



UNIVERSITÉ COTE D'AZUR/FDS(UEH)

MongoDB : 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 « Necessiter » 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 MONGODB et classes java	9
4.1	Spécification des Modèles de Documents	9
4.1.1	Collection "Medecin" (document JSON/BSON)	9
4.1.2	Collection "Patient" (document JSON)	11
4.2	Spécification des classes et des méthodes JAVA	15
4.2.1	Méthodes CRUD	15
4.2.2	Indexation secondaire	28
4.2.3	Méthode de Consultation (Jointure, Groupement)	28
5	Compléments sur le moteur NoSql MONGODB	29
5.1	Modèles de données supportés	29
5.1.1	Modèle de données : Document (JSON/BSON)	29
5.1.2	Usage	30
5.2	Procédure d'installation du moteur et des utilitaires	30
5.2.1	Installation	30
5.2.2	Installation (Windows)	30
5.2.3	Installation (Linux)	31
5.2.4	Installation via Docker	31
5.2.5	Utilitaires	31
5.3	Architecture du moteur NoSQL (schéma)	32
5.3.1	Architecture	32
5.3.2	Schéma	32
5.4	Méthode de partitionnement	33
5.4.1	Sharding	33
5.4.2	Schéma	33

5.5	Méthode de réplication	33
5.5.1	Réplication	33
5.5.2	Schéma	33
5.6	Montée en charge	34
5.6.1	Montée en charge verticale	34
5.6.2	Montée en charge horizontale	35
5.7	Gestion du cache mémoire	35
5.7.1	Schéma	35
6	Génération automatique des données	35
6.1	Script de génération de 1000 médecins	36
6.2	Script de génération de 1000 patients	37
6.3	Génération des données dans des fichiers JSON	39
6.3.1	Données des médecins	39
6.3.2	Données des patients	40

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.

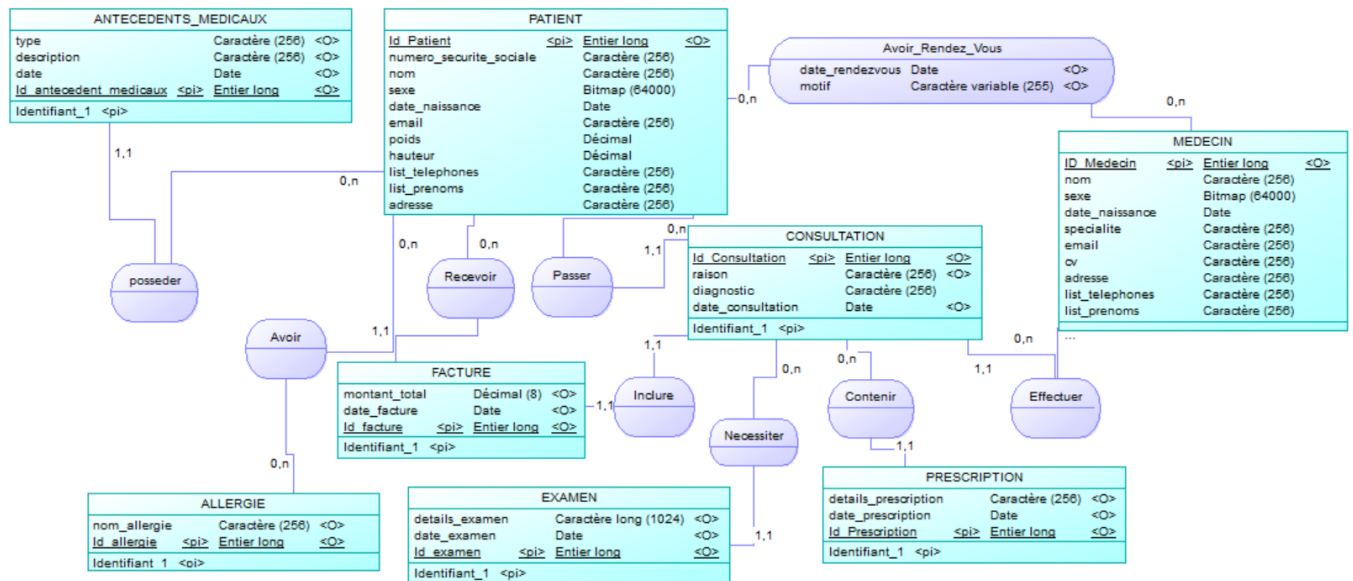
3.8 Association « Posséder » entre ANTECEDENTS _ MEDICAUX et PATIENT

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 MONGODB et classes java

4.1 Spécification des Modèles de Documents

MongoDB est un système de base de données NoSQL orienté document, qui stocke les données sous forme de documents JSON/BSON. Les objets sont donc organisés en collections et chaque document peut être indépendant en termes de structure.

Dans notre cas de figure nous avons identifié deux collections (Medecin et Patient) suite à la conversion du MCD en objets MONGODB en respectant les préconisations et règles d'organisation des bases de données NOSQL et MONGODB en particulier, notamment en dénormalisant les données autant que possible.

4.1.1 Collection "Medecin" (document JSON/BSON)

```

1  {
2    "_id": "ObjectId",
3    "nom": "String",
4    "sexe": "String",
5    "date_naissance": "Date",
6    "specialite": "String",
7    "email": "String",

```

```

8      "cv": "String",
9      "list_telephones": ["String"],
10     "list_prenoms": ["String"],
11     "adresse": {
12         "numero": "Number",
13         "rue": "String",
14         "code_postal": "Number",
15         "ville": "String"
16     },
17     "list_rendez_vous": [
18         {
19             "patient": "ObjectId",
20             "date_rendez_vous": "Date",
21             "motif": "String"
22         }
23     ],
24     "list_consultations": [
25         {
26             "patient": "ObjectId",
27             "raison": "String",
28             "diagnostic": "String",
29             "date_consultation": "Date",
30             "list_examens": [
31                 {
32                     "details_examen": "String",
33                     "date_examen": "Date"
34                 }
35             ],
36             "list_prescriptions": [
37                 {
38                     "details_prescription": "String",
39                     "date_prescription": "Date"
40                 }
41             ]
42         }
43     ]
44 }

```

Exemple de document de médecin

```

1  {
2      "_id": "66f0820f580fe26567ae4f8d",
3      "nom": "Kertzmann",
4      "sexe": "Other",
5      "date_naissance": "1997-08-08",
6      "specialite": "Gynecologist",
7      "email": "susy_mohr@metz.test",
8      "cv": "Qui doloremque debitis et autem placeat natus qui quia
          voluptatibus.",
9      "list_telephones": [
10         "409.780.8066",

```

```

11     "740-880-4984"
12 ],
13 "list_prenoms": [
14     "Beau",
15     "Jessica"
16 ],
17 "adresse": {
18     "numero": "7777",
19     "rue": "Francesco Lights",
20     "code_postal": "32176-8834",
21     "ville": "Isabellashire"
22 },
23 "list_rendez_vous": [
24     {
25         "patient": "66f0820f580fe26567ae4f8e",
26         "date_rendez_vous": "2024-10-04",
27         "motif": "Quaerat doloremque nobis."
28     }
29 ],
30 "list_consultations": [
31     {
32         "patient": "66f0820f580fe26567ae4f8f",
33         "raison": "Magnam dolorum suscipit.",
34         "diagnostic": "Et nihil pariatur minima quam.",
35         "date_consultation": "2024-08-24",
36         "list_examens": [
37             {
38                 "details_examen": "Rerum in et similique voluptates.",
39                 "date_examen": "2024-09-07"
40             }
41         ],
42         "list_prescriptions": [
43             {
44                 "details_prescription": "Et et non facere non.",
45                 "date_prescription": "2024-09-14"
46             }
47         ]
48     }
49 ]
50 }
```

NB : Le modèle n'est pas un schéma strict. Les documents ne doivent pas nécessairement respecter en tout point le modèle de document. Le document de médecin ci-dessus est présenté à titre indicatif.

4.1.2 Collection "Patient" (document JSON)

```

1 {
2     "_id": "ObjectId",
3     "numero_securite_sociale": "String",
```

```

4  "nom": "String",
5  "sexe": "String",
6  "date_naissance": "Date",
7  "email": "String",
8  "poids": "Number",
9  "hauteur": "Number",
10 "list_telephones": ["String"],
11 "list_prenoms": ["String"],
12 "adresse": {
13     "numero": "Number",
14     "rue": "String",
15     "code_postal": "Number",
16     "ville": "String"
17 },
18 "list_rendez_vous": [
19     {
20         "medecin": "ObjectId",
21         "date_rendez_vous": "Date",
22         "motif": "String"
23     }
24 ],
25 "list_consultations": [
26     {
27         "medecin": "ObjectId",
28         "raison": "String",
29         "diagnostic": "String",
30         "date_consultation": "Date",
31         "list_examens": [
32             {
33                 "details_examen": "String",
34                 "date_examen": "Date"
35             }
36         ],
37         "list_prescriptions": [
38             {
39                 "details_prescription": "String",
40                 "date_prescription": "Date"
41             }
42         ],
43         "facture": {
44             "Montant_Total": "Number",
45             "date_facture": "Date"
46         }
47     }
48 ],
49 "antecedents_medicaux": [
50     {
51         "type": "String",
52         "description": "String",
53         "date": "Date"

```

```

54     }
55   ],
56   "allergies": ["String"]
57 }

```

Exemple de document de patient

```

1  {
2    "_id": "66f08628c97d6838ff4b3b48",
3    "numero_securite_sociale": "847-37-8324",
4    "nom": "Abernathy",
5    "sexe": "0",
6    "date_naissance": "2009-08-12",
7    "email": "matthew_hand@donnelly.test",
8    "poids": 77.64,
9    "hauteur": 1.66,
10   "list_telephones": [
11     "357-824-3744",
12     "(365) 603-4456"
13   ],
14   "list_prenoms": [
15     "Tom",
16     "Jamar"
17   ],
18   "adresse": {
19     "numero": "2945",
20     "rue": "Walter Mills",
21     "code_postal": "28820",
22     "ville": "Friesenport"
23   },
24   "list_rendez_vous": [
25     {
26       "medecin": "66f08628c97d6838ff4b3b49",
27       "date_rendez_vous": "2024-10-02",
28       "motif": "Qui culpa delectus."
29     }
30   ],
31   "list_consultations": [
32     {
33       "medecin": "66f08628c97d6838ff4b3b4a",
34       "raison": "Et qui labore.",
35       "diagnostic": "Porro totam ut ut possimus.",
36       "date_consultation": "2024-09-21",
37       "list_examens": [
38         {
39           "details_examen": "Laboriosam quo vitae quam quasi.",
40           "date_examen": "2024-09-11"
41         }
42       ],
43       "list_prescriptions": [
44         {

```

```

45         "details_prescription": "Tenetur sunt quae ex iusto.",
46         "date_prescription": "2024-09-21"
47     }
48 ],
49     "facture": {
50         "Montant_Total": 329.15,
51         "date_facture": "2024-09-16"
52     }
53 },
54 {
55     "medecin": "66f08628c97d6838ff4b3b4b",
56     "raison": "Exercitationem animi ex.",
57     "diagnostic": "Perspiciatis sequi placeat dolor amet.",
58     "date_consultation": "2024-09-20",
59     "list_examens": [
60         {
61             "details_examen": "Est eius excepturi fuga tenetur.",
62             "date_examen": "2024-09-11"
63         }
64     ],
65     "list_prescriptions": [
66         {
67             "details_prescription": "Quis aut dolores quia rerum.",
68             "date_prescription": "2024-09-15"
69         }
70     ],
71     "facture": {
72         "Montant_Total": 480.4,
73         "date_facture": "2024-09-19"
74     }
75 }
76 ],
77 "antecedents_medicaux": [
78     {
79         "type": "Chronic",
80         "description": "Ducimus repellat eos labore quis.",
81         "date": "2023-08-05"
82     },
83     {
84         "type": "Acute",
85         "description": "Eum dolorem labore et occaecati.",
86         "date": "2024-06-21"
87     }
88 ],
89 "allergies": [
90     "eligendi",
91     "ab"
92 ]
93 }

```

NB : Le modèle n'est pas un schéma strict. Les documents ne doivent pas nécessairement respecter en tout point le modèle de document. Le document de patient ci-dessus est présenté à titre indicatif.

4.2 Spécification des classes et des méthodes JAVA

4.2.1 Méthodes CRUD

1. Définition des Classes :

```

1 // Classe Medecin
2 package org.example.entity;
3
4 import org.bson.types.ObjectId;
5
6 import java.util.Date;
7 import java.util.List;
8
9 public class Medecin {
10     private ObjectId _id;
11     private String nom;
12     private String sexe;
13     private Date dateNaissance;
14     private String specialite;
15     private String email;
16     private String cv;
17     private List<String> listTelephones;
18     private List<String> listPrenoms;
19     private Adresse adresse;
20     private List<RendezVous> listRendezVous;
21     private List<Consultation> listConsultations;
22
23     public ObjectId getId() {
24         return _id;
25     }
26
27     public void setId(ObjectId id) {
28         this._id = id;
29     }
30
31     public String getNom() {
32         return nom;
33     }
34
35     public void setNom(String nom) {
36         this.nom = nom;
37     }
38
39     public String getSexe() {
40         return sexe;
41     }

```



```

42
43     public void setSexe(String sexe) {
44         this.sexe = sexe;
45     }
46
47     public Date getDateNaissance() {
48         return dateNaissance;
49     }
50
51     public void setDateNaissance(Date dateNaissance) {
52         this.dateNaissance = dateNaissance;
53     }
54
55     public String getSpecialite() {
56         return specialite;
57     }
58
59     public void setSpecialite(String specialite) {
60         this.specialite = specialite;
61     }
62
63     public String getEmail() {
64         return email;
65     }
66
67     public void setEmail(String email) {
68         this.email = email;
69     }
70
71     public String getCv(){
72         return this.cv;
73     }
74
75     public void setCv(String cv){
76         this.cv = cv;
77     }
78
79     public List<String> getListTelephones() {
80         return listTelephones;
81     }
82
83     public void setListTelephones(List<String> listTelephones
84         ) {
85         this.listTelephones = listTelephones;
86     }
87
88     public List<String> getListPrenoms() {
89         return listPrenoms;
90     }

```

```

91     public void setListPrenoms(List<String> listPrenoms) {
92         this.listPrenoms = listPrenoms;
93     }
94
95     public Adresse getAdresse() {
96         return adresse;
97     }
98
99     public void setAdresse(Adresse adresse) {
100         this.adresse = adresse;
101     }
102
103     public List<RendezVous> getListRendezVous() {
104         return listRendezVous;
105     }
106
107     public void setListRendezVous(List<RendezVous>
108         listRendezVous) {
109         this.listRendezVous = listRendezVous;
110     }
111
112     public List<Consultation> getListConsultations() {
113         return listConsultations;
114     }
115
116     public void setListConsultations(List<Consultation>
117         listConsultations) {
118         this.listConsultations = listConsultations;
119     }
120
121     public String toString(){
122         return "Medecin => nom : " + this.nom + ", sexe : " +
123             this.sexe + ", specialite : " + this.specialite +
124             ", email : " + this.email;
125     }
126 }
127
128 // Classe Patient
129 package org.example.entity;
130
131 import org.bson.types.ObjectId;
132
133 import java.util.Date;
134 import java.util.List;
135
136 public class Patient {
137     private ObjectId _id;
138     private String numeroSecuriteSociale;
139     private String nom;
140     private String sexe;

```

```

137     private Date dateNaissance;
138     private String email;
139     private double poids;
140     private double hauteur;
141     private List<String> listTelephones;
142     private List<String> listPrenoms;
143     private Adresse adresse;
144     private List<RendezVous> listRendezVous;
145     private List<Consultation> listConsultations;
146     private List<AntecedentMedical> antecedentsMedicaux;
147     private List<String> allergies;
148
149     public ObjectId getId() {
150         return _id;
151     }
152
153     public void setId(ObjectId id) {
154         this._id = id;
155     }
156
157     public String getNumeroSecuriteSociale() {
158         return numeroSecuriteSociale;
159     }
160
161     public void setNumeroSecuriteSociale(String
        numeroSecuriteSociale) {
162         this.numeroSecuriteSociale = numeroSecuriteSociale;
163     }
164
165     public String getNom() {
166         return nom;
167     }
168
169     public void setNom(String nom) {
170         this.nom = nom;
171     }
172
173     public String getSexe() {
174         return sexe;
175     }
176
177     public void setSexe(String sexe) {
178         this.sexe = sexe;
179     }
180
181     public Date getDateNaissance() {
182         return dateNaissance;
183     }
184
185     public void setDateNaissance(Date dateNaissance) {

```

```

186         this.dateNaissance = dateNaissance;
187     }
188
189     public String getEmail() {
190         return email;
191     }
192
193     public void setEmail(String email) {
194         this.email = email;
195     }
196
197     public double getPoids() {
198         return poids;
199     }
200
201     public void setPoids(double poids) {
202         this.poids = poids;
203     }
204
205     public double getHauteur() {
206         return hauteur;
207     }
208
209     public void setHauteur(double hauteur) {
210         this.hauteur = hauteur;
211     }
212
213     public List<String> getListTelephones() {
214         return listTelephones;
215     }
216
217     public void setListTelephones(List<String> listTelephones
218         ) {
219         this.listTelephones = listTelephones;
220     }
221
222     public List<String> getListPrenoms() {
223         return listPrenoms;
224     }
225
226     public void setListPrenoms(List<String> listPrenoms) {
227         this.listPrenoms = listPrenoms;
228     }
229
230     public Adresse getAdresse() {
231         return adresse;
232     }
233
234     public void setAdresse(Adresse adresse) {
235         this.adresse = adresse;

```

```

235     }
236
237     public List<RendezVous> getListRendezVous() {
238         return listRendezVous;
239     }
240
241     public void setListRendezVous(List<RendezVous>
242         listRendezVous) {
243         this.listRendezVous = listRendezVous;
244     }
245
246     public List<Consultation> getListConsultations() {
247         return listConsultations;
248     }
249
250     public void setListConsultations(List<Consultation>
251         listConsultations) {
252         this.listConsultations = listConsultations;
253     }
254
255     public List<AntecedentMedical> getAntecedentsMedicaux() {
256         return antecedentsMedicaux;
257     }
258
259     public void setAntecedentsMedicaux(List<AntecedentMedical
260         > antecedentsMedicaux) {
261         this.antecedentsMedicaux = antecedentsMedicaux;
262     }
263
264     public List<String> getAllergies() {
265         return allergies;
266     }
267
268     public void setAllergies(List<String> allergies) {
269         this.allergies = allergies;
270     }
271
272     public String toString() {
273         return "nom : " + this.nom + " sexe : " + this.sexe +
274             " email : " + this.email;
275     }
276 }
277
278 // Classe Examen
279 package org.example.entity;
280
281 import java.util.Date;
282
283 public class Examen {
284     private String detailsExamen;

```

```

281     private Date dateExamen;
282
283     public String getDetailsExamen() {
284         return detailsExamen;
285     }
286
287     public void setDetailsExamen(String detailsExamen) {
288         this.detailsExamen = detailsExamen;
289     }
290
291     public Date getDateExamen() {
292         return dateExamen;
293     }
294
295     public void setDateExamen(Date dateExamen) {
296         this.dateExamen = dateExamen;
297     }
298 }
299
300 // Classe Prescription
301 package org.example.entity;
302
303 import java.util.Date;
304
305 public class Prescription {
306     private String detailsPrescription;
307     private Date datePrescription;
308
309     public String getDetailsPrescription() {
310         return detailsPrescription;
311     }
312
313     public void setDetailsPrescription(String
        detailsPrescription) {
314         this.detailsPrescription = detailsPrescription;
315     }
316
317     public Date getDatePrescription() {
318         return datePrescription;
319     }
320
321     public void setDatePrescription(Date datePrescription) {
322         this.datePrescription = datePrescription;
323     }
324 }
325
326 // Classe Facture
327 package org.example.entity;
328
329 import java.util.Date;

```

```

330
331 public class Facture {
332     private double montantTotal;
333     private Date dateFacture;
334
335     public double getMontantTotal() {
336         return montantTotal;
337     }
338
339     public void setMontantTotal(double montantTotal) {
340         this.montantTotal = montantTotal;
341     }
342
343     public Date getDateFacture() {
344         return dateFacture;
345     }
346
347     public void setDateFacture(Date dateFacture) {
348         this.dateFacture = dateFacture;
349     }
350 }
351
352 // Classe AntecedentMedical
353 package org.example.entity;
354
355 import java.util.Date;
356
357 public class AntecedentMedical {
358     private String type;
359     private String description;
360     private Date date;
361
362     public String getType() {
363         return type;
364     }
365
366     public void setType(String type) {
367         this.type = type;
368     }
369
370     public String getDescription() {
371         return description;
372     }
373
374     public void setDescription(String description) {
375         this.description = description;
376     }
377
378     public Date getDate() {
379         return date;

```

```

380     }
381
382     public void setDate(Date date) {
383         this.date = date;
384     }
385 }
386
387 // Classe Adresse
388 package org.example.entity;
389
390 public class Adresse {
391     private int numero;
392     private String rue;
393     private int codePostal;
394     private String ville;
395
396     public int getNumero() {
397         return numero;
398     }
399
400     public void setNumero(int numero) {
401         this.numero = numero;
402     }
403
404     public String getRue() {
405         return rue;
406     }
407
408     public void setRue(String rue) {
409         this.rue = rue;
410     }
411
412     public int getCodePostal() {
413         return codePostal;
414     }
415
416     public void setCodePostal(int codePostal) {
417         this.codePostal = codePostal;
418     }
419
420     public String getVille() {
421         return ville;
422     }
423
424     public void setVille(String ville) {
425         this.ville = ville;
426     }
427 }
428
429 // Classe RendezVous

```



```

430 package org.example.entity;
431
432 import java.util.Date;
433
434 public class RendezVous {
435     private Medecin medecin;
436     private Patient patient;
437     private Date dateRendezVous;
438     private String motif;
439
440     public Medecin getMedecin() {
441         return medecin;
442     }
443
444     public void setMedecin(Medecin medecin) {
445         this.medecin = medecin;
446     }
447
448     public Patient getPatient() {
449         return patient;
450     }
451
452     public void setPatient(Patient patient) {
453         this.patient = patient;
454     }
455
456     public Date getDateRendezVous() {
457         return dateRendezVous;
458     }
459
460     public void setDateRendezVous(Date dateRendezVous) {
461         this.dateRendezVous = dateRendezVous;
462     }
463
464     public String getMotif() {
465         return motif;
466     }
467
468     public void setMotif(String motif) {
469         this.motif = motif;
470     }
471 }
472
473 // Classe Consultation
474 package org.example.entity;
475
476 import java.util.Date;
477 import java.util.List;
478
479 public class Consultation {

```

```

480     private Medecin medecin;
481     private Patient patient;
482     private String raison;
483     private String diagnostic;
484     private Date dateConsultation;
485     private List<Examen> listExamens;
486     private List<Prescription> listPrescriptions;
487     private Facture facture;
488
489     public Medecin getMedecin() {
490         return medecin;
491     }
492
493     public void setMedecin(Medecin medecin) {
494         this.medecin = medecin;
495     }
496
497     public Patient getPatient() {
498         return patient;
499     }
500
501     public void setPatient(Patient patient) {
502         this.patient = patient;
503     }
504
505     public String getRaison() {
506         return raison;
507     }
508
509     public void setRaison(String raison) {
510         this.raison = raison;
511     }
512
513     public String getDiagnostic() {
514         return diagnostic;
515     }
516
517     public void setDiagnostic(String diagnostic) {
518         this.diagnostic = diagnostic;
519     }
520
521     public Date getDateConsultation() {
522         return dateConsultation;
523     }
524
525     public void setDateConsultation(Date dateConsultation) {
526         this.dateConsultation = dateConsultation;
527     }
528
529     public List<Examen> getListExamens() {

```

```

530         return listExamens;
531     }
532
533     public void setListExamens(List<Examen> listExamens) {
534         this.listExamens = listExamens;
535     }
536
537     public List<Prescription> getListPrescriptions() {
538         return listPrescriptions;
539     }
540
541     public void setListPrescriptions(List<Prescription>
542         listPrescriptions) {
543         this.listPrescriptions = listPrescriptions;
544     }
545
546     public Facture getFacture() {
547         return facture;
548     }
549
550     public void setFacture(Facture facture) {
551         this.facture = facture;
552     }
553 }

```

2. Méthode pour insérer un médecin :

```

1     public InsertOneResult insertOneMedecin(Medecin medecin)
2     {
3         Document doc = new Document("nom", medecin.getNom())
4             .append("sexe", medecin.getSexe())
5             .append("date_naissance", medecin.
6                 getDateNaissance())
7             .append("specialite", medecin.getSpecialite())
8             .append("email", medecin.getEmail());
9         return medecinCollection.insertOne(doc);
10    }

```

3. Méthodes pour insérer plusieurs médecins :

```

1     public void insertMedecins(List<Medecin> medecins) {
2         List<Document> documents = new ArrayList<>();
3         for (Medecin medecin : medecins) {
4             Document doc = new Document("nom", medecin.getNom()
5                 .append("sexe", medecin.getSexe())
6                 .append("date_naissance", medecin.
7                     getDateNaissance())
8                 .append("specialite", medecin.
9                     getSpecialite())

```

```

8         .append("email", medecin.getEmail());
9         documents.add(doc);
10    }
11    medecinCollection.insertMany(documents);
12 }

```

4. Méthode pour lire un medecin par son id

```

1    public Medecin getMedecinById(String id) {
2        Document query = new Document("_id", new ObjectId(id)
3        );
4        Document result = medecinCollection.find(query).first
5        ();
6        if (result != null) {
7            Medecin medecin = new Medecin();
8            medecin.setId(result.getObjectId(id));
9            medecin.setNom(result.getString("nom"));
10           medecin.setSpecialite(result.getString("
11           specialite"));
12           medecin.setEmail(result.getString("email"));
13           medecin.setSexe(result.getString("sexe"));
14           return medecin;
15       }
16       return null;
17   }

```

5. Méthode pour rechercher tous les médecins ayant une spécialité donnée

```

1    public List<Document> findMedecinsBySpecialite(String
2    specialite) {
3        return medecinCollection.find(Filters.eq("specialite"
4        , specialite)).into(new ArrayList<>());
5    }

```

6. Méthode pour mettre à jour un medecin

```

1    public void updateOneMedecin(String id, String email) {
2        medecinCollection.updateOne(Filters.eq("_id", new
3        ObjectId(id)), Updates.set("email", email));
4    }

```

7. Méthode pour mettre à jour plusieurs médecins

```

1    public void updateMedecins(String specialite, String
2    newEmailDomain) {
3        medecinCollection.updateMany(Filters.eq("specialite",
4        specialite),
5        Updates.set("email", newEmailDomain));
6    }

```

8. Méthode pour supprimer un médecin

```

1 public void deleteMedecin(String id) {
2     MongoCollection<Document> medecinCollection = database.
        getCollection("Medecin");
3     medecinCollection.deleteOne(Filters.eq("_id", new
        ObjectId(id)));
4 }

```

9. Méthode pour supprimer plusieurs médecins selon leur spécialité

```

1 public void deleteMedecins(String specialite) {
        medecinCollection.deleteMany(Filters.eq("specialite",
        specialite));
2 }

```

4.2.2 Indexation secondaire

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

```

1 medecinCollection.createIndex(Indexes.ascending("nom"));

```

4.2.3 Méthode de Consultation (Jointure, Groupement)

```

1 // Groupement des medecins par specialite
2 public void groupMedecinsBySpecialite() {
3     MongoCollection<Document> medecinCollection = database.
        getCollection("Medecin");
4     AggregateIterable<Document> groupResult = medecinCollection
        .aggregate(Arrays.asList(
5         Aggregates.group("$specialite", Accumulators.sum("total", 1))
6     ));
7
8     for (Document doc : groupResult) {
9         System.out.println(doc.toJson());
10    }
11 }
12
13 // Grouper les medecins par specialite et compter le nombre de
    medecins par groupe
14 public List<Document> groupMedecinsBySpecialite() {
15     return medecinCollection.aggregate(
16         Arrays.asList(
17             Aggregates.group("$specialite",
18                 Accumulators.sum("count", 1))
19         )
20     ).into(new ArrayList<>());

```

```

21
22 // Effectuer une jointure entre la collection "Medecin" et "
    Patient" pour afficher les rendez-vous
23     public List<Document> joinMedecinWithPatients() {
24         return medecinCollection.aggregate(
25             Arrays.asList(
26                 Aggregates.lookup("Patient", "_id", "
                    medecin", "list_rendez_vous")
27             )
28         ).into(new ArrayList<>());
29     }
30
31 // Rechercher les medecins ayant plus de N consultations
32 public List<Document> findMedecinsWithMoreThanNConsultations(
    int n) {
33     return medecinCollection.find(Filters.size("
        list_consultations", n)).into(new ArrayList<>());
34 }
35
36 // Grouper par ville et calculer le nombre total de medecins
    dans chaque ville
37 public List<Document> groupMedecinsByCity() {
38     return medecinCollection.aggregate(
39         Arrays.asList(
40             Aggregates.group("$adresse.ville",
                Accumulators.sum("count", 1))
41         )
42     ).into(new ArrayList<>());
43 }
44
45 // Filtrer les medecins par sexe et trier par date de
    naissance
46 public List<Document> filterMedecinsBySexAndSortByDate(String
    sexe) {
47     return medecinCollection.find(Filters.eq("sexe", sexe))
48         .sort(new Document("date_naissance", 1))
49         .into(new ArrayList<>());
50 }

```

5 Compléments sur le moteur NoSql MONGODB

5.1 Modèles de données supportés

5.1.1 Modèle de données : Document (JSON/BSON)

MongoDB stocke les données sous forme de documents JSON (ou BSON, une version binaire de JSON). Chaque document est un ensemble de paires clé-valeur. Cela permet de gérer des structures complexes, des tableaux et des sous-documents imbriqués. Le modèle est flexible, sans schéma strict (schema-less).

5.1.2 Usage

Convient aux cas où la structure des données peut varier ou être modifiée fréquemment.

5.2 Procédure d'installation du moteur et des utilitaires

5.2.1 Installation

Relativement simple. MongoDB offre des packages pour les systèmes d'exploitation populaires (Linux, Windows, macOS). Il suffit d'installer MongoDB et de démarrer le service MongoDB ("mongod").

5.2.2 Installation (Windows)

Voici un guide étape par étape pour installer MongoDB sur un système Windows :

1. Téléchargement de MongoDB :
 - (a) Accédez à la page de téléchargement de [MongoDB Community Edition](#).
 - (b) Choisissez la version de MongoDB compatible avec votre système (en général, Windows 64-bit).
 - (c) Téléchargez l'installateur .msi pour Windows.
2. Installation de MongoDB :
 - (a) Ouvrez le fichier .msi téléchargé pour lancer l'installation.
 - (b) Suivez les étapes de l'assistant d'installation :
 - i. Sélectionnez l'option **Complete** pour installer MongoDB avec les options par défaut.
 - ii. Cochez l'option **Install MongoDB as a Service** afin que MongoDB démarre automatiquement au démarrage de Windows.
 - iii. Vous pouvez également installer **MongoDB Compass**, l'interface graphique de MongoDB (facultatif mais recommandé pour gérer vos bases de données).
3. Configurer MongoDB
 - (a) Après l'installation, vous devez configurer MongoDB :
 - i. Créer le répertoire de données : MongoDB utilise le dossier `C:\data\db` Par défaut pour stocker ses bases de données. Vous devez créer ce répertoire manuellement.
 - i. Ouvrez l'explorateur de fichiers.
 - ii. Allez dans le disque `C:\`.
 - iii. Créez un dossier nommé `data`.
 - iv. À l'intérieur de `data`, créez un sous-dossier nommé `db` : `C:\data\db`
4. Vérifier que MongoDB est bien installé : Ouvrez un terminal (invité de commandes) et tapez la commande suivante pour vérifier que MongoDB est correctement installé :

```
mongod --version
```
5. Lancer MongoDB : MongoDB devrait s'exécuter en tant que service, mais vous pouvez aussi le démarrer manuellement.

- Via l'invité de commandes : Tapez cette commande `mongod` pour démarrer MongoDB. MongoDB sera lancé et écoutera par défaut sur le port 27017.
- Accéder à MongoDB avec le shell. Ouvrez un nouvel onglet dans l'invité de commandes et tapez : `mongo`. Cela lancera le shell MongoDB, où vous pourrez interagir avec votre base de données.
- Utiliser MongoDB Compass (facultatif) Si vous avez installé MongoDB Compass, vous pouvez l'utiliser pour interagir avec MongoDB via une interface graphique en lançant l'application.

Si MongoDB a été installé en tant que service, vous pouvez démarrer, arrêter ou redémarrer MongoDB directement depuis le gestionnaire de services de Windows :

- (a) Ouvrez "Services" (tapez `services.msc` dans l'Invite de commandes).
- (b) Recherchez le service "MongoDB".
- (c) Cliquez avec le bouton droit pour démarrer, arrêter ou redémarrer le service.

5.2.3 Installation (Linux)

```
bash > sudo apt-get install -y mongodb & systemctl start mongod
```

5.2.4 Installation via Docker

1. Installer *Docker Desktop*.
2. Dans un terminal ou invité de commandes : `docker run -name mongod -p 27017:27017 -d mongodb/mongodb-community-server:latest`

5.2.5 Utilitaires

1. Outils de ligne de commande comme "mongo", "mongodump", et "mongoimport".
 - Mongo fait référence au client interactif utilisé pour interagir avec une base de données MongoDB.
 - Vous pouvez l'utiliser pour exécuter des commandes MongoDB, écrire des requêtes, gérer des collections, etc.
 - Commande typique pour se connecter à une base MongoDB : `mongo` Si vous voulez spécifier une base de données particulière : `mongo <nom_de_la_base>`
 - Mongodump est un outil utilisé pour sauvegarder une base de données MongoDB. Il crée un *snapshot* des collections sous forme de fichiers BSON (Binary JSON).
 - Cela est utile pour les sauvegardes ou pour déplacer des données d'un environnement à un autre. Commande de base pour sauvegarder une base de données : `mongodump -db <nom_de_la_base>` Vous pouvez aussi spécifier un répertoire de sortie : `mongodump -db <nom_de_la_base> -out /chemin/vers/sauvegarde`
 - Mongoimport permet de restaurer ou importer des données dans MongoDB depuis des fichiers au format JSON, CSV ou TSV. Commande de base pour importer un fichier JSON dans une collection MongoDB :
`mongoimport --db <nom_de_la_base> --collection <nom_collection> --file <chemin_vers_le_fichier.json>`

. Pour des fichiers CSV, vous pouvez ajouter l'option `-type csv` :

```
mongoimport --db <nom\_de\_la\_base> --collection
<nom\_de\_la\_collection> --type csv --file
<chemin\_vers\_le\_fichier.csv> --headerline
```

Ces trois outils sont essentiels pour la gestion, la sauvegarde et la restauration des données dans MongoDB.

2. MongoDB Compass offre une interface utilisateur graphique pour interagir avec les données.

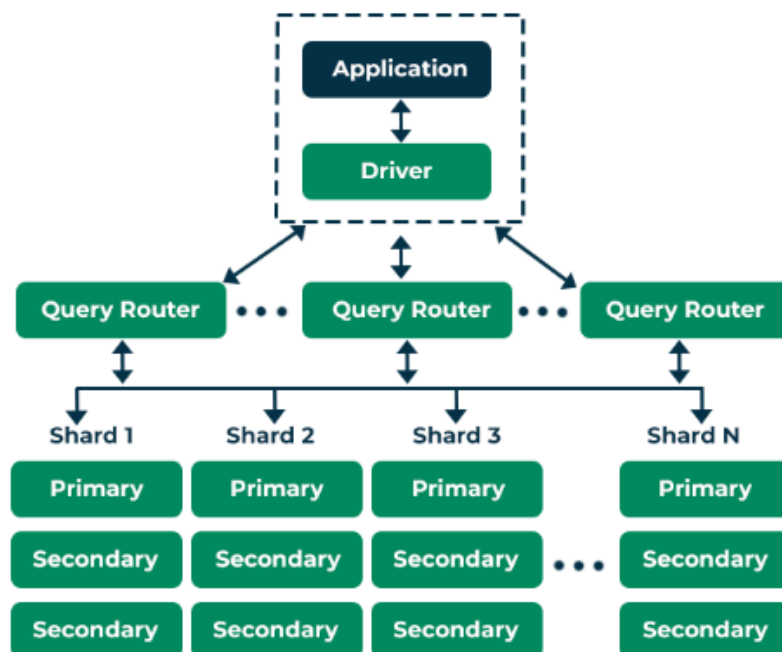
5.3 Architecture du moteur NoSQL (schéma)

5.3.1 Architecture

MongoDB suit une architecture maître-esclave (ou primaire-secondaire) dans un ensemble de *replicas* ou dans une partition. Les données sont distribuées en partitions (sharding) pour faciliter la scalabilité horizontale. Les partitions sont réparties entre plusieurs serveurs.

5.3.2 Schéma

- Sharding pour le partitionnement.
- Réplication pour la haute disponibilité.
- Chaque ensemble de replicas a un replica *primaire* ou maître qui gère les écritures.



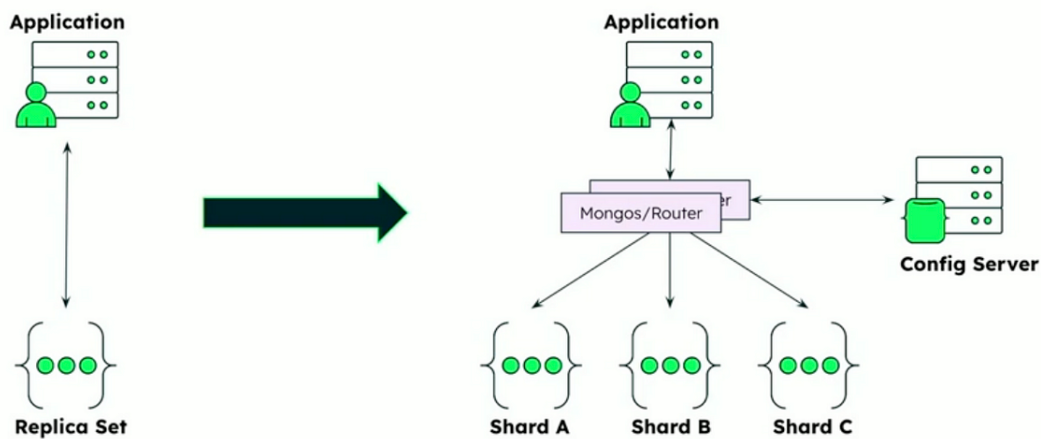
5.4 Méthode de partitionnement

5.4.1 Sharding

MongoDB utilise un mécanisme de *sharding* pour partitionner horizontalement les données. Les documents sont distribués entre différents partitions en fonction de la clé de sharding (clé de partitionnement).

5.4.2 Schéma

- Clé de partitionnement choisie pour diviser les documents.
- Chaque partition stocke une portion des données.



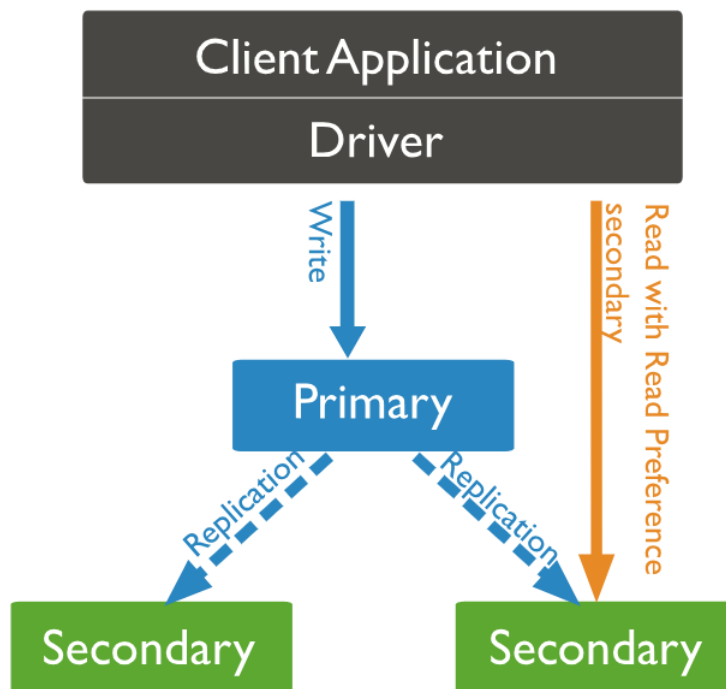
5.5 Méthode de réplication

5.5.1 Réplication

MongoDB utilise des *replicas* pour la haute disponibilité. Un ensemble de replicas contient un replica primaire et plusieurs secondaires. Le primaire gère les écritures, et les réplicas (secondaires) répliquent les données.

5.5.2 Schéma

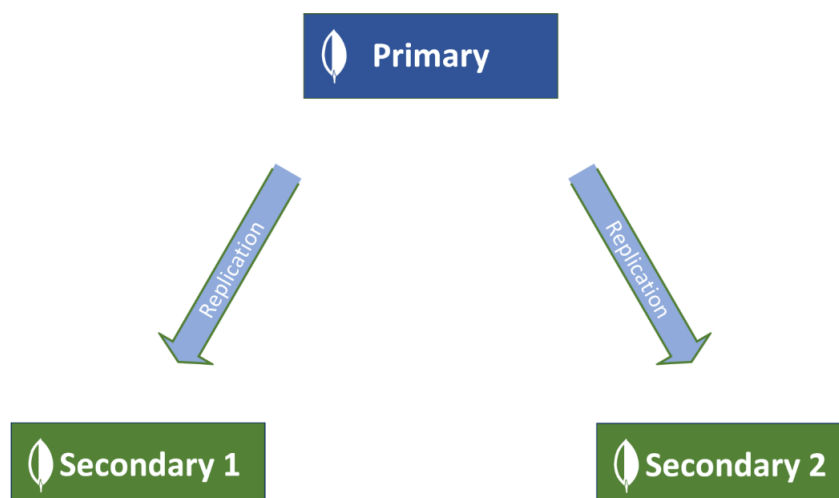
- Primaire pour les écritures.
- Secondaires pour la redondance et les lectures.



5.6 Montée en charge

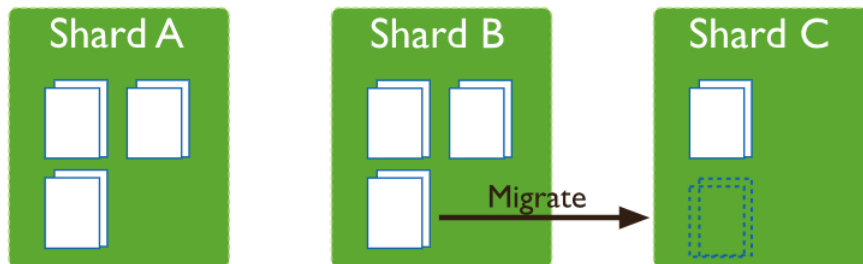
5.6.1 Montée en charge verticale

MongoDB permet une montée en charge verticale grâce au mécanisme de répliqua-tion sur plusieurs serveurs dans une même partition. En ajoutant de nouveaux serveurs ou replicas, les données peuvent être répliquées de manière synchrone par défaut pour mieux répartir la charge. L'ajout de nouveaux esclaves ou replicas dans une partition pour étendre la base de données verticalement permet de faciliter les opérations de lec-ture garantir et améliorer la disponibilité.



5.6.2 Montée en charge horizontale

MongoDB permet une montée en charge horizontale grâce au partitionnement sur plusieurs serveurs. En ajoutant de nouveaux serveurs, les partitions peuvent être redistribuées pour mieux répartir la charge. L'ajout de partitions pour étendre la base de données horizontalement permet de faciliter les opérations d'écriture.

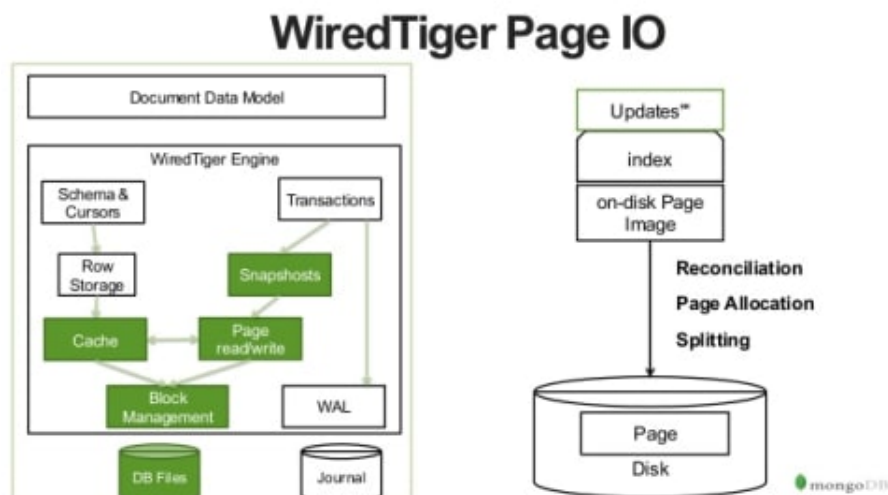


5.7 Gestion du cache mémoire

MongoDB utilise le cache de *mappage mémoire*. Le moteur *WiredTiger* utilise la mémoire disponible pour les lectures, mais laisse au système d'exploitation la gestion précise de l'utilisation de la mémoire.

5.7.1 Schéma

WiredTiger, mémoire tampon pour les écritures et mappage des pages de disque dans la mémoire.



6 Génération automatique des données

Dans cette section nous allons générer des données pour nos deux collections *Médecin* et *Patient*. Pour cela nous utilisons le package *gem* du langage Ruby. Ci-dessous la procédure d'installation de Ruby et *Gem* :

1. Installation de Ruby
 - macOS : Ruby est généralement déjà pré-installé.
 - Windows : Ruby est téléchargeable a l'adresse <https://rubyinstaller.org/>.
 - Linux (Ubuntu) : `sudo apt install ruby`.
2. Ouvrir un Terminal/Command Prompt :
 - macOS/Linux : Open a terminal.
 - Windows : Open Command Prompt or PowerShell.
3. `gem install faker`
4. `gem install mongo`
5. `gem list` (vérification de l'installation)

6.1 Script de génération de 1000 médecins

```

1  require 'faker'
2  require 'json'
3  require 'date'
4  require 'mongo'
5
6  def generate_random_document
7    {
8      "_id" => BSON::ObjectId.new,
9      "nom" => Faker::Name.last_name,
10     "sexe" => ["Male", "Female", "Other"].sample,
11     "date_naissance" => Faker::Date.birthday(min_age: 25,
12       max_age: 65),
13     "specialite" => ["Cardiologist", "Dermatologist", "
14       Pediatrician", "Orthopedic Surgeon", "Neurologist", "
15       Gynecologist"].sample,
16     "email" => Faker::Internet.email,
17     "cv" => Faker::Lorem.sentence(word_count: 10),
18     "list_telephones" => [Faker::PhoneNumber.phone_number,
19       Faker::PhoneNumber.phone_number],
20     "list_prenoms" => [Faker::Name.first_name, Faker::Name.
21       first_name],
22     "adresse" => {
23       "numero" => Faker::Address.building_number,
24       "rue" => Faker::Address.street_name,
25       "code_postal" => Faker::Address.zip_code,
26       "ville" => Faker::Address.city
27     },
28     "list_rendez_vous" => [
29       {
30         "patient" => BSON::ObjectId.new,
31         "date_rendez_vous" => Faker::Date.forward(days: 23),
32         "motif" => Faker::Lorem.sentence(word_count: 3)
33       }
34     ],
35     "list_consultations" => [

```

```

31     {
32       "patient" => BSON::ObjectId.new,
33       "raison" => Faker::Lorem.sentence(word_count: 3),
34       "diagnostic" => Faker::Lorem.sentence(word_count: 5),
35       "date_consultation" => Faker::Date.backward(days: 30),
36       "list_examens" => [
37         {
38           "details_examen" => Faker::Lorem.sentence(
39             word_count: 5),
40           "date_examen" => Faker::Date.backward(days: 15)
41         },
42       ],
43       "list_prescriptions" => [
44         {
45           "details_prescription" => Faker::Lorem.sentence(
46             word_count: 5),
47           "date_prescription" => Faker::Date.backward(days:
48             10)
49         }
50       ]
51     }
52   end
53
54   documents = Array.new(1000) { generate_random_document }
55
56   File.open("medecins_data.json", "w") do |f|
57     f.write(JSON.pretty_generate(documents))
58   end

```

6.2 Script de génération de 1000 patients

```

1   require 'mongo'
2   require 'faker'
3   require 'json'
4   require 'date'
5
6   def generate_random_patient
7     {
8       "_id" => BSON::ObjectId.new,
9       "numero_securite_sociale" => Faker::IdNumber.valid,
10      "nom" => Faker::Name.last_name,
11      "sexe" => ["M", "F", "O"].sample,
12      "date_naissance" => Faker::Date.birthday(min_age: 0,
13        max_age: 100),
14      "email" => Faker::Internet.email,
15      "poids" => Faker::Number.between(from: 45.0, to: 120.0).
16        round(2),

```

```

15     "hauteur" => Faker::Number.between(from: 1.5, to: 2.0).
      round(2),
16     "list_telephones" => [Faker::PhoneNumber.phone_number,
      Faker::PhoneNumber.phone_number],
17     "list_prenoms" => [Faker::Name.first_name, Faker::Name.
      first_name],
18     "adresse" => {
19         "numero" => Faker::Address.building_number,
20         "rue" => Faker::Address.street_name,
21         "code_postal" => Faker::Address.zip_code,
22         "ville" => Faker::Address.city
23     },
24     "list_rendez_vous" => [
25         {
26             "medecin" => BSON::ObjectId.new,
27             "date_rendez_vous" => Faker::Date.forward(days: 23),
28             "motif" => Faker::Lorem.sentence(word_count: 3)
29         }
30     ],
31     "list_consultations" => generate_consultations(Faker),
32     "antecedents_medicaux" => generate_antecedents_medicaux(
      Faker),
33     "allergies" => [Faker::Lorem.word, Faker::Lorem.word]
34 }
35 end
36
37 def generate_consultations(faker)
38     consultations_list = []
39
40     2.times do
41         examens_list = [
42             {
43                 "details_examen" => Faker::Lorem.sentence(word_count:
44                     5),
45                 "date_examen" => Faker::Date.backward(days: 15)
46             }
47         ]
48
49         prescriptions_list = [
50             {
51                 "details_prescription" => Faker::Lorem.sentence(
52                     word_count: 5),
53                 "date_prescription" => Faker::Date.backward(days: 10)
54             }
55         ]
56
57         facture = {
58             "Montant_Total" => faker::Number.between(from: 50.0, to:
59                 500.0).round(2),
60             "date_facture" => Faker::Date.backward(days: 7)
61         }
62     end
63 end

```

```

58     }
59
60     consultations_list << {
61       "medecin" => BSON::ObjectId.new,
62       "raison" => Faker::Lorem.sentence(word_count: 3),
63       "diagnostic" => Faker::Lorem.sentence(word_count: 5),
64       "date_consultation" => Faker::Date.backward(days: 30),
65       "list_examens" => examens_list,
66       "list_prescriptions" => prescriptions_list,
67       "facture" => facture
68     }
69   end
70
71   consultations_list
72 end
73
74 def generate_antecedents_medicaux(faker)
75   [
76     {
77       "type" => "Chronic",
78       "description" => Faker::Lorem.sentence(word_count: 5),
79       "date" => Faker::Date.backward(days: 500)
80     },
81     {
82       "type" => "Acute",
83       "description" => Faker::Lorem.sentence(word_count: 5),
84       "date" => Faker::Date.backward(days: 300)
85     }
86   ]
87 end
88
89 patients = Array.new(1000) { generate_random_patient }
90
91 File.open("patients_data.json", "w") do |f|
92   f.write(JSON.pretty_generate(patients))
93 end

```

6.3 Génération des données dans des fichiers JSON

6.3.1 Données des médecins

1. Mettre le script pour la génération des données pour les médecins dans un fichier, par exemple `medecins_script.rb`
2. `ruby medecins_script.rb`
3. le fichier `medecins_data.json` a été généré et contient les données des médecins au format *JSON*.

6.3.2 Données des patients

1. Mettre le script pour la génération des données pour les patients dans un fichier, par exemple `patients_script.rb`
2. `ruby patients_script.rb`
3. le fichier `patients_data.json` a été généré et contient les données des patients au format *JSON*.