Gestion d'un hôpital

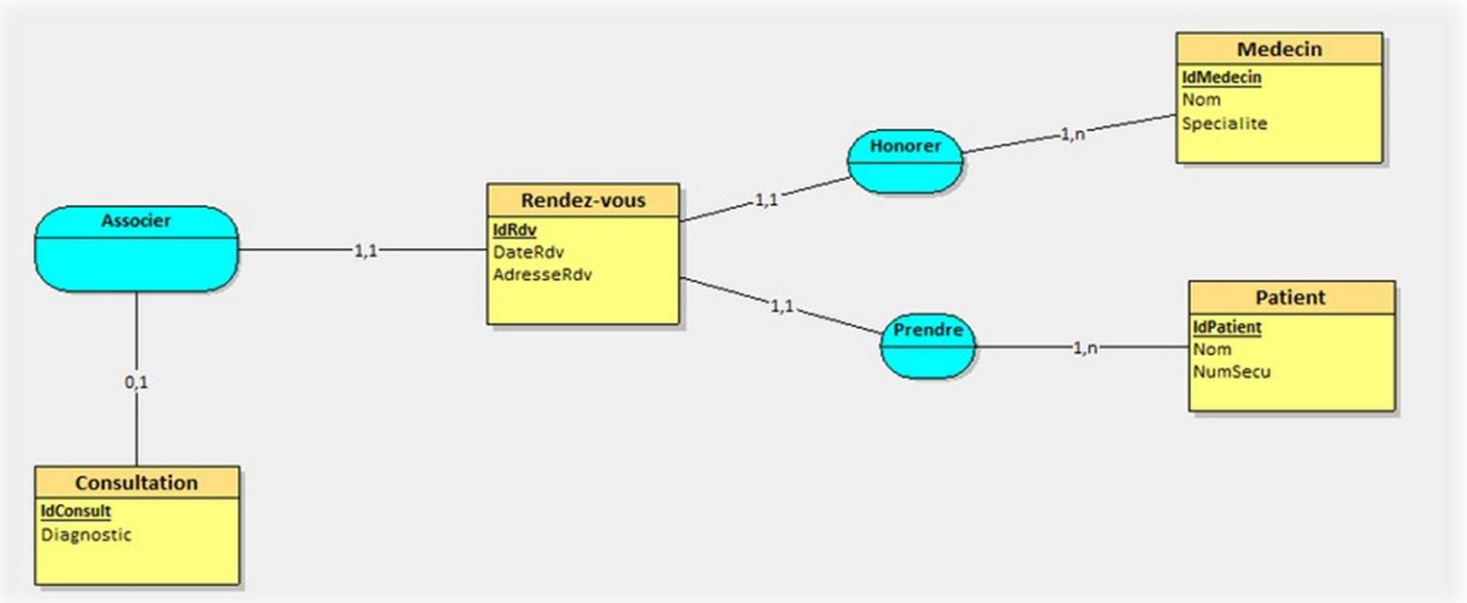
Lara Abi Rizk

Cahier de Charge

- Un hôpital souhaite gérer les rendez-vous des consultations des patients effectuées par les médecins.
- Chaque rendez-vous concerne un patient et un médecin.
- Pour chaque rendez-vous, on associe une seule consultation à l’issue du rendez-vous.
- Un patient peut prendre plusieurs rendez-vous.

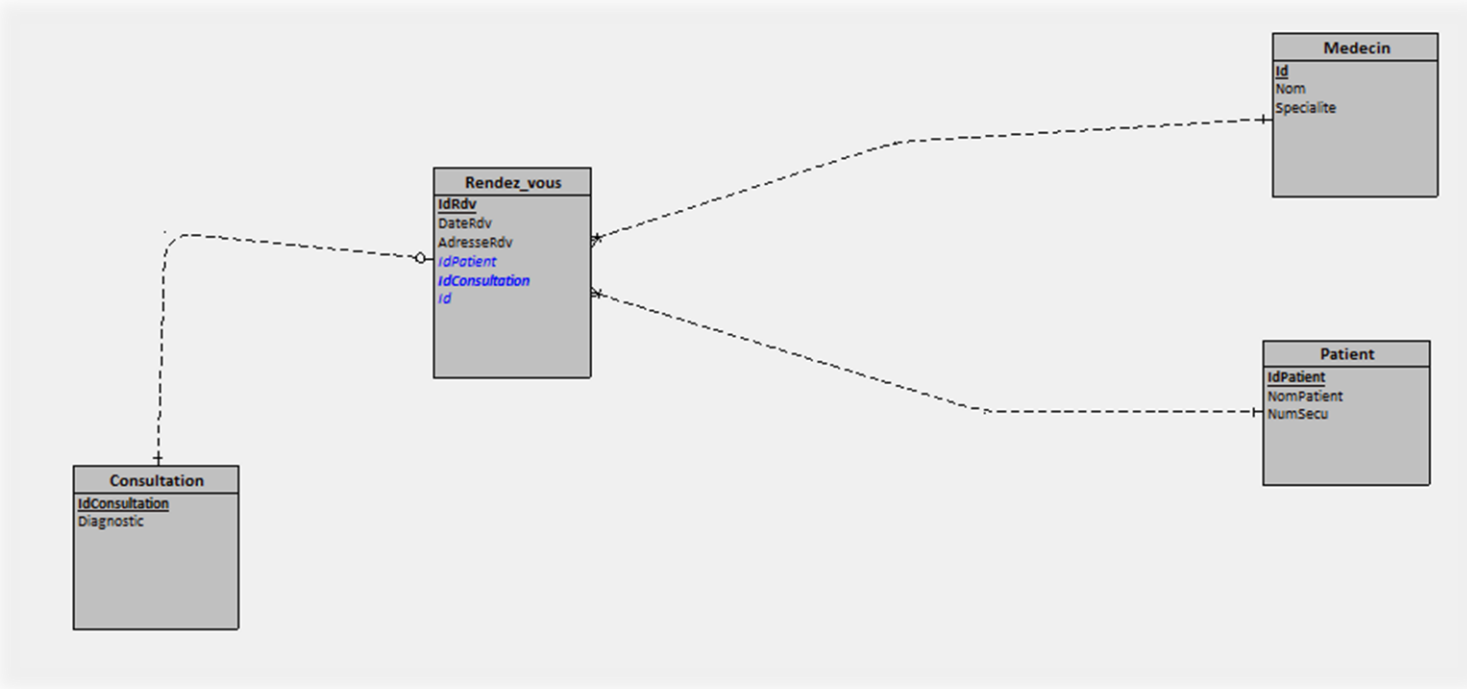
Conception

Modèle Conceptuel des Données (MCD)



Nous avons opté pour un modèle conceptuel simple dans l'optique de pouvoir Développer facilement l'application conformément au dossier de conception. Notre choix du MCD nous a permis de nous focaliser sur les méthodes dans java et faire quelques tests unitaires.

Modèle Physique des Données



Modèle logique de données (MLD)

- **Medecin = (Id**, Nom Specialite**)**
- **Patient = (IdPatient**, NomPatient ,NumSecu **)**
- **Consultation = (IdConsultation**, Diagnostic**)**
- **Rendez_vous = (IdRdv** ,DateRdv, AdresseRdv, *#IdPatient, **#IdConsultation**, #Id***);**

Analyse

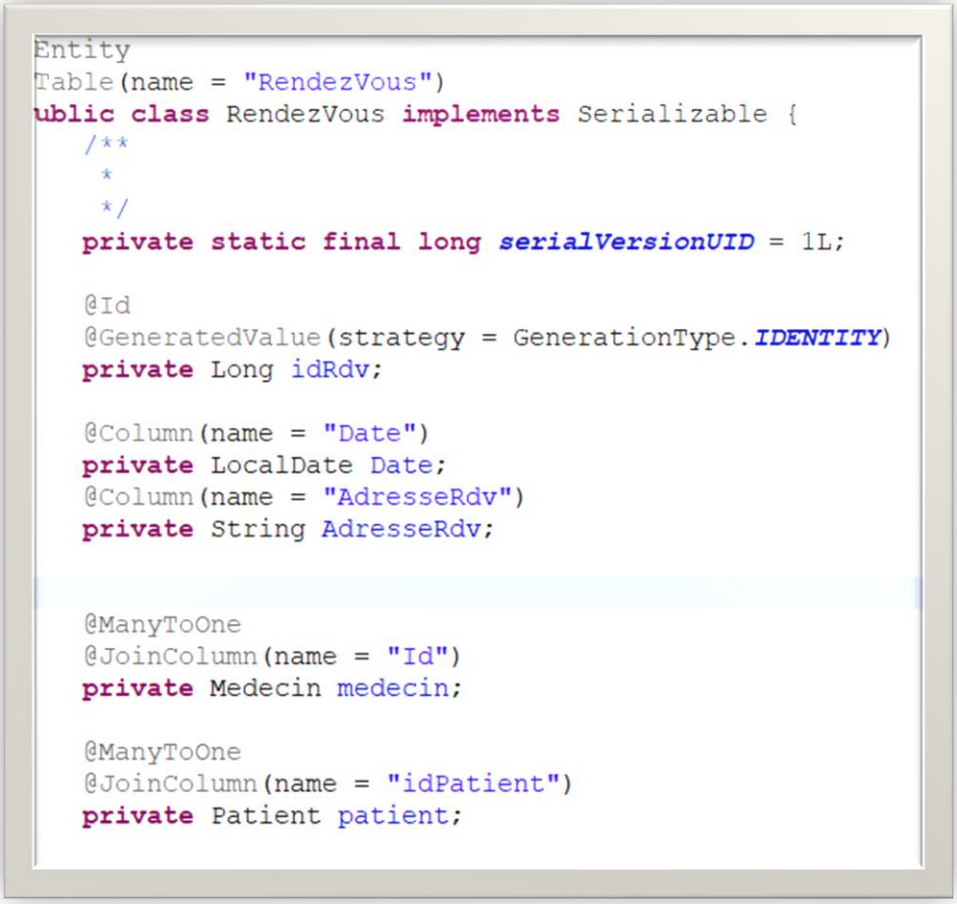
- **Un Médecin peut honorer par un ou plusieurs rendez-vous.**
- **Dans notre code, cela se traduit par une relation @OneToMany dans la classe Medecin.models**

```
@Entity
@Table(name="medecin")

public class Medecin implements Serializable {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long Id;
    @Column(name = "Nom")
    private String Nom;
    @Column(name = "Specialite")
    private String Specialite;


    //FK de Medecin s'affiche dans la table RendezVous

    @OneToMany(mappedBy = "medecin", fetch = FetchType.LAZY)
    private Collection<RendezVous> lesRendezVous;


    @JsonIgnore
    public Collection<RendezVous> getLesRendezVous() {
        return lesRendezVous;
    }
```

Rien n'empêche un patient de prendre un ou plusieurs **Rendez-Vous**. Par conséquent, nous ajoutons la relation **@OneToMany** dans la classe **Patient.Models.**

```
Entity
Table(name = "Patient")
ublic class Patient {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idPatient;
    @Column(name = "NomPatient")
    private String NomPatient;
    @Column(name = "NumSecu")
    private Long NumSecu;


    @OneToMany(mappedBy = "Patient", fetch = FetchType.LAZY)
    private Collection<RendezVous> lesRendezVous;

    @JsonIgnore
    public Collection<RendezVous> getLezRendezVous() {
        return lesRendezVous;
    }
```

Connectons maintenant la table de l'objet Rendez-Vous au deux autres tables Medecin et Patient.
Nous savons qu'un Rendez-Vous peut être honoré par un et un seul médecin et qu'un patient peut prendre un ou plusieurs rendez-Vous. Dans notre classe RendezVous.models, nous ajoutons deux relations **@ManyToOne** vers les deux tables des objets Medecin et Patient.



```
Entity
Table(name = "RendezVous")
ublic class RendezVous implements Serializable {
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idRdv;

    @Column(name = "Date")
    private LocalDate Date;
    @Column(name = "AdresseRdv")
    private String AdresseRdv;


    @ManyToOne
    @JoinColumn(name = "Id")
    private Medecin medecin;

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

Passons à l'objet **Consultation.** Une Consultation peut être associée à un et un seul **Rendez-vous**. Un Rendez-Vous peut être associé à une et une seule consultation. Dans les deux classes Rendez-Vous et Consultation, nous ajoutons une relation @OneToOne.

Dans RendezVous.models :

```
@OneToOne

private Consultation consultation
```

Dans Consultation.models :

```
@OneToOne(mappedBy = "Consultation", fetch = FetchType.LAZY)

private RendezVous lesRendezVous
```

P.S : Pour faire le MCD Looping nous ne permet pas d'ajouter la relation @OneToOne de Consultation vers Rendez-Vous.

Il nous impose un ZeroToOne. Pour remédier à ça dans notre code, nous avons ajouté dans la classe consultation la relation

```
@OnetoOne

private Consultation consultation
```

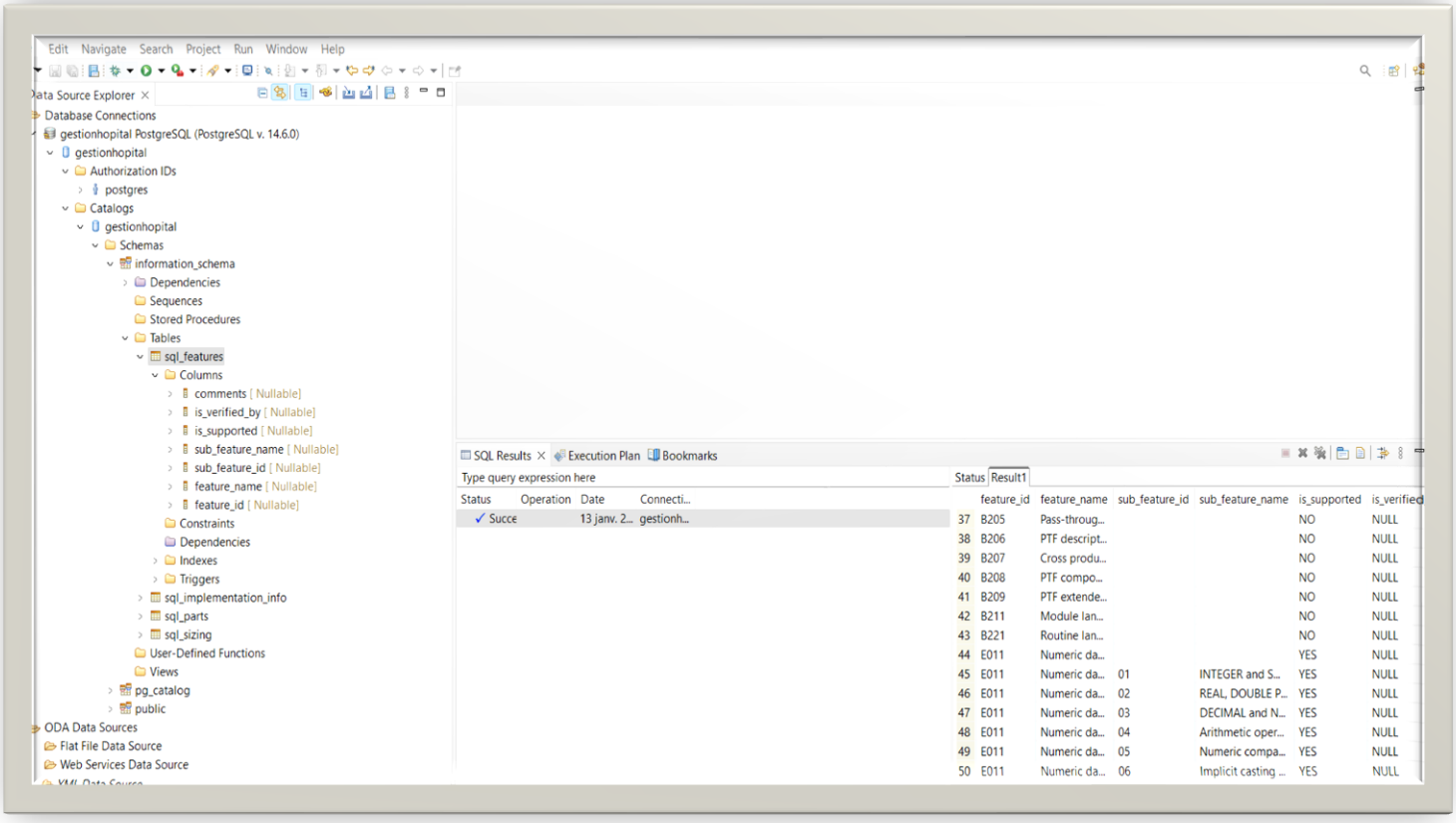
Pour aussi avoir une trace de rendezVous dans Consultation

Le MLD nous parait maintenant très logique, les PK de Medecin ,Patient et Consultation migrents dans la table RendezVous et dans Consultation nous avons bien une trace de RendezVous.

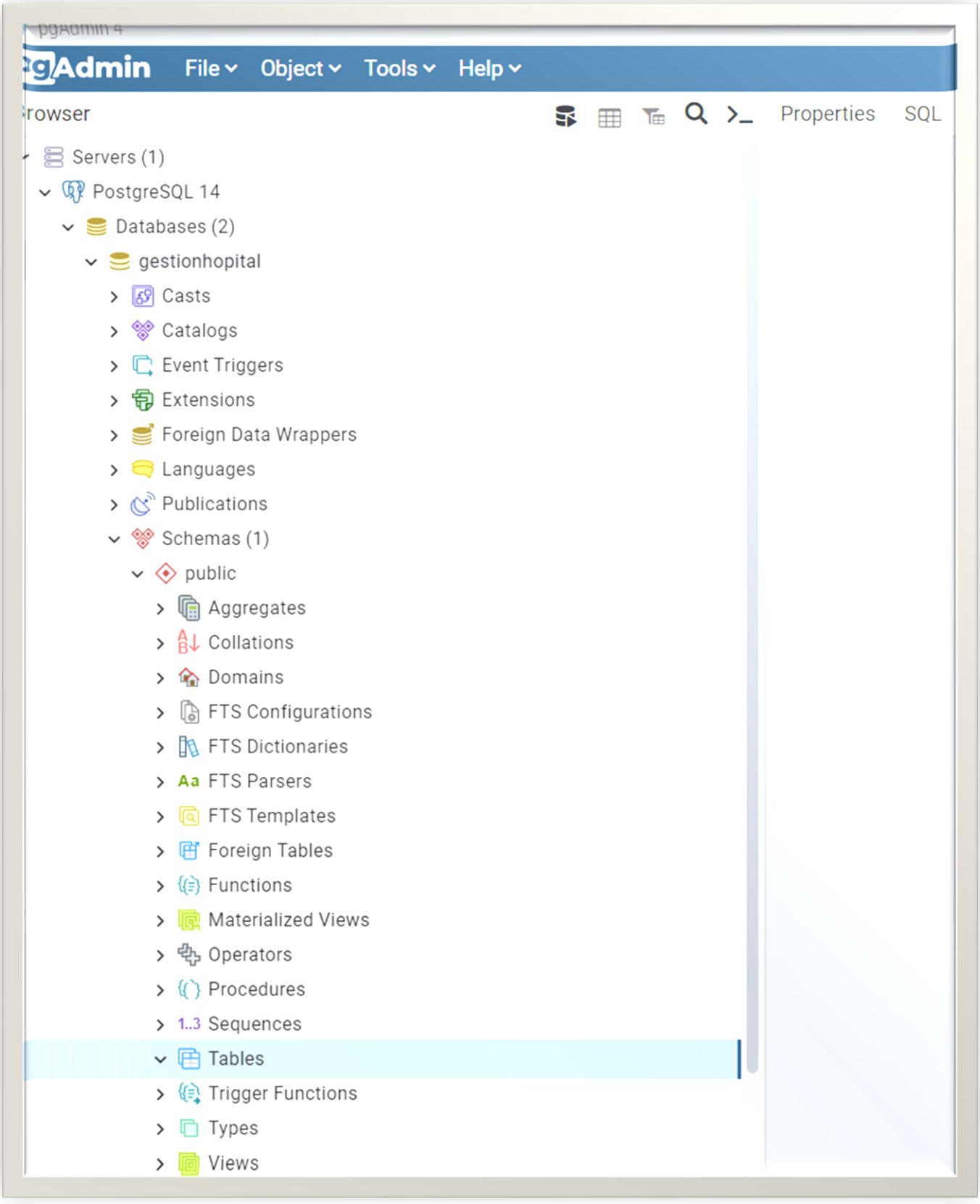
- **Medecin = (Id**, Nom Specialite**)**
- **Patient = (IdPatient**, NomPatient ,NumSecu **)**
- **Consultation = (IdConsultation**, Diagnostic***,#RendezVous*****)**
- **Rendez_vous = (IdRdv** ,DateRdv, AdresseRdv*, #IdPatient, #IdConsultation, #Id*****);**

Difficultés et contraintes

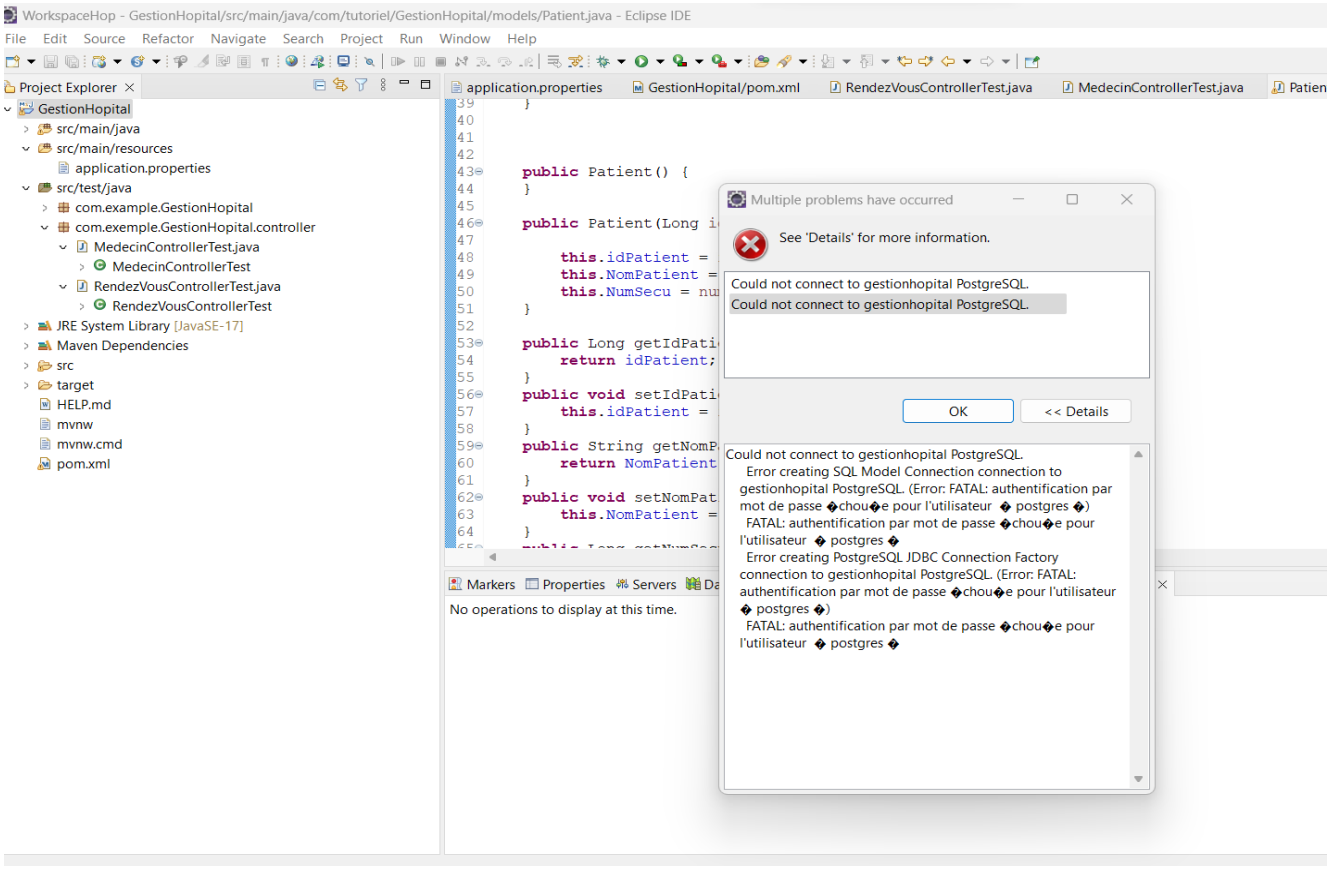
Je n’ai malheureusement pas réussi la connection à une base de donées. J’ai essayé avec Maria ainsi qu’avec Postgresql en téléchargeant postgresql-42.5.1.jar , la création d’une base de donnée dans Eclipse à bien eu lieu mais les tables n’ont pas été créés.



Le même problème dans PgAdmin, la base est vide…

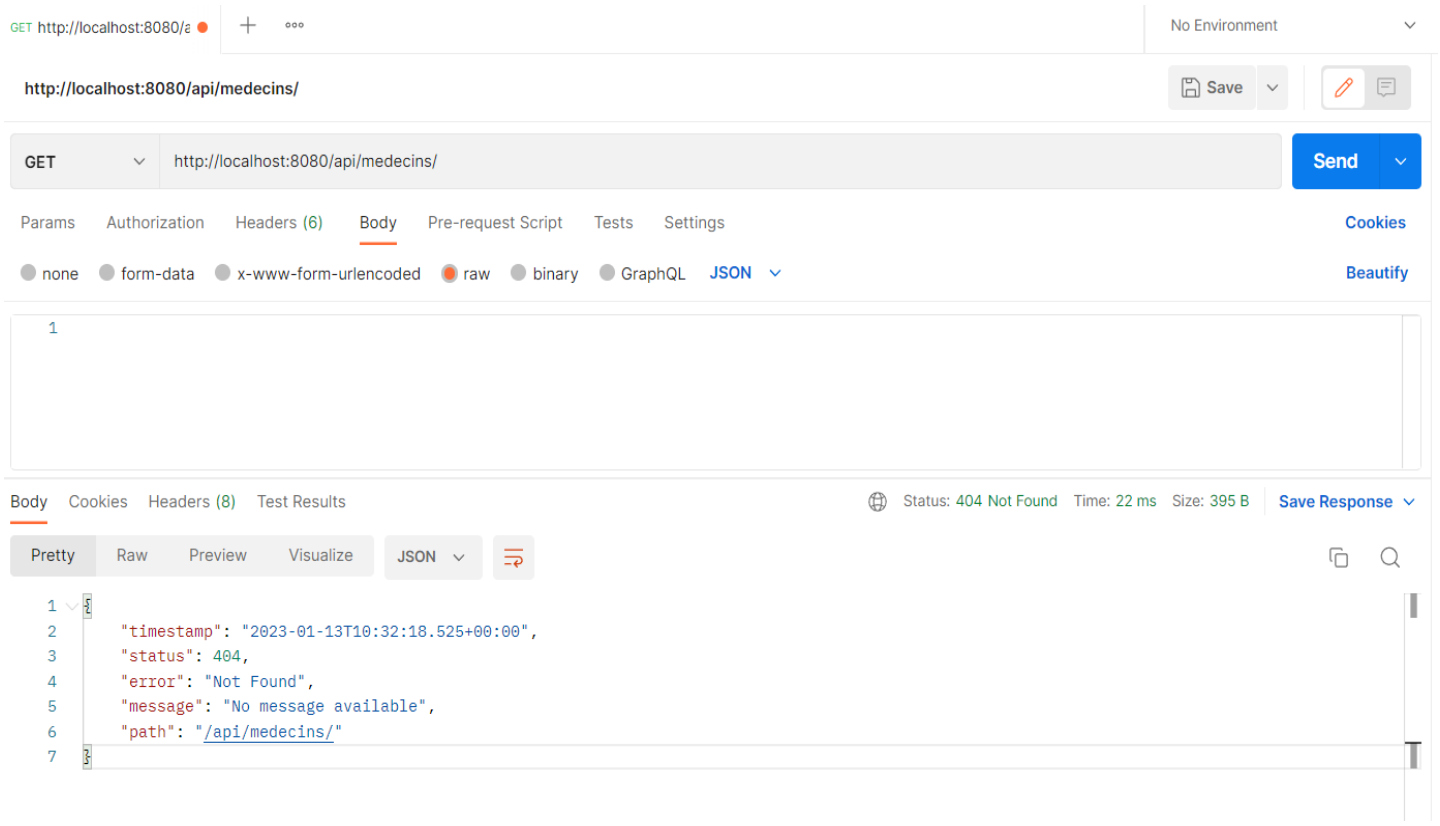


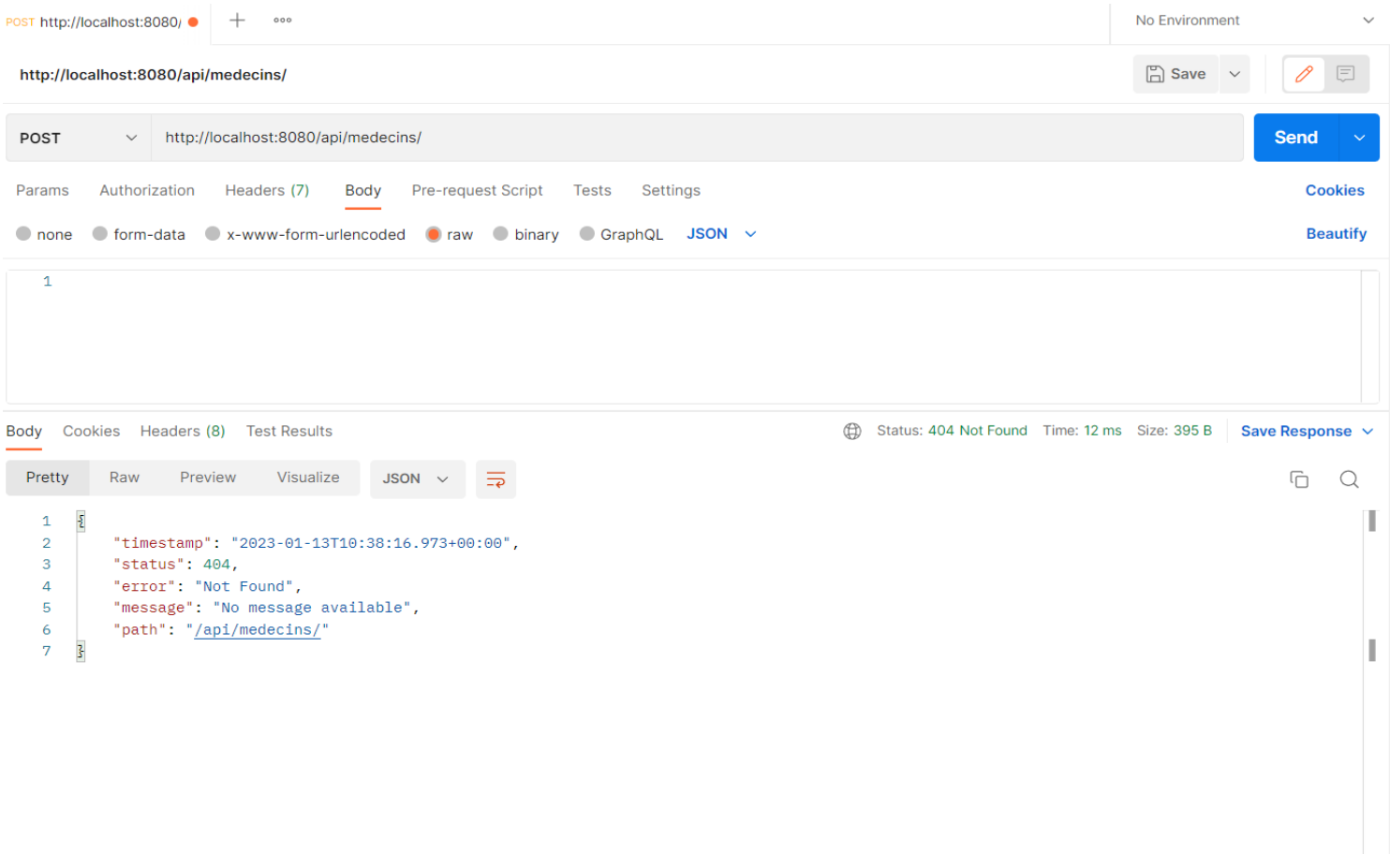
J’ai essayé de contourner ce problème mais l’erreur ci-dessous persiste et la BD reste vide.



Je n’ai malheureusement pas testé mes méthodes sur Postman, l’erreur à persister malgré toutes mes tentatives à contourner ce problème.

*





Merci