



UNIVERSITÉ COTE D'AZUR/FDS(UEH)

CASSANDRA : Gestion d'un cabinet médical

Etudiants :

Douilly RODELY
Djimy SURLIN
Pierre Rubens MILORME
Bob Charlemagne PIERRE

Professeur :

Gabriel MOPOLO-MOKE

Table des matières

1	Choix du sujet	3
1.1	SUJET	3
1.2	Description du sujet	3
2	MCD MERISE	4
2.1	Dictionnaire de données MERISE	4
2.2	La description textuelle des associations	7
3	Descriptions textuelles des associations	7
3.1	Association « Effectuer » entre MEDECIN et CONSULTATION	8
3.2	Association « Inclure » entre FACTURE et CONSULTATION	8
3.3	Association « Passer » entre PATIENT et CONSULTATION	8
3.4	Association « Nécessiter » entre CONSULTATION et EXAMEN	8
3.5	Association « Recevoir » entre PATIENT et FACTURE	8
3.6	Association « Contenir » entre CONSULTATION et PRESCRIPTION	8
3.7	Association « Avoir_Rendez_Vous » entre PATIENT et MEDECIN	8
3.8	Association « Posséder » entre ANTECEDENTS_MEDICAUX et PATIENT	8
3.9	Association « Avoir » entre PATIENT et ALLERGIE	9
3.10	La définition du Modèle Entité-Association MERISE	9
4	Conversion du MCD MERISE en des objets CASSANDRA et classes java	9
4.1	Spécification des Modèles de données pour CASSANDRA	9
4.1.1	Table "Medecin_By_Speciality"	9
4.1.2	Table "Patient_By_BirthDay"	10
4.1.3	Table "RendezVous_By_Date"	10
4.1.4	Table "Consultation_By_Date"	10
4.2	Spécification des classes et des méthodes JAVA	11
4.2.1	Méthodes CRUD	11
4.2.2	Indexation secondaire	22
4.2.3	Méthode de Consultation (Jointure, Groupement)	22
5	Compléments sur le moteur NoSql CASSANDRA	23
5.1	Modèles de données supportés	23
5.1.1	Modèle de données : orienté famille de colonnes (Wide Column)	23
5.1.2	Usage	23
5.2	Procédure d'installation du moteur et des utilitaires	23
5.2.1	Installation Windows	23
5.2.2	Installation (Linux)	25
5.2.3	Installation via Docker	25
5.2.4	Utilitaires	25
5.3	Architecture du moteur NoSQL (schéma)	25
5.3.1	Architecture	25
5.3.2	Schéma	25
5.4	Méthode de partitionnement	26

5.4.1	Sharding	26
5.4.2	Schéma	26
5.5	Méthode de réplication	26
5.5.1	Réplication	26
5.5.2	Schéma	27
5.6	Montée en charge	27
5.6.1	Schéma	27
5.7	Gestion du cache mémoire	27
5.7.1	Schéma	28
6	Génération automatique des données	28
6.1	Script de génération des données	28

1 Choix du sujet

1.1 SUJET

Gestion d'un cabinet médical

1.2 Description du sujet

Cette application pour un cabinet médical permet la gestion des patients, des médecins, des rendez-vous, des consultations, des prescriptions, des examens et de la facturation. Les patients peuvent être enregistrés avec leurs détails personnels, tandis que les médecins sont répertoriés avec leur spécialité respective. Les rendez-vous entre patients et médecins sont programmés. Chaque consultation est consignée et associée à un patient, un médecin et une facture. Les prescriptions médicales sont enregistrées et liées à la consultation correspondante. Les détails des examens sont stockés et également liés à la consultation. Chaque consultation génère une facture avec le montant total à payer.

En résumé, cette application fournit un système complet pour gérer les opérations quotidiennes d'un cabinet médical, optimisant le suivi des patients et la gestion des consultations et des facturations.

2 MCD MERISE

2.1 Dictionnaire de données MERISE

Entité : PATIENT					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Patient#	Identifiant unique du patient	Entier	Long	Valeur unique, non nulle	Oui
Num_Sec_Social	Numéro de Sécurité Sociale du patient	Chaîne	Caractère variable, 50 caractères max	non nulle	Non
Nom	Nom de famille du patient	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
Sexe	Sexe du patient	Chaîne	Caractère, 4 caractères max	Non nulle	Non
Date_naissance	Date de naissance du patient	Date		Non nulle	Non
Poids	Poids du patient	Décimal	Numérique	Non nulle	Non
Hauteur	Poids du patient	Décimal	Numérique	Non nulle	Non
listTelephones	Chaîne	Caractère variable, 50 caractères max		Non nulle	Non
listPrenoms	Prénoms du patient	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
Adresse	Adresse du patient	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non

Email	Email du patient	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
Entité : ALLERGIE					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id _Allergie#	Identifiant allergie	Entier	Long	Valeur unique, oui nulle	Oui
nom _allergie	nom de l'allergie	Chaîne	Caractère variable, 50 caractères max	Valeur unique, oui nulle	non
Entité : ANTECEDENTS _MEDICAUX					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id _ant _medic#	Identifiant	Entier	Long	Valeur unique, oui nulle	Oui
description	description	Chaîne	Caractère variable, 50 caractères max	non nulle	non
date _antec _medic	date de l'allergie	Date		non nulle	non
Entité : FACTURE					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id _Facture#	Identifiant unique de la facture	Entier	Long	Valeur unique, non nulle	Oui
Montant _Total	Montant total de la facture	Décimal	Numérique	Non nulle	Non
Date _Facture	Date d'émission de la facture	Date		Non nulle	Non
Entité : EXAMEN					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id _Examen#	Identifiant unique de l'examen	Entier	Long	Valeur unique, non nulle	Oui

Details_Examen	Détails de l'examen	Chaîne	200 caractères max	Non nulle	Non
Date_Examen	Date de l'examen	Date		Non nulle	Non
Entité : PRESCRIPTION					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Prescription#	Identifiant unique de la prescription	Entier	Long	Valeur unique, non nulle	Oui
Details_Prescription	Détails de la prescription	Chaîne	200 caractères max	Non null	Non
Date_Prescription	Date de la prescription	Date		Non null	Non
Entité : CONSULTATION					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Consultation#	Identifiant unique de la consultation	Entier	Long	Valeur unique, non nulle	Oui
Raison	Raison de la consultation	Chaîne	Caractère variable, 200 caractères max	Non nulle	Non
Diagnostic	Diagnostic au terme de la consultation	Chaîne	Caractère variable, 300 caractères max	Non nulle	Non
Date_Consultation	Date de la consultation	Date		Non nulle	Non
Entité : MEDECIN					
Attributs	Description	Types	Formats	Contraintes	Identifiants
Id_Medecin#	Identifiant unique du médecin	Entier	Long	Valeur unique, non nulle	Oui
listTelephones	telephones du médecin	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non

Sexe	Sexe du médecin	Chaîne	Caractère, 4 caractères max	Non nulle	Non
Spécialité	Spécialité du médecin	Chaîne	Caractère, 4 caractères max	Non nulle	Non
Nom	Nom de famille du médecin	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
listPrenoms	Prénoms du médecin	Chaîne		Non nulle	Non
Adresse	Adresse du médecin	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
Email	Email du médecin	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
listTelephones	Numéros de téléphone du médecin	Chaîne	Caractère variable, 50 caractères max	Non nulle	Non
Date_naissance	Date de naissance du médecin	Date		Non nulle	Non
CV	CV du médecin	CLOB		Non nulle	Non

TABLE 1 – Dictionnaire de données MERISE

2.2 La description textuelle des associations

3 Descriptions textuelles des associations

Dans cette section sont décrites les associations entre les différentes entités. Une association permet de mettre en relation deux ou plusieurs entités :

3.1 Association « Effectuer » entre MEDECIN et CONSULTATION

Cette association indique que chaque consultation est effectuée par un médecin. Un médecin peut effectuer plusieurs consultations, mais une consultation est effectuée par un seul médecin à la fois.

3.2 Association « Inclure » entre FACTURE et CONSULTATION

Cette association représente le fait qu'une facture inclut une consultation. Chaque consultation peut être incluse dans une seule facture, et réciproquement une facture peut référencer une seule consultation.

3.3 Association « Passer » entre PATIENT et CONSULTATION

Cette association signifie qu'un patient passe une consultation. Chaque consultation est passée par un seul patient, mais un patient peut passer plusieurs consultations.

3.4 Association « Nécessiter » entre CONSULTATION et EXAMEN

Cette association indique que chaque consultation peut nécessiter plusieurs examens. Chaque examen est associé à une seule consultation.

3.5 Association « Recevoir » entre PATIENT et FACTURE

Cette association représente le fait qu'un patient peut recevoir une facture. Chaque facture est destinée à un seul patient, mais un patient peut recevoir plusieurs factures.

3.6 Association « Contenir » entre CONSULTATION et PRESCRIPTION

Cette association signifie qu'une consultation peut contenir plusieurs prescriptions. Chaque prescription est associée à une seule consultation.

3.7 Association « Avoir _ Rendez _ Vous » entre PATIENT et MEDECIN

Cette association indique que chaque rendez-vous est entre un patient et un médecin. Chaque rendez-vous est pris par un patient avec un médecin spécifique.

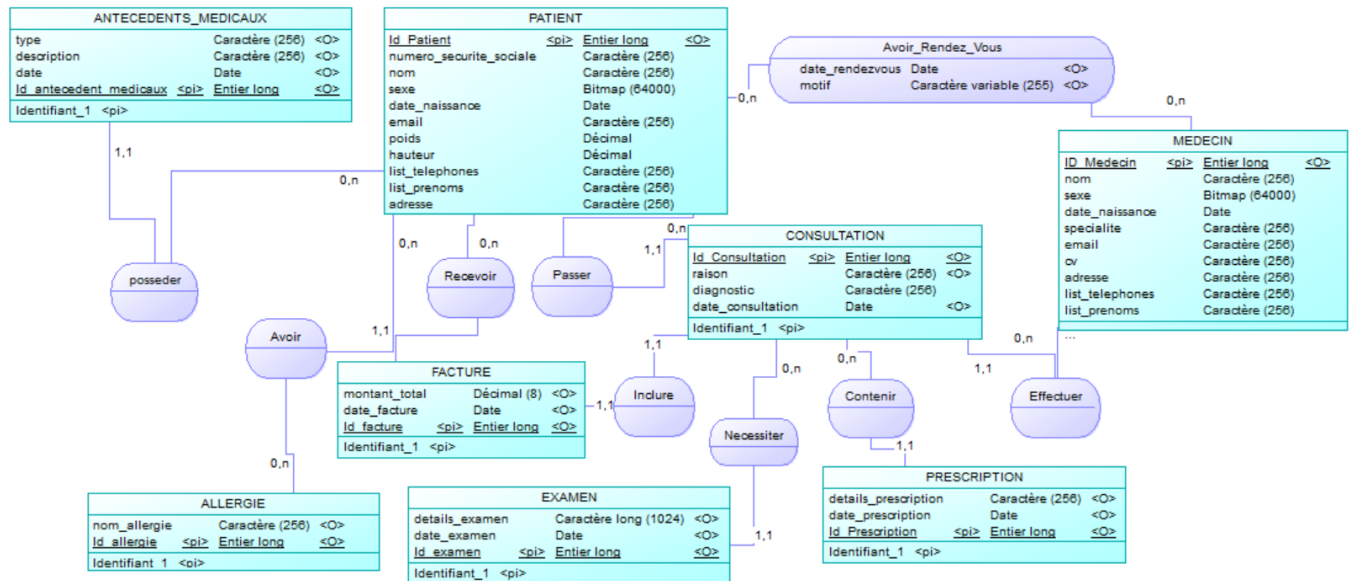
3.8 Association « Posséder » entre ANTECEDENTS _ MEDICAUX et PATIENT

Cette association indique que chaque patient peut avoir plusieurs antécédents médicaux et un antécédent médical ne peut être lié qu'à un seul patient.

3.9 Association « Avoir » entre PATIENT et ALLERGIE

Cette association indique que chaque patient peut avoir plusieurs allergies et plusieurs patients peuvent avoir une même allergie.

3.10 La définition du Modèle Entité-Association MERISE



4 Conversion du MCD MERISE en des objets CASSANDRA et classes java

4.1 Spécification des Modèles de données pour CASSANDRA

Cassandra est une base de données NoSQL orientée colonnes, idéale pour les applications nécessitant une gestion distribuée et une tolérance aux pannes. Les données sont organisées sous forme de tables, mais ne suivent pas un modèle strictement relationnel.

4.1.1 Table "Medecin_By_Speciality"

```

1 CREATE TABLE Medecin_By_Speciality (
2     id UUID,
3     nom text,
4     sexe text,
5     date_naissance date,
6     specialite text,
7     email text,
8     cv text,
9     adresse FROZEN<map<text, text>>,
10    list_telephones set<text>,
11    list_prenoms set<text>,
12    PRIMARY KEY(specialite, date_naissance, nom, sexe, email, id)

```

```
13 );
```

Clé de partition : specialite

4.1.2 Table "Patient_By_BirthDay"

```
1 CREATE TABLE Patient_By_BirthDay (
2     id UUID,
3     numero_securite_sociale text,
4     nom text,
5     sexe text,
6     date_naissance date,
7     email text,
8     poids double,
9     hauteur double,
10    list_telephones set<text>,
11    list_prenoms set<text>,
12    adresse FROZEN<map<text, text>>,
13    allergies list<text>,
14    PRIMARY KEY(date_naissance, nom, sexe, email, id)
15 );
```

Clé de partition : date_naissance

4.1.3 Table "RendezVous_By_Date"

```
1 CREATE TABLE RendezVous_By_Date(
2     rendezvous_date date,
3     patient_id UUID,
4     doctor_id UUID,
5     motif text,
6     PRIMARY KEY(rendezvous_date, patient_id, doctor_id)
7 );
```

Clé de partition : rendezvous_date

4.1.4 Table "Consultation_By_Date"

```
1 CREATE TABLE Consultation_By_Date (
2     consultation_date date,
3     patient_id UUID,
4     doctor_id UUID,
5     raison text,
6     diagnostic text,
7     facture FROZEN<map<text, text>>,
8     prescriptions list<FROZEN<map<text, text>>>,
9     examens list<FROZEN<map<text, text>>>,
10    PRIMARY KEY(consultation_date, patient_id, doctor_id)
```

```
11      );
```

Clé de partition : consultation_date

4.2 Spécification des classes et des méthodes JAVA

4.2.1 Méthodes CRUD

```
1.      // Classe Medecin
2      package org.example.entities;
3
4      import java.time.LocalDate;
5      import java.util.List;
6      import java.util.Map;
7      import java.util.Set;
8      import java.util.UUID;
9
10     public class Medecin {
11         private UUID id;
12         private String nom;
13         private String sexe;
14         private LocalDate dateNaissance;
15         private String specialite;
16         private String email;
17         private String cv;
18         private Map<String, String> adresse;
19         private Set<String> listTelephones;
20         private Set<String> listPrenoms;
21         private List<Consultation> consultations;
22         private List<RendezVous> rendezVous;
23
24         public Medecin(UUID id, String nom, String sexe,
25             LocalDate dateNaissance, String specialite,
26             String email, String cv, Map<String,
27                 String> adresse,
28             Set<String> listTelephones, Set<String
29                 > listPrenoms) {
30
31             this.id = id;
32             this.nom = nom;
33             this.sexe = sexe;
34             this.dateNaissance = dateNaissance;
35             this.specialite = specialite;
36             this.email = email;
37             this.cv = cv;
38             this.adresse = adresse;
39             this.listTelephones = listTelephones;
40             this.listPrenoms = listPrenoms;
41         }
42
43         public UUID getId() {
```

```

40         return id;
41     }
42
43     public void setId(UUID id) {
44         this.id = id;
45     }
46
47     public String getNom() {
48         return nom;
49     }
50
51     public void setNom(String nom) {
52         this.nom = nom;
53     }
54
55     public String getSexe() {
56         return sexe;
57     }
58
59     public void setSexe(String sexe) {
60         this.sexe = sexe;
61     }
62
63     public LocalDate getDateNaissance() {
64         return dateNaissance;
65     }
66
67     public void setDateNaissance(LocalDate dateNaissance)
68     {
69         this.dateNaissance = dateNaissance;
70     }
71
72     public String getSpecialite() {
73         return specialite;
74     }
75
76     public void setSpecialite(String specialite) {
77         this.specialite = specialite;
78     }
79
80     public String getEmail() {
81         return email;
82     }
83
84     public void setEmail(String email) {
85         this.email = email;
86     }
87
88     public String getCv() {
89         return cv;

```

```

89         }
90
91         public void setCv(String cv) {
92             this.cv = cv;
93         }
94
95         public Map<String, String> getAdresse() {
96             return adresse;
97         }
98
99         public void setAdresse(Map<String, String> adresse) {
100             this.adresse = adresse;
101         }
102
103         public Set<String> getListTelephones() {
104             return listTelephones;
105         }
106
107         public void setListTelephones(Set<String>
listTelephones) {
108             this.listTelephones = listTelephones;
109         }
110
111         public Set<String> getListPrenoms() {
112             return listPrenoms;
113         }
114
115         public void setListPrenoms(Set<String> listPrenoms) {
116             this.listPrenoms = listPrenoms;
117         }
118
119         public List<Consultation> getConsultations() {
120             return consultations;
121         }
122
123         public void setConsultations(List<Consultation>
consultations) {
124             this.consultations = consultations;
125         }
126
127         public List<RendezVous> getRendezVous() {
128             return rendezVous;
129         }
130
131         public void setRendezVous(List<RendezVous> rendezVous
) {
132             this.rendezVous = rendezVous;
133         }
134
135         @Override

```

```

136         public String toString() {
137             return "Medecin{" +
138                 "id=" + id +
139                 ", nom='" + nom + '\\', ' +
140                 ", sexe='" + sexe + '\\', ' +
141                 ", dateNaissance=" + dateNaissance +
142                 ", specialite='" + specialite + '\\', ' +
143                 ", email='" + email + '\\', ' +
144                 ", cv='" + cv + '\\', ' +
145                 ", adresse=" + adresse +
146                 ", listTelephones=" + listTelephones +
147                 ", listPrenoms=" + listPrenoms +
148                 '}'';
149         }
150     }
151
152     // Classe Patient
153     package org.example.entities;
154
155     import java.time.LocalDate;
156     import java.util.Map;
157     import java.util.Set;
158     import java.util.UUID;
159     import java.util.List;
160
161     public class Patient {
162         private UUID id;
163         private String numeroSecuriteSociale;
164         private String nom;
165         private String sexe;
166         private LocalDate dateNaissance;
167         private String email;
168         private double poids;
169         private double hauteur;
170
171         private Set<String> listTelephones;
172         private Set<String> listPrenoms;
173         private Map<String, String> adresse;
174         private List<String> allergies;
175         private List<Consultation> consultations;
176         private List<RendezVous> rendezVous;
177
178         public Patient(){}
179
180         public Patient(UUID id, String numeroSecuriteSociale,
181             String nom, String sexe, LocalDate dateNaissance,
182             String email, double poids, double
183             hauteur, Set<String> listTelephones
184             ,

```

```

182         Set<String> listPrenoms, Map<String,
                String> adresse, List<String>
                allergies) {
183         this.id = id;
184         this.numeroSecuriteSociale =
                numeroSecuriteSociale;
185         this.nom = nom;
186         this.sexe = sexe;
187         this.dateNaissance = dateNaissance;
188         this.email = email;
189         this.poids = poids;
190         this.hauteur = hauteur;
191         this.listTelephones = listTelephones;
192         this.listPrenoms = listPrenoms;
193         this.adresse = adresse;
194         this.allergies = allergies;
195     }
196
197     public UUID getId() { return id; }
198     public void setId(UUID id) { this.id = id; }
199
200     public String getNumeroSecuriteSociale() { return
                numeroSecuriteSociale; }
201     public void setNumeroSecuriteSociale(String
                numeroSecuriteSociale) { this.
                numeroSecuriteSociale = numeroSecuriteSociale; }
202
203     public String getNom() { return nom; }
204     public void setNom(String nom) { this.nom = nom; }
205
206     public String getSexe() { return sexe; }
207     public void setSexe(String sexe) { this.sexe = sexe;
                }
208
209     public LocalDate getDateNaissance() { return
                dateNaissance; }
210     public void setDateNaissance(LocalDate dateNaissance)
                { this.dateNaissance = dateNaissance; }
211
212     public String getEmail() { return email; }
213     public void setEmail(String email) { this.email =
                email; }
214
215     public double getPoids() { return poids; }
216     public void setPoids(double poids) { this.poids =
                poids; }
217
218     public double getHauteur() { return hauteur; }
219     public void setHauteur(double hauteur) { this.hauteur
                = hauteur; }

```



```

220
221     public Set<String> getListTelephones() { return
        listTelephones; }
222     public void setListTelephones(Set<String>
        listTelephones) { this.listTelephones =
        listTelephones; }
223
224     public Set<String> getListPrenoms() { return
        listPrenoms; }
225     public void setListPrenoms(Set<String> listPrenoms) {
        this.listPrenoms = listPrenoms; }
226
227     public Map<String, String> getAdresse() { return
        adresse; }
228     public void setAdresse(Map<String, String> adresse) {
        this.adresse = adresse; }
229
230     public List<String> getAllergies() { return allergies
        ; }
231     public void setAllergies(List<String> allergies) {
        this.allergies = allergies; }
232
233     public List<Consultation> getConsultations() {
234         return consultations;
235     }
236
237     public void setConsultations(List<Consultation>
        consultations) {
238         this.consultations = consultations;
239     }
240
241     public List<RendezVous> getRendezVous() {
242         return rendezVous;
243     }
244
245     public void setRendezVous(List<RendezVous> rendezVous
        ) {
246         this.rendezVous = rendezVous;
247     }
248
249     @Override
250     public String toString() {
251         return "Patient{" +
252             "id=" + id +
253             ", numeroSecuriteSociale=" +
                numeroSecuriteSociale + '\',' +
254             ", nom=" + nom + '\',' +
255             ", sexe=" + sexe + '\',' +
256             ", dateNaissance=" + dateNaissance +
257             ", email=" + email + '\',' +

```

```

258         ", poids=" + poids +
259         ", hauteur=" + hauteur +
260         ", listTelephones=" + listTelephones +
261         ", listPrenoms=" + listPrenoms +
262         ", adresse=" + adresse +
263         ", allergies=" + allergies +
264         '}',
265     }
266 }
267
268 // Classe Consultation
269 package org.example.entities;
270
271 import java.time.LocalDate;
272 import java.util.List;
273 import java.util.Map;
274 import java.util.UUID;
275
276 public class Consultation {
277     private LocalDate consultationDate;
278     private UUID patientId;
279     private UUID doctorId;
280     private String raison;
281     private String diagnostic;
282     private Map<String, String> facture;
283     private List<Map<String, String>> prescriptions;
284     private List<Map<String, String>> examens;
285
286     public Consultation(LocalDate consultationDate, UUID
        patientId, UUID doctorId, String raison, String
        diagnostic,
287                             Map<String, String> facture, List
                                <Map<String, String>>
                                prescriptions, List<Map<String
                                    , String>> examens) {
288         this.consultationDate = consultationDate;
289         this.patientId = patientId;
290         this.doctorId = doctorId;
291         this.raison = raison;
292         this.diagnostic = diagnostic;
293         this.facture = facture;
294         this.prescriptions = prescriptions;
295         this.examens = examens;
296     }
297
298     public LocalDate getConsultationDate() { return
        consultationDate; }
299     public void setConsultationDate(LocalDate
        consultationDate) { this.consultationDate =
        consultationDate; }

```

```

300
301     public UUID getPatientId() { return patientId; }
302     public void setPatientId(UUID patientId) { this.
        patientId = patientId; }
303
304     public UUID getDoctorId() { return doctorId; }
305     public void setDoctorId(UUID doctorId) { this.
        doctorId = doctorId; }
306
307     public String getRaison() { return raison; }
308     public void setRaison(String raison) { this.raison =
        raison; }
309
310     public String getDiagnostic() { return diagnostic; }
311     public void setDiagnostic(String diagnostic) { this.
        diagnostic = diagnostic; }
312
313     public Map<String, String> getFacture() { return
        facture; }
314     public void setFacture(Map<String, String> facture) {
        this.facture = facture; }
315
316     public List<Map<String, String>> getPrescriptions() {
        return prescriptions; }
317     public void setPrescriptions(List<Map<String, String
        >> prescriptions) { this.prescriptions =
        prescriptions; }
318
319     public List<Map<String, String>> getExamens() {
        return examens; }
320     public void setExamens(List<Map<String, String>>
        examens) { this.examens = examens; }
321
322     @Override
323     public String toString() {
324         return "Consultation{" +
325             "consultationDate=" + consultationDate +
326             ", patientId=" + patientId +
327             ", doctorId=" + doctorId +
328             ", raison='" + raison + '\'' +
329             ", diagnostic='" + diagnostic + '\'' +
330             ", facture=" + facture +
331             ", prescriptions=" + prescriptions +
332             ", examens=" + examens +
333             '}'
334     }
335 }
336
337 // Classe RendezVous
338 package org.example.entities;

```

```

339
340 import java.time.LocalDate;
341 import java.util.UUID;
342
343 public class RendezVous {
344     private LocalDate rendezvousDate;
345     private UUID patientId;
346     private UUID doctorId;
347     private String motif;
348
349     public RendezVous(LocalDate rendezvousDate, UUID
        patientId, UUID doctorId, String motif) {
350         this.rendezvousDate = rendezvousDate;
351         this.patientId = patientId;
352         this.doctorId = doctorId;
353         this.motif = motif;
354     }
355
356     public LocalDate getRendezvousDate() { return
        rendezvousDate; }
357     public void setRendezvousDate(LocalDate
        rendezvousDate) { this.rendezvousDate =
        rendezvousDate; }
358
359     public UUID getPatientId() { return patientId; }
360     public void setPatientId(UUID patientId) { this.
        patientId = patientId; }
361
362     public UUID getDoctorId() { return doctorId; }
363     public void setDoctorId(UUID doctorId) { this.
        doctorId = doctorId; }
364
365     public String getMotif() { return motif; }
366     public void setMotif(String motif) { this.motif =
        motif; }
367
368     @Override
369     public String toString() {
370         return "RendezVous{" +
371             "rendezvousDate=" + rendezvousDate +
372             ", patientId=" + patientId +
373             ", doctorId=" + doctorId +
374             ", motif='" + motif + '\'' +
375             '}';
376     }
377 }

```

2. Méthode pour insérer un médecin :

```

1 // Insérer un medecin
2 public void insertOneMedecin(Medecin medecin) {

```

```

3      String query = "INSERT INTO Medecin_By_Speciality (id
      , nom, sexe, date_naissance, specialite, email, cv
      , adresse, list_telephones, list_prenoms) " +
4          "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
5
6      PreparedStatement preparedStatement = session.prepare
      (query);
7
8      BoundStatement boundStatement = preparedStatement.
      bind(
9          medecin.getId(),
10         medecin.getNom(),
11         medecin.getSexe(),
12         medecin.getDateNaissance(),
13         medecin.getSpecialite(),
14         medecin.getEmail(),
15         medecin.getCv(),
16         medecin.getAdresse(),
17         medecin.getListTelephones(),
18         medecin.getListPrenoms()
19     );
20
21     session.execute(boundStatement);
22 }

```

3. Méthode pour lire un médecin

```

1      // Recuperer un medecin
2      public Medecin getOneMedecin(String specialite, LocalDate
      dateNaissance, String nom) {
3          String query = "SELECT id, nom, sexe, date_naissance,
      specialite, email, cv, adresse, list_telephones,
      list_prenoms " +
4              "FROM Medecin_By_Speciality WHERE specialite
      = ? AND date_naissance = ? AND nom = ?";
5
6          PreparedStatement preparedStatement = session.prepare
      (query);
7          BoundStatement boundStatement = preparedStatement.
      bind(specialite, dateNaissance, nom);
8          ResultSet resultSet = session.execute(boundStatement)
      ;
9
10         Row row = resultSet.one();
11
12         if (row != null) {
13             UUID id = row.getUuid("id");
14             String sexe = row.getString("sexe");
15             String email = row.getString("email");
16             String cv = row.getString("cv");

```

```

17         Map<String, String> adresse = row.getMap("adresse
18             ", String.class, String.class);
19         Set<String> listTelephones = row.getSet("
20             list_telephones", String.class);
21         Set<String> listPrenoms = row.getSet("
22             list_prenoms", String.class);
23
24         return new Medecin(id, nom, sexe, dateNaissance,
25             specialite, email, cv, adresse, listTelephones
26             , listPrenoms);
27     }
28     return null;
29 }

```

4. Méthode pour mettre à jour un médecin

```

1 // Modifier un medecin
2 public void updateOneMedecin(Medecin medecin) {
3     String query = "UPDATE Medecin_By_Speciality " +
4         "SET cv = ?, adresse = ?, list_telephones =
5         ?, list_prenoms = ? " +
6         "WHERE specialite = ? AND date_naissance = ?
7         AND nom = ? AND sexe = ? AND email = ? AND
8         id = ?";
9
10    PreparedStatement preparedStatement = session.prepare
11        (query);
12
13    BoundStatement boundStatement = preparedStatement.
14        bind(
15        medecin.getCv(),
16        medecin.getAdresse(),
17        medecin.getListTelephones(),
18        medecin.getListPrenoms(),
19        medecin.getSpecialite(),
20        medecin.getDateNaissance(),
21        medecin.getNom(),
22        medecin.getSexe(),
23        medecin.getEmail(),
24        medecin.getId()
25    );
26
27    session.execute(boundStatement);
28 }

```

5. Méthode pour supprimer un médecin

```

1 // Supprimer un medecin
2 public void deleteOneMedecin(String specialite, LocalDate
3     dateNaissance, String nom, String sexe, String email,
4     UUID id) {

```

```

3      String query = "DELETE FROM Medecin_By_Speciality " +
4                      "WHERE specialite = ? AND date_naissance = ?
                        AND nom = ? AND sexe = ? AND email = ? AND
                        id = ?";
5
6      PreparedStatement preparedStatement = session.prepare
          (query);
7
8      BoundStatement boundStatement = preparedStatement.
          bind(
9          specialite,
10         dateNaissance,
11         nom,
12         sexe,
13         email,
14         id
15     );
16     session.execute(boundStatement);
17 }

```

4.2.2 Indexation secondaire

1. Création d'index secondaire sur le champ "nom"

```

1  // Creation d'index
2  public void createIndexes(){
3      session.execute("CREATE INDEX IF NOT EXISTS ON
                        Medecin_By_Speciality (nom)");
4  }

```

4.2.3 Méthode de Consultation (Jointure, Groupement)

Dans Cassandra, les jointures ne sont pas supportées directement, donc il est recommandé d'organiser les données pour éviter les jointures en dénormalisant les données. On peut utiliser la requête de sélection par clé de partition ou le groupement via des agrégats externes.

```

1  // Groupement
2  public long countRecords(String specialite) {
3      String query = "SELECT COUNT(*) FROM Medecin_By_Speciality
                        WHERE specialite = '" + specialite + "'";
4      ResultSet resultSet = session.execute(query);
5      return resultSet.one().getLong(0);
6  }

```

5 Compléments sur le moteur NoSql CASSANDRA

5.1 Modèles de données supportés

5.1.1 Modèle de données : orienté famille de colonnes (Wide Column)

Cassandra utilise un modèle orienté colonne, où les données sont organisées en lignes et colonnes. C'est un modèle plus proche des bases de données relationnelles, mais avec une architecture distribuée et sans transactions complexes comme celles des bases SQL.

5.1.2 Usage

Idéal pour les applications qui nécessitent des écritures à haute vitesse et des lectures avec des clés spécifiques (par exemple, journaux, métadonnées).

5.2 Procédure d'installation du moteur et des utilitaires

L'installation de Cassandra peut être plus complexe, en particulier sur des clusters distribués. Elle nécessite l'installation de Java, puis Cassandra via les distributions officielles.

5.2.1 Installation Windows

Pour installer Apache Cassandra sur Windows, suivez ces étapes :

1. Télécharger Apache Cassandra :
 - (a) Allez sur la page de téléchargement officielle d'Apache Cassandra : <https://cassandra.apache.org/download/>.
 - (b) Téléchargez la dernière version de Cassandra sous forme de fichier `.tar.gz` ou `.zip`.
2. Installer Java :

Cassandra nécessite Java pour fonctionner. Installez le Java Development Kit (JDK) :

 - (a) Allez sur la page de téléchargement du <https://www.oracle.com/java/technologies/javase-jdk11-downloads.html> ou <https://jdk.java.net/>.
 - (b) Téléchargez et installez la dernière version.

Après l'installation, configurez la variable d'environnement `JAVA_HOME` :

- (a) Faites un clic droit sur **This PC** et sélectionnez **Propriétés**.
- (b) Cliquez sur **Paramètres système avancés**.
- (c) Cliquez sur **Variables d'environnement**.
- (d) Sous **Variables système**, cliquez sur **Nouveau** et ajoutez :
 - i. Nom de la variable : `JAVA_HOME`
 - ii. Valeur de la variable : `C:\Program Files\Java\jdk-<version>`
- (e) Ajoutez le dossier `bin` du JDK à la variable `PATH` :
 - Trouvez `Path` dans la liste des variables système, cliquez sur **Modifier** et ajoutez : `%JAVA_HOME%\bin`.

3. Extraire Apache Cassandra
Extrayez l'archive Cassandra téléchargée (.tar.gz ou .zip) dans un répertoire, par exemple `C:\cassandra`.
4. Configurer les variables d'environnement
Pour rendre les commandes Cassandra accessibles, vous devez configurer les variables d'environnement :
 - (a) Allez dans **Paramètres système avancés** → **Variables d'environnement**.
 - (b) Sous **Variables système**, trouvez **Path** et ajoutez : `C:\cassandra\bin`. Cela vous permettra d'utiliser les commandes Cassandra dans l'invite de commandes.
5. Configurer Cassandra
 - (a) Allez dans `C:\cassandra\conf`.
 - (b) Ouvrez le fichier `cassandra.yaml` avec un éditeur de texte.
 - (c) Modifiez le `cluster_name` en le renommant selon votre préférence.
 - (d) Configurez les répertoires : `data_file_directories`, `commitlog_directory` et `saved_caches_directory` pour définir les chemins de stockage des données.
 - (e) Assurez-vous que l'adresse `listen_address` est configurée sur l'IP locale de votre machine ou sur `localhost`.
6. Installer et configurer Python (pour cqlsh)
 - (a) Cassandra utilise **Python** pour exécuter `cqlsh`.
 - (b) Téléchargez et installez Python depuis <https://www.python.org/downloads/>.
 - (c) Assurez-vous que la commande `python` est accessible en ajoutant Python à votre `PATH`.
 - (d) Lors de l'installation, cochez l'option **Ajouter Python au PATH**.
7. Démarrer Cassandra
 - (a) Ouvrez une **Invite de commandes** et allez dans le dossier Cassandra : `cd C:\cassandra\bin`
 - (b) Démarrez Cassandra en exécutant : `cassandra -f`
 - (c) Cassandra démarrera en mode premier plan, et vous verrez les messages du journal.
8. Accéder à Cassandra Shell (cqlsh)
 - (a) Une fois Cassandra démarré, ouvrez une autre fenêtre d'Invite de commandes.
 - (b) Allez à nouveau dans le dossier `bin` : `cd C:\cassandra\bin`
 - (c) Démarrez le shell de langage de requête Cassandra (CQLSH) en exécutant : `cqlsh`
 - (d) Cela vous connectera à l'instance Cassandra en cours d'exécution, et vous pourrez commencer à exécuter des commandes CQL.
9. Vérifier l'installation
 - (a) Pour vérifier l'installation, exécutez la commande suivante dans `cqlsh` pour vérifier la version de Cassandra : `SELECT release_version FROM system.local;`
 - (b) Si tout fonctionne correctement, vous devriez voir la version de Cassandra affichée.

10. Optionnel : Exécuter Cassandra en tant que service Windows. Vous pouvez utiliser des outils comme NSSM (Non-Sucking Service Manager) pour exécuter Cassandra en tant que service Windows, ce qui permet à Cassandra de démarrer automatiquement au démarrage de Windows.

En suivant ces étapes, Cassandra devrait être installé et fonctionner sur votre machine Windows.

5.2.2 Installation (Linux)

```
sudo apt install cassandra & sudo systemctl start cassandra
```

5.2.3 Installation via Docker

```
docker run -name some-cassandra -d -e CASSANDRA_BROADCAST_ADDRESS=10.42.42.42
-p 7000:7000 cassandra:latest
```

5.2.4 Utilitaires

Cassandra utilise `cqlsh`, un utilitaire en ligne de commande pour interagir avec la base de données à l'aide de CQL (Cassandra Query Language). L'administration de Cassandra peut également se faire via `nodetool`.

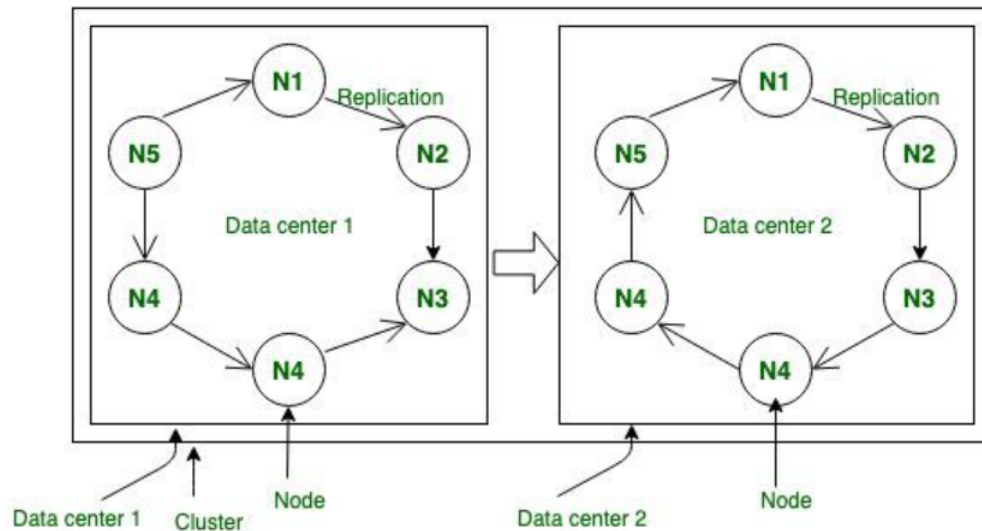
5.3 Architecture du moteur NoSQL (schéma)

5.3.1 Architecture

Cassandra a une architecture distribuée peer-to-peer, où tous les nœuds sont égaux. Il n'y a pas de maître, chaque nœud peut accepter des lectures et des écritures. Cette architecture est conçue pour la haute disponibilité et l'évolutivité.

5.3.2 Schéma

- Les nœuds sont connectés dans un anneau.
- Pas de nœud central (évitant le single point of failure).
- Lecture/écriture à haute disponibilité.



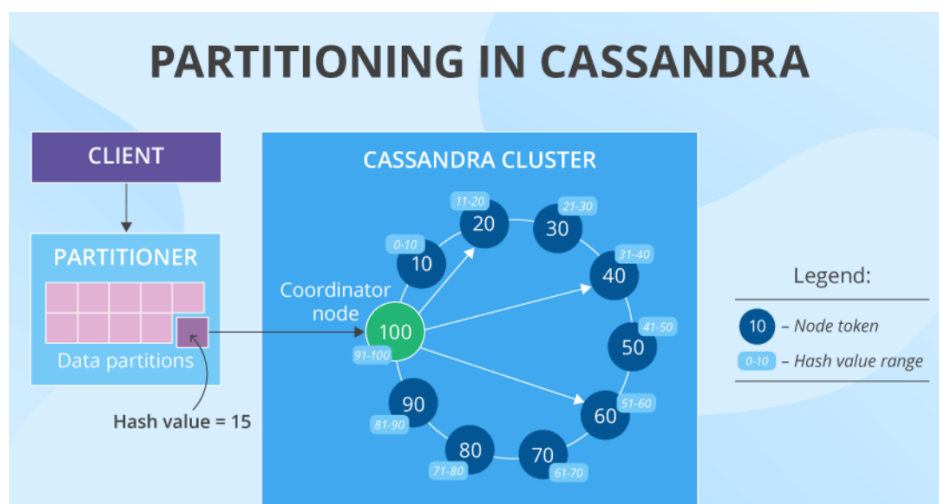
5.4 Méthode de partitionnement

5.4.1 Sharding

Cassandra utilise le **partitionnement par clé** pour répartir les données entre les nœuds du cluster. Une clé de partition est choisie, et les données sont distribuées uniformément en utilisant un **hash** de la clé.

5.4.2 Schéma

- Chaque nœud stocke une plage de valeurs de clé.
- Le partitionneur détermine quel nœud recevra quelles données.



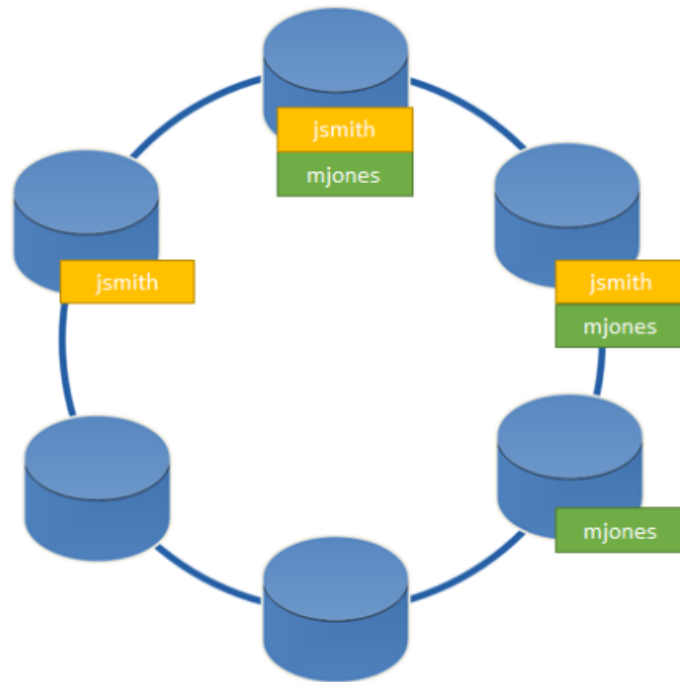
5.5 Méthode de réplication

5.5.1 Réplication

Cassandra utilise un modèle de réplication où chaque nœud réplique les données vers **N** autres nœuds en fonction du facteur de réplication.

5.5.2 Schéma

- Réplication en anneau, avec chaque nœud répliquant les données sur les nœuds voisins.



5.6 Montée en charge

Cassandra est conçu pour une montée en charge horizontale sans goulot d'étranglement central. Vous pouvez ajouter des nœuds au cluster, et Cassandra redistribue automatiquement les données.

5.6.1 Schéma

Nœuds ajoutés au cluster sans point de contrôle central.

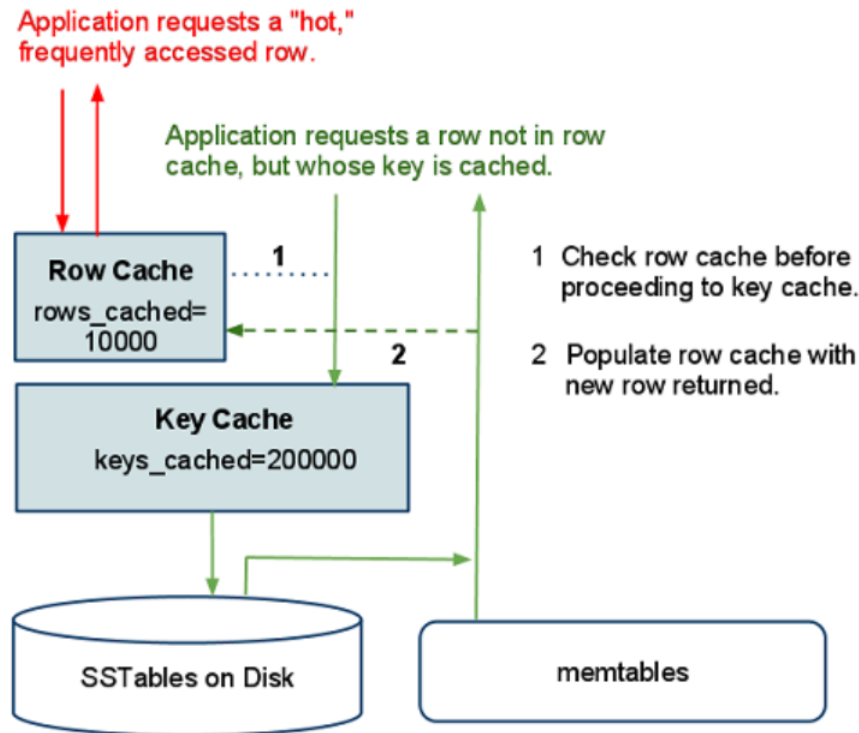


5.7 Gestion du cache mémoire

Cassandra utilise plusieurs caches, tels que le **key cache** (pour accélérer l'accès aux clés) et le **row cache** (pour stocker des lignes complètes en mémoire).

5.7.1 Schéma

Cache pour les clés et les lignes pour accélérer les lectures fréquentes.



6 Génération automatique des données

Dans cette section nous allons générer des données pour nos tables cassandra.

6.1 Script de génération des données

Pour générer les données pour les tables (1000 enregistrements par table) nous avons utilisé le script Ruby suivant :

```

1  require 'json'
2  require 'securerandom'
3  require 'date'
4
5  # Helper functions
6  def random_date(start_date, end_date)
7    rand(start_date..end_date).to_s
8  end
9
10 def random_text(length)
11   ('a'..'z').to_a.sample(length).join
12 end
13
14 def random_sex
15   ['Male', 'Female'].sample

```

```

16     end
17
18     def random_phone
19         "+#{rand(1000000000..9999999999)}"
20     end
21
22     def random_address
23         { "numero" => rand(1..100).to_s, "street" => random_text(10),
24           "city" => random_text(6), "postal_code" => rand
25             (10000..99999).to_s }
26     end
27
28     def random_specialty
29         ['Cardiology', 'Neurology', 'Pediatrics', 'Orthopedics', '
30           Dermatology'].sample
31     end
32
33     def random_motif
34         ['Routine Checkup', 'Emergency Visit', 'Follow-up', '
35           Consultation'].sample
36     end
37
38     # Generating 1000 records for Medecin_By_Speciality
39     medecin_data = 1000.times.map do
40         {
41             "id" => SecureRandom.uuid,
42             "nom" => random_text(8),
43             "sexe" => random_sex,
44             "date_naissance" => random_date(Date.new(1960, 1, 1), Date.
45               new(1995, 12, 31)),
46             "specialite" => random_specialty,
47             "email" => "#{random_text(5)}@domain.com",
48             "cv" => random_text(20),
49             "adresse" => random_address,
50             "list_telephones" => [random_phone, random_phone],
51             "list_prenoms" => [random_text(6), random_text(5)]
52         }
53     end
54
55     # Generating 1000 records for Patient_By_Birthday
56     patient_data = 1000.times.map do
57         {
58             "id" => SecureRandom.uuid,
59             "numero_securite_sociale" => rand(1000000000..9999999999).
60               to_s,
61             "nom" => random_text(8),
62             "sexe" => random_sex,
63             "date_naissance" => random_date(Date.new(1960, 1, 1), Date.
64               new(2005, 12, 31)),
65             "email" => "#{random_text(5)}@domain.com",

```

```

59     "poids" => rand(50.0..100.0).round(2),
60     "hauteur" => rand(150.0..200.0).round(2),
61     "list_telephones" => [random_phone, random_phone],
62     "list_prenoms" => [random_text(6), random_text(5)],
63     "adresse" => random_address,
64     "allergies" => [random_text(5), random_text(6)]
65   }
66 end
67
68 # Generating 1000 records for RendezVous_By_Date
69 rendezvous_data = 1000.times.map do
70   {
71     "rendezvous_date" => random_date(Date.new(2020, 1, 1), Date
72       .new(2024, 12, 31)),
73     "patient_id" => SecureRandom.uuid,
74     "doctor_id" => SecureRandom.uuid,
75     "motif" => random_motif
76   }
77 end
78
79 # Generating 1000 records for Consultation_By_Date
80 consultation_data = 1000.times.map do
81   {
82     "consultation_date" => random_date(Date.new(2020, 1, 1),
83       Date.new(2024, 12, 31)),
84     "patient_id" => SecureRandom.uuid,
85     "doctor_id" => SecureRandom.uuid,
86     "raison" => random_motif,
87     "diagnostic" => random_text(20),
88     "facture" => { "montant_total" => "#{rand(100..1000)} USD",
89       "date_facture": random_date(Date.new(2020, 1, 1), Date.
90         new(2024, 12, 31))},
91     "prescriptions" => 3.times.map { { "details_prescription"
92       => random_text(20), "date_prescription" => random_date(
93         Date.new(2020, 1, 1), Date.new(2024, 12, 31)) } },
94     "examens" => 2.times.map { { "details_examen" =>
95       random_text(20), "date_examen" => random_date(Date.new
96         (2020, 1, 1), Date.new(2024, 12, 31)) } }
97   }
98 end
99
100 # Writing to JSON files
101 File.write('Medecin_By_Speciality.json', JSON.pretty_generate(
102   medecin_data))
103 File.write('Patient_By_Birthday.json', JSON.pretty_generate(
104   patient_data))
105 File.write('RendezVous_By_Date.json', JSON.pretty_generate(
106   rendezvous_data))
107 File.write('Consultation_By_Date.json', JSON.pretty_generate(
108   consultation_data))

```

```

97
98 puts "Files generated successfully!"

```

Nous obtenons des fichiers au format json que nous convertissons au format csv avec le script Ruby ci-dessous :

```

1  require 'json'
2  require 'csv'
3
4  # Function to handle flattening nested structures like arrays
   and hashes
5  def flatten_value(value)
6    case value
7    when Array
8      value.join('; ') # Join array elements with ';' for CSV
9    when Hash
10     value.map { |k, v| "#{k}: #{v}" }.join('; ') # Flatten hash
        into 'key: value' format
11    else
12      value
13    end
14  end
15
16 # Function to convert JSON to CSV
17 def json_to_csv(json_file, csv_file)
18   # Load the JSON data from the file
19   json_data = JSON.parse(File.read(json_file))
20
21   # Open a CSV file for writing
22   CSV.open(csv_file, 'w') do |csv|
23     # Extract column names (keys from the first JSON object)
24     headers = json_data.first.keys
25     csv << headers
26
27     # Write each JSON object as a CSV row
28     json_data.each do |hash|
29       csv << hash.values.map { |value| flatten_value(value) }
30     end
31   end
32   puts "CSV file #{csv_file} has been generated successfully!"
33 end
34
35 # Converting JSON files for all 4 tables
36
37 # 1. Medecin_By_Speciality
38 json_to_csv('Medecin_By_Speciality.json', '
    Medecin_By_Speciality.csv')
39
40 # 2. Patient_By_BirthDay
41 json_to_csv('Patient_By_BirthDay.json', 'Patient_By_BirthDay.

```



```
        csv')
42
43 # 3. RendezVous_By_Date
44 json_to_csv('RendezVous_By_Date.json', 'RendezVous_By_Date.csv'
45             )
46 # 4. Consultation_By_Date
47 json_to_csv('Consultation_By_Date.json', 'Consultation_By_Date.
            csv')
```

Nous obtenons 4 fichiers **CSV** contenant des données qu'il convient d'importer pour chacune des tables modélisée précédemment :

- Medecin_By_Speciality.csv
- Patient_By_BirthDay.csv
- RendezVous_By_Date.csv
- Consultation_By_Date.csv