Phase 5: Apex Programming (Developer)

Project: Hospital Management System - Salesforce CRM

Step 1: Classes & Objects

Apex is object-oriented, allowing creation of classes (blueprints) and objects (instances) to organize and reuse healthcare business logic.

In this project, a Trigger Handler Class approach was implemented to follow best practices for medical data processing.

Source Code:

```
apex
```

```
public class AppointmentTriggerHandler {

    public static void validateAppointmentSlots(List<Appointment__c> appointmentList) {
        for (Appointment__c appointment : appointmentList) {

            // Emergency appointments can be scheduled anytime
            if (appointment.Appointment_Type__c == 'Emergency') {
                continue;
            }

            // Regular appointments validation
            if (appointment.Appointment_Type__c == 'Regular') {
                if (appointment.Start_Date_Time__c == null) {
                    appointment.Start_Date_Time__c.addError('Regular appointments must have a scheduled time.')
                }
                // Check if appointment is within doctor's availability hours
                Time appointmentTime = appointment.Start_Date_Time__c.time();
                if (appointmentTime < Time.newInstance(9, 0, 0, 0) ||
                    appointmentTime > Time.newInstance(17, 0, 0, 0)) {
                    appointment.Start_Date_Time__c.addError('Regular appointments must be between 9 AM and 5
                }
            }

            // Follow-up appointments validation
            else if (appointment.Appointment_Type__c == 'Follow-up') {
                if (appointment.Patient__c == null) {
                    appointment.Patient__c.addError('Follow-up appointments must be linked to an existing patient
                }
            }

            // Consultation fee validation
            if (appointment.Fee__c == null || appointment.Fee__c <= 0) {
                appointment.Fee__c.addError('Appointment fee must be greater than 0.');
            }
        }
        System.debug('All appointment records validated successfully.');
    }
}
```

Step 2: Apex Triggers (before/after insert/update/delete)

Two custom Apex triggers were implemented to automate critical processes in the Hospital Management System:

a) Appointment Fee Trigger

Purpose:

- Automatically set the appointment fee based on doctor's consultation rate
- Formula: Doctor's Consultation Fee = Appointment Fee

Source Code:

```
apex
```

```
trigger AppointmentFeeTrigger on Appointment__c (before insert, before update) {

    Set<Id> doctorIds = new Set<Id>();

    // Collect all doctor IDs from the appointments
    for (Appointment__c appointment : Trigger.new) {
        if (appointment.Doctor__c != null) {
            doctorIds.add(appointment.Doctor__c);
        }
    }

    // Query doctor consultation fees
    Map<Id, Doctor__c> doctorMap = new Map<Id, Doctor__c>(
        [SELECT Id, Consultation_Fee__c FROM Doctor__c WHERE Id IN :doctorIds]
    );

    // Set appointment fees based on doctor's consultation fee
    for (Appointment__c appointment : Trigger.new) {
        if (appointment.Doctor__c != null &&
            doctorMap.containsKey(appointment.Doctor__c)) {

            Doctor__c doctor = doctorMap.get(appointment.Doctor__c);
            appointment.Fee__c = doctor.Consultation_Fee__c;

            // Apply emergency surcharge if applicable
            if (appointment.Appointment_Type__c == 'Emergency') {
                appointment.Fee__c = appointment.Fee__c * 1.5; // 50% emergency surcharge
            }
        }
    }
}
```

b) Doctor Availability Trigger

Purpose:

- Update doctor's daily appointment count and availability status
- Prevent overbooking by tracking appointment limits

Source Code:

```
apex
```

```
trigger DoctorAvailabilityTrigger on Appointment__c (after insert, after update, after delete) {

    Set<Id> doctorIds = new Set<Id>();
    Date today = Date.today();

    // Collect doctor IDs from appointments scheduled for today
    if (Trigger.isInsert || Trigger.isUpdate) {
        for (Appointment__c appointment : Trigger.new) {
            if (appointment.Doctor__c != null &&
                appointment.Start_Date_Time__c != null &&
                appointment.Start_Date_Time__c.date() == today) {
                doctorIds.add(appointment.Doctor__c);
            }
        }
    }

    // Handle deleted appointments
    if (Trigger.isDelete) {
        for (Appointment__c appointment : Trigger.old) {
            if (appointment.Doctor__c != null &&
                appointment.Start_Date_Time__c != null &&
                appointment.Start_Date_Time__c.date() == today) {
                doctorIds.add(appointment.Doctor__c);
            }
        }
    }

    if (doctorIds.isEmpty()) return;

    // Query current appointment counts for today
    Map<Id, Integer> appointmentCounts = new Map<Id, Integer>();

    for (AggregateResult ar : [SELECT Doctor__c, COUNT(Id) appointmentCount
                      FROM Appointment__c
                      WHERE Doctor__c IN :doctorIds
                      AND DAY_ONLY(Start_Date_Time__c) = TODAY
                      GROUP BY Doctor__c]) {
        appointmentCounts.put((Id)ar.get('Doctor__c'), (Integer)ar.get('appointmentCount'));
    }

    // Update doctor availability status
    List<Doctor__c> doctorsToUpdate = new List<Doctor__c>();

    for (Id doctorId : doctorIds) {
        Doctor__c doctor = new Doctor__c(Id = doctorId);
        Integer todayAppointments = appointmentCounts.get(doctorId) != null ?
                                   appointmentCounts.get(doctorId) : 0;

        doctor.Today_Appointments__c = todayAppointments;

        // Set availability status based on appointment count
        if (todayAppointments >= 8) {
            doctor.Availability_Status__c = 'Fully Booked';
        } else if (todayAppointments >= 6) {
            doctor.Availability_Status__c = 'Limited Availability';
        } else {
            doctor.Availability_Status__c = 'Available';
        }

        doctorsToUpdate.add(doctor);
    }

    if (!doctorsToUpdate.isEmpty()) {
        update doctorsToUpdate;
    }
}
```

Step 3: Patient History Trigger

Purpose:

- Automatically update patient's visit count and last visit date
- Track patient engagement for follow-up care

Source Code:

```
apex
```

```
trigger PatientHistoryTrigger on Appointment__c (after update) {

    Set<Id> patientIds = new Set<Id>();

    // Collect patient IDs from completed appointments
    for (Appointment__c appointment : Trigger.new) {
        Appointment__c oldAppointment = Trigger.oldMap.get(appointment.Id);

        // Check if appointment status changed to 'Completed'
        if (appointment.Status__c == 'Completed' &&
            oldAppointment.Status__c != 'Completed' &&
            appointment.Patient__c != null) {
            patientIds.add(appointment.Patient__c);
        }
    }

    if (patientIds.isEmpty()) return;

    // Update patient records
    List<Patient__c> patientsToUpdate = new List<Patient__c>();

    for (Id patientId : patientIds) {
        Patient__c patient = new Patient__c(Id = patientId);
        patient.Last_Visit_Date__c = Date.today();
        patient.Total_Visits__c = (patient.Total_Visits__c != null ?
                                   patient.Total_Visits__c + 1 : 1);
        patientsToUpdate.add(patient);
    }

    if (!patientsToUpdate.isEmpty()) {
        update patientsToUpdate;
    }
}
```

Key Implementation Benefits:

Data Integrity: Validation ensures accurate medical scheduling **Automation:** Triggers handle fee calculations and availability tracking **Business Logic:** Handler classes organize complex healthcare rules **Performance:** Efficient querying prevents system overload **Scalability:** Modular design supports future healthcare modules

---

Best Practices Implemented:

- **Trigger Handler Pattern:** Separates logic from trigger code
- **Bulk Processing:** Handles multiple records efficiently
- **SOQL Optimization:** Minimizes database queries
- **Error Handling:** Provides clear validation messages
- **Governor Limits:** Follows Salesforce limits and best practices

---

This Apex programming framework ensures robust healthcare data processing while maintaining system performance and reliability.