 PROCEDURE UpdateDetails(id NUMBER, name VARCHAR2);
 FUNCTION CalcAnnualSalary(id NUMBER) RETURN NUMBER;
END;
/
CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS
 PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, pos VARCHAR2, sal
NUMBER, dept VARCHAR2) IS
 BEGIN
  INSERT INTO Employees VALUES (id, name, pos, sal, dept, SYSDATE);
 END;

 PROCEDURE UpdateDetails(id NUMBER, name VARCHAR2) IS
 BEGIN
  UPDATE Employees SET Name = name WHERE EmployeeID = id;
 END;

 FUNCTION CalcAnnualSalary(id NUMBER) RETURN NUMBER IS
  sal NUMBER;
 BEGIN
  SELECT Salary INTO sal FROM Employees WHERE EmployeeID = id;
  RETURN sal * 12;
 END;
END;
/
```

Output:

```
SQL> CREATE OR REPLACE PACKAGE EmployeeManagement AS
       PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, pos VARCHAR2, sal NUMBER, dept VARCHAR2);
       PROCEDURE UpdateDetails(id NUMBER, name VARCHAR2);
       FUNCTION CalcAnnualSalary(id NUMBER) RETURN NUMBER;...
Show more...



Package EMPLOYEEMANAGEMENT compiled

Elapsed: 00:00:00.014
```

```
SQL> CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS
       PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, pos VARCHAR2, sal NUMBER, dept VARCHAR2) IS
       BEGIN
         INSERT INTO Employees VALUES (id, name, pos, sal, dept, SYSDATE);...
Show more...



Package Body EMPLOYEEMANAGEMENT compiled

Elapsed: 00:00:00.012
```

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.

  Code:

```
CREATE OR REPLACE PACKAGE AccountOperations AS
 PROCEDURE OpenAccount(acc_id NUMBER, cust_id NUMBER, type VARCHAR2, bal
NUMBER);
 PROCEDURE CloseAccount(acc_id NUMBER);
 FUNCTION GetTotalBalance(cust_id NUMBER) RETURN NUMBER;
END;
/
CREATE OR REPLACE PACKAGE BODY AccountOperations AS
 PROCEDURE OpenAccount(acc_id NUMBER, cust_id NUMBER, type VARCHAR2, bal
NUMBER) IS
 BEGIN
  INSERT INTO Accounts VALUES (acc_id, cust_id, type, bal, SYSDATE);
 END;

 PROCEDURE CloseAccount(acc_id NUMBER) IS
 BEGIN
  DELETE FROM Accounts WHERE AccountID = acc_id;
 END;

 FUNCTION GetTotalBalance(cust_id NUMBER) RETURN NUMBER IS
  total NUMBER;
 BEGIN
```

```
  SELECT SUM(Balance) INTO total FROM Accounts WHERE CustomerID = cust_id;
  RETURN total;
 END;
END;
/
```

Output:

```
SQL> CREATE OR REPLACE PACKAGE AccountOperations AS
       PROCEDURE OpenAccount(acc_id NUMBER, cust_id NUMBER, type VARCHAR2, bal NUMBER);
       PROCEDURE CloseAccount(acc_id NUMBER);
       FUNCTION GetTotalBalance(cust_id NUMBER) RETURN NUMBER;...
Show more...


Package ACCOUNTOPERATIONS compiled

Elapsed: 00:00:00.014
```

```
SQL> CREATE OR REPLACE PACKAGE BODY AccountOperations AS
       PROCEDURE OpenAccount(acc_id NUMBER, cust_id NUMBER, type VARCHAR2, bal NUMBER) IS
       BEGIN
         INSERT INTO Accounts VALUES (acc_id, cust_id, type, bal, SYSDATE);...
Show more...


Package Body ACCOUNTOPERATIONS compiled

Elapsed: 00:00:00.013
```

Submitted By:

Name : Lingaraj Nayak

Superset ID : 6387607