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.

Ans :

CODE :

```
DELIMITER //
CREATE PROCEDURE ApplyDiscountToLoanInterestRates()
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE cust_id INT;
   DECLARE dob DATE;
   DECLARE current_rate DECIMAL(5,2);
   DECLARE cur CURSOR FOR
      SELECT c.CustomerID, c.DOB, l.InterestRate
      FROM customers c
      JOIN loans l ON c.CustomerID = l.CustomerID
      WHERE TIMESTAMPDIFF(YEAR, c.DOB, CURDATE()) > 60;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
   OPEN cur;
   read_loop: LOOP
      FETCH cur INTO cust_id, dob, current_rate;
      IF done THEN
      END IF;
      UPDATE loans
      SET InterestRate = InterestRate - 1
      WHERE CustomerID = cust_id;
      SELECT CONCAT('Customer ID: ', cust_id, ', Old Rate: ', current_rate, ', New Rate: ', current_rate - 1)
AS RateUpdate;
   END LOOP;
   CLOSE cur;
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE PROCEDURE PromoteToVIP()
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE cust_id INT;
   DECLARE balance DECIMAL(10,2);
   DECLARE cur CURSOR FOR
      SELECT CustomerID, Balance
      FROM customers
      WHERE Balance > 10000;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
   OPEN cur;
   read_loop: LOOP
      FETCH cur INTO cust_id, balance;
      IF done THEN
         LEAVE read_loop;
      END IF;
      UPDATE customers
      SET IsVIP = TRUE
      WHERE CustomerID = cust_id;
      SELECT CONCAT('Customer ID: ', cust_id, ', Balance: ', balance, ' is now a VIP.') AS VIPStatus;
   END LOOP;
   CLOSE cur;
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE PROCEDURE SendLoanReminders()
BEGIN
   DECLARE done INT DEFAULT FALSE;
   DECLARE loan_id INT;
   DECLARE due_date DATE;
   DECLARE cur CURSOR FOR
```

```
        SELECT LoanID, EndDate
        FROM loans
        WHERE EndDate BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 30 DAY);
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO loan_id, due_date;
        IF done THEN
            LEAVE read_loop;
        END IF;
        SELECT CONCAT('Loan ID: ', loan_id, ', Due Date: ', due_date, ' - Reminder: Your loan is due soon.')
AS ReminderMessage;
    END LOOP;
    CLOSE cur;
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE PROCEDURE SafeTransferFunds(
    IN from_account INT,
    IN to_account INT,
    IN amount DECIMAL(10,2)
)
BEGIN
    DECLARE insufficient_funds EXCEPTION;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' ROLLBACK;
    START TRANSACTION;
    IF (SELECT Balance FROM accounts WHERE AccountID = from_account) < amount THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient funds';
    ELSE
        UPDATE accounts
        SET Balance = Balance - amount
        WHERE AccountID = from_account;
        UPDATE accounts
        SET Balance = Balance + amount
        WHERE AccountID = to_account;
    END IF;
```

```
   COMMIT;
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE PROCEDURE UpdateSalary(
   IN emp_id INT,
   IN percentage DECIMAL(5,2)
)
BEGIN
   DECLARE CONTINUE HANDLER FOR SQLSTATE '42S22'
   BEGIN
      SELECT 'Employee ID does not exist.' AS ErrorMessage;
   END;
   UPDATE employees
   SET Salary = Salary * (1 + percentage / 100)
   WHERE EmployeeID = emp_id;
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE PROCEDURE AddNewCustomer(
   IN p_CustomerID INT,
   IN p_Name VARCHAR(100),
   IN p_DOB DATE,
   IN p_Balance DECIMAL(10,2)
)
BEGIN
   DECLARE duplicate_entry EXCEPTION;
   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'
   BEGIN
      SELECT 'Customer with this ID already exists.' AS ErrorMessage;
```

```
    END;
    INSERT INTO customers (CustomerID, Name, DOB, Balance, LastModified)
    VALUES (p_CustomerID, p_Name, p_DOB, p_Balance, NOW());
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE PROCEDURE ProcessMonthlyInterest()
BEGIN
    UPDATE accounts
    SET Balance = Balance * 1.01
    WHERE AccountType = 'Savings';
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE PROCEDURE UpdateEmployeeBonus(
    IN dept VARCHAR(50),
    IN bonus_percentage DECIMAL(5,2)
)
BEGIN
    UPDATE employees
    SET Salary = Salary * (1 + bonus_percentage / 100)
    WHERE Department = dept;
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE PROCEDURE TransferFunds(
  IN from_account INT,
  IN to_account INT,
  IN amount DECIMAL(10,2)
)
BEGIN
  IF (SELECT Balance FROM accounts WHERE AccountID = from_account) < amount THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient funds';
  ELSE
    UPDATE accounts
    SET Balance = Balance - amount
    WHERE AccountID = from_account;
    UPDATE accounts
    SET Balance = Balance + amount
    WHERE AccountID = to_account;
  END IF;
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE FUNCTION CalculateAge(dob DATE)
RETURNS INT
BEGIN
  RETURN TIMESTAMPDIFF(YEAR, dob, CURDATE());
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE FUNCTION CalculateMonthlyInstallment(
    loan_amount DECIMAL(10,2),
    interest_rate DECIMAL(5,2),
    duration_years INT
)
RETURNS DECIMAL(10,2)
BEGIN
    DECLARE monthly_rate DECIMAL(5,2);
    DECLARE total_months INT;
    DECLARE installment DECIMAL(10,2);

    SET monthly_rate = interest_rate / 12 / 100;
    SET total_months = duration_years * 12;
    SET installment = loan_amount * (monthly_rate * POWER(1 + monthly_rate, total_months)) /
(POWER(1 + monthly_rate, total_months) - 1);
    RETURN installment;
END //
DELIMITER ;
```

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.

Ans

CODE :

```
DELIMITER //
CREATE FUNCTION HasSufficientBalance(
    account_id INT,
    amount DECIMAL(10,2)
)
RETURNS BOOLEAN
BEGIN
    DECLARE current_balance DECIMAL(10,2);
    DECLARE sufficient BOOLEAN;
    SELECT Balance INTO current_balance
    FROM accounts
    WHERE AccountID = account_id;
    IF current_balance >= amount THEN
        SET sufficient = TRUE;
    ELSE
        SET sufficient = FALSE;
    END IF;
    RETURN sufficient;
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON customers
FOR EACH ROW
BEGIN
   SET NEW.LastModified = NOW();
END //
DELIMITER ;
```

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.

Ans :

CODE:

```
DELIMITER //
CREATE TRIGGER LogTransaction
AFTER INSERT ON transactions
FOR EACH ROW
BEGIN
   INSERT INTO audit_log (TransactionID, AccountID, Amount, TransactionType, Timestamp)
   VALUES (NEW.TransactionID, NEW.AccountID, NEW.Amount, NEW.TransactionType, NOW());
END //
DELIMITER ;
```

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.

Ans :

CODE :

```
DELIMITER //
CREATE TRIGGER CheckTransactionRules
BEFORE INSERT ON transactions
FOR EACH ROW
BEGIN
   IF NEW.TransactionType = 'Withdrawal' AND
      (SELECT Balance FROM accounts WHERE AccountID = NEW.AccountID) < NEW.Amount THEN
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient funds for withdrawal';
```

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

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.

```
DELIMITER //
CREATE PROCEDURE GenerateMonthlyStatements()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE cust_id INT;
  DECLARE trans_date DATE;
  DECLARE trans_amount DECIMAL(10,2);
  DECLARE cur CURSOR FOR
    SELECT c.CustomerID, t.TransactionDate, t.Amount
    FROM customers c
    JOIN transactions t ON c.CustomerID = t.CustomerID
    WHERE MONTH(t.TransactionDate) = MONTH(CURDATE())
      AND YEAR(t.TransactionDate) = YEAR(CURDATE());
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  OPEN cur;
  read_loop: LOOP
    FETCH cur INTO cust_id, trans_date, trans_amount;
    IF done THEN
      LEAVE read_loop;
    END IF;
    SELECT CONCAT('Customer ID: ', cust_id, ', Date: ', trans_date, ', Amount: ', trans_amount) AS
Statement;
  END LOOP;
  CLOSE cur;
END //
DELIMITER ;
```

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.

```
DELIMITER //
CREATE PROCEDURE ApplyAnnualFee()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE acc_id INT;
  DECLARE balance DECIMAL(10,2);
  DECLARE cur CURSOR FOR
    SELECT AccountID, Balance
    FROM accounts;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  OPEN cur;
  read_loop: LOOP
    FETCH cur INTO acc_id, balance;
    IF done THEN
      LEAVE read_loop;
    END IF;
    UPDATE accounts
    SET Balance = Balance - 50
    WHERE AccountID = acc_id;
    SELECT CONCAT('Account ID: ', acc_id, ', New Balance: ', Balance - 50) AS UpdateMessage;
  END LOOP;
  CLOSE cur;
END //
DELIMITER ;
```

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.

```
DELIMITER //
CREATE PROCEDURE UpdateLoanInterestRates()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE loan_id INT;
  DECLARE current_rate DECIMAL(5,2);
  DECLARE cur CURSOR FOR
    SELECT LoanID, InterestRate
    FROM loans;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  OPEN cur;
  read_loop: LOOP
    FETCH cur INTO loan_id, current_rate;
    IF done THEN
```

```
         LEAVE read_loop;
      END IF;
      UPDATE loans
      SET InterestRate = InterestRate + 0.5
      WHERE LoanID = loan_id;
      SELECT CONCAT('Loan ID: ', loan_id, ', New Rate: ', InterestRate + 0.5) AS UpdateMessage;
   END LOOP;
   CLOSE cur;
END //
DELIMITER ;
```

Exercise 7: Packages

Scenario 1: Group all customer-related procedures and functions into a package.

- o **Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

```
DELIMITER //
CREATE PACKAGE CustomerManagement AS
   PROCEDURE AddNewCustomer(p_Name VARCHAR(100), p_DOB DATE, p_Balance DECIMAL(10,2));
   PROCEDURE UpdateCustomerDetails(p_CustomerID INT, p_Name VARCHAR(100), p_Balance
DECIMAL(10,2));
   FUNCTION GetCustomerBalance(p_CustomerID INT) RETURN DECIMAL(10,2);
END CustomerManagement //

DELIMITER //
CREATE PACKAGE BODY CustomerManagement AS
   PROCEDURE AddNewCustomer(p_Name VARCHAR(100), p_DOB DATE, p_Balance DECIMAL(10,2)) IS
   BEGIN
      INSERT INTO customers (Name, DOB, Balance, LastModified)
      VALUES (p_Name, p_DOB, p_Balance, NOW());
   END;
   PROCEDURE UpdateCustomerDetails(p_CustomerID INT, p_Name VARCHAR(100), p_Balance
DECIMAL(10,2)) IS
   BEGIN
      UPDATE customers
      SET Name = p_Name, Balance = p_Balance, LastModified = NOW()
      WHERE CustomerID = p_CustomerID;
   END;
   FUNCTION GetCustomerBalance(p_CustomerID INT) RETURN DECIMAL(10,2) IS
      v_Balance DECIMAL(10,2);
   BEGIN
      SELECT Balance INTO v_Balance FROM customers WHERE CustomerID = p_CustomerID;
      RETURN v_Balance;
   END;
END CustomerManagement //
```

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.

```
DELIMITER //
CREATE PACKAGE EmployeeManagement AS
   PROCEDURE HireNewEmployee(p_Name VARCHAR(100), p_Position VARCHAR(50), p_Salary
DECIMAL(10,2), p_Department VARCHAR(50));
   PROCEDURE UpdateEmployeeDetails(p_EmployeeID INT, p_Name VARCHAR(100), p_Position
VARCHAR(50), p_Salary DECIMAL(10,2));
   FUNCTION CalculateAnnualSalary(p_Salary DECIMAL(10,2)) RETURN DECIMAL(10,2);
END EmployeeManagement //

DELIMITER //
CREATE PACKAGE BODY EmployeeManagement AS
   PROCEDURE HireNewEmployee(p_Name VARCHAR(100), p_Position VARCHAR(50), p_Salary
DECIMAL(10,2), p_Department VARCHAR(50)) IS
   BEGIN
      INSERT INTO employees (Name, Position, Salary, Department, HireDate)
      VALUES (p_Name, p_Position, p_Salary, p_Department, NOW());
   END;
   PROCEDURE UpdateEmployeeDetails(p_EmployeeID INT, p_Name VARCHAR(100), p_Position
VARCHAR(50), p_Salary DECIMAL(10,2)) IS
   BEGIN
      UPDATE employees
      SET Name = p_Name, Position = p_Position, Salary = p_Salary
      WHERE EmployeeID = p_EmployeeID;
   END;
   FUNCTION CalculateAnnualSalary(p_Salary DECIMAL(10,2)) RETURN DECIMAL(10,2) IS
      v_AnnualSalary DECIMAL(10,2);
   BEGIN
      v_AnnualSalary := p_Salary * 12;
      RETURN v_AnnualSalary;
   END;
END EmployeeManagement //
```

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.

```
DELIMITER //
CREATE PACKAGE AccountOperations AS
  PROCEDURE OpenNewAccount(p_CustomerID INT, p_AccountType VARCHAR(50), p_Balance
DECIMAL(10,2));
  PROCEDURE CloseAccount(p_AccountID INT);
  FUNCTION GetTotalBalance(p_CustomerID INT) RETURN DECIMAL(10,2);
END AccountOperations //

DELIMITER //
CREATE PACKAGE BODY AccountOperations AS
  PROCEDURE OpenNewAccount(p_CustomerID INT, p_AccountType VARCHAR(50), p_Balance
DECIMAL(10,2)) IS
  BEGIN
    INSERT INTO accounts (CustomerID, AccountType, Balance, LastModified)
    VALUES (p_CustomerID, p_AccountType, p_Balance, NOW());
  END;

  PROCEDURE CloseAccount(p_AccountID INT) IS
  BEGIN
    DELETE FROM accounts WHERE AccountID = p_AccountID;
  END;
  FUNCTION GetTotalBalance(p_CustomerID INT) RETURN DECIMAL(10,2) IS
    v_TotalBalance DECIMAL(10,2);
  BEGIN
    SELECT SUM(Balance) INTO v_TotalBalance
    FROM accounts
    WHERE CustomerID = p_CustomerID;
    RETURN v_TotalBalance;
  END;
END AccountOperations //
```