Detailed Document on Entity Mapping and Persistence for Hospital Management System

Introduction

This document provides an in-depth explanation of the entity mapping and persistence strategy for the hospital management system, designed using Spring Boot and JPA. The system models a hospital environment, including employees (doctors and nurses), departments, wards, patients, and hospitalizations. The relationships between these entities are managed through JPA annotations, which handle the underlying database mappings and persistence.

1. Entity Overview and Relationships

1.1 Employee Entity

The `Employee` entity serves as a base class for both `Doctor` and `Nurse` entities, representing a common set of attributes shared by all employees in the hospital.

- **Attributes**:
    - `employeeNumber`: Primary key, unique identifier for each employee.
    - `surname`, `firstName`, `address`, `telephoneNumber`: Basic employee details.
- **Inheritance Strategy**:
    - The `Employee` entity uses the `JOINED` inheritance strategy. This ensures that shared fields are stored in the `Employee` table, while fields specific to `Doctor` and `Nurse` are stored in their respective tables.

Code:

```
@Entity
@Data
@Inheritance(strategy = InheritanceType.JOINED)
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long employeeNumber;

    private String surname;
    private String firstName;
    private String address;
    private String telephoneNumber;
}
```

1.2 Doctor Entity

The `Doctor` entity inherits from `Employee` and adds specific attributes and relationships relevant to doctors.

- **Attributes**:
  - `speciality`: The medical specialisation of the doctor.
- **Relationships**:
  - `departments`: A doctor can be the director of multiple departments.
  - `patients`: A doctor can treat multiple patients.

Code:

```
@Entity
@Data
public class Doctor extends Employee {
    private String speciality;

    @OneToMany(mappedBy = "director")
    private Set<Department> departments;

    @OneToMany(mappedBy = "doctor")
    private Set<Patient> patients;
}
```

1.3 Nurse Entity

The `Nurse` entity also inherits from `Employee` and includes nurse-specific attributes.

- **Attributes**:
  - `salary`: The salary of the nurse.
  - `rotation`: The work rotation schedule of the nurse.
- **Relationships**:
  - `department`: Each nurse is assigned to a single department.
  - `ward`: A nurse can supervise one ward.

Code:

```
@Entity
@Data
public class Nurse extends Employee {
    private double salary;
    private String rotation;
    @ManyToOne
    @JoinColumn(name = "department_id")
    private Department department;

    @OneToOne(mappedBy = "supervisor")
    private Ward ward;
}
```

1.4 Department Entity

The `Department` entity represents various departments within the hospital.

- **Attributes**:
  - `code`: Primary key, unique identifier for each department.
  - `name`, `building`: Details about the department.
- **Relationships**:
  - `director`: A doctor directs the department.
  - `wards`: A department contains multiple wards.

Code:

```
@Entity
@Data
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    Private String code;
    private String building;

    @ManyToOne
    @JoinColumn(name = "director_id")
    private Doctor director;

    @OneToMany(mappedBy = "department")
    private Set<Ward> wards;
}
```

1.5 Ward Entity

The `Ward` entity represents individual wards within a department.

- **Attributes**:
    - `id`: Primary key, unique identifier for each ward.
    - `wardNumber`: Local identifier within the department.
    - `numberOfBeds`: Number of beds in the ward.
- **Relationships**:
    - `department`: Each ward belongs to a specific department.
    - `supervisor`: A nurse supervises the ward.
    - `hospitalizations`: A ward can have multiple hospitalizations.

Code:

```
@Entity
@Data
public class Ward {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private int wardNumber;
    private int numberOfBeds;

    @ManyToOne
    @JoinColumn(name = "department_id")
    private Department department;

    @OneToOne
    @JoinColumn(name = "supervisor_id")
    private Nurse supervisor;

    @OneToMany(mappedBy = "ward")
    private Set<Hospitalisation> hospitalisations;

}
```

1.6 Patient Entity

The `Patient` entity represents patients treated in the hospital.

- **Attributes**:
  - `patientNumber`: Primary key, unique identifier for each patient.
  - `surname`, `firstName`, `address`, `telephoneNumber`, `diagnosis`: Patient details.
- **Relationships**:
  - `doctor`: A patient is treated by a doctor.
  - `hospitalisations`: A patient can have multiple hospitalisations.

Code:

```java
@Entity
@Data
public class Patient {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long patientNumber;

    private String surname;
    private String firstName;
    private String address;
    private String telephoneNumber;
    private String diagnosis;

    @ManyToOne
    @JoinColumn(name = "doctor_id")
    private Doctor doctor;

    @OneToMany(mappedBy = "patient")
    private Set<Hospitalisation> hospitalisations;

    // Getters and Setters
}
```

1.7 Hospitalisation Entity

The `Hospitalisation` entity tracks the history of a patient's stays in different wards.

- **Attributes**:
  - `id`: Primary key, unique identifier for each hospitalisation.
  - `bedNumber`: The specific bed in the ward assigned to the patient.

- **Relationships**:
    - `patient`: The patient who is hospitalised.
    - `ward`: The ward where the patient is hospitalised.

Code:

```
@Entity
@Data
public class Hospitalisation {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private int bedNumber;

    @ManyToOne
    @JoinColumn(name = "patient_id")
    private Patient patient;

    @ManyToOne
    @JoinColumn(name = "ward_id")
    private Ward ward;

}
```

2. Persistence and Database Interaction

2.1 Inheritance Strategy

The `JOINED` inheritance strategy is used for the `Employee` entity to ensure that common attributes (like `surname`, `firstName`, etc.) are stored in a single table, while `Doctor` and `Nurse`-specific attributes are stored in separate tables. This approach normalises the database schema and avoids redundancy.

2.2 One-to-Many and Many-to-One Relationships

Most relationships in this model use `@ManyToOne` and `@OneToMany` annotations:

- **Department to Ward**: A `Department` has many `Wards`, but a `Ward` belongs to one `Department`.
- **Doctor to Patient**: A `Doctor` treats many `Patients`, but each `Patient` is treated by one `Doctor`.

2.3 One-to-One Relationship

The `Ward` entity has a one-to-one relationship with the `Nurse` entity, where a `Ward` is supervised by a single `Nurse`, and a `Nurse` can supervise only one `Ward`.

2.4 Many-to-Many Relationship

This system does not directly involve a many-to-many relationship. However, if needed, it could be introduced, for example, if patients could be treated by multiple doctors.

2.5 Cascade and Fetch Types

Proper use of `cascade` and `fetch` strategies in relationships ensures the integrity of operations and performance:

- **Cascade**: For example, `CascadeType.ALL` might be used for `Department` to `Ward` to automatically persist or remove associated `Wards` when a `Department` is persisted or removed.
- **Fetch**: `FetchType.LAZY` is typically used for large collections (like `wards` in `Department`) to optimise performance.

3. Database Schema Generation

When the Spring Boot application runs, JPA automatically generates the database schema based on these entities and relationships. The `ddl-auto` property in `application.properties` controls this behaviour:

properties code
```
spring.jpa.hibernate.ddl-auto=update
```

This setting ensures that the database schema is updated automatically with any changes in the entity classes.

4. Conclusion

This document outlines the JPA entity mapping and persistence strategy for a hospital management system in a Spring Boot application. By following these guidelines, the system ensures efficient database operations, robust data integrity, and a clear representation of relationships within the hospital domain.