Apache Cassandra : Modelisation

v1.0

Objectifs

Dans ce TP nous allons étudier la modélisation des données dans Apache Cassandra.

- Dans un premier temps nous allons regarder plusieurs modélisations existantes pour comprendre le modéle de stockage physique de Cassandra ainsi que les possibilités de requetage offertes par chaque modélisation.
- Nous allons ensuite proposer un schema Cassandra pour enregistrer les données produites par un réseau de capteurs.

Primary key, partition key et clustering columns

1) Via **ccm** creez un nouveau cluster ***temperatures_NOM*** en spécifiant la version de cassandra 3.0.15 (remplacer NOM par votre NOM).

```bash
[bigdata@bigdata ~]$ccm create temperatures_andrei -v 3.0.15 -n 3 -s
```

2) Créez un keyspace ***temperature*** avec un RF de 3. Créez une table *temperature1* qui a deux colonnes (*ville* et *temperature*) et une clé primaire composée d'une seule colonne. Expliquez le choix de la clé primaire et ses impacts. Faire quelques insertions avec des valeurs de villes/temperatures différentes/identiques et tester quelles sont les requêtes supportées par cette table.

SQL

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

USE temperature;

-- Plusieurs modelisation possibles
CREATE TABLE temperature1 (
    ville text,
    temperature int,
    PRIMARY KEY (ville)
);

CREATE TABLE temperature1 (
    ville text,
    temperature int,
    PRIMARY KEY (temperature)
);

-- Pour ce qui suit nous allons utiliser cette modelisation comme example
CREATE TABLE temperature1 (
    ville text,
    temperature int,
    PRIMARY KEY (ville)
);

INSERT INTO temperature1(ville, temperature ) VALUES ( 'Paris', 30);
INSERT INTO temperature1(ville, temperature ) VALUES ( 'Paris', 29);
INSERT INTO temperature1(ville, temperature ) VALUES ( 'Rennes', 30);

INSERT INTO temperature1(ville) VALUES ( 'Bas-terre'); -- On peut inserer des villes sans temperatures


SELECT * FROM temperature;

 ville  | temperature
--------+-------------
  Paris |          29
 Rennes |          30


-- Ce modèle est mis en défaut à cause de l'upsert car pour la même clef ville et la meme temperature nous
allons ecraser sa valeur.



SELECT * FROM temperature1; -- OK
SELECT * FROM temperature1 WHERE ville='Paris'; -- OK
SELECT * FROM temperature1 WHERE temperature=30; -- KO

cqlsh:temperature> SELECT * FROM temperature1 WHERE temperature=30; -- KO
InvalidRequest: Error from server: code=2200 [Invalid query] message="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"
cqlsh:temperature> SELECT * FROM temperature1 WHERE temperature=30 ALLOW FILTERING;

 ville  | temperature
--------+-------------
 Rennes |          30

(1 rows)

cqlsh:temperature> SELECT * FROM temperature1 WHERE ville='Paris' AND temperature>20; -- KO
InvalidRequest: Error from server: code=2200 [Invalid query] message="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"

-- Chauque condition dans une clause WHERE doit etre faite uniquement sur des colonnes de la  cle primaire ! Si
c'est pas le cas (la colonne temperature fait pas partie de la cle) la requete sera potentiellement pas
```

```
efficace (dans ce cas toutes les lignes de la partition qui correspond a la ville de 'Paris' seront chargeer
pour filtrer la temperature).

cqlsh:temperature> SELECT count(*) FROM temperature1; --KO

 count
-------
     2

(1 rows)

Warnings :
Aggregation query used without partition key


cqlsh:temperature> SELECT count(*) FROM temperature1 WHERE ville='Paris'; --OK

 count
-------
     1

(1 rows)


cqlsh:temperature> SELECT ville, max(temperature) FROM temperature1 GROUP BY ville;
SyntaxException: line 1:49 no viable alternative at input 'GROUP' (...ville, max(temperature) FROM
[temperature1] GROUP...) -- Supportee a partir de Cassandra 3.10 !
```

3) Pour comprendre le stocquage physique de Cassandra nous allons utiliser sstabedump pour avoir une representation des données qui sont dans les SSTables d'un noeud spécifique. Pour s'assurer que les *memtables* ont bien ete sauvegardées dans des *sstables* (donc persistés sur le disk) nous allons faire un *ccm flush* (qui va faire un nodetool flush sur tous les noeuds du cluster). Dans le cas des suppressions une compaction du keyspace peut etre nécessaire.

> Pour la commande *sstabledump* adaptez le nom de la SSTable ainsi que le repertoire:
>
> ***.ccm/{CLUSTER_NAME}/{NODE_NAME}/data/{KEYSPACE_NAME}/{TABLE_NAME}-{TABLE_ID}/mc-{SSTABLE_ID}-big-Data.db***

BASH

```
[bigdata@bigdata ~]$ ccm flush; ccm node1 nodetool compact temperature
```

```
[bigdata@bigdata ~]$.ccm/repository/3.0.15/tools/bin/sstabledump
.ccm/temperatures_andrei/node1/data/temperature/temperature1-512a7380c95111e7a6f381a6e4064663/mc-1-big-Data.db
```

JAVASCRIPT


```
[
  {
    "partition" : {
      "key" : [ "Paris" ],
      "position" : 0
    },
    "rows" : [
      {
        "type" : "row",
        "position" : 30,
        "liveness_info" : { "tstamp" : "2017-11-14T15:34:40.012400Z" },
        "cells" : [
          { "name" : "temperature", "value" : 29 }
        ]
      }
    ]
  },
  {
    "partition" : {
      "key" : [ "Rennes" ],
      "position" : 31
    },
    "rows" : [
      {
        "type" : "row",
        "position" : 62,
        "liveness_info" : { "tstamp" : "2017-11-14T15:34:40.039478Z" },
        "cells" : [
          { "name" : "temperature", "value" : 30 }
        ]
      }
    ]
  }
]
```


4) Nous voulons changer cette modélisation pour pouvoir récupérer les temperatures enregistrées dans une ville donnée pour les deux derniers jours disponibles (avec la donnée la plus fraîche en premier). Créez une nouvelle table temperature2 qui a une colonne de plus → *record_date* de type text dans laquelle on va stocker la date des relevés de temperature. Quelle est la modélisation qui ne permettra de répondre a notre besoin? Créer cette table, insérez quelques valeurs et écrivez la requete demandée. Utiliser ***sstabledump*** pour explorer le stocquage physique.

SQL

```
CREATE TABLE temperature2 (
    ville text,
    record_date text,
    temperature int,
    PRIMARY KEY (ville, record_date)
) WITH CLUSTERING ORDER BY (record_date DESC) ;

INSERT INTO temperature2 (ville, record_date, temperature ) VALUES ( 'Paris', '2017/11/14', 30);
INSERT INTO temperature2 (ville, record_date, temperature ) VALUES ( 'Paris', '2017/11/13', 29);
INSERT INTO temperature2 (ville, record_date, temperature ) VALUES ( 'Rennes', '2016/11/10', 30);
INSERT INTO temperature2 (ville, record_date, temperature ) VALUES ( 'Paris', '2017/11/15', 29);


cqlsh:temperature> SELECT * FROM temperature2;

 ville  | record_date | temperature
--------+-------------+-------------
  Paris |  2017/11/15 |          29
  Paris |  2017/11/14 |          30
  Paris |  2017/11/13 |          29
 Rennes |  2016/11/10 |          40


-- Le fait d'avoir forcé le clustering order nous permmet d'avoir les resultats naturellement triees

 cqlsh:temperature> SELECT * FROM temperature2 WHERE ville = 'Paris' LIMIT 2;

  ville | record_date | temperature
 -------+-------------+-------------
  Paris |  2017/11/15 |          29
  Paris |  2017/11/14 |          30

 (2 rows)
```

JAVASCRIPT

```
[bigdata@bigdata ~]$ ccm flush; ccm node1 nodetool compact temperature
```

```
[bigdata@bigdata ~]$ .ccm/repository/3.0.15/tools/bin/sstabledump
.ccm/temperatures_andrei/node1/data/temperature/temperature1-268f3470c95211e7a6f381a6e4064663/mc-1-big-Data.db
[
  {
    "partition" : {
      "key" : [ "Paris" ],
      "position" : 0
    },
    "rows" : [
      {
        "type" : "row",
        "position" : 43,
        "clustering" : [ "2017/11/15" ],
        "liveness_info" : { "tstamp" : "2017-11-14T15:44:32.305629Z" },
        "cells" : [
          { "name" : "temperature", "value" : 29 }
        ]
      },
      {
        "type" : "row",
        "position" : 43,
        "clustering" : [ "2017/11/14" ],
        "liveness_info" : { "tstamp" : "2017-11-14T15:40:36.844827Z" },
        "cells" : [
          { "name" : "temperature", "value" : 30 }
        ]
      },
      {
        "type" : "row",
        "position" : 64,
        "clustering" : [ "2017/11/13" ],
        "liveness_info" : { "tstamp" : "2017-11-14T15:40:36.870250Z" },
        "cells" : [
          { "name" : "temperature", "value" : 29 }
        ]
      }
    ]
  },
  {
    "partition" : {
      "key" : [ "Rennes" ],
      "position" : 88
    },
    "rows" : [
      {
        "type" : "row",
        "position" : 132,
        "clustering" : [ "2016/11/10" ],
        "liveness_info" : { "tstamp" : "2017-11-14T15:42:30.031849Z" },
        "cells" : [
          { "name" : "temperature", "value" : 40 }
        ]
      }
    ]
  }
]
```

5) Le modèle precedent est bien meilleur car on peut insérer 1 température par jour, sans écraser la précédente. Cependant, on ne pourra pas stocker plus de 2 milliards de valeurs pour la meme ville. Nous nous rendons compte aussi qu'on commence a avoir des partitions trop grosses pour certaines villes. Créez une nouvelle table pour palier a ces problèmes en permettant le requetage de la temperature heure par heure pour un jour donné.

Créez les requêtes pour interroger Cassandra en spécifiant soit la ville, soit la ville + la date. Vérifier qu'on ne peut pas accéder à la donnée avec une requête qui spécifie simplement la date.

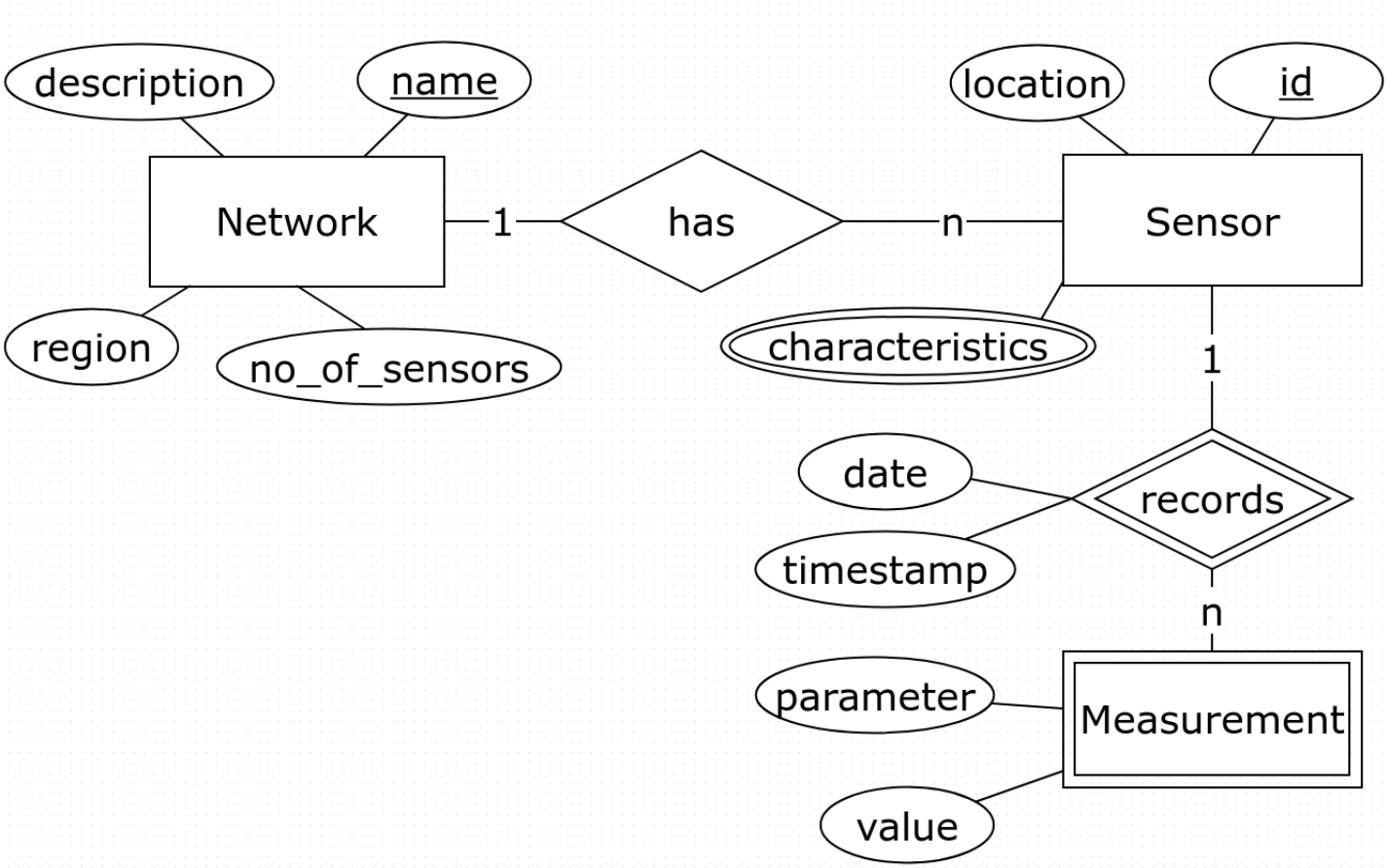
SQL

```sql
CREATE TABLE temperature3 (
    ville text,
    jour text,
    heure text,
    temperature int,
    PRIMARY KEY ((ville, jour), heure)
);

INSERT INTO temperature3 (ville, jour, heure, temperature)
VALUES ('Paris', '2014-01-20', '8', 7);
INSERT INTO temperature3 (ville, jour, heure, temperature)
VALUES ('Paris', '2014-01-20', '9', 8);
INSERT INTO temperature3 (ville, jour, heure, temperature)
VALUES ('Paris', '2014-01-20', '10', 7);
INSERT INTO temperature3 (ville, jour, heure, temperature)
VALUES ('Paris', '2014-01-21', '10', 7);

SELECT * FROM temperature3;
SELECT * FROM temperature3 WHERE ville = 'Paris';
SELECT * FROM temperature3 WHERE jour = '2014-01-20';
SELECT * FROM temperature3 WHERE ville = 'Paris' AND jour = '2014-01-20';
```

Modélisation

Dans cette partie nous allons écrire les requêtes pour modéliser un systeme qui gère les données produites par un réseau de capteurs. Nous allons utiliser l'approche décrite ici (http://www.cs.wayne.edu/andrey/papers/bigdata2015.pdf) et utilisée par KDM (http://kdm.dataview.org) .

Nous avons fait une premiere étape d'analyse des besoins et on a construit le modéle conceptuel avec les entités, leurs attributes et leurs clefs ainsi que les relations entre entités (rectangle = entité, cercle = attribut, dimant=relation, double rectangle/diamant = entite faible, double cercle = collection).



Les principales requêtes auxquelles notre modélisation devrait pouvoir répondre sont:

- *Q1*: Trouver tous les paramètres des capteurs d'un réseau donné
- *Q2*: Trouver tous les relevées d'un capteur donné pour un jour donneé. Afficher les résultats comme une liste triée en ordre décroissante sur le timestamp
- *Q3*: Trouver toutes les mesures (*parameter*, *value*) d'un capteur donne. Trier les résultats par l'heure de l'enregistrement (décroisant).

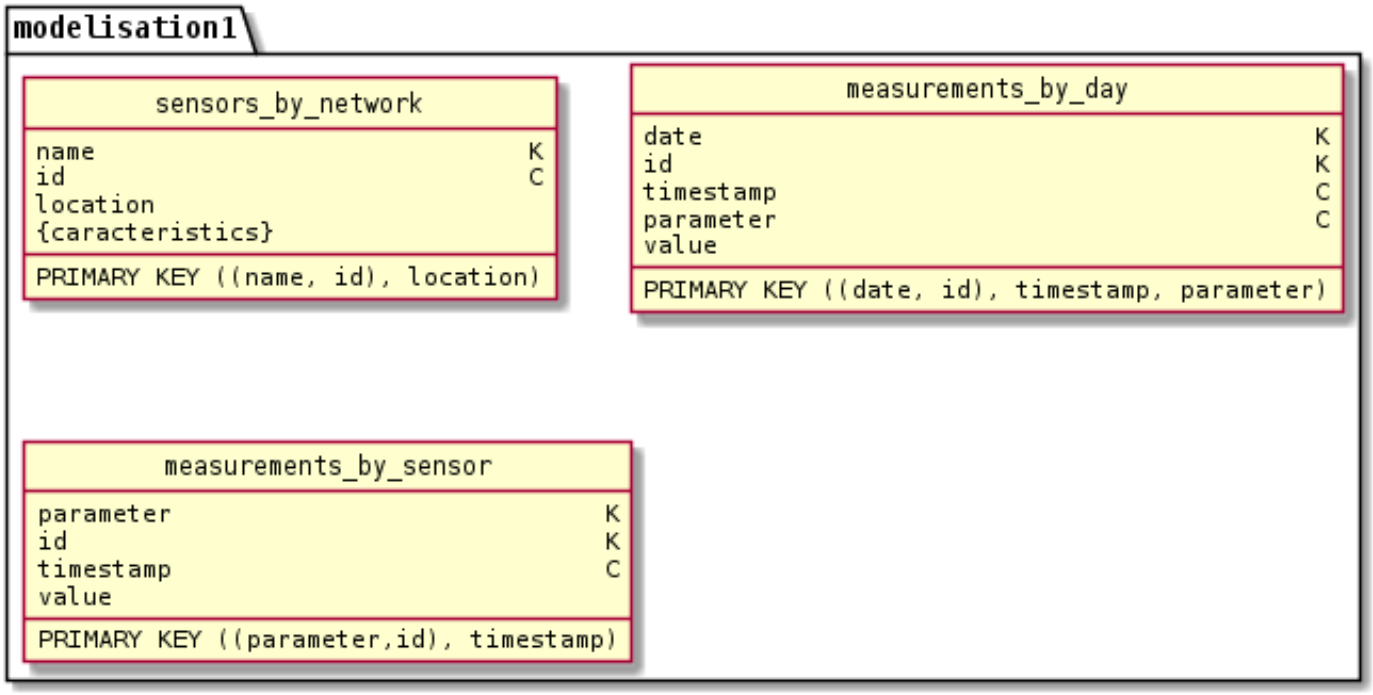
1) ***Modélisation logique***: Créez un ensemble des tables Cassandra en spécifiant pour chaque table le nom des attributes et les propriétés (cle primaire, cle de partition, colonne de clustering.

Par exemple le modèle logique suivant correspond a notre table *temperature3* définie précédemment (Legende: **K** = partie de la cle de partition, **C** = clustering column, **S** = static column)

temperature3	
ville	K
jour	K
heure	C
temperature	
PRIMARY KEY ((ville, jour), heure)	

SQL

```
-- Plusieurs modélisation logiques sont possibles, par exemple :
```



2) ***Modélisation physique***: ecrire les requêtes CQL pour créer les tables qui correspondent au modéle logique precedement défini (prenez en compte les contraintes exposes en cours sur la taille des partition, des colonnes etc..). Traduire en CQL les requêtes presentées dans l'expression des besoins.

SQL

```
-- Pour le modele logique choisi plus haut on a la
// Q1:
CREATE TABLE sensors_by_network (name TEXT, id UUID, location TEXT, characteristics SET<TEXT>, PRIMARY KEY
(name,id)) WITH CLUSTERING ORDER BY (id ASC);

/* SELECT id, location, characteristics FROM sensors_by_network WHERE name=?; */

// Q2:
CREATE TABLE measurements_by_day (date TEXT, id UUID, timestamp TEXT, parameter TEXT, value TEXT, PRIMARY KEY
((date,id),timestamp,parameter)) WITH CLUSTERING ORDER BY (timestamp DESC, parameter ASC);

/* SELECT timestamp, parameter, value FROM table0 WHERE date=? and id=? ORDER BY timestamp DESC; */

// Q3:
CREATE TABLE measurements_by_sensor (parameter TEXT, id UUID, timestamp TEXT, value TEXT, PRIMARY KEY
((parameter,id),timestamp)) WITH CLUSTERING ORDER BY (timestamp DESC);

/* SELECT timestamp, value FROM measurements_by_sensor WHERE parameter=? and id=? ORDER BY timestamp DESC; */
```

☑ Afficher les reponses

BASH

```
Les reponses sont maintenant affichees !
```

Last updated 2019-11-20 08:21:27 +0100