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

25 septembre 2024

Table des matières

1 Choix du sujet

SUJET : Gestion d'un cabinet médical

2 MCD MERISE

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

3.1 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.

3.1.1 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

3.1.2 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>>,
```

```
    allergies list<text>,
    PRIMARY KEY(date_naissance, nom, sexe, email, id)
);
```

Clé de partition : `date_naissance`

3.1.3 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`

3.1.4 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`

3.2 Spécification des classes et des méthodes JAVA

3.2.1 Méthodes CRUD

```
1. // 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;
```

```
11      private String nom;
12      private String sexe;
13      private LocalDate dateNaissance;
14      private String specialite;
15      private String email;
16      private String cv;
17      private Map<String, String> adresse;
18      private Set<String> listTelephones;
19      private Set<String> listPrenoms;
20
21      public Medecin(UUID id, String nom, String sexe,
           LocalDate dateNaissance, String specialite,
22                     String email, String cv, Map<String,
                          String> adresse,
23                     Set<String> listTelephones, Set<String>
                          listPrenoms) {
24          this.id = id;
25          this.nom = nom;
26          this.sexe = sexe;
27          this.dateNaissance = dateNaissance;
28          this.specialite = specialite;
29          this.email = email;
30          this.cv = cv;
31          this.adresse = adresse;
32          this.listTelephones = listTelephones;
33          this.listPrenoms = listPrenoms;
34      }
35
36      // Getters and Setters
37      public UUID getId() {
38          return id;
39      }
40
41      public void setId(UUID id) {
42          this.id = id;
43      }
44
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);
6
7         BoundStatement boundStatement = preparedStatement.
              bind(
8                 medecin.getId(),
9                 medecin.getNom(),
10                medecin.getSexe(),
11                medecin.getDateNaissance(),
12                medecin.getSpecialite(),
13                medecin.getEmail(),
14                medecin.getCv(),
15                medecin.getAdresse(),
16                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);

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

        session.execute(boundStatement);
```

```
17      }
```

3.2.2 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       }
```

3.2.3 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.

4 Compléments sur le moteur NoSql CASSANDRA

4.1 Modèles de données supportés

4.1.1 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.

4.1.2 Usage

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

4.2 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.

4.2.1 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.

4.2.2 Installation (Linux)

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

4.2.3 Installation via Docker

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

4.2.4 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`.

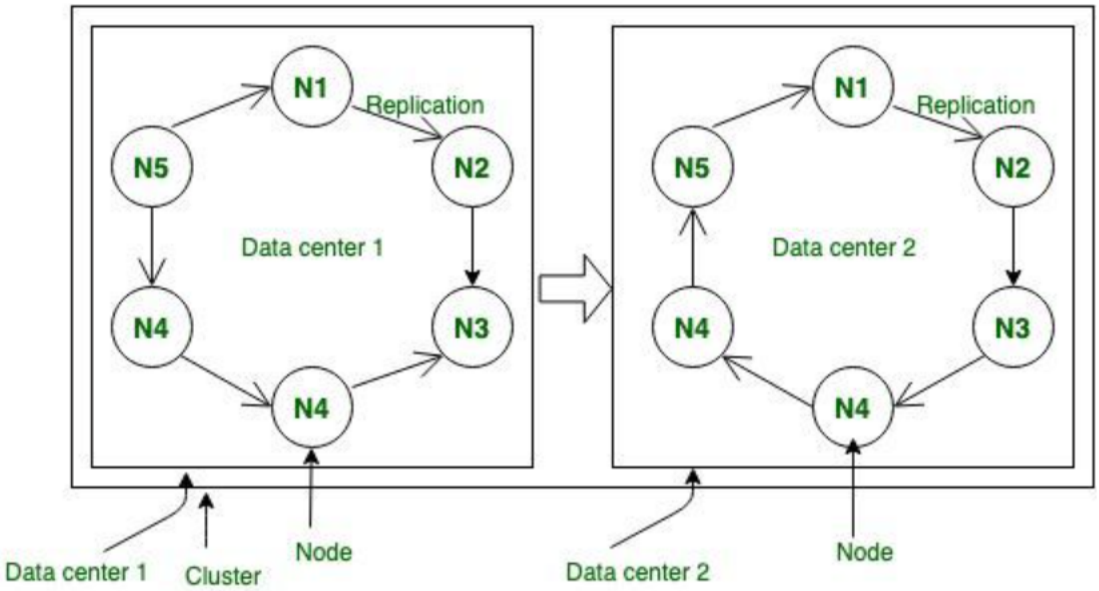
4.3 Architecture du moteur NoSQL (schéma)

4.3.1 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.

4.3.2 Schéma

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



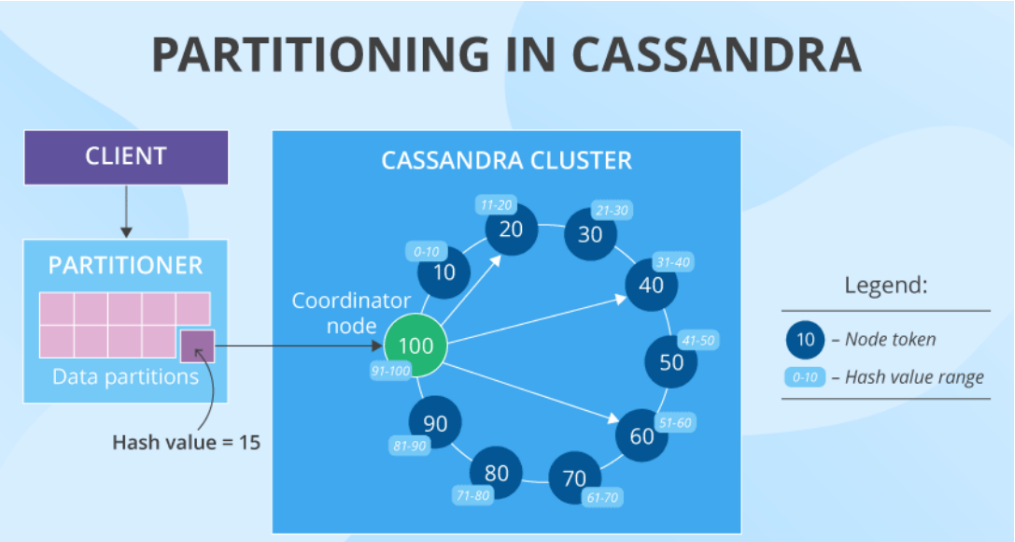
4.4 Méthode de partitionnement

4.4.1 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.

4.4.2 Schéma

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



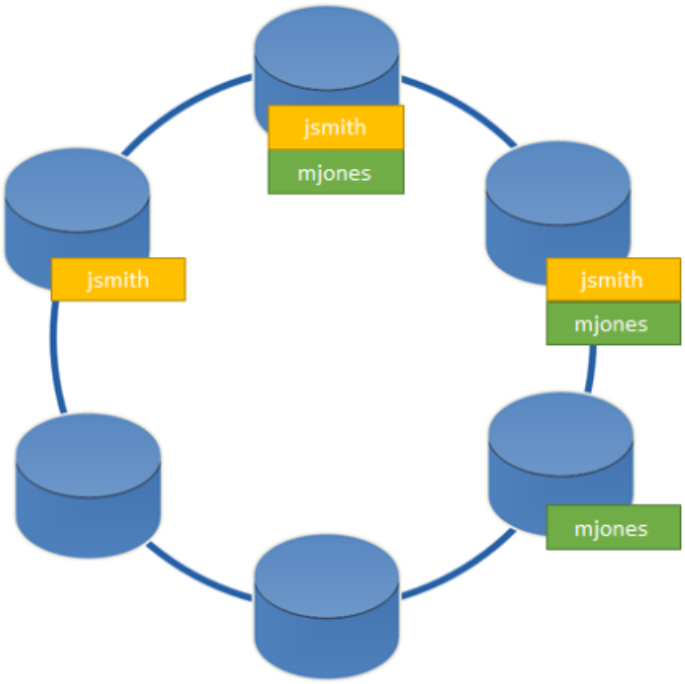
4.5 Méthode de réplication

4.5.1 Réplication

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

4.5.2 Schéma

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



4.6 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.

4.6.1 Schéma

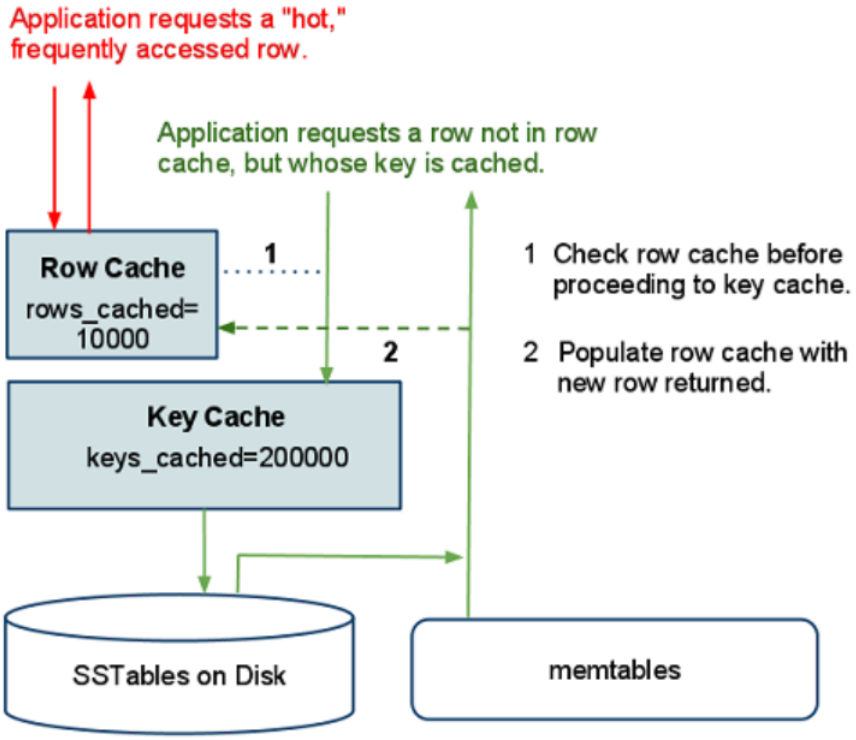
Nœuds ajoutés au cluster sans point de contrôle central.

4.7 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).

4.7.1 Schéma

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



5 Génération automatique des données

Dans cette section nous allons générer des données.

5.1 Script de génération des données

5.2 Génération des données dans des fichiers `JSON`