

Université Mohammed V – Rabat

Ecole Mohammadia d’Ingénieurs

Département Génie Informatique

Ingénierie et Qualité Logicielles



Khalil Bennis

Houda Boukham

El Mokhtar Nourhira

Encadrés par Mme Laïla Benhlima

2018-2019

Rapport final

Bases de Données Complexes

Table des matières

[Introduction à Apache Cassandra 2](#_Toc535267391)

[Architecture 2](#_Toc535267392)

[L’écriture des données 2](#_Toc535267393)

[Configuration d’Apache Cassandra 3](#_Toc535267394)

[Installation d’Apache Cassandra 4](#_Toc535267395)

[Prérequis 4](#_Toc535267396)

[Installation 4](#_Toc535267397)

[Exemple de manipulation dans Apache Cassandra 6](#_Toc535267398)

[Créer un keyspace 6](#_Toc535267399)

[Créer une table 6](#_Toc535267400)

[Insérer dans une table 6](#_Toc535267401)

[Interroger la base de données 7](#_Toc535267402)

[L’indexation dans Apache Cassandra 8](#_Toc535267403)

[Primary key 8](#_Toc535267404)

[Partition key 8](#_Toc535267405)

[Clustering key 9](#_Toc535267406)

[Secondary index 9](#_Toc535267407)

[Présentation technique de l’application 10](#_Toc535267408)

[Situation 10](#_Toc535267409)

[Modélisation des données 10](#_Toc535267410)

[Modélisation par le modèle de Chebotko 10](#_Toc535267411)

[Environnement de développement 11](#_Toc535267412)

[Interfaces de l’application 11](#_Toc535267413)

[Création de la base de données 12](#_Toc535267414)

[Création du keyspace 12](#_Toc535267415)

[Création des tables 12](#_Toc535267416)

[Création des index 13](#_Toc535267417)

[Insertion des données 13](#_Toc535267418)

[Insertion des images de radiographies 14](#_Toc535267419)

[La communication avec la base de données 14](#_Toc535267420)

[L’interface graphique avec JavaFX 14](#_Toc535267421)

[La manipulation des fichiers image 15](#_Toc535267422)

[Un exemple de Controller 16](#_Toc535267423)

[Accéder au projet 20](#_Toc535267424)

[Conclusion 21](#_Toc535267425)

# 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[[1]](#footnote-1)

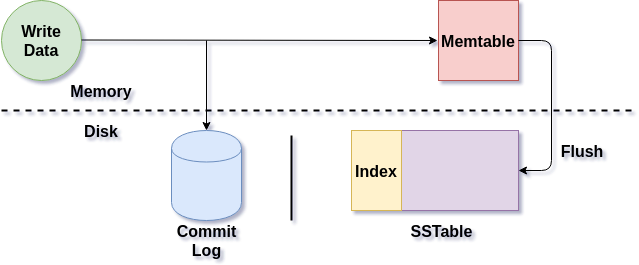
L’architecture de Cassandra est constituée des composants principaux suivants :

* 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 réplications 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.



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:

PRIMARY KEY (X , Y , Z)

Clustering key

Partition key

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 réplication : le nombre de copies pour chaque ligne dans un cluster.

• La stratégie de réplication : comment les réplications seront distribuées dans le cluster (Network Topology Strategy est la stratégie la plus recommandée).

# Installation d’Apache Cassandra

## Prérequis

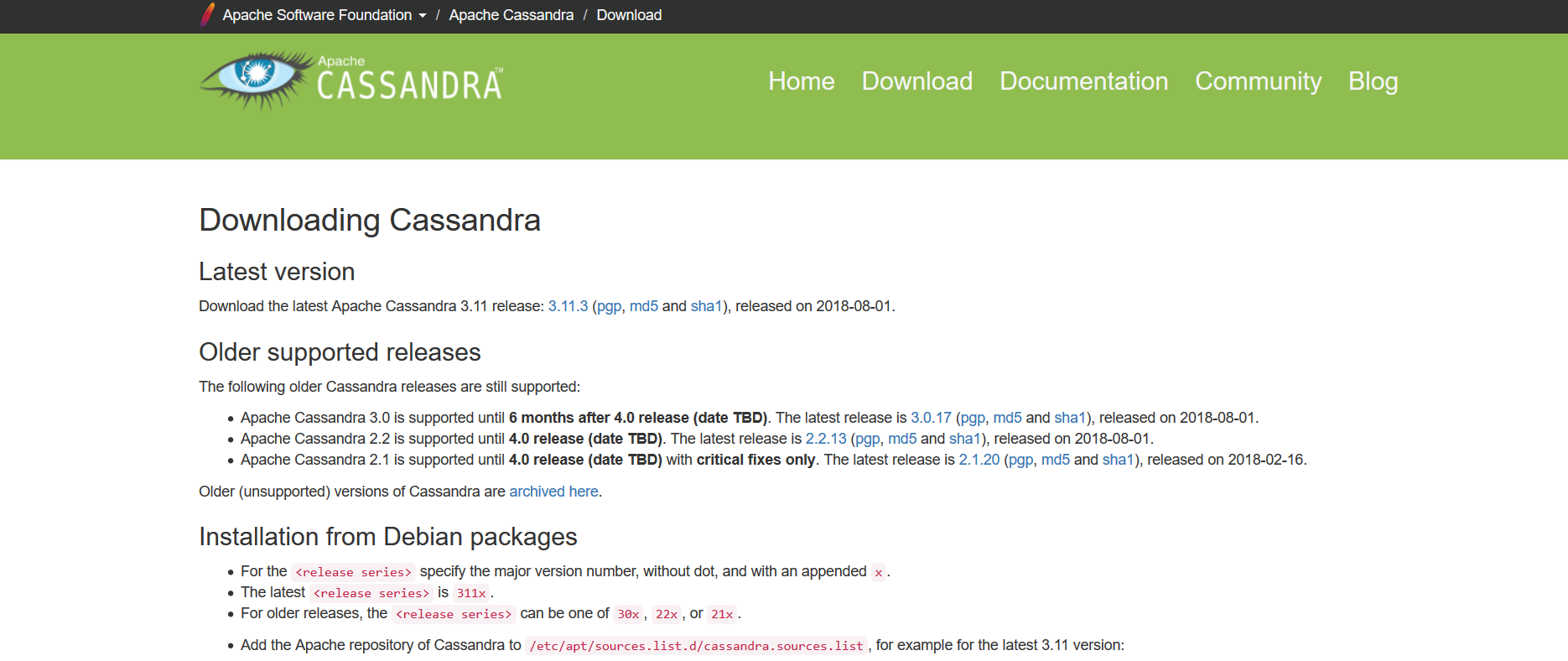
• Windows OS

• JDK doit être installé

• Python 2.7 doit être installé

## Installation

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

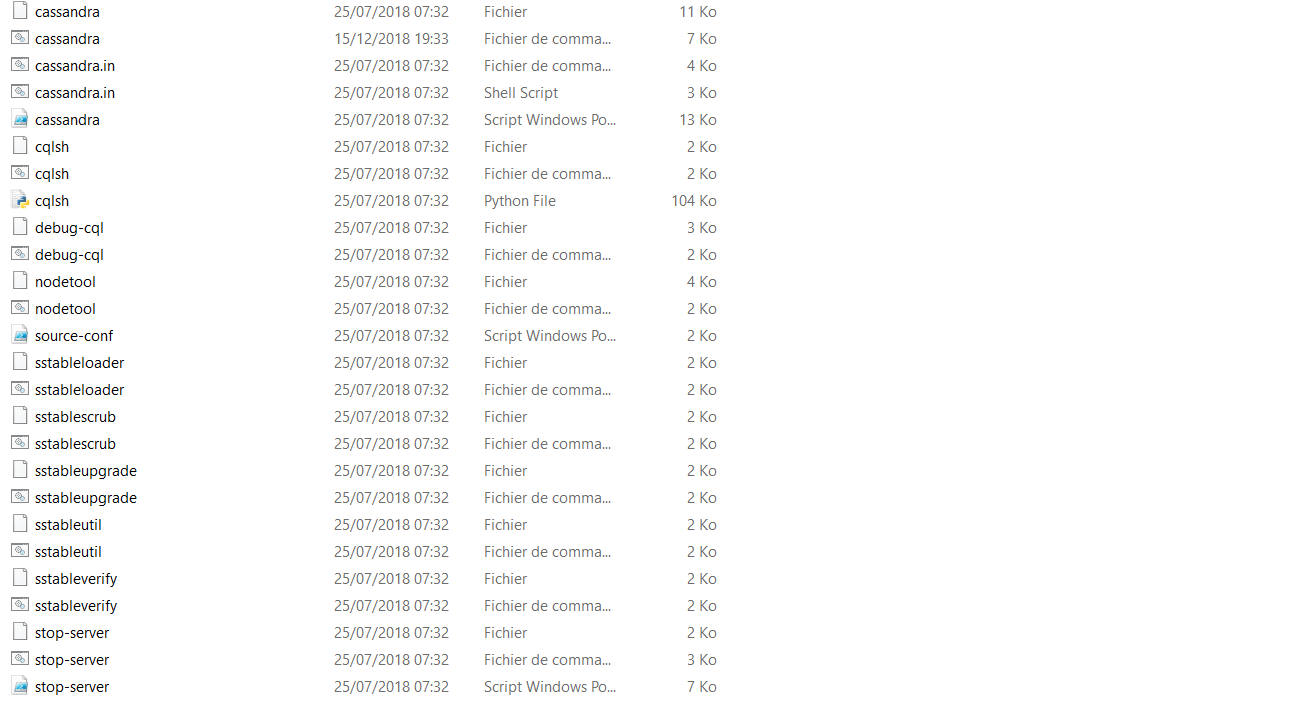


* Télécharger la dernière version de Cassandra.
* Extraire le document téléchargé.



**extraction**

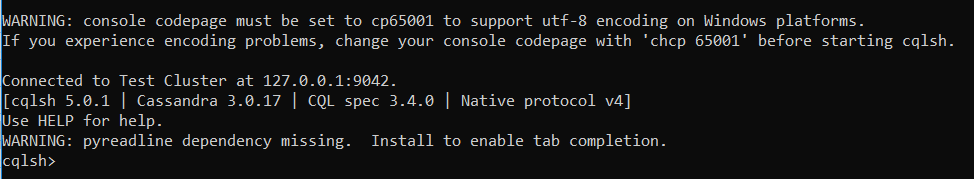
* Accéder au dossier bin (emplacementdudossier\apache-cassandra-3.0.17\bin)



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



* 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).



* 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 queOracle. 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 :



## 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 :



## 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 :

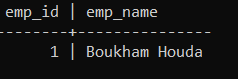


## Interroger la base de données

Exemple :



Le résultat est le suivant :



# L’indexation dans Apache Cassandra[[2]](#footnote-2)

## Primary key

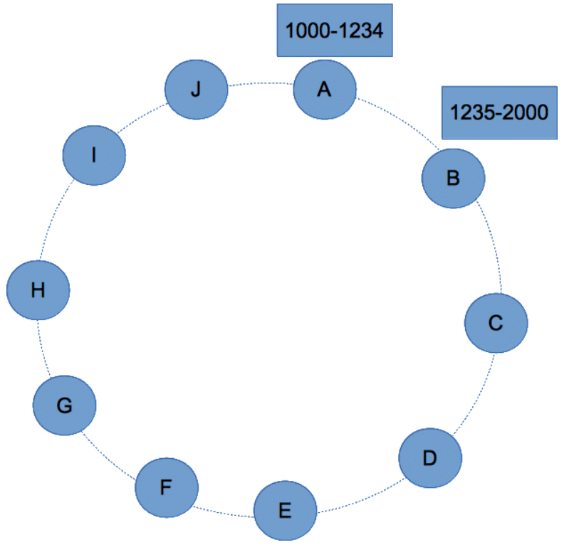
Chaque ligne est identifiée par une clé primaire ou primary key, dite aussi row key. Cette clé peut être composée, auquel cas le 1er composant constitue le partition key, et le reste le clustering key.

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=IwAR0wwIuGusn78ZtSnyJQJ46bXMIf\_bgBb0s2S53hVbL5bWnnSQ0QWj5Fz4w*

## 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.

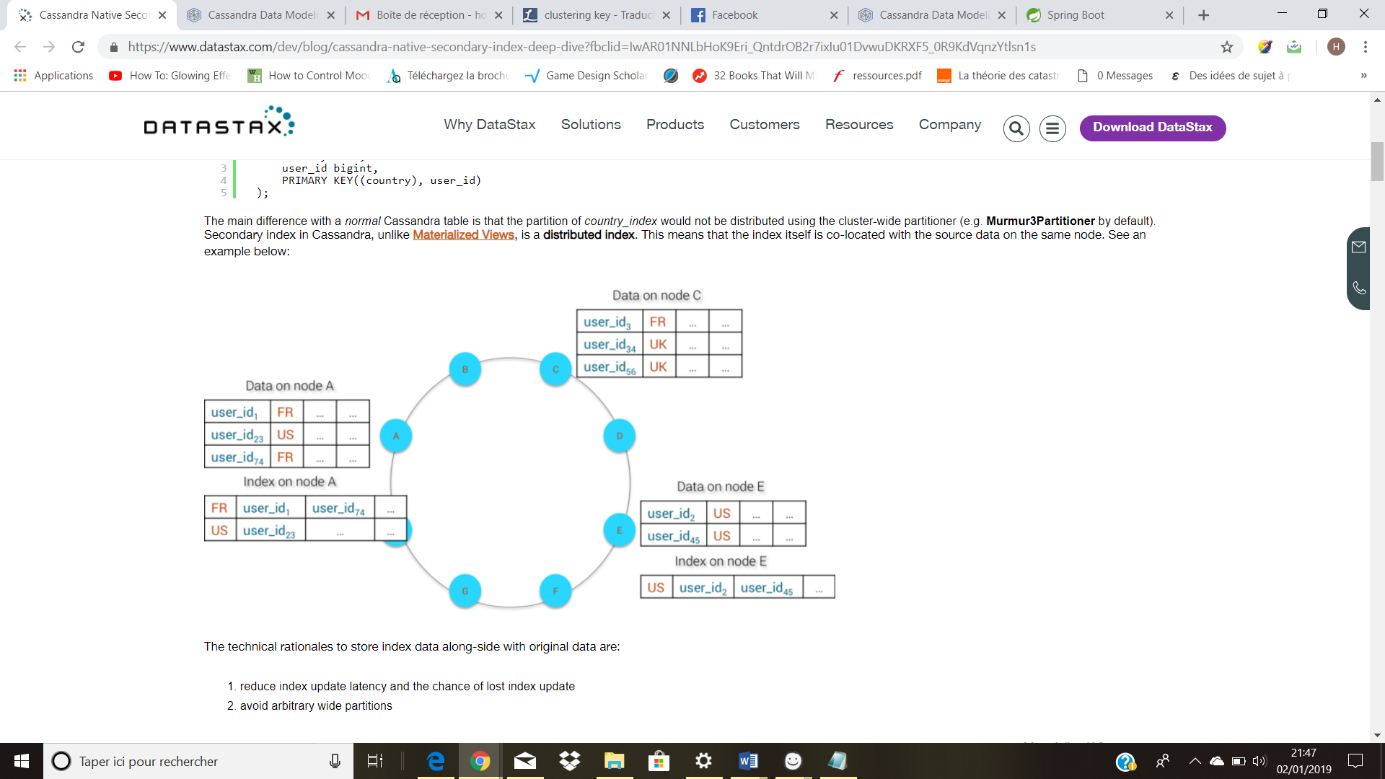
En effet, si l’on essaie de faire une requête sur des données autre que la clé primaire, le nom de patient par exemple, le message d’erreur suivant apparaît : InvalidRequest: code=2200 [Invalid query] message="No supported secondary index found for the non primary key columns restrictions".

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\_QntdrOB2r7ixIu01DvwuDKRXF5\_0R9KdVqnzYtlsn1s*

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 devrons 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[[3]](#footnote-3)

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.

|  |
| --- |
| 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 ) |

|  |
| --- |
| 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) |

R1

R2

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, écographie…), 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

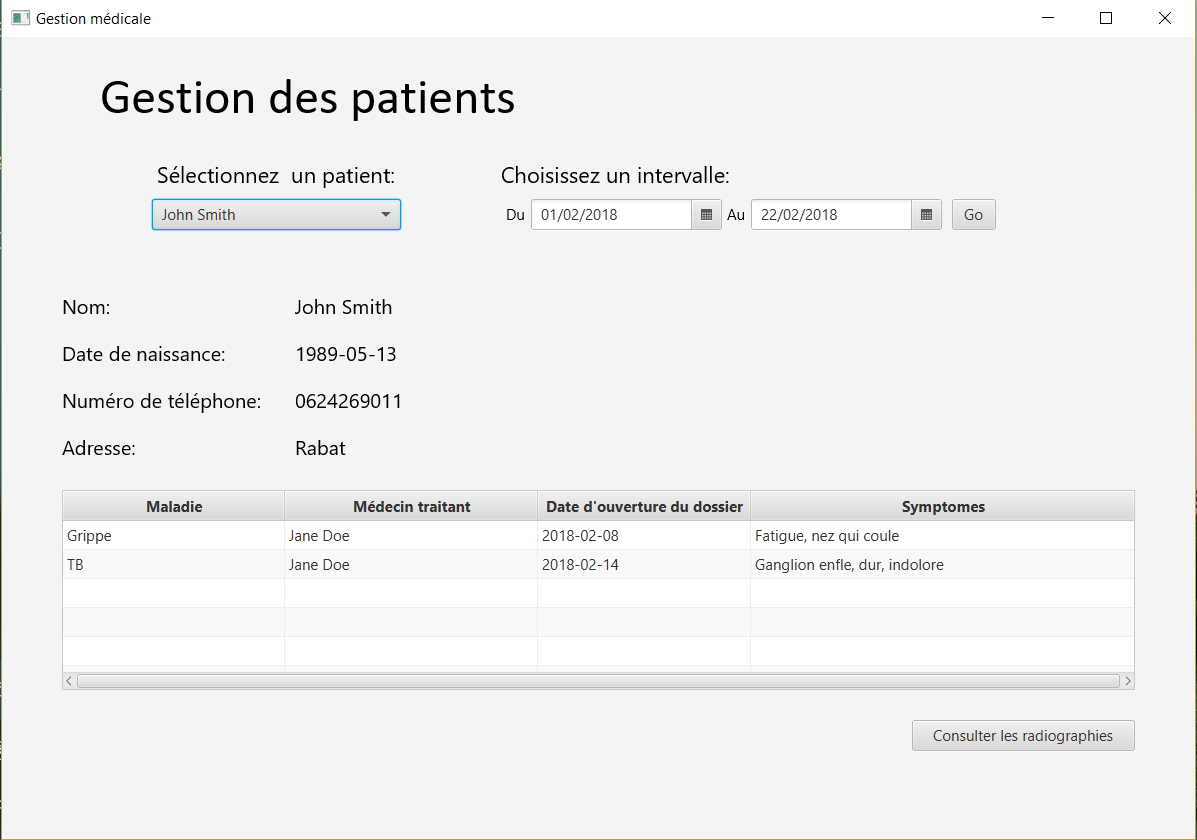
Notre projet est conçu sous forme d’une application desktop, développée sous Java et utilisant le framework JavaFX. La connexion à la base de données est assurée par le driver Datastax (disponible sur [*https://docs.datastax.com/en/developer/java-driver/3.0/*](https://docs.datastax.com/en/developer/java-driver/3.0/) ). 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.

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



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

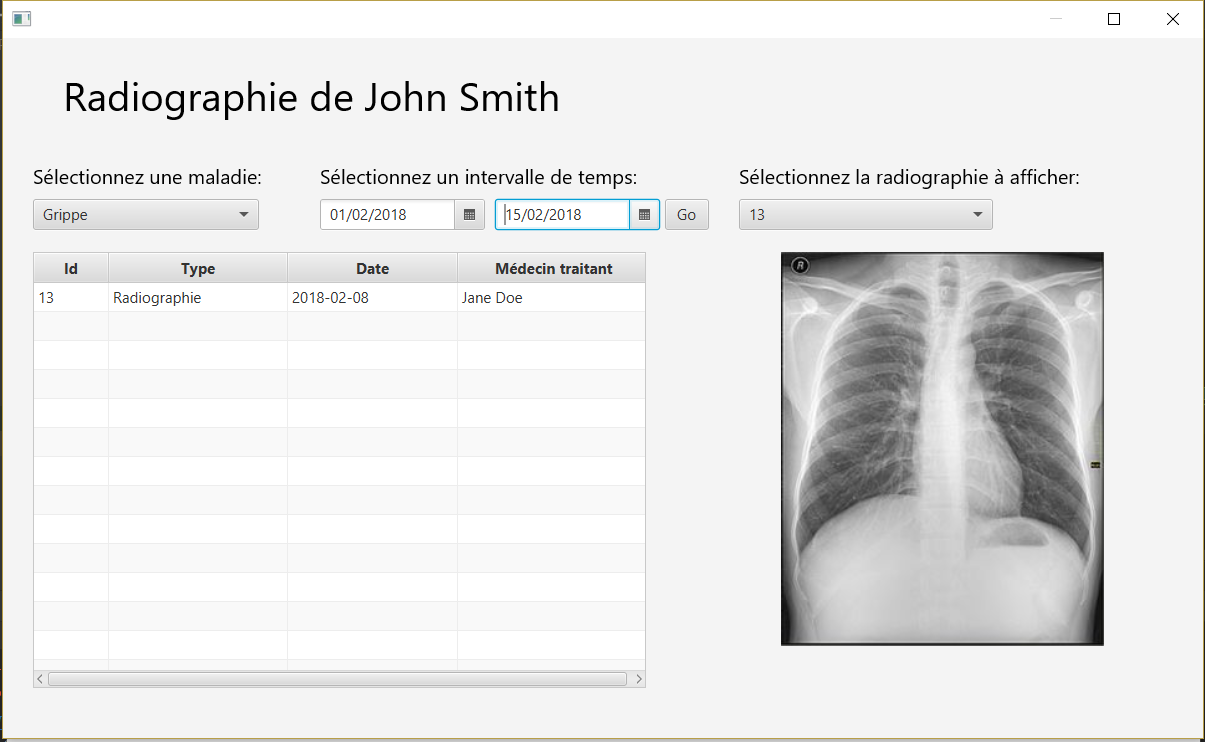
4 Consulter les radiographies du patient sélectionné

1 Choisir le patient à afficher

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.

6 Choisir un intervalle où seront comprises les dates de radiographie

5 Sélectionner une maladie par laquelle filtrer



8 Sélectionner l’identifiant de la radiographie dont on souhaite afficher l’image

7 Consulter l’identifiant de la radiographie dont on souhaite afficher l’image

## Création de la base de données

### Création du keyspace

Pour créer notre base de données, nous créons un [keyspace](#_Architecture). 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\_facteur, qui spécifie le nombre de réplications.

CREATE KEYSPACE gestionMedicale

WITH replication = {'class':'SimpleStrategy', 'replication\_factor' : 3};

### Création des tables

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

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](#_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 écrirons :

@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.*getArray*(bImage);  
fis = new ByteArrayInputStream(image);  
image.setImage(new Image(fis);

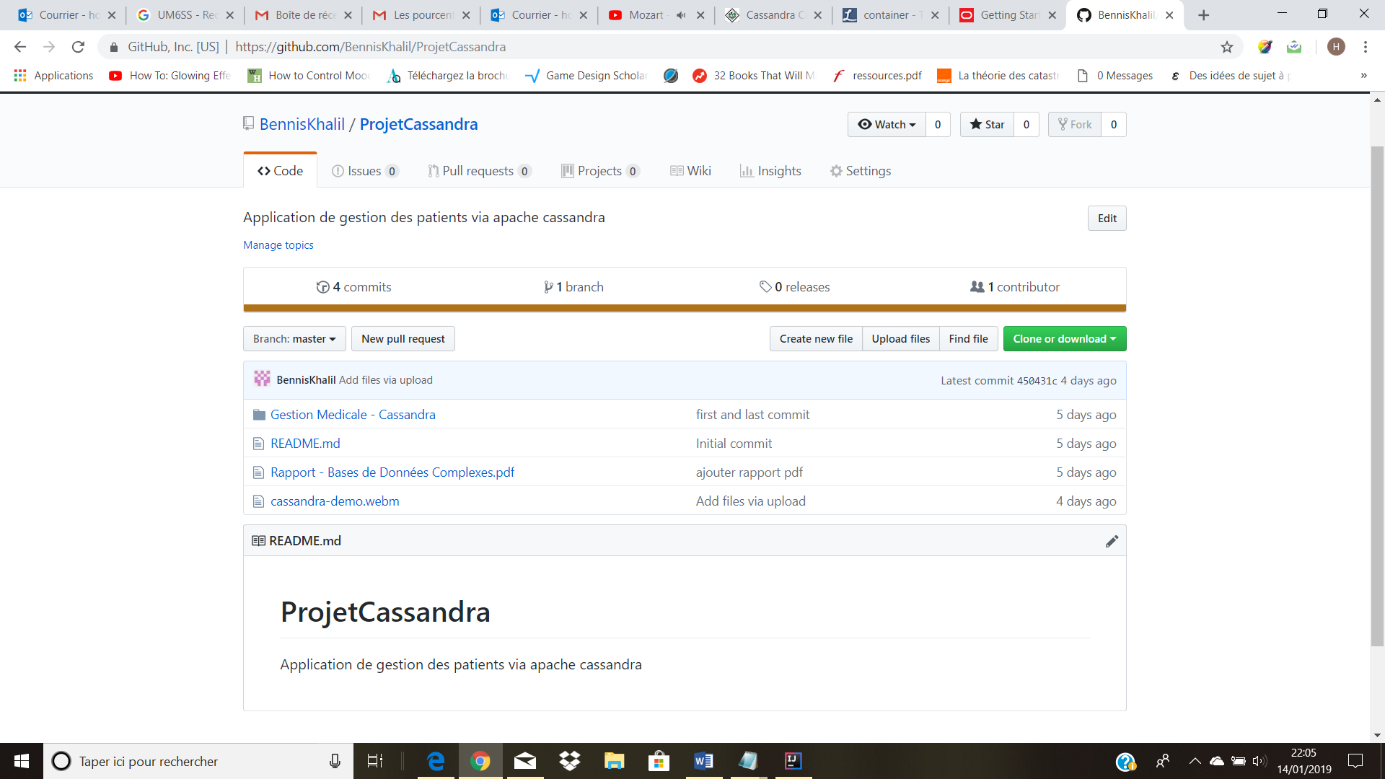
## Un exemple de Controller

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 fonctionnent 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 récuperer 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 boutton " 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>

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.

1. Référence : https://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archTOC.html [↑](#footnote-ref-1)
2. Référence : https://www.datastax.com/dev/blog/cassandra-native-secondary-index-deep-dive?fbclid=IwAR01NNLbHoK9Eri\_QntdrOB2r7ixIu01DvwuDKRXF5\_0R9KdVqnzYtlsn1s [↑](#footnote-ref-2)
3. Référence : https://www.oreilly.com/ideas/cassandra-data-modeling [↑](#footnote-ref-3)