Base de données NoSQL – TD 3
Cassandra

Master 2 BI&BD - 2021–2022 - 25/10/2021

1 Introduction

Cassandra est un système de gestion de données à grande échelle, conçu à l'origine (2007) par les ingénieurs de Facebook pour répondre à des problématiques liées au stockage et à l'utilisation de gros volumes de données. En 2008, ils essayèrent de le démocratiser en fournissant une version stable, documentée, disponible sur Google Code. Cependant, Cassandra ne reçut pas un accueil particulièrement enthousiaste. Les ingénieurs de Facebook décidèrent donc en 2009 de faire porter Cassandra par l'Apache Incubator. En 2010, Cassandra était promu au rang de top-level Apache Project.

Aujourd'hui, c'est la société Datastax qui assure la distribution et le support de Cassandra qui reste un projet Open Source de la fondation Apache.

Cassandra a beaucoup évolué depuis l'origine, ce qui explique une terminologie assez erratique qui peut prêter à confusion. L'inspiration initiale est le système BigTable de Google, et l'évolution a ensuite plutôt porté Cassandra vers un modèle proche du relationnel, avec quelques différences significatives, notamment sur les aspects internes. C'est un système NoSQL très utilisé, et sans doute un bon point de départ pour passer du relationnel à un système distribué.

2 Le modèle de données

Dans Cassandra, les données sont regroupées en familles de colonnes. Une famille de colonnes (column family) est plus ou moins l'équivalent d'une table dans le modèle relationnel, à la différence que son schéma n'est pas fixé à l'avance : chaque ligne peut avoir un nombre différent de colonnes. Chaque famille de colonnes est stockée dans son propre fichier sur le disque.

Une colonne est définie par son nom, qui n'est pas forcément une chaîne de caractères : le nom de colonne peut être d'un autre type, comme un entier ou un UUID, ce qui est pratique pour créer manuellement des index secondaires. La colonne contient une valeur et un horodatage (timestamp). La version 2 de Cassandra, sortie en 2014, a un peu dissimulé ce stockage interne. Au ressenti, un développeur Cassandra se retrouve comme devant des tables de SGBDR.

Cassandra est un SGBD qui s'est progressivement orienté vers un modèle relationnel étendu, avec un typage fort et un schéma contraint. Initialement, Cassandra était beaucoup plus permissif et permettait d'insérer à peu près n'importe quoi.

```
##Cassandra 2
CREATE COLUMNFAMILY livres (
       KEY bigint PRIMARY KEY,
       "auteur:prenom" varchar,
       "auteur:nom" varchar
);
INSERT INTO articles (KEY, "auteur:prenom", "auteur:nom", "auteur:email")
VALUES (1, 'Leo', 'Tolstoy', 'Leo.Tolstoy@mail.com');
```

```
##Cassandra 3
CREATE TABLE livres (
       auteur_id bigint,
       auteur_prenom" varchar,
       auteur_nom" varchar,
       PRIMARY KEY (auteur_id)
);
INSERT INTO articles (auteur_id, auteur_prenom, auteur_nom)
VALUES (1, 'Leo', 'Tolstoy');
```

Comme dans un système relationnel, une base de données Cassandra est constituée de tables. Chaque table a un nom et est constituée de colonnes. Toute ligne (row) de la table doit respecter le schéma de cette dernière. Si une table a 5 colonnes, alors à l'insertion d'une entrée, la donnée devra être composée de 5 valeurs respectant le typage. Une colonne peut avoir différents types :

— des types atomiques, par exemple entier, texte, date ;
— des types complexes (ensembles, listes, dictionnaires) ;
— des types construits et nommés.

Nous sommes effectivement proches d'un modèle de documents structurés de type JSON, avec imbrication de structures, mais avec un schéma qui assure le contrôle des données insérées. La gestion de la base est donc très contrainte et doit se faire en cohérence avec la structure de chaque table (son schéma). C'est une différence notable avec de nombreux systèmes NoSQL.

2.1 La paire clé-valeur (columns) et documents (rows)

La structure de base d'un document dans Cassandra est la paire (clé, valeur), autrement dit la structure atomique de représentation des informations semi-structurées, à la base de XML ou JSON par exemple. Une valeur peut être atomique (entier, chaîne de caractères) ou complexe (dictionnaire, liste).

2.1.1 Vocabulaire

Dans Cassandra, cette structure est parfois appelée colonne, ce qui est difficilement explicable au premier abord (vous êtes d'accord qu'une paire-clé/valeur n'est pas une colonne ?). Il s'agit en fait d'un héritage de l'inspiration initiale de Cassandra, le système BigTable de Google dans lequel les données sont stockées en colonnes. Même si l'organisa-

tion finale de Cassandra a évolué, le vocabulaire est resté. Bilan : chaque fois que vous lisez "colonne" dans le contexte Cassandra, comprenez "paire clé-valeur" et tout s'éclaircira.

Un document dans Cassandra est un identifiant unique associé à un ensemble de paires (clé, valeur). Cassandra appelle row les documents, et row key l'identifiant unique. La notion de ligne (row) vient également de BigTable.

2.2 Les tables (column families)

Les documents sont groupés dans des tables qui, sous Cassandra, sont parfois appelées des *`column families`* pour des raisons historiques.

2.3 Les bases de données (Keyspaces)

Le troisième niveau d'organisation dans Cassandra est le keyspace, qui contient un ensemble de tables (column families). C'est l'équivalent de la notion de base de données, ensemble de tables dans le modèle relationnel, ou ensemble de collections dans des systèmes comme MongoDB.

2.4 Conception de schéma

De nombreux conseils sont disponibles pour la conception d'un schéma Cassandra. Cette conception est nécessairement différente de celle d'un schéma relationnel à cause de l'absence du système de clé étrangère et de l'opération de jointure.

C'est la raison pour laquelle de nombreux design patterns sont proposés pour guider la mise en place d'une architecture de données dans Cassandra qui soit cohérente avec les besoins métiers, et la performance que peut offrir la base de données.

Cassandra oblige à réfléchir en priorité à la façon dont le modèle de données va être utilisé. Quelles requêtes vont être exécutées ? Dans quel sens mes données seront-elles traitées ? C'est à partir de ces questions que pourra s'élaborer un modèle optimisé, dénormalisé et donc performant.

En résumé

1. Cassandra permet de stocker des tables dénormalisées dans lesquelles les valeurs ne sont pas nécessairement atomiques ; il s'appuie sur une plus grande diversité de types (pas uniquement des entiers et des chaînes de caractères, mais des types construits comme les listes ou les dictionnaires).
2. La modélisation d'une architecture de données dans Cassandra est beaucoup plus ouverte qu'en relationnel, ce qui rend notamment la modélisation plus difficile à évaluer, surtout à long terme.
3. La dénormalisation (souvent considérée comme la bête noire à pourchasser dans un modèle relationnel) devient recommandée avec Cassandra, en restant conscient que ses inconvénients (notamment la duplication de l'information, et les incohérences possibles) doivent être envisagés sérieusement.
4. En contrepartie des difficultés accrues de la modélisation, et surtout de l'impossibilité de garantir formellement la qualité d'un schéma grâce à des méthodes adaptées, Cassandra assure un passage à l'échelle par distribution basée sur des techniques

de partitionnement et de réplication que nous détaillerons ultérieurement. C'est un système qui offre des performances jugées très satisfaisantes dans un environnement Big Data.

3 Mise en oeuvre

3.1 Installation

Avec Docker, il vous sera possible d'utiliser Cassandra dans un environnement virtuel. C'est de loin le mode d'installation le plus simple, il est rapide et ne pollue pas la machine avec des services qui tournent en tâche de fond et dont on ne se sert pas.

Docker sous Windows

Si vous avez Windows 10, vous pouvez installer Docker depuis le lien :

$$https://docs.docker.com/desktop/windows/install/$$

Docker sous Linux

Pour installer Docker, vous avez besoin de la version 64 bits de l'une de ces versions d'Ubuntu : Zesty 17.04, Xenial 16.04 (LTS), Trusty 14.04 (LTS).
sudo apt-get install docker-ce

```
$ docker search cassandra
$ docker pull cassandra:latest
$ docker run -v ~/Developpement/env/cassandra:/var/lib/cassandra
\--name cassandra_instance -p 3000:9042
\-e "CASSANDRA_TOKEN=1" -d cassandra:latest
```

```
$ docker ps
```

Vérifiez que votre conteneur est bien lancé. Si oui, vous êtes prêts à vous connecter au serveur Cassandra et à interagir avec la base de données.

3.2 Hello world

1) Commande : docker run hello-world
A la première exécution, Docker ne doit pas trouver l'image de l'application hello-word en local. Il va alors tenter de télécharger la dernière version. En cas de réussite, il exécute l'application qui affiche une simple page d'explication sur la sortie standard et s'arrête.
2) Commande : docker –version
Maintenant, ce serait aussi le bon moment pour vous assurer que vous utilisez la version récente de Docker.

3.3 Quelques commande Docker

— docker ps : Pour afficher uniquement les conteneurs en cours d'exécution.
— docker ps -a : Pour afficher tous les conteneurs.
— docker ps -l : Pour afficher le dernier conteneur créé.
— docker container ls : Pour afficher tous les conteneurs en cours d'exécution. Dans la nouvelle version de Docker, les commandes sont mises à jour, certaines commandes de gestion sont ajoutées.

Plus de commandes sur le site : $https://docs.docker.com/engine/reference/commandline/docker/$

4 À vos claviers

Nous allons reprendre notre base de données de livres, mais cette fois-ci on va séparer les auteurs et les livres en deux tables (Cassandra) différentes.

4.1 CQLSH

Les commandes de base sont données ci-dessous ; elles peuvent être entrées directement dans l'interpréteur de commande `cqlsh`.

1. Entrez dans votre conteneur `cassandra_instance` :

```
docker exec -it cassandra_instance bash
```

2. Vérifiez que le serveur Cassandra est en marche :

```
nodetool status
```

```
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Owns (effective)  Host ID
UN  172.17.0.2  36.22 KB   100.0%            ---  1
```

3. Lancez le client `cqlsh` :

```
cqlsh 172.17.0.2
```

N.B. L'adresse IP après le `cqlsh` correspond à l'adresse issue du teste de votre serveur Cassandra (étape 2).

4.1.1 Le keyspace

```
cqlsh > CREATE KEYSPACE IF NOT EXISTS bibliotheque
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor': 1 };
```

Explorez le `keyspace` en utilisant les commandes suivantes :

```
cqlsh > DESCRIBE keyspaces;
cqlsh > DESCRIBE KEYSPACE bibliotheque;
```

4.1.2 Les données

On peut traiter Cassandra comme une base relationnelle (en se plaçant du point de vue de la modélisation en tout cas). On crée alors des tables destinées à contenir des données avec des types atomiques. Commençons par créer une table pour nos auteurs.

```
cqlsh > use bibliotheque;
cqlsh:bibliotheque> create table IF NOT EXISTS auteurs (id text,
                nom text, prenom text,
                primary key (id)
               );
```

La commande `describe table` vous affichera les détails concernant votre table.
L'insertion de données suit elle aussi la syntaxe SQL.

```
cqlsh:bibliotheque> insert into auteurs (id, nom, prenom)
values ('auteur1', 'Tolstoy', 'Leo');
```

On peut vérifier que l'insertion a bien fonctionné en sélectionnant les données.

```
cqlsh:bibliotheque> select * from auteurs;

 id      | nom     | prenom
---------+---------+--------
 auteur1 | Tolstoy |    Leo

(1 rows)
```

Il est également possible d'insérer à partir d'un document JSON en ajoutant le mot-clé `JSON`.

```
insert into auteurs JSON '{
     "id": "auteur2",
     "nom": "Tolkien",
     "prenom": "J.R.R."
 }';
```

N.B. La structure du document doit correspondre très précisément (types compris) au schéma de la table, sinon Cassandra rejette l'insertion.

4.2 Le client graphique

Des clients graphiques existent. Le plus complet (jusqu'à 2018 !) est le Datastax DevCenter. L'installable est sur le lien Moodle. Téléchargez et installez Datastax DevCenter .
Attention : si des soucis se présentent pendant l'insatllation, vous pouvez chercher et installer le client graphique le plus récent de Datastax ou continuer en mode ligne de commandes.

4.2.1 Connnection au serveur

Après l'installation, ouvrez le client DevCenter et créez une connexion à votre serveur Cassandra.

1. Donnez un nom à votre connexion ;
2. Le `Contact host` : `localhost` si vous utilisez un conteneur docker ;
3. Le `Native Protocol port` : `3000` si vous utilisez un conteneur, car nous avons fait une redirection de port lors du lancement du conteneur ;

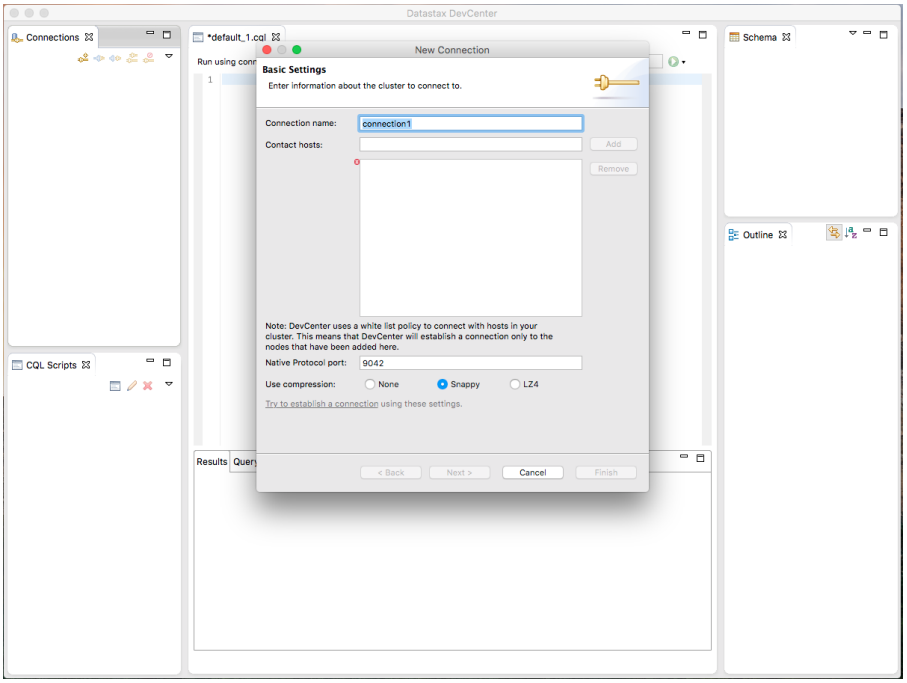

Figure 1 – Nouvelle connexion

4. Connectez-vous au serveur ;

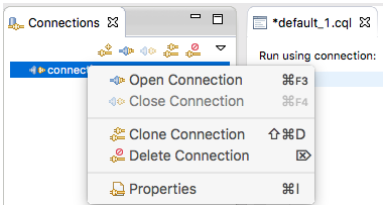

Figure 2 – Ouvrir une connexion

5. Sélectionnez le keyspace que vous souhaitez utiliser ;

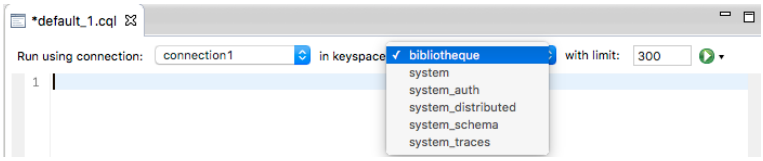

Figure 3 – Sélectionner une base de données

6. Vous pouvez maintenant exécuter vos requêtes depuis le DevCenter. **Attention** les commandes `DESCRIBE` ne fonctionnent pas, car vous pouvez accéder directement aux structures de la base de données et des tables depuis la fenêtre `Schema:NomDeVotreconnexion`.

4.3 Documents structurés (avec imbrications)

À partir de cette section, nous allons utiliser le DevCenter.

Cassandra va au-delà de la norme relationnelle en permettant des données dénormalisées dans lesquelles certaines valeurs sont complexes. C'est le principe de base que nous avons étudié pour la modélisation de document : en permettant l'imbrication, on s'autorise la création de structures beaucoup plus riches, et potentiellement suffisantes pour représenter intégralement les informations relatives à une entité.

Prenons le cas des livres. En relationnel, il faudrait associer chaque ligne de la table livres à une ligne de la table représentant les auteurs. Cassandra permet l'imbrication de la représentation d'un auteur dans la représentation d'un livre ; une seule table suffit donc. Il nous faut au préalable définir le `type` auteur de la manière suivante :

```
create type IF NOT EXISTS auteur (id text,
                    nom text,
                    prenom text);
```

Et on peut alors créer la table `livres` en spécifiant que l'un des champs a pour type `auteur`.

```
create table IF NOT EXISTS livres (id text,
                     titre text,
                     annee int,
                     ecrivain frozen<auteur>,
                     primary key (id) );
```

Notez le champ `ecrivain`, avec pour type `frozen <auteur>` indiquant l'utilisation d'un type défini dans le schéma.

Il devient alors possible d'insérer des documents structurés, comme celui de l'exemple ci-dessous. Ce qui montre l'équivalence entre le modèle Cassandra et les modèles des documents structurés que nous avons étudiés. Il est important de noter que les concepteurs de Cassandra ont décidé de se tourner vers un typage fort : tout document non conforme au schéma précédent est rejeté, ce qui garantit que la base de données est saine et respecte les contraintes.

```
INSERT INTO livres JSON '{
    "id": "16",
    "titre": "Harry Potter à école des sorciers",
    "annee": 1997,
    "ecrivain": {
        "id":"auteur1",
        "nom": "Rowling",
        "prenom": "J. K."
        }
    }';
```

```
select * from livres;
select ecrivain.id from livres;
select ecrivain.nom from livres;
select ecrivain.prenom from livres;
select titre, ecrivain.nom, ecrivain.prenom from livres;
```

Sur le même principe, on peut ajouter un niveau d'imbrication pour représenter l'ensemble des auteurs d'un livre. Le constructeur `list<...>` déclare un type liste. Voici un exemple parlant :

```
create table livres (id text,
                     titre text,
                     annee int,
                     ecrivains list<frozen<auteur>>,
                     primary key (id) );
```

Pour résumer :

1. Cassandra propose un modèle relationnel étendu, basé sur la capacité à imbriquer des types complexes dans la définition d'un schéma, et à sortir en conséquence de la première règle de normalisation (ce type de modèle est d'ailleurs appelé depuis longtemps N1NF pour Non First Normal Form) ;
2. Cassandra a choisi d'imposer un typage fort : toute insertion doit être conforme au schéma ;
3. L'imbrication des constructeurs de type, notamment les dictionnaires (nuplets) et les ensembles (set) rend le modèle comparable aux documents structurés JSON ou XML.

4.4 Les requêtes de sélection

```
SELECT * [or select_expression]
FROM [keyspace_name.] table_name
[WHERE partition_value
   [AND clustering_filters
   [AND static_filters]]]
[ORDER BY PK_column_name ASC / DESC]
[LIMIT N]
[ALLOW FILTERING]
```

Le CQL est très proche du SQL mis à part les jointures qui ne sont pas permises. Pour plus de détails :
`https://docs.datastax.com/en/cql/3.3/cql/cql_reference/cqlSelect.html`

5 Travail à faire

Nous allons maintenant traiter un cas d'utilisation basé sur les "Points d'intérêt touristique" représentant l'ensemble de l'offre touristique et de loisirs sur les communes du Grand Lyon.

1. Créez un nouveau `keyspace`, nommez le `tourismes`
2. créez la table `poi_lyon` avec les champs suivants :

```
"id","adresse","classement","codepostal","commune","date_creation"
"email","facebook","fax","gid","id_sitra1",
"last_update","last_update_fme","nom","ouverture","producteur",
"siteweb","tarifsenclair","tarifsmax","tarifsmin"
"telephone","telephonefax","type","type_detail"
```

   source : `https://data.grandlyon.com/`
3. dans le terminal :
   (a) téléchargez le fichier poi.csv disponible sur Moodle.
   (b) copiez le fichier dans votre instance cassandra

```
docker cp poi.csv cassandra_instance:/
```

   (c) entrez dans le `bash` de votre instance

```
docker exec -it cassandra_instance bash
```

   (d) lancez le `cqlsh`
   (e) exécutez les commandes suivantes pour charger le `poi.csv` dans votre table `poi_lyon`

```
cqlsh> use tourismes;
cqlsh:tourismes>
cqlsh:tourismes> COPY poi_lyon (classement, siteweb, commune ,
telephone, codepostal, id, telephonefax, last_update, gid,
tarifsenclair, type, email, nom, fax, adresse, tarifsmax,
facebook, producteur, last_update_fme, tarifsmin, date_creation,
id_sitra1, ouverture, type_detail)
FROM 'poi.csv' WITH DELIMITER=',';
```

   N.B Faites attention à l'ordre des champs dans le fichier .csv
4. vérifiez, dans `DevCenter`, que toutes les données (1001 lignes) ont été chargées

```
select count(*) from poy_lyon;
```

5. **requêtes** :
   (a) liste de tous les POI
   (b) liste des Noms de POI
   (c) nom et commune des restaurants de `type 'RESTAURATION'`. Cette requête ne sera pas exécutée car le champ `type` n'est pas indexé.

```
Unable to execute CQL script on 'connection1':Cannot execute this
query as it might involve data filtering and thus may have
unpredictable performance. If you want to execute this query despite
the performance unpredictability,  use ALLOW FILTERING"
```

   Indexez le champ `type` ou ajoutez `ALLOW FILTERING` à votre requête
   (d) comptez le nombre de 'PATRIMOINE_CULTUREL' dans le Grand Lyon
   (e) les noms des 'COMMERCE_ET_SERVICE' dans le premier arrondissement de Lyon
   (f) le nom et l'adresse du POI 206112
   (g) liste de tous les POI créés après le 2013-01-01 (YYYY-MM-DD)
   (h) combien de POI crées après le 2013-01-01 sont dans la 'HEBERGEMENT_LOCATIF'
   (i) les noms et adresses des POI id (190563, 129583,156377, 128673)
   (j) la liste des hotels 4 étoiles du Grand Lyon

6. **évolution du schéma** :on souhaite imbriqué deux `types` (contact et tarification) à notre table `poi_lyon`
   (a) créez un `type` **`contact`** qui regroupera les données suivantes : "email", "telephone","adresse", "facebook", "fax", "telephonefax"
   (b) créez un `type` **`tarification`**qui regroupera les données suivantes : "tarifsenclair", "tarifsmax","tarifsmin"
   (c) créez une nouvelle table `poi_lyon_evo` qui utilisera les nouveaux types créés précédemment
   (d) remplissez la table avec les données json du fichier *samples* disponible sur Moodle.
   (e) listez les `noms` et les `emails` de contact des poi
   (f) listez les `noms` et les `tarifsenclair` des poi