Bases de données documentaires et distribuées, http://b3d.bdpedia.fr

Cassandra: modélisation des données

Qu'est-ce que Cassandra ?

Cassandra, conçu à l'origine par Facebook, maintenant un projet de la fondation Apache, distribué par la société Datastax.

- Initialement basé sur le système BigTable de Google (stockage orienté colonnes)
- A fortement évolué vers un *modèle relationnel étendu*
- Un des rares systèmes NoSQL à proposer un typage fort
- Un langage de définition de schéma et de requêtes, CQL (pas de jointure)

Construit dès l'origine comme un système **scalable** et **distribué**.

Dans ce cours, nous étudions le modèle de données.

Modèle = relationnel + types complexes

Cassandra gère des **bases** (*keyspaces*), contenant des **tables** (a.k.a. *column families*).

Une table contient des **lignes** (*rows*) constituées de **valeurs simples** ou **complexes**.

Perspective A : c'est du relationnel étendu en rompant la première règle de normalisation (type atomique).

Perspective B : chaque ligne est un document structuré au sens où nous l'avons vu jusqu'ici.

Attention

Vocabulaire confus et parfois déroutant : *columns*, *column families*, etc.

Paires et documents

La structure de base est la paire (clé, valeur)

- Clé : un identifiant
- Valeur : atomique (entier, chaîne de caractères), ou complexe (dictionnaire, ensemble, liste)

Une ligne (*row*, document) est un identifiant (*row key*) associé à un ensemble de paires (clé, valeur).

Les lignes sont **typées** par un schéma, y compris les données imbriquées.

Cassandra n'autorise pas (au moins avec CQL) l'insertion de données ne respectant pas le schéma.

Quelques aspects ignorés ici, le versionnement notamment.

Tables et base de données

Une **table** (anciennement *column family*) est un ensemble de *rows* conformes au schéma de la table.

Les tables sont **dénormalisées** : on peut avoir des attributs dont la valeur est un type construit (objet ou ensemble).

Des notions de colonnes et super-colonnes sont (semble-t-il) en voie de disparition.

Une **base** est un ensemble de tables. On l'appelle *keyspace*. Explication ? Plus tard.

Conception d'un schéma

Cassandra ne propose pas de jointure. C'est du NoSQL après tout.

Il est donc préférable de **regrouper** les données le plus possible dans des lignes (notion de document autonome, déjà vue).

Au pire, on peut représenter les mêmes données sous plusieurs formes.

- Tous les films, avec leurs réalisateurs et acteurs.
- Tous les artistes, avec les films qu'ils ont tournés ou dans lesquels ils ont joué.

Le regroupement a aussi des inconvénients (redondance, difficulté d'accès aux données "profondes"). La conception Cassandra s'appuie sur des heuristiques spécifiques : une autre session lui est consacrée.

Application !

Création d'un *keyspace*.

```
create keyspace if not exists Movies
       with replication = { 'class' : 'SimpleStrategy',
               'replication_factor': 3 };
```

Création d'une table (sans imbrication).

```
 create table artists (id text,
                last_name text, first_name text,
                birth_date int, primary key (id)
            );
```

Insertions

A la SQL

```
insert into artists (id, last_name, first_name, birth_date)
      values ('artist1', 'Depardieu', 'Gerard', 1948);
insert into artists (id, last_name, first_name, birth_date)
          values ('artist2', 'Baye', 'Nathalie', 1948);
insert into artists (id, last_name, first_name)
          values ('artist3', 'Marceau', 'Sophie');
```

Avec JSON

```
insert into artists JSON '{
     "id": 1,
     "last_name": "Coppola",
     "first_name": "Sofia",
     "birth_date": "1971"
   }'
```

Imbrication de structures

C'est le principe de dénormalisation : on regroupe les données le plus possible dans des lignes pour éviter les jointures.

Une valeur d'attribut peut correspondre :

- à un ensemble de valeurs (non ordonnées) : **SET**
- à une liste de valeurs : **LIST**
- à un dictionnaire : **MAP**
- à un nuplet : **TUPLE**
- à une instance d'un type utilisateur

Les ensembles doivent rester de taille raisonnable

Avec imbrication

Création d'un type

```
create type artist (id text,
                    last_name text,
                    first_name text,
                    birth_date int,
```

Table des films, avec imbrication d'un object conforme au type `artist`.

```
create table movies (id text,
                     title text,
                     year int,
                     genre text,
                     country text,
                     director frozen<artist>,
                     primary key (id) );
```

Insertion de documents

À partir d'un document JSON

```
insert into movies JSON '{
    "id": "movie:1",
    "title": "Vertigo",
    "year": 1958,
    "genre": "drama",
    "country": "USA",
    "director": {
        "id": "artist:3",
        "last_name": "Hitchcock",
        "first_name": "Alfred",
    "   birth_date": "1899"
    },
}';
```

Table des films, complète

On peut avoir des valeurs instances d'un type, en **ensemble** d'instance d'un type (mot-clé `set`). Exemple :

```
create table movies (id text,
                     title text,
                     year int,
                     genre text,
                     country text,
                     director frozen<artist>,
                     actors set< frozen<artist>>,
                  primary key (id) );
```

À retenir

- Cassandra propose un modèle relationnel étendu, basé sur la capacité à imbriquer des types complexes dans la définition d'un schéma
- Cassandra a choisi d'imposer un typage fort : toute insertion doit être conforme au schéma ;
- L'imbrication des constructeurs de type rend le modèle comparable aux documents structurés JSON ou XML.