

# Bases de Données Complexes

Rapport final



# *cassandra*

Khalil Bennis  
Houda Boukham  
El Mokhtar Nourhira

Encadrés par Mme  
Laïla Benhlila

## Table des matières

Introduction à Apache Cassandra .....	2
Architecture .....	2
L'écriture des données.....	2
Configuration d'Apache Cassandra .....	3
Installation d'Apache Cassandra .....	4
Prérequis .....	4
Installation .....	4
Exemple de manipulation dans Apache Cassandra.....	6
Créer un keyspace.....	6
Créer une table .....	6
Insérer dans une table .....	6
Interroger la base de données .....	7
L'indexation dans Apache Cassandra .....	8
Primary key .....	8
Partition key.....	8
Clustering key.....	9
Secondary index.....	9
Présentation technique de l'application .....	10
Situation.....	10
Modélisation des données.....	10
Modélisation par le modèle de Chebotko .....	10
Environnement de développement .....	11
Interfaces de l'application.....	11
Création de la base de données.....	12
Création du keyspace.....	12
Création des tables .....	12
Création des index .....	13
Insertion des données .....	13
Insertion des images de radiographies .....	14
La communication avec la base de données.....	14
L'interface graphique avec JavaFX .....	14
La manipulation des fichiers image .....	15
Un exemple de Controller .....	16
Accéder au projet.....	20
Conclusion .....	21

## Introduction à Apache Cassandra

---

Apache Cassandra est un système de gestion de base de données de type NoSQL orienté colonnes, conçu pour gérer des quantités massives de données sur un grand nombre de serveurs.

### Architecture<sup>1</sup>

L'architecture de Cassandra est constituée des composants principaux suivants :

- Node : la machine où les données sont stockées.
- Datacenter : un ensemble de nœuds.
- Cluster : un ensemble de datacenters.
- Keyspace : Un keyspace est plus ou moins l'équivalent d'une base de données dans le modèle relationnel. De la même manière qu'une base de données contient plusieurs tables dans le modèle relationnel, un keyspace contient plusieurs tables dans Cassandra.
- Commit log : log où les données sont écrites pour la récupération, avant d'être écrites dans les SSTable.
- SSTable (Sorted String Table) est un fichier immuable, fonctionnant en mode ajouter seulement, permettant ainsi aux memtables (des tables temporaires dans la mémoire cache) d'écrire périodiquement sur ce fichier. Il s'agit de l'implémentation physique de la table.
- CQL Table : une collection ordonnée de colonnes possédant une clé primaire. C'est la représentation logique de la table.
- Colonne : La colonne est l'unité de données la plus basique dans le modèle de données Cassandra. Elle possède un nom et une valeur. La structure des données est flexible : il n'est pas nécessaire de spécifier des valeurs pour toutes les colonnes de la table lors de l'insertion des données. Il est également possible d'ajouter une colonne après la création de la table.

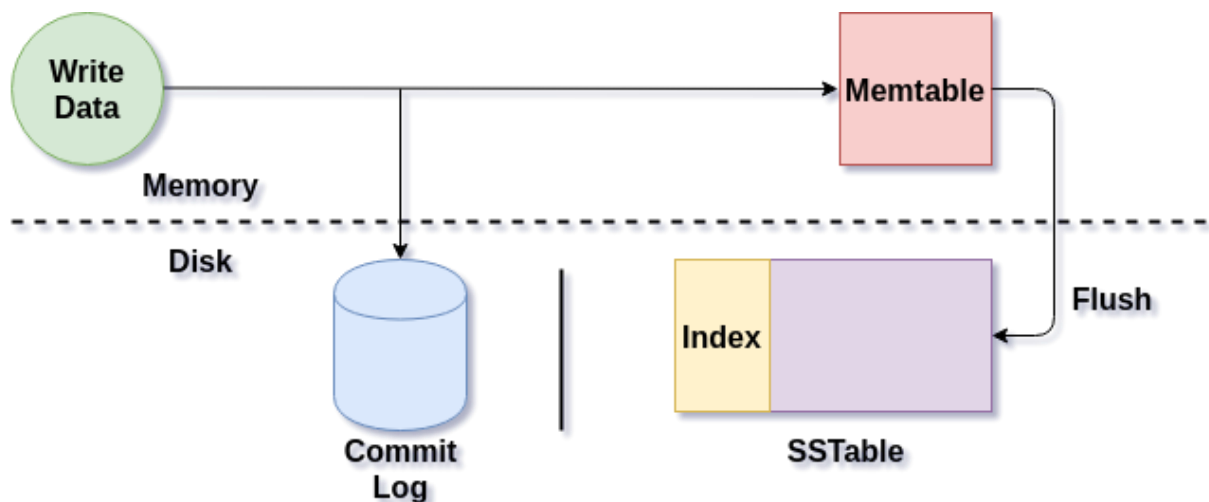
Cassandra possède une architecture peer-to-peer, c'est-à-dire que les données et leurs répliques sont distribuées de manière homogène sur les différents nœuds. Cela la distingue du modèle master-slave, adopté par HDFS et HBase par exemple. Chaque nœud du cluster échange son état avec ses pairs en utilisant un protocole de communication appelé *gossip*.

### L'écriture des données

Les données sont d'abord écrites dans un fichier log appelé Commit Log. Elles sont ensuite écrites dans un memtable – une table temporaire dans la mémoire cache. Une fois la mémoire saturée, les données sont persistées dans un fichier dit SSTable.

---

<sup>1</sup> Référence : <https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archTOC.html>

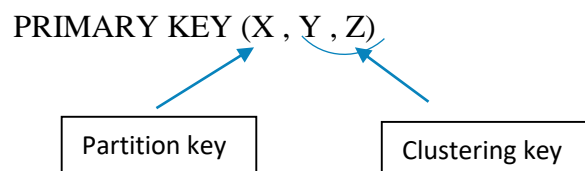


La procédure d'écriture des données dans Apache Cassandra

Source: <http://sudotutorials.com/tutorials/cassandra/how-data-is-written-to-cassandra.html>

## Configuration d'Apache Cassandra

Lors d'une requête Cassandra, la clé primaire est un élément essentiel puisqu'elle permet de connaître l'emplacement d'une partition au sein d'un cluster. En effet, la clé primaire est utilisée pour le partitionnement d'une table CQL et permet d'identifier une ligne de façon unique. Elle est définie comme suit:



Ainsi, pour configurer Cassandra, il faut prendre en compte plusieurs éléments:

- Le *partitioner* : c'est le composant responsable de distribuer les données dans un cluster.
- Le facteur de réplcation : le nombre de copies pour chaque ligne dans un cluster.
- La stratégie de réplcation : comment les réplcations seront distribuées dans le cluster (Network Topology Strategy est la stratégie la plus recommandée).

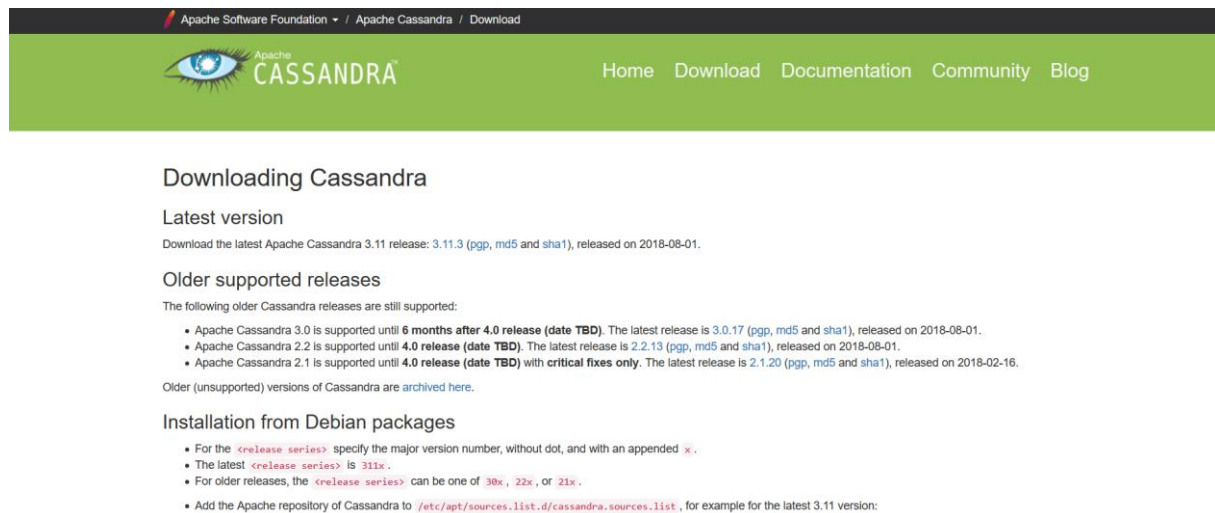
# Installation d'Apache Cassandra

## Prérequis

- Windows OS
- JDK doit être installé
- Python 2.7 doit être installé

## Installation

- Accéder au site web <http://cassandra.apache.org/download/>



Apache Software Foundation / Apache Cassandra / Download

Home Download Documentation Community Blog

### Downloading Cassandra

**Latest version**  
Download the latest Apache Cassandra 3.11 release: 3.11.3 (pgp, md5 and sha1), released on 2018-08-01.

**Older supported releases**  
The following older Cassandra releases are still supported:

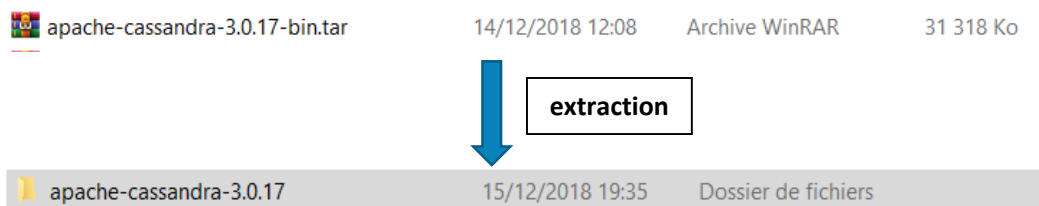
- Apache Cassandra 3.0 is supported until **6 months after 4.0 release (date TBD)**. The latest release is 3.0.17 (pgp, md5 and sha1), released on 2018-08-01.
- Apache Cassandra 2.2 is supported until **4.0 release (date TBD)**. The latest release is 2.2.13 (pgp, md5 and sha1), released on 2018-08-01.
- Apache Cassandra 2.1 is supported until **4.0 release (date TBD)** with **critical fixes only**. The latest release is 2.1.20 (pgp, md5 and sha1), released on 2018-02-16.

Older (unsupported) versions of Cassandra are [archived here](#).

**Installation from Debian packages**

- For the `<release series>` specify the major version number, without dot, and with an appended `x`.
- The latest `<release series>` is `311x`.
- For older releases, the `<release series>` can be one of `30x`, `22x`, or `21x`.
- Add the Apache repository of Cassandra to `/etc/apt/sources.list.d/cassandra.sources.list`, for example for the latest 3.11 version:

- Télécharger la dernière version de Cassandra.
- Extraire le document téléchargé.



apache-cassandra-3.0.17-bin.tar 14/12/2018 12:08 Archive WinRAR 31 318 Ko

extraction

apache-cassandra-3.0.17 15/12/2018 19:35 Dossier de fichiers

- Accéder au dossier bin (emplacementdudossier\apache-cassandra-3.0.17\bin)

cassandra	25/07/2018 07:32	Fichier	11 Ko
cassandra	15/12/2018 19:33	Fichier de comma...	7 Ko
cassandra.in	25/07/2018 07:32	Fichier de comma...	4 Ko
cassandra.in	25/07/2018 07:32	Shell Script	3 Ko
cassandra	25/07/2018 07:32	Script Windows Po...	13 Ko
cqlsh	25/07/2018 07:32	Fichier	2 Ko
cqlsh	25/07/2018 07:32	Fichier de comma...	2 Ko
cqlsh	25/07/2018 07:32	Python File	104 Ko
debug-cql	25/07/2018 07:32	Fichier	3 Ko
debug-cql	25/07/2018 07:32	Fichier de comma...	2 Ko
nodetool	25/07/2018 07:32	Fichier	4 Ko
nodetool	25/07/2018 07:32	Fichier de comma...	2 Ko
source-conf	25/07/2018 07:32	Script Windows Po...	2 Ko
sstableloader	25/07/2018 07:32	Fichier	2 Ko
sstableloader	25/07/2018 07:32	Fichier de comma...	2 Ko
sstablescrub	25/07/2018 07:32	Fichier	2 Ko

- Ouvrir cassandra.bat avec le bloc-notes et ajouter l'emplacement du JDK dans l'emplacement souligné.

```
goto runLegacy

REM -----
:runPowerShell
echo Detected powershell execution permissions. Running with enhanced startup scripts.
set errorlevel=
powershell /file "%CASSANDRA_HOME%\bin\cassandra.ps1" %*
exit /b %errorlevel%

REM -----
:runLegacy
echo WARNING! Powershell script execution unavailable.
echo Please use 'powershell Set-ExecutionPolicy Unrestricted'
echo on this user-account to run cassandra with fully featured
echo functionality on this platform.

echo Starting with legacy startup options

if NOT DEFINED CASSANDRA_MAIN set CASSANDRA_MAIN=org.apache.cassandra.service.CassandraDaemon
if NOT DEFINED JAVA_HOME set JAVA_HOME=C:\Program Files\Java\jdk1.8.0_181

REM -----
REM JVM Opts we'll use in legacy run or installation
set JAVA_OPTS=-ea^
-javaagent:"%CASSANDRA_HOME%\lib\jamm-0.3.0.jar"^
-Xms2G^
-Xmx2G^
-XX:+HeapDumpOnOutOfMemoryError^
-XX:+UseParNewGC^
-XX:+UseConcMarkSweepGC^
-XX:+CMSParallelRemarkEnabled^
-XX:+CMSClassUnloadingEnabled^
```

- Enregistrer et fermer le bloc-notes.
- Exécuter « cassandra.bat » pour démarrer Cassandra (cliquer deux fois sur le fichier).
- Exécuter « cqlsh.bat » pour manipuler les données dans Cassandra à partir de la console (cliquer deux fois sur le fichier).

```
WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.0.17 | CQL spec 3.4.0 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing. Install to enable tab completion.
cqlsh>
```

- Maintenant vous pouvez exécuter des requêtes CQL à travers cette console.

## Exemple de manipulation dans Apache Cassandra

---

### Créer un keyspace

Un « Keyspace » est l'équivalent d'une « database » dans une Base de données relationnelle telle que Oracle. On peut créer un Keyspace avec la syntaxe suivante :

```
CREATE KEYSPACE "nom du KeySpace" WITH replication = {'class': 'nom  
de la stratégie de réplication utilisé', 'replication_factor' :  
'Nombre de réplication '};
```

Exemple :

```
CREATE KEYSPACE FirstKeySpace WITH replication = {'class':'SimpleStrategy','replication_factor':3};
```

### Créer une table

Pour créer une table, on utilise la syntaxe suivante :

```
CREATE TABLE tablename(  
    column1 name data type PRIMARYKEY,  
    column2 name data type,  
    column3 name data type.  
)
```

Exemple :

```
create table emp( emp_id int PRIMARY KEY , emp_name text);
```

### Insérer dans une table

Pour insérer dans une table, on utilise la syntaxe suivante :

```
INSERT INTO <tablename>  
(<column1 name>, <column2 name>....)  
VALUES (<value1>, <value2>....)  
USING <option>
```

Exemple :

```
INSERT INTO emp (emp_id, emp_name) VALUES (1,'Boukham Houda');
```

## Interroger la base de données

Exemple :

```
select * from emp;
```

Le résultat est le suivant :

emp_id	emp_name
1	Boukham Houda



## L'indexation dans Apache Cassandra<sup>2</sup>

### Primary key

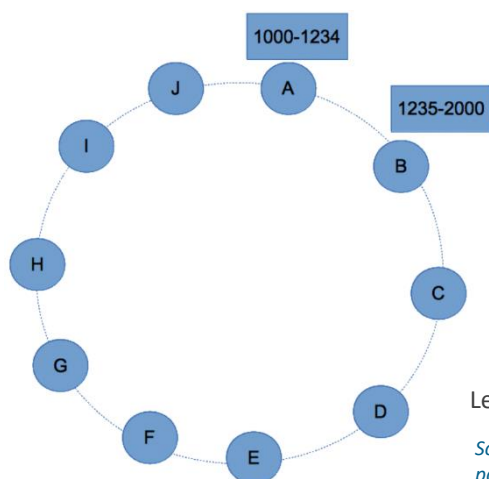
Chaque ligne est identifiée par une clé primaire ou primary key, dite aussi row key. Cette clé peut être composée, auquel cas le 1<sup>er</sup> composant constitue le partition key, et le reste le clustering key.

Différents cas:

Primary key	Partition key	Clustering key
(id)	id	aucun
(id, nom)	id	nom
(id, nom, prenom)	id	nom, prenom
(id, (nom, prenom))	id	nom, prenom
((id, nom, prenom), (date_naissance, lieu_naissance))	id, nom, prenom	date_naissance, lieu_naissance

### Partition key

Le partition key permet de retrouver le nœud dans le cluster où est stockée une ligne donnée. Lorsqu'une donnée est lue ou écrite, une fonction dite The Partionner associe à son partition key un hash code. C'est ce code-là qui permet de déterminer le nœud qui contient la donnée. Considérons par exemple la figure ci-dessous :



Le nœud A contient les lignes dont les partition keys sont compris entre 1000 et 1234, le neoud B ceux compris entre 1235 et 2000. Si une ligne possède un partition key dont le hash code est de 1200, elle sera stockée sur le nœud A.

Le partitioning dans Apache Cassandra

Source : [https://dzone.com/articles/cassandra-data-modeling-primary-clustering-partiti?fbclid=IwAR0wwluGusn78ZtSnyJQJ46bXMlf\\_bgBb0s2S53hVbL5bWnnSQ0QWj5Fz4w](https://dzone.com/articles/cassandra-data-modeling-primary-clustering-partiti?fbclid=IwAR0wwluGusn78ZtSnyJQJ46bXMlf_bgBb0s2S53hVbL5bWnnSQ0QWj5Fz4w)

<sup>2</sup> Référence : [https://www.datastax.com/dev/blog/cassandra-native-secondary-index-deep-dive?fbclid=IwAR01NNLbHoK9Eri\\_QntdrOB2r7ixlu01DvwuDKRXF5\\_OR9KdVqnzYtIsn1s](https://www.datastax.com/dev/blog/cassandra-native-secondary-index-deep-dive?fbclid=IwAR01NNLbHoK9Eri_QntdrOB2r7ixlu01DvwuDKRXF5_OR9KdVqnzYtIsn1s)

## Clustering key

Le clustering key permet de trier les données d'une même partition, afin de pouvoir récupérer les données de manière plus efficace.

## Secondary index

Dans Apache Cassandra, l'index – dit Secondary Index – permet d'accéder à des colonnes qui ne font pas partie de la clé primaire.

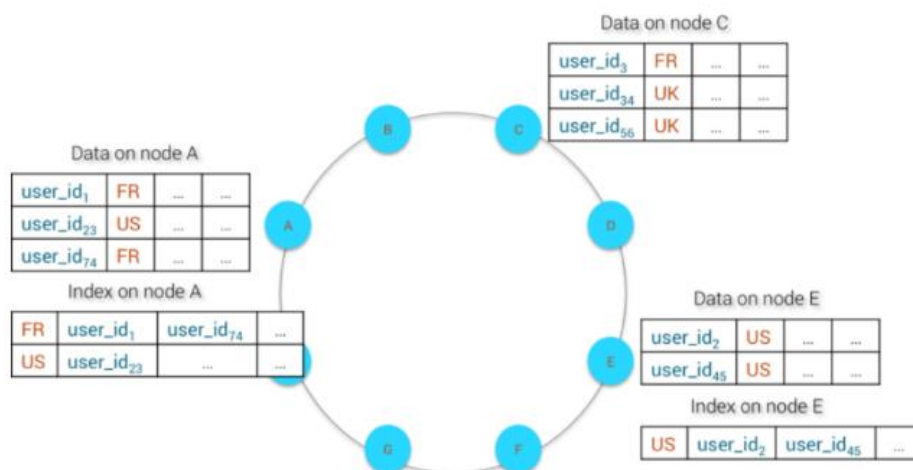
En effet, si l'on essaie de faire une requête sur des données autre que la clé primaire, le nom de patient par exemple, le message d'erreur suivant apparaît : `InvalidRequest: code=2200 [Invalid query] message="No supported secondary index found for the non primary key columns restrictions"`.

La solution serait d'ajouter un index secondaire sur la colonne nom. Pour cela, il faudrait écrire :

```
cqlsh:gestion_medicale> CREATE INDEX ON patient ( nom );
```

A présent, l'on pourrait effectuer la requête suivante :

```
cqlsh:my_keyspace> SELECT * FROM patient WHERE nom = 'Smith';
```



Source : [https://www.datastax.com/dev/blog/cassandra-native-secondary-index-deep-dive?fbclid=IwAR01NNLbHoK9Eri\\_QntdrOB2r7ixlu01DvwuDKRXF5\\_OR9KdVqnzYtIsn1s](https://www.datastax.com/dev/blog/cassandra-native-secondary-index-deep-dive?fbclid=IwAR01NNLbHoK9Eri_QntdrOB2r7ixlu01DvwuDKRXF5_OR9KdVqnzYtIsn1s)

Considérons la situation suivante, représentée dans la figure ci-dessus. Nous avons une table `users`, composée d'une clé primaire `user_id`, et d'une colonne `country`. Pour pouvoir effectuer une recherche sur `country`, nous devons ajouter un index secondaire sur cette colonne. L'index sera alors une table cachée possédant cette structure :

```
CREATE TABLE country_index(  
    country text,  
    user_id bigint,  
    PRIMARY KEY((country), user_id)  
);
```

L'index est stocké dans le même nœud que la table, et est donc lui aussi distribué.

Note : Bien que Cassandra soit orienté colonne, la recherche s'y fait par ligne.

## Présentation technique de l'application

### Situation

Notre application doit permettre la gestion des patients, de leurs maladies et des radiographies associées à celles-ci.

Les données à modéliser sont les suivantes :

Patients : nom, prénom, date naissance,

Maladies du patient : médecin traitant, nom maladie, symptômes.

Radios du patient : image, date radio, médecin ayant demandé la radio.

### Modélisation des données

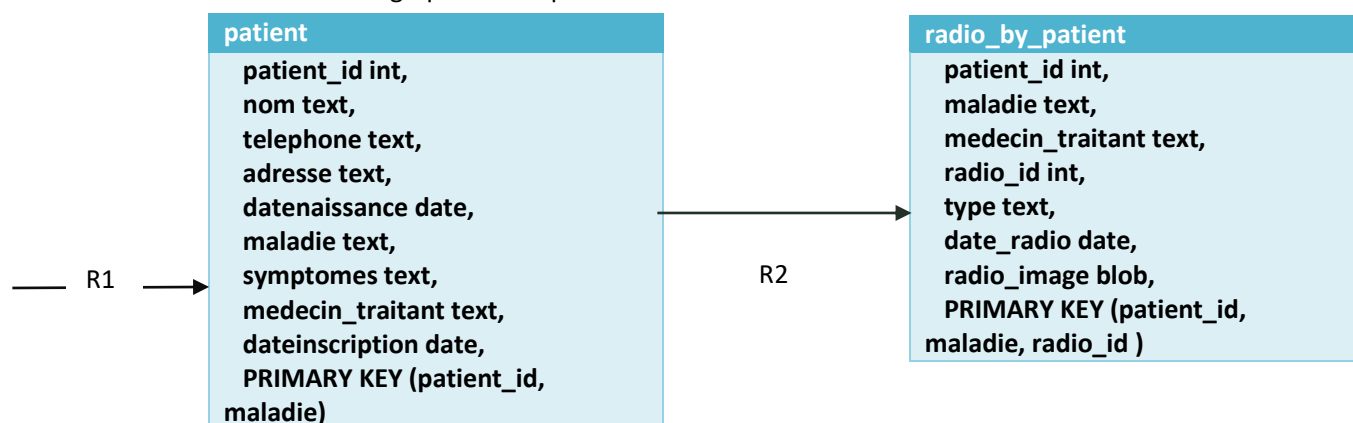
Les soucis de modélisation diffèrent entre les bases de données relationnelles et les bases de données NoSQL. Alors que dans les premières, la normalisation était un souci primordial, dans la seconde, la dénormalisation est « normale ». Il n'importe pas autant d'éviter les redondances que d'optimiser la performance, par exemple. Comme les jointures n'existent pas dans Apache Cassandra, il sert toujours de réunir les données qui vont ensemble dans une même table.

Modélisation par le modèle de Chebotko<sup>3</sup>

La communauté Cassandra a proposé plusieurs notations pour modéliser les données sous forme d'un diagramme. L'une de ces notations est le modèle Chebotko représenté ci-dessous.

R1. Afficher les informations sur un patient.

R2. Afficher les radiographies d'un patient.



<sup>3</sup> Référence : <https://www.oreilly.com/ideas/cassandra-data-modeling>

Le patient est identifié par un `patient_id`. Il est représenté par son nom, son numéro de téléphone, son adresse et sa date de naissance. Un patient peut être traité pour une ou plusieurs maladies. Pour chaque maladie, le patient est traité par un médecin.

Les radiographies d'un patient sont liées à sa maladie. Une radiographie est définie par un identifiant `radio_id`, un type (scanner, cliché pulmonaire, échographie...), la date à laquelle elle a été prise, le médecin l'ayant demandé, et la maladie pour laquelle elle a été demandée. Le fichier image est lui-même stocké dans la table des radiographies.

## Environnement de développement

Notre projet est conçu sous forme d'une application desktop, développée sous Java et utilisant le framework JavaFX. La connexion à la base de données est assurée par le driver Datastax (disponible sur <https://docs.datastax.com/en/developer/java-driver/3.0/>). Nous avons employé Scene Builder pour la construction de l'interface graphique.

## Interfaces de l'application

L'application comporte deux interfaces.

La première interface permet de sélectionner un patient et d'afficher ses informations.

The screenshot shows a desktop application window titled 'Gestion médicale' with a sub-header 'Gestion des patients'. It features a patient selection dropdown, date range pickers, a patient information form, a table of medical history, and a button to view radiographs. Four blue callout boxes with numbers 1 through 4 provide instructions on how to use the interface.

**1 Choisir le patient à afficher**

Sélectionnez un patient: John Smith

Choisissez un intervalle: Du 01/02/2018 Au 22/02/2018 Go

**2 Choisir un intervalle où seront comprises les dates d'ouverture de dossier**

Nom: John Smith  
Date de naissance: 1989-05-13  
Numéro de téléphone: 0624269011  
Adresse: Rabat

Maladie	Médecin traitant	Date d'ouverture du dossier	Symptômes
Grippe	Jane Doe	2018-02-08	Fatigue, nez qui coule
TB	Jane Doe	2018-02-14	Ganglion enflé, dur, indolore

**3 Consulter la liste des maladies pour lesquelles le patient a été traité**

Consulter les radiographies

**4 Consulter les radiographies du patient sélectionné**

En cliquant sur le bouton « Consulter les radiographies », l'utilisateur est dirigé vers la deuxième interface. Celle-ci permet en effet de consulter la liste des radiographies effectuées par le patient sélectionné.

[illegible]

## Création de la base de données

## Création du keyspace

Pour créer notre base de données, nous créons un `keyspace`. Nous devons spécifier deux propriétés : la propriété `class` qui décide de la stratégie de réplication à utiliser (nous utilisons `SimpleStrategy`, qui spécifie un facteur de réplication simple pour le cluster), et la propriété `repliation_factor`, qui spécifie le nombre de répliquions.

```
CREATE KEYSPACE gestionMedicale
WITH replication = {'class':'SimpleStrategy', 'replication_factor' :
3};
```

## Création des tables

Nous créons ensuite nos tables, patient et patient\_id (voir [Modélisation](#)).

```
CREATE TABLE patient (
```

```
patient_id int,  
nom text,  
telephone text,  
adresse text,  
datenaissance date,  
maladie text,  
symptomes text,  
medecin_traitant text,  
dateinscription date,  
PRIMARY KEY (patient_id, maladie) );
```

```
CREATE TABLE radio_by_patient(  
patient_id int,  
maladie text,  
medecin_traitant text,  
radio_id int,  
type text,  
date_radio date,  
radio_image blob,  
PRIMARY KEY (patient_id, maladie, radio_id ));
```

## Création des index

Pour pouvoir effectuer la recherche par date et par nom du patient (des colonnes qui ne font pas partie de la clé primaire, voir [Secondary Index](#)), nous créons les index suivants :

```
CREATE INDEX recherche_dateins ON patient (dateinscription);  
CREATE INDEX recherche_nom ON patient (nom);  
CREATE INDEX recherche_dateradio ON radio_by_patient (date_radio);
```

## Insertion des données

Nous avons inséré quelques patients à noms génériques dans notre base de données.

```
INSERT INTO patient (patient_id, maladie, nom, telephone, adresse,  
datenaissance, medecin_traitant, dateinscription, symptomes) VALUES (1,  
'TB', 'John Smith', '0624269011', 'Rabat', '1989-05-13', 'Jane Doe',  
'2018-02-14', 'Ganglion enfle, dur, indolore');
```

## Insertion des images de radiographies

La fonction ci-dessous permet d'effectuer le stockage des fichiers image dans la base de données. Pour les images des radiographies, nous avons utilisé des sites stock qui offrent des images gratuites pour réutilisation à but non commercial.

```
public static void insertRadios(String image) throws IOException {
    //Nous creons un flux de donnees qui permet de recuperer l'image
    FileInputStream fis=new FileInputStream(image);
    //Nous recuperons la taille de l'image
    int lengthImage = fis.available()+1;
    //Nous creons un tableau de bytes a partir de la taille de l'image
    byte[] b= new byte[lengthImage];
    //Nous stockons l'image dans le tableau precedent
    fis.read(b);
    //Nous creons un buffer qui permet de transformer le tableau en format
    //binaire
    ByteBuffer buffer =ByteBuffer.wrap(b);
    //Nous effectuons la requete qui permet d'injecter l'image en format
    //binaire dans la colonne de type blob de la table radios_by_patient
    PreparedStatement ps = session.prepare("update radio_by_patient set
radio_image = ? where patient_id = ? AND maladie = ? AND radio_id = ?;");
    BoundStatement boundStatement = new BoundStatement(ps);
    session.execute( boundStatement.bind( buffer ,5,"Intoxication",51));
}
```

## La communication avec la base de données

Le Java driver de Datastax pour Apache Cassandra rend possibles la communication entre notre application et notre base de données. Il suffit de créer un cluster, une session, de déterminer l'adresse et le port sur lesquels Cassandra est configuré, et d'introduire la requête à exécuter.

```
cluster =
Cluster.builder().addContactPoint("127.0.0.1").WithPort(9042).build();
//connexion à la base de données
session = cluster.connect("gestionmedicale");
//connexion au keyspace gestionmedicale
ResultSet results = session.execute("Select * from patient");
//execution d'une requête CQL et enregistrement du résultat dans un
ResultSet
Cluster.close();
//fermer la connexion
```

## L'interface graphique avec JavaFX

Pour la conception de l'interface graphique, nous avons utilisé JavaFX (intégré dans l'IDE IntelliJ).

Les interfaces JavaFX sont définies dans un fichier basé sur XML dit FXML. Par exemple, le fichier FXML suivant correspond à une page contenant un texte Welcome.

```
<GridPane fx:controller="fxmlexample.FXMLExampleController"
xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10" vgap="10">
```

```
<padding><Insets top="25" right="25" bottom="10" left="25"/></padding>
    <Text fx:id="title" text="Welcome" GridPane.columnIndex="0"
    GridPane.rowIndex="0" GridPane.columnSpan="2"/>
</GridPane>
```

Nous avons utilisé le Scene Builder pour la construction de l'interface. Scene Builder est un outil qui facilite la conception d'interfaces graphiques, dont le fonctionnement est basé sur le Drag and Drop. (Pour l'utiliser avec l'IDE, il faut le télécharger depuis <https://gluonhq.com/products/scene-builder/>, puis indiquer à l'IDE le lien vers l'exécutable Scene Builder).

Pour lier l'interface graphique à l'application, notre classe Main doit étendre la classe Application et avoir l'emplacement du fichier FXML, ainsi que le nom de sa balise racine.

```
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception{
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(new URL("file:/D:\\IntelliJ Projects\\Gestion
Medicale - Cassandra\\src\\main\\java\\sample\\sample.fxml"));
        //emplacement de notre fichier FXML
        GridPane root = loader.<GridPane>load();
        primaryStage.setTitle("Gestion médicale"); //titre de l'interface
        primaryStage.setScene(new Scene(root));
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args); //la méthode launch() démarre l'interface graphique
    }
}
```

Nous injectons les objets du Scene Builder (définis dans le fichier dans le fichier FXML) dans notre code en utilisant @FXML. Par exemple, pour injecter notre élément graphique de texte ayant l'id title dans notre code, nous écrivons :

```
@FXML
public Text title;
```

## La manipulation des fichiers image

Les fichiers image des radiographies sont stockés sous format Blob sur la base de données.

L'image est récupérée sous format Blob et est ensuite transformée en un tableau de bytes. Nous créons un flux streaming qui permet d'injecter ce tableau de bytes dans un objet ImageView (un élément graphique qui permet d'afficher les images sur JavaFX).

Nous injectons notre objet graphique Image dans notre code :



```
@FXML  
public ImageView image;
```

Ensuite, nous insérons le code suivant dans la fonction responsable d’afficher les images des radiographies (la fonction `oncbselection()` définie dans le `Controller2`, qui est déclenchée par la sélection d’une ligne du combobox).

```
byte image[] = Bytes.getBytes(bImage);  
fis = new ByteArrayInputStream(image);  
image.setImage(new Image(fis);
```

## Un exemple de Controller

Le fonctionnement de l’application est défini dans les classes `Controller`. Nous avons deux classes `Controller` : un pour chaque interface. C’est là que nous retrouvons les fonctionnements qui remplissent les tableaux et les combobox de l’application, et qui définissent les fonctions déclenchées par un événement (clic sur un bouton, sélection d’une ligne dans le combobox).

Ci-dessous est le code du `Controller` de la première interface graphique, donnant les informations relatives à un patient. La classe `Controller` implémente l’interface `Initializable` pour pouvoir utiliser la méthode `Initialize()`, invoquée implicitement avec la méthode `launch()`. Cette fonction définit les actions à exécuter au démarrage de l’application (remplissage du combobox à partir de la base de données).

```
public class Controller implements Initializable {  
  
    // variables utilisées pour la connection et le requetage  
    public static Cluster cluster;  
    public static Session session;  
    public static PreparedStatement prepared;  
    public static BoundStatement bound;  
    public static ResultSet results;  
  
    // variable determinant l'id du patient selectionné  
    public static int idpatient;  
  
    public Controller() {  
    }  
  
    // éléments graphiques  
    @FXML  
    public ComboBox<String> cb;  
  
    @FXML  
    public DatePicker dpmin;  
    @FXML  
    public DatePicker dpmax;  
  
    @FXML  
    public TableView<Patient> tv;  
    @FXML  
    public TableColumn<Patient, LocalDate> c_dateins;  
    @FXML  
    public TableColumn<Patient, String> c_maladie;
```

```
@FXML
public TableColumn<Patient, String> c_symptomes;
@FXML
public TableColumn<Patient, String> c_medecin;

@FXML
public Text t_nom;
@FXML
public Text t_datenaissance;
@FXML
public Text t_telephone;
@FXML
public Text t_adresse;

@FXML
public Button btn;
@FXML
public Button btnRadio;

//Cette fonction permet de selectionner un patient dans une liste
deroulante
@FXML
public void oncbselection() {
    // creation du query pour récupérer le patient ayant un nom donné

    prepared = session.prepare(
        "select * from patient where nom = ? ;");
    String nom_select = cb.getSelectionModel().getSelectedItem();
    bound = prepared.bind(nom_select);
    ResultSet rs = session.execute(bound);
    ObservableList<Patient> patientinfo =
FXCollections.observableArrayList();

    //iterer sur le resultat du query pour soustraire le patient

    for (Row row : rs) {
        patientinfo.add(new Patient(row.getInt("patient_id"),
row.getString("maladie"), row.getString("symptomes"),
row.getDate("dateinscription"), row.getString("medecin traitant")));
    }

    Row r = session.execute(bound).one();

    //populer les champs vides et le tableau avec les données du
patient

    idpatient = r.getInt("patient_id");
    t_nom.setText(r.getString("nom"));
    t_telephone.setText(r.getString("telephone"));
    t_adresse.setText(r.getString("adresse"));
    t_datenaissance.setText(r.getDate("datenaissance").toString());
    c_maladie.setCellValueFactory(new PropertyValueFactory<Patient,
String>("maladie"));
    c_symptomes.setCellValueFactory(new PropertyValueFactory<Patient,
```

```
String>("symptomes"));
    c_dateins.setCellValueFactory(new PropertyValueFactory<Patient,
LocalDate>("dateinscription"));
    c_medecin.setCellValueFactory(new PropertyValueFactory<Patient,
String>("medecin_traitant"));
    tv.getItems().setAll(patientinfo);
    btn.setDisable(false);
    btnRadio.setDisable(false);

}

//cette fonction se declenche lors de la recherche par date
d'inscription
@FXML
public void onbtnselection() {
    //creation de la requete pour la recherche par dateinscription

    prepared = session.prepare(
        "select * from patient where dateinscription > ? and
dateinscription < ? and nom = ? ALLOW FILTERING;");
    LocalDate datemin =
LocalDate.fromYearMonthDay(dpmin.getValue().getYear(),
dpmin.getValue().getMonthValue(), dpmin.getValue().getDayOfMonth());
    LocalDate datemax =
LocalDate.fromYearMonthDay(dpmax.getValue().getYear(),
dpmax.getValue().getMonthValue(), dpmax.getValue().getDayOfMonth());
    String nom_select = cb.getSelectionModel().getSelectedItem();
    bound = prepared.bind(datemin, datemax, nom_select);
    ResultSet rs = session.execute(bound);
    ObservableList<Patient> patientinfo =
FXCollections.observableArrayList();

    //iterer sur le resultat du query pour soustraire les consultation

    for (Row row : rs) {
        patientinfo.add(new Patient(row.getInt("patient_id"),
row.getString("maladie"), row.getString("symptomes"),
row.getDate("dateinscription"), row.getString("medecin_traitant")));
    }

    //populer le tableau par le resultat de la recherche

    c_maladie.setCellValueFactory(new PropertyValueFactory<Patient,
String>("maladie"));
    c_symptomes.setCellValueFactory(new PropertyValueFactory<Patient,
String>("symptomes"));
    c_dateins.setCellValueFactory(new PropertyValueFactory<Patient,
LocalDate>("dateinscription"));
    tv.getItems().setAll(patientinfo);
}

//lors du clic sur le bouton " consulter les radios "
@FXML
public void onradiobtnclick() throws Exception {
```

```
//fournir à a nouvelle page le nom du patient

Controller2.patient = idpatient;
Controller2.nompatient = cb.getSelectionModel().getSelectedItem();
    final Stage dialog = new Stage();
    dialog.initModality(Modality.APPLICATION_MODAL);
FXMLLoader loader = new FXMLLoader();
loader.setLocation(new URL("file:/D:\\IntelliJ Projects\\Gestion
Medicale - Cassandra\\src\\main\\java\\sample\\sample2.fxml"));
    AnchorPane root = loader.<AnchorPane>load();
    Scene dialogScene = new Scene(root);
    dialog.setScene(dialogScene);
    dialog.show();

}

// cette fonction est executée à l'ouverture de la fenetre
@FXML
public void initialize(URL location, ResourceBundle resources){
    //connection à cassandra
    cluster =
Cluster.builder().addContactPoint("127.0.0.1").withPort(9042).build();
    session = cluster.connect("gestionmedicale");

    //creation du query qui recupere le nom de tout les patients

    results = session.execute("select nom from patient group by
patient_id allow filtering;");
    ObservableList<String> data = FXCollections.observableArrayList();
    for (Row row : results) {
        data.add(row.getString("nom"));
    }
    cb.setItems(data);
    btn.setDisable(true);
    btnRadio.setDisable(true);

}

}
```

## Accéder au projet

Vous pouvez accéder au projet depuis GitHub sur le lien <https://github.com/BennisKhalil/ProjetCassandra>

The screenshot shows the GitHub repository page for 'BennisKhalil / ProjetCassandra'. The repository is titled 'Application de gestion des patients via apache cassandra'. It has 4 commits, 1 branch, 0 releases, and 1 contributor. The 'Code' tab is selected, showing a list of files: 'Gestion Medicale - Cassandra', 'README.md', 'Rapport - Bases de Données Complexes.pdf', and 'cassandra-demo.webm'. A 'Clone or download' button is visible. Callouts point to specific elements: 'Code source du projet' points to the repository name, 'Rapport du projet' points to the PDF file, 'Une vidéo demo' points to the webm file, and a larger callout on the right says 'Ici, vous pouvez télécharger le dossier contenant le projet, le rapport et une demo' pointing to the 'Clone or download' button.

Code source du projet

Rapport du projet

Une vidéo demo

Ici, vous pouvez télécharger le dossier contenant le projet, le rapport et une demo

A priori, le dossier est téléchargé en format ZIP. Dans un environnement Mac, il est possible d'y accéder par un double clic sur le fichier ZIP.

## Conclusion

---

Ce projet a servi de bonne introduction aux bases de données NoSQL en général, et à Cassandra Apache en particulier. Il nous aura permis de comprendre le principe des bases de données NoSQL orientées colonne, et de prendre connaissance des spécificités de Cassandra.

Cassandra, en effet, est intéressante, en cela qu'elle possède une architecture peer-to-peer, par opposé à l'architecture master-slave à laquelle nous sommes accoutumés. D'autant plus que ce choix d'architecture nous évite les Single Point of Failure, un problème récurrent dans plusieurs bases de données NoSQL, HBase à titre d'exemple.

Les bases de données NoSQL sont désormais moins intimidantes, et nous sommes motivés à en explorer d'autre, et à continuer notre découverte du domaine du Big Data.

Nous trouvons, somme toute, que les projets académiques de ce semestre – les derniers de notre formation – ont été bien instructifs et riches en valeur ajoutée, et c'est avec enthousiasme et motivation que nous débarquons en PFE.