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

29 septembre 2024

Table des matières

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 complète 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, enregistrés dans la table RENDEZ-VOUS. Chaque consultation est consignée dans la table CONSULTATION, associée 'a un patient, un médecin et une facture. Les prescriptions médicales sont enregistrées dans la table PRESCRIPTION, liées la consultation correspondante. Les détails des examens sont stockés dans la table EXAMEN, également lies a la consultation. Chaque consultation génère une facture, enregistrée dans la table FACTURE avec le montant total a payer. Des contraintes de clé étrangère garantissent l'intégrité des données et les relations entre les tables. 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	Chaine	Caractère variable, 50 caractères max	non nulle	Oui
Nom	Nom de famille du patient	Chaîne	Caractère variable, 50 caractères max	Non null	Non
Sexe	Sexe du patient	Chaîne	Caractère, 4 caractères max	Non null	Non
Date_naissance	Date de naissance du patient	Date		Non null	Non
Poids	Poids du patient	Entier	Numerique	Non null	Non
Hauteur	Entier	Numerique		Non null	Non
listTelephones	Chaine	Caractère variable, 50 caractères max		Non null	Non
listPrenoms	Prénoms du patient	Chaine	Caractère variable, 50 caractères max	Non null	Non
Adresse	Adresse du patient	Chaine	Caractère variable, 50 caractères max	Non null	Non

Email	Email du patient	Chaîne	Caractère variable, 50 caractères max	Non null	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	Entier	Long	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	Chaine		Valeur unique, non nulle	non
date_antec_medic	date de l'allergie	Date		Valeur unique, 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	Chaîne	Numérique	Non null	Non
Date_Facture	Date d'émission de la facture	Date		Non null	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 null	Non
Date_Examen	Date de l'examen	Date		Non null	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 null	Non
Diagnostic	Diagnostic au terme de la consultation	Chaîne	Caractère variable, 300 caractères max	Non null	Non
Date_Consultation	Date de la consultation	Date		Non null	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 medecin	Chaine	Caractère variable, 50 caractères max	Non null	Non
Sexe	Sexe du medecin	Chaîne	Caractère, 4 caractères max	Non null	Non

Specialite	Specialite du medecin	Caractère variable, 50 caractères max	Caractère, 4 caractères max	Non null	Non
Nom	Nom de famille du médecin	Chaîne	Caractère variable, 50 caractères max	Non null	Non
listPrenoms	Prénoms du médecin	Chaine		Non null	Non
Adresse	Adresse du médecin	Chaine		Non null	Non
Email	Email du médecin	Chaîne	Caractère variable, 50 caractères max	Non null	Non
listTelephones	Numéros de téléphone du médecin	Chaine		Non null	Non
Date_naissance	Date de naissance du médecin	Date		Non null	Non
CV	CV du médecin	CLOB		Non null	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 entres 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 « Necessiter » 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.

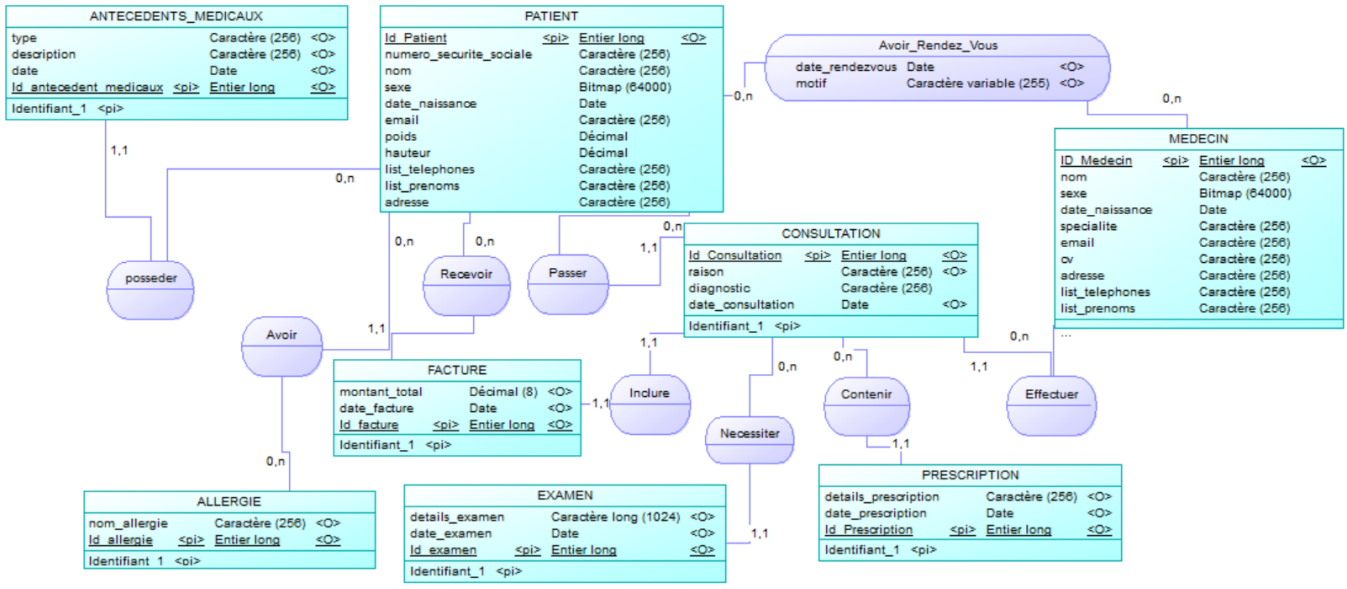
3.8 Association « Posséder » entre ANTECEDENTS_MEDICAUX et PATIENT

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

```
CREATE TABLE Medecin_By_Speciality (
  id UUID,
  nom text,
  sexe text,
  date_naissance date,
  specialite text,
  email text,
  cv text,
  adresse FROZEN<map<text, text>>,
  list_telephones set<text>,
  list_prenoms set<text>,
  PRIMARY KEY(specialite, date_naissance, nom, sexe, email, id)
);
```

Clé de partition : `specialite`

4.1.2 Table "Patient_By_BirthDay"

```
CREATE TABLE Patient_By_BirthDay (
  id UUID,
  numero_securite_sociale text,
  nom text,
  sexe text,
  date_naissance date,
  email text,
  poids double,
  hauteur double,
  list_telephones set<text>,
  list_prenoms set<text>,
  adresse FROZEN<map<text, text>>,
  allergies list<text>,
  PRIMARY KEY(date_naissance, nom, sexe, email, id)
);
```

Clé de partition : `date_naissance`

4.1.3 Table "RendezVous_By_Date"

```
CREATE TABLE RendezVous_By_Date(
  rendezvous_date date,
  patient_id UUID,
  doctor_id UUID,
  motif text,
  PRIMARY KEY(rendezvous_date, patient_id, doctor_id)
);
```

Clé de partition : `rendezvous_date`

4.1.4 Table "Consultation_By_Date"

```
CREATE TABLE Consultation_By_Date (
  consultation_date date,
  patient_id UUID,
  doctor_id UUID,
  raison text,
  diagnostic text,
  facture FROZEN<map<text, text>>,
  prescriptions list<FROZEN<map<text, text>>>,
  examens list<FROZEN<map<text, text>>>,
  PRIMARY KEY(consultation_date, patient_id, doctor_id)
);
```

Clé de partition : `consultation_date`

4.2 Spécification des classes et des méthodes JAVA

4.2.1 Méthodes CRUD

```java
// Classe Medecin
package org.example.entities;

import java.time.LocalDate;
import java.util.Map;
import java.util.Set;
import java.util.UUID;

public class Medecin {
    private UUID id;
    private String nom;
    private String sexe;
    private LocalDate dateNaissance;
    private String specialite;
    private String email;
    private String cv;
    private Map<String, String> adresse;
    private Set<String> listTelephones;
    private Set<String> listPrenoms;

    public Medecin(UUID id, String nom, String sexe,
       LocalDate dateNaissance, String specialite,
                   String email, String cv, Map<String,
                      String> adresse,
                   Set<String> listTelephones, Set<String>
                      listPrenoms) {
        this.id = id;
        this.nom = nom;
        this.sexe = sexe;
        this.dateNaissance = dateNaissance;
        this.specialite = specialite;
        this.email = email;
        this.cv = cv;
        this.adresse = adresse;
        this.listTelephones = listTelephones;
        this.listPrenoms = listPrenoms;
    }

    // Getters and Setters
    public UUID getId() {
        return id;
    }

    public void setId(UUID id) {
        this.id = id;
    }

```

```
45      public String getNom() {
46          return nom;
47      }
48
49      public void setNom(String nom) {
50          this.nom = nom;
51      }
52
53      public String getSexe() {
54          return sexe;
55      }
56
57      public void setSexe(String sexe) {
58          this.sexe = sexe;
59      }
60
61      public LocalDate getDateNaissance() {
62          return dateNaissance;
63      }
64
65      public void setDateNaissance(LocalDate dateNaissance) {
66          this.dateNaissance = dateNaissance;
67      }
68
69      public String getSpecialite() {
70          return specialite;
71      }
72
73      public void setSpecialite(String specialite) {
74          this.specialite = specialite;
75      }
76
77      public String getEmail() {
78          return email;
79      }
80
81      public void setEmail(String email) {
82          this.email = email;
83      }
84
85      public String getCv() {
86          return cv;
87      }
88
89      public void setCv(String cv) {
90          this.cv = cv;
91      }
92
93      public Map<String, String> getAdresse() {
94          return adresse;
```

```
95      }
96
97      public void setAdresse(Map<String, String> adresse) {
98          this.adresse = adresse;
99      }
100
101     public Set<String> getListTelephones() {
102         return listTelephones;
103     }
104
105     public void setListTelephones(Set<String> listTelephones)
            {
106         this.listTelephones = listTelephones;
107     }
108
109     public Set<String> getListPrenoms() {
110         return listPrenoms;
111     }
112
113     public void setListPrenoms(Set<String> listPrenoms) {
114         this.listPrenoms = listPrenoms;
115     }
116
117     @Override
118     public String toString() {
119         return "Medecin{" +
120                 "id=" + id +
121                 ", nom='" + nom + '\'' +
122                 ", sexe='" + sexe + '\'' +
123                 ", dateNaissance=" + dateNaissance +
124                 ", specialite='" + specialite + '\'' +
125                 ", email='" + email + '\'' +
126                 ", cv='" + cv + '\'' +
127                 ", adresse=" + adresse +
128                 ", listTelephones=" + listTelephones +
129                 ", listPrenoms=" + listPrenoms +
130                 '}';
131     }
132 }
133
134 // Classe Patient
```

2. Méthode pour insérer un médecin :

```
1       public void insertOneMedecin(Medecin medecin) {
2           String query = "INSERT INTO Medecin_By_Speciality (id
                , nom, sexe, date_naissance, specialite, email, cv
                , adresse, list_telephones, list_prenoms) " +
3                   "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
4
5           PreparedStatement preparedStatement = session.prepare
```

```
            (query);

        BoundStatement boundStatement = preparedStatement.
            bind(
                medecin.getId(),
                medecin.getNom(),
                medecin.getSexe(),
                medecin.getDateNaissance(),
                medecin.getSpecialite(),
                medecin.getEmail(),
                medecin.getCv(),
                medecin.getAdresse(),
                medecin.getListTelephones(),
                medecin.getListPrenoms()
        );

        session.execute(boundStatement);
    }
```

3. Méthode pour lire un medecin

```
    public Medecin getMedecin(String specialite, LocalDate
        dateNaissance, String nom) {
        String query = "SELECT id, nom, sexe, date_naissance,
            specialite, email, cv, adresse, list_telephones,
            list_prenoms " +
                "FROM Medecin_By_Speciality WHERE specialite
                    = ? AND date_naissance = ? AND nom = ?";

        PreparedStatement preparedStatement = session.prepare
            (query);

        BoundStatement boundStatement = preparedStatement.
            bind(specialite, dateNaissance, nom);

        ResultSet resultSet = session.execute(boundStatement)
            ;

        Row row = resultSet.one();

        if (row != null) {
            UUID id = row.getUuid("id");
            String sexe = row.getString("sexe");
            String email = row.getString("email");
            String cv = row.getString("cv");
            Map<String, String> adresse = row.getMap("adresse
                ", String.class, String.class);
            Set<String> listTelephones = row.getSet("
                list_telephones", String.class);
            Set<String> listPrenoms = row.getSet("
```

```
                list_prenoms", String.class);

            return new Medecin(id, nom, sexe, dateNaissance,
                specialite, email, cv, adresse, listTelephones
                , listPrenoms);
        }
        return null;
    }
```

4. Méthode pour mettre à jour un medecin

```
    public void updateMedecin(Medecin medecin) {
        String query = "UPDATE Medecin_By_Speciality " +
                "SET cv = ?, adresse = ?, list_telephones =
                   ?, list_prenoms = ? " +
                "WHERE specialite = ? AND date_naissance = ?
                   AND nom = ? AND sexe = ? AND email = ? AND
                    id = ?";

        PreparedStatement preparedStatement = session.prepare
            (query);

        BoundStatement boundStatement = preparedStatement.
            bind(
                medecin.getCv(),
                medecin.getAdresse(),
                medecin.getListTelephones(),
                medecin.getListPrenoms(),
                medecin.getSpecialite(),
                medecin.getDateNaissance(),
                medecin.getNom(),
                medecin.getSexe(),
                medecin.getEmail(),
                medecin.getId()
        );

        session.execute(boundStatement);
    }
```

5. Méthode pour supprimer un medecin

```
public void deleteOneMedecin(String specialite, LocalDate
    dateNaissance, String nom, String sexe, String email, UUID
     id) {
        String query = "DELETE FROM Medecin_By_Speciality " +
                "WHERE specialite = ? AND date_naissance = ?
                   AND nom = ? AND sexe = ? AND email = ? AND
                    id = ?";

        PreparedStatement preparedStatement = session.prepare
            (query);
```

```
6
7           BoundStatement boundStatement = preparedStatement.
               bind(
8                   specialite,
9                   dateNaissance,
10                  nom,
11                  sexe,
12                  email,
13                  id
14          );
15
16          session.execute(boundStatement);
17      }
```

4.2.2 Indexation secondaire

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

```
1       public void createIndex(){
2           session.execute("CREATE INDEX IF NOT EXISTS ON
               Medecin (nom)");
3       }
```

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.

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`.

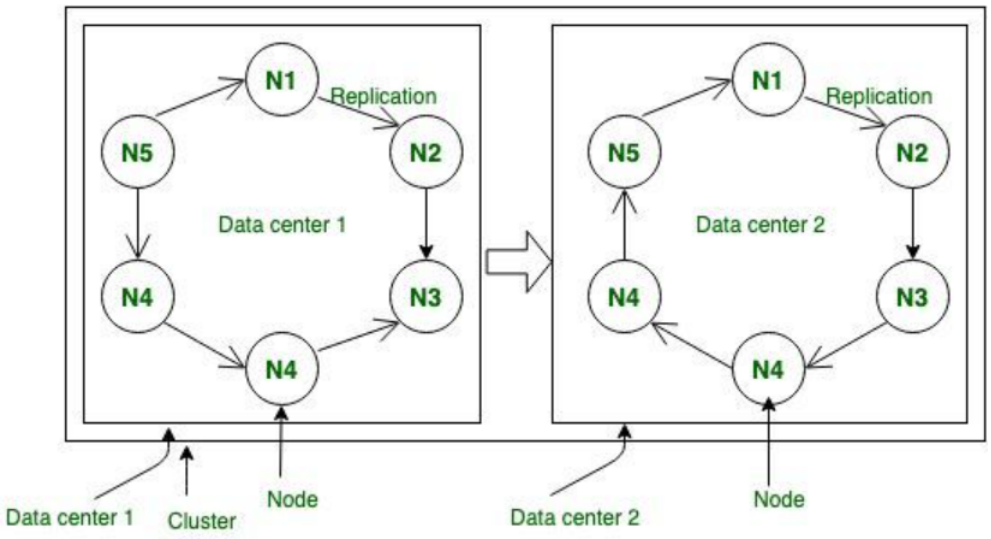
5.3 Architecture du moteur NoSQL (schéma)

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.



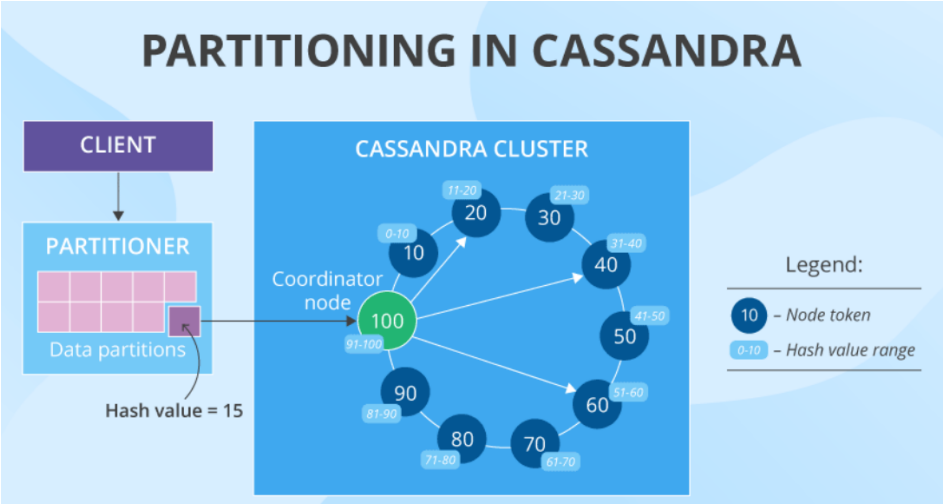
5.4 Méthode de partitionnement

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.



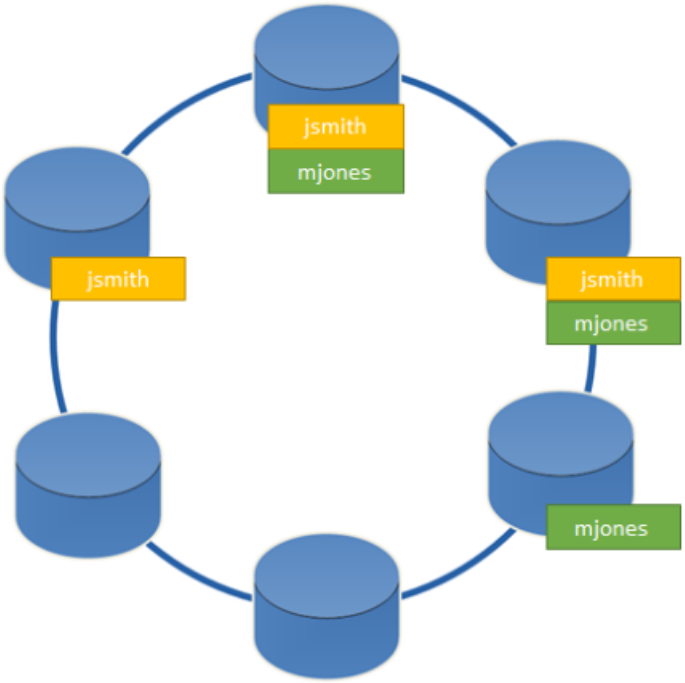
5.5 Méthode de réplication

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

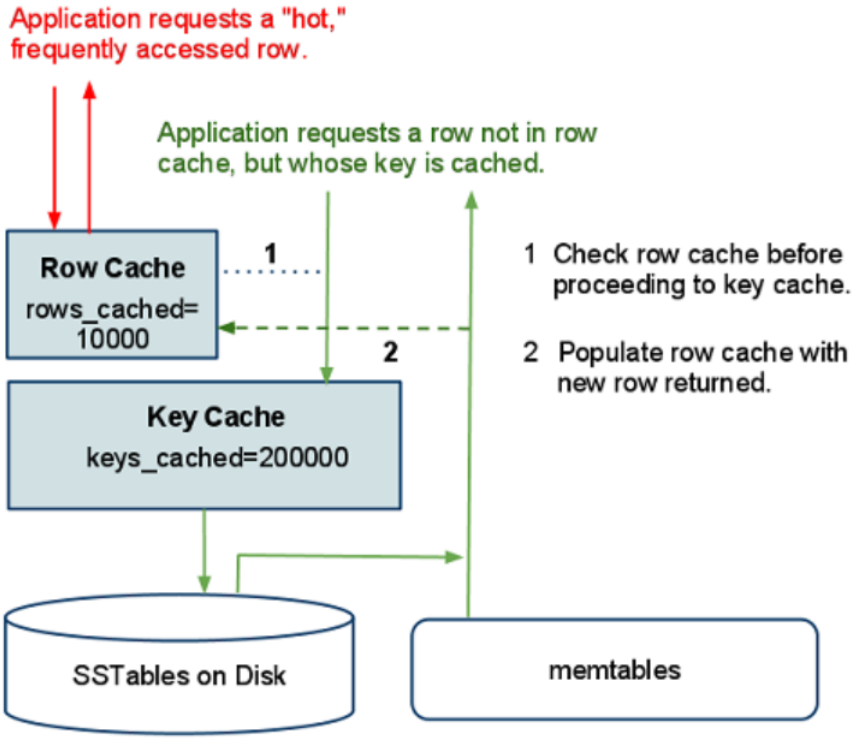
Nœuds ajoutés au cluster sans point de contrôle central.

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 utilise le script Ruby suivant :

```ruby
require 'json'
require 'securerandom'
require 'date'

# Helper functions
def random_date(start_date, end_date)
  rand(start_date..end_date).to_s
end

def random_text(length)
  ('a'..'z').to_a.sample(length).join
end

def random_sex
  ['Male', 'Female'].sample
end

def random_phone
  "+#{rand(1000000000..9999999999)}"
end

def random_address
  { "numero" => rand(1..100).to_s, "street" => random_text(10),
    "city" => random_text(6), "postal_code" => rand
   (10000..99999).to_s }
end

def random_specialty
  ['Cardiology', 'Neurology', 'Pediatrics', 'Orthopedics', '
     Dermatology'].sample
end

def random_motif
  ['Routine Checkup', 'Emergency Visit', 'Follow-up', '
     Consultation'].sample
end

# Generating 1000 records for Medecin_By_Speciality
medecin_data = 1000.times.map do
  {
    "id" => SecureRandom.uuid,
```

```
38          "nom" => random_text(8),
39          "sexe" => random_sex,
40          "date_naissance" => random_date(Date.new(1960, 1, 1), Date.
              new(1995, 12, 31)),
41          "specialite" => random_specialty,
42          "email" => "#{random_text(5)}@domain.com",
43          "cv" => random_text(20),
44          "adresse" => random_address,
45          "list_telephones" => [random_phone, random_phone],
46          "list_prenoms" => [random_text(6), random_text(5)]
47        }
48      end
49
50      # Generating 1000 records for Patient_By_BirthDay
51      patient_data = 1000.times.map do
52        {
53          "id" => SecureRandom.uuid,
54          "numero_securite_sociale" => rand(1000000000..9999999999).
              to_s,
55          "nom" => random_text(8),
56          "sexe" => random_sex,
57          "date_naissance" => random_date(Date.new(1960, 1, 1), Date.
              new(2005, 12, 31)),
58          "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
              .new(2024, 12, 31)),
72          "patient_id" => SecureRandom.uuid,
73          "doctor_id" => SecureRandom.uuid,
74          "motif" => random_motif
75        }
76      end
77
78      # Generating 1000 records for Consultation_By_Date
79      consultation_data = 1000.times.map do
80        {
81          "consultation_date" => random_date(Date.new(2020, 1, 1),
              Date.new(2024, 12, 31)),
82          "patient_id" => SecureRandom.uuid,
```

```
        "doctor_id" => SecureRandom.uuid,
        "raison" => random_motif,
        "diagnostic" => random_text(20),
        "facture" => { "montant_total" => "#{rand(100..1000)} USD",
            "date_facture": random_date(Date.new(2020, 1, 1), Date.
          new(2024, 12, 31))},
        "prescriptions" => 3.times.map { { "details_prescription"
          => random_text(20), "date_prescription" => random_date(
          Date.new(2020, 1, 1), Date.new(2024, 12, 31))  } },
        "examens" => 2.times.map { { "details_examen" =>
          random_text(20) , "date_examen" => random_date(Date.new
          (2020, 1, 1), Date.new(2024, 12, 31))  } }
      }
    end

    # Writing to JSON files
    File.write('Medecin_By_Speciality.json', JSON.pretty_generate(
       medecin_data))
    File.write('Patient_By_BirthDay.json', JSON.pretty_generate(
       patient_data))
    File.write('RendezVous_By_Date.json', JSON.pretty_generate(
       rendezvous_data))
    File.write('Consultation_By_Date.json', JSON.pretty_generate(
       consultation_data))

    puts "Files generated successfully!"
```

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

```
    require 'json'
    require 'csv'

    # Function to handle flattening nested structures like arrays
       and hashes
    def flatten_value(value)
      case value
      when Array
        value.join('; ') # Join array elements with ';' for CSV
      when Hash
        value.map { |k, v| "#{k}: #{v}" }.join('; ') # Flatten hash
            into 'key: value' format
      else
        value
      end
    end

    # Function to convert JSON to CSV
    def json_to_csv(json_file, csv_file)
      # 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