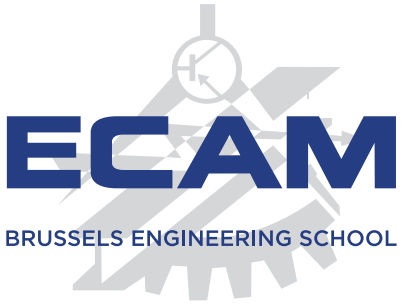


I404C NoSQL

Séance 3

Modèle orienté-colonnes Cassandra, HBase

Sébastien Combéfis

2019

Rappels

- Modèle de base de données orienté clé-valeur
  - Définition et principe du stockage de paires clé-valeur
  - Moteur Riak avec les buckets
  - Base de données en RAM avec moteur Memcached
  - Moteur Redis et types de données spécifiques
- Modèles de distribution des données sur un cluster
  - Sharding et répartitions des données sur des nœuds
  - Réplication et copies des données sur des nœuds
  - Approches hybrides combinant le sharding et la réplication

Objectifs

- Le modèle orienté-colonnes
  - Stockage sur disque de lignes ou colonnes
  - Le modèle de données
  - Principaux types de requêtes
- Exemples de bases de données orientée-colonnes
  - HBase
  - Cassandra

Modèle orienté-colonnes

Famille de colonnes (1)

- Base de données orientée-colonnes proche des relationnelles

  Comportent des colonnes avec un type de données

- Suit l'approche BigTable apportée par Google

  Et dont HBase est une implémentation open source

- Accès rapide aux données et très bonne scalability

  En particulier avec Cassandra et une distribution peer-to-peer

Famille de colonnes (2)

- Ensemble de clés de ligne et de familles de colonnes

  Organisation d'une base sous forme de plusieurs tables

- Regroupement des données souvent accédées ensemble

  Chaque famille de colonnes est une map de données

Ligne vs colonne (1)

- Stockage sur disque par tuples ou par colonnes

  Initialement uniquement une question de stockage

- Les requêtes ne concernent souvent pas toutes les colonnes

  Récupération directe de colonnes depuis le disque plus efficace

ID	**Firstname**	**Class**
16139	Alexis	3BE
10003	Damien	5MIN

Stockage de lignes

ID
16139
10003

Firstname
Alexis
Damien

Class
3BE
5MIN

Stockage de colonnes

Ligne vs colonne (2)

- Choix du stockage sur disque pour l'efficacité des opérations
  - Stockage de lignes efficace pour les écritures
  - Stockage de colonnes efficace pour les lectures
- Lecture de quelques colonnes de beaucoup de lignes

  Améliore les performances des requêtes de sélection

	Stockage de lignes	**Stockage de colonnes**
+	Facile d'ajouter un enregistrement	Seules les données voulues sont lues
−	Lecture de données inutiles	Écriture d'un tuple nécessite plusieurs accès

C-Store (1)

- Stockage des données de la base en colonnes

  Créé par les universités Brown, Brandeis, MIT et UMass Boston

- Basé sur le modèle relationnel et utilise SQL

  Ne fait pas partie de la sphère NoSQL, mais va l'inspirer

- Deux espaces de stockage différents sur le disque

  Pour optimiser au mieux les opérations de lecture et écriture

C-Store (2)

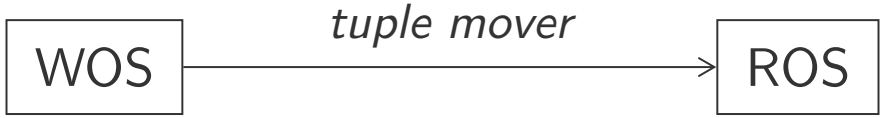
- ROS (*Read Optimized Store*)
  - Stockage de fichiers contenant des colonnes
  - Compression des fichiers selon le type de données inclus
  - Données triées selon un attribut de la table de la colonne
- WOS (*Write Optimized Store*)
  - Buffer temporaire utilisé lors d'écritures (INSERT, UPDATE)
  - Pas de compression et pas de partitionnement vertical

C-Store (3)

- Migration régulière des données du WOS vers le ROS

  Réalisé par un tuple mover qui est autorisé à écrire dans le ROS

- Les requêtes doivent pouvoir agir sur les deux stores
  - Insertions directement envoyées au WOS
  - Suppression marquée dans ROS, puis gérée par tuple mover
  - Update est une combinaison d'insertion et suppression



Ligne vs colonne (3)

- Pas de meilleur choix absolu entre lignes ou colonnes

  Tout dépend du type d'opération effectuée

	Lignes	**Colonnes**
Agrégation d'éléments d'une colonne	Lent	Rapide
Compression	–	Haute
Sélection de quelques colonnes	Lent (skip de données)	Rapide
Insertion/Mise à jour	Rapide	Lent
Sélection d'un enregistrement	Rapide	Lent

Modèle de données (1)

- Une base orientée-colonnes est un map à deux niveaux

  Plutôt qu'une structure en tables organisées par colonnes

- Au premier niveau, une paire clé-valeur identifie une ligne

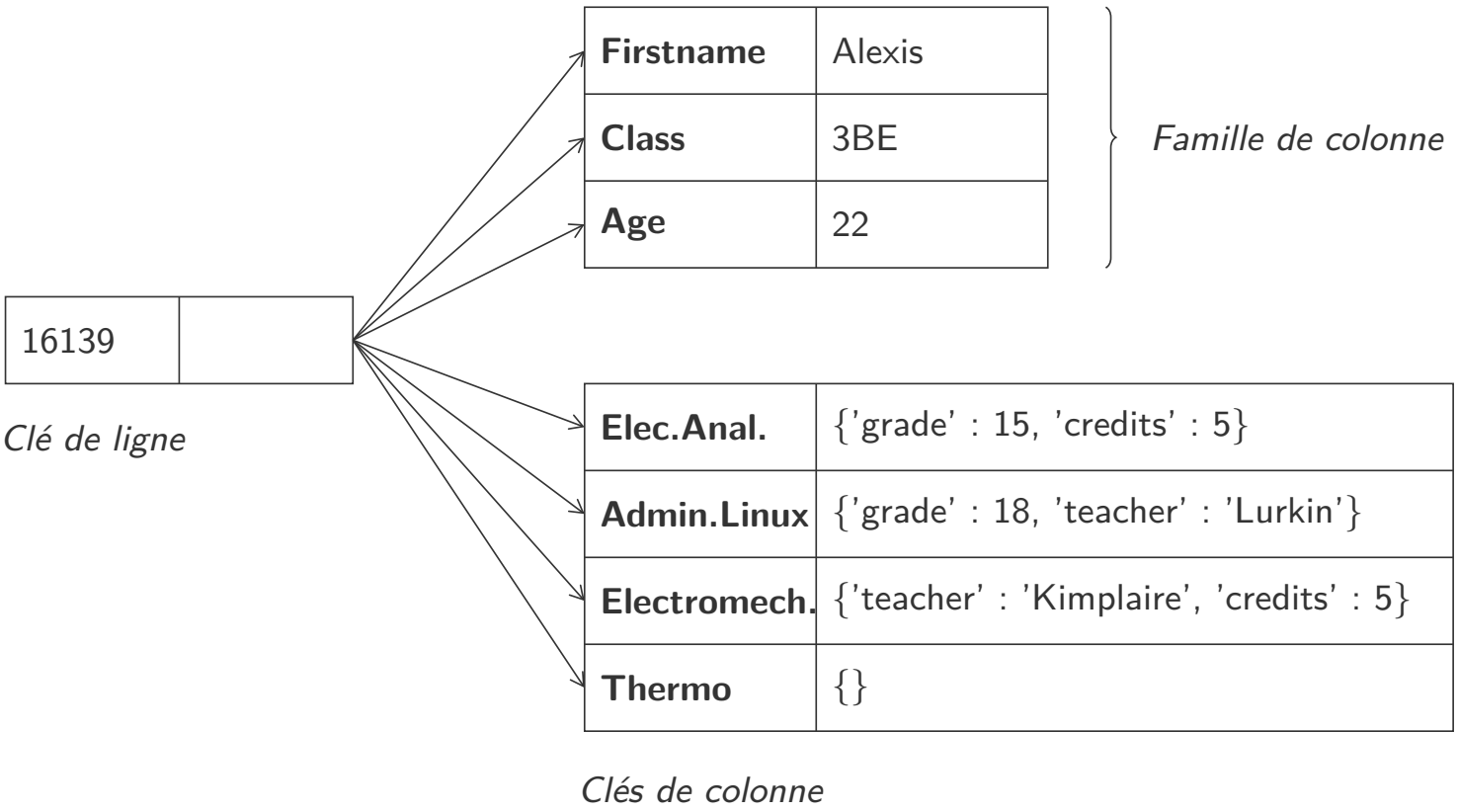
  La clé est un identificateur de ligne

- Au second niveau, une map de colonnes formant des familles
  - Nombre arbitraire de paires clé-valeur par ligne
  - Les familles permettent un accès commun à des colonnes

Modèle de données (2)

- Structure à deux niveaux combinant lignes et colonnes

  Ligne est la jointure des enregistrements des familles de colonnes



Modèle de données (3)

- Base orientée-colonnes ne sont pas vraiment des tables
  - On peut ajouter des colonnes à n'importe quelle ligne
  - Les lignes peuvent avoir des clés de colonnes différentes
- Définir de nouvelles familles de colonnes est rare

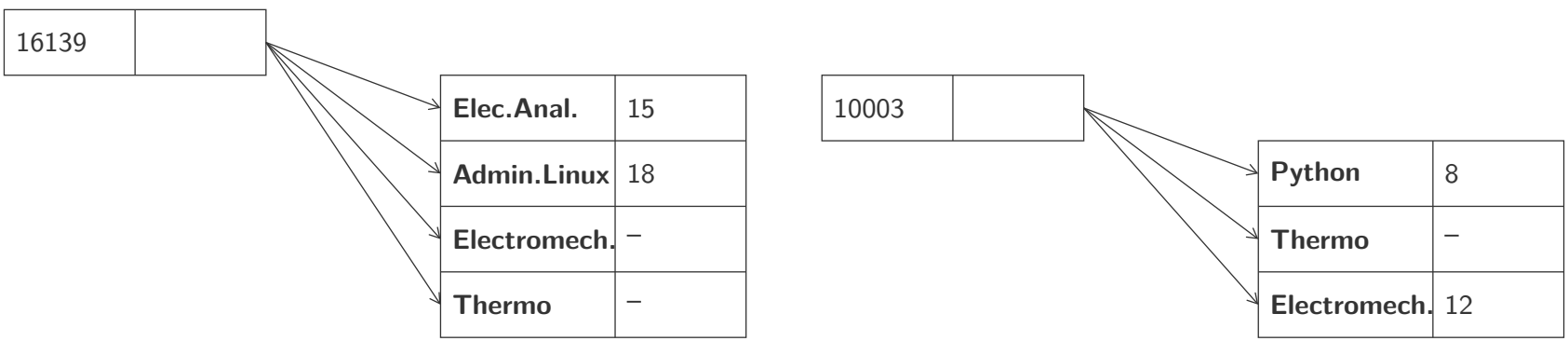
  Alors que l'ajout de colonnes peut se faire à la volée
- Deux types de lignes selon le nombre de colonnes
  - **Skinny row** quelques colonnes et les mêmes partout (*field-like*)
  - **Wide row** avec des milliers de colonnes (*list-like*)

Table vs Colonne

- Base orientée-colonnes permet d'éviter la présence de `NULL`

Chaque ligne ne possède que les colonnes qu'elle doit avoir

Matricule	Admin.Linux	Elec.Anal.	Electromech.	Python	Thermo
16139	18	15	–	NULL	–
10003	NULL	NULL	12	8	–



Avantages des colonnes

- Lecture efficace des données des seules colonnes demandées

  Attention à la reconstruction des tuples lorsqu'on lit tout

- Meilleur taux de compression, mais utilisation du CPU

  Moins d'entropie puisque toutes les données du même domaine

- Efficacité du tri et de l'indexation des données

  Par stockage redondant grâce à l'espace gagné par compression

Projection (1)

- Possibilité d'avoir des projections stockées physiquement

  Permet d'améliorer les performances pour des types de requêtes

Table logique

Region	Client	Produit	Vente
A	G	C	789
B	C	C	743
D	F	D	675
C	C	A	23
A	R	B	654

Super-projection

Region	A	B	D	C	A
Client	G	C	F	C	R
Produit	C	C	D	A	B
Vente	789	743	675	23	654

Projection (2)

- Projections peuvent êtres triées sur une ou plusieurs colonnes

  Améliore les performances pour requêtes `SORT` et `GROUP BY`

Table logique

Region	Client	Produit	Vente
A	G	C	789
B	C	C	743
D	F	D	675
C	C	A	23
A	R	B	654

Projection 1

Region	A	A	B	C	D
Produit	B	C	C	A	D
Vente	654	789	743	23	675

Facilite des requêtes du style

```
SELECT Region, Produit, SUM(Vente)
GROUP BY Region, Produit
```

Projection (3)

- Peuvent être créées manuellement ou on-the-fly

  Un peu la même logique que d'avoir des vues matérialisées

Table logique

Region	Client	Produit	Vente
A	G	C	789
B	C	C	743
D	F	D	675
C	C	A	23
A	R	B	654

Projection 2

Client	C	C	F	G	R
Vente	743	23	675	789	654

Facilite des requêtes du style

```
SELECT Client, SUM(Sales)
GROUP BY Client
```

Compression (1)

- Run-Length Encoding sur les valeurs dans les colonnes

  Pratique lorsque beaucoup de données similaires

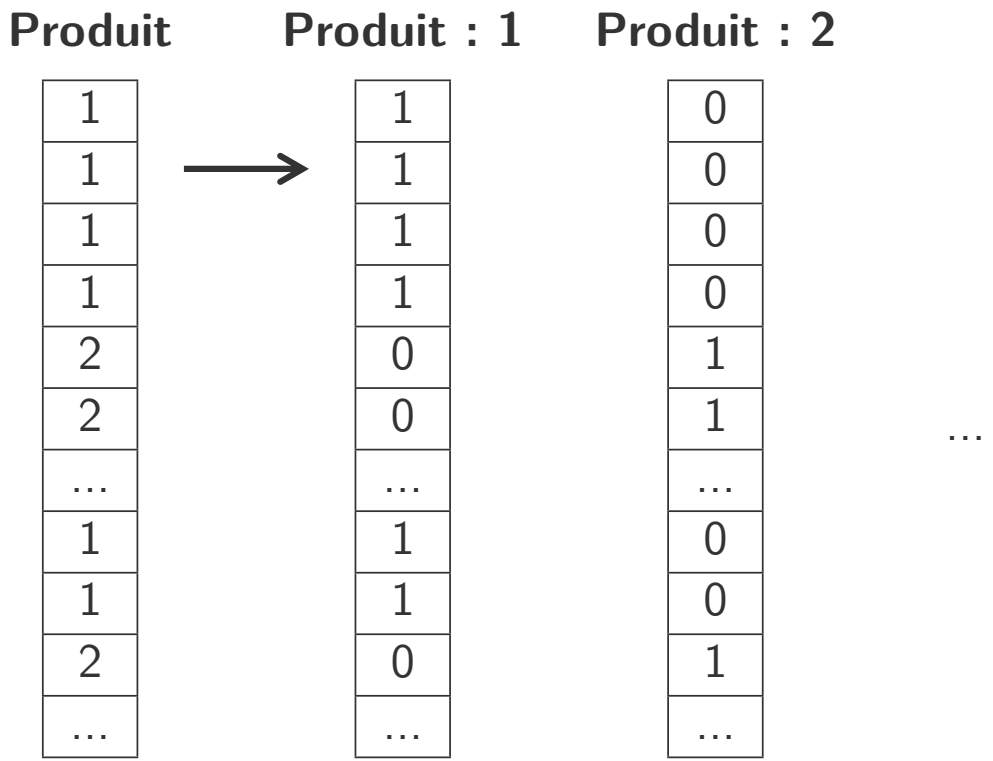
Quadri	Produit	Prix
Q1	1	5
Q1	1	7
Q1	1	2
Q1	1	9
Q1	2	6
Q1	2	8
...	...	...
Q2	1	3
Q2	1	8
Q2	2	1
...	...	...

Quadri	Produit	Prix
(Q1, 1, 300)	(1, 1, 4)	5
(Q2, 301, 350)	(2, 5, 2)	7
...	...	2
	(1, 301, 2)	9
	(2, 303, 1)	6
	...	8
		...
		3
		8
		1
		...

Compression (2)

- Bit-Vector Encoding pour chaque valeur unique des colonnes

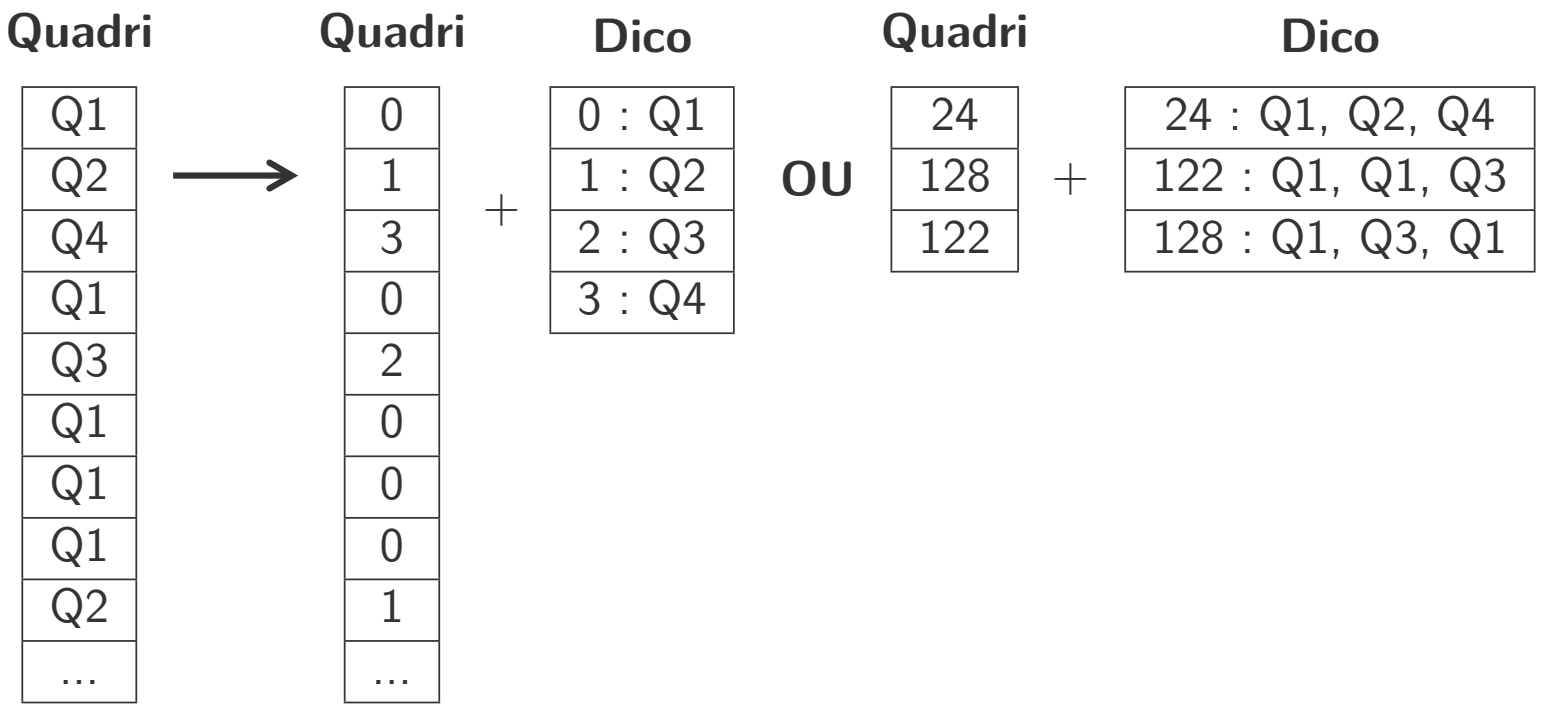
  Pratique lorsque peu de valeurs uniques, RLE en plus possible



Compression (3)

- Dictionnaire pour chaque valeur ou bloc de valeurs

  Pratique lorsque répétitions de motifs



Cas d'utilisation

- Stockage de logs d'évènements

  Changement d'états ou erreurs relevés dans une application

- Billets d'un blog dans le cadre d'un CMS

  Tags, catégories, liens... dans différentes colonnes d'une famille

- Compter et catégoriser les visiteurs d'une page web

  Utilisation d'une colonne particulière de type compteur

Cas de non utilisation

- Problèmes pour qui ACID doit être satisfait en lecture/écriture

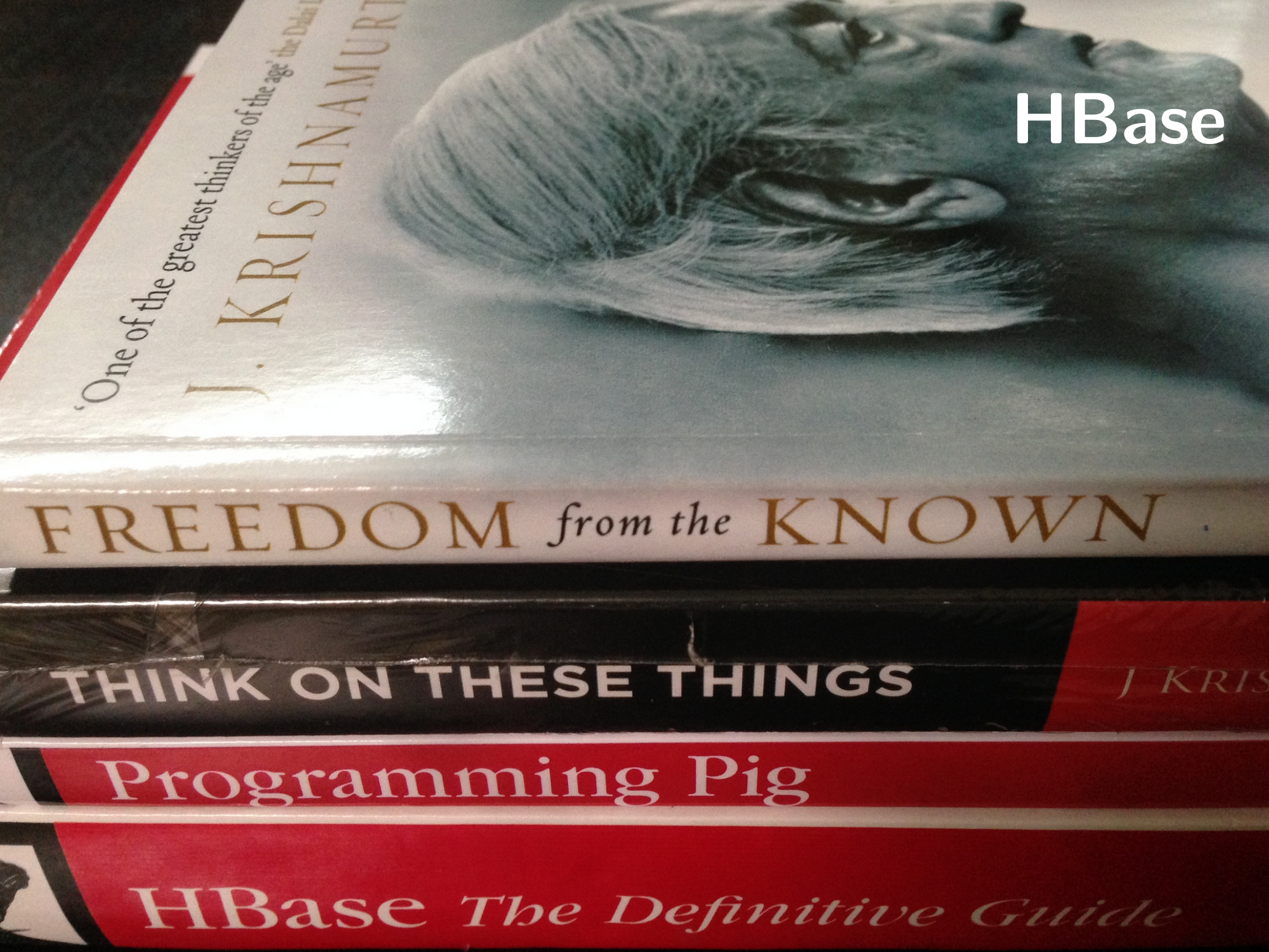
  Ne permet pas de faire des transactions ACID

- Requêtes d'agrégation des données (SUM, AVG...)

  Nécessite d'abord de rapatrier toutes les rangées côté client

- Ne pas utiliser dans une phase de prototypage

  Le design des familles de colonnes change avec les requêtes à faire


HBase
'One of the greatest thinkers of the age'
J. KRISHNAMURTI
FREEDOM from the KNOWN
THINK ON THESE THINGS
J KRIS
Programming Pig
HBase The Definitive Guide

HBase

- Implémentation libre du moteur BigTable de Google

  Fait partie du projet Hadoop de Apache

- Exécution par dessus le système de fichiers HDFS

  Stockage de données creuses en étant tolérant aux pannes

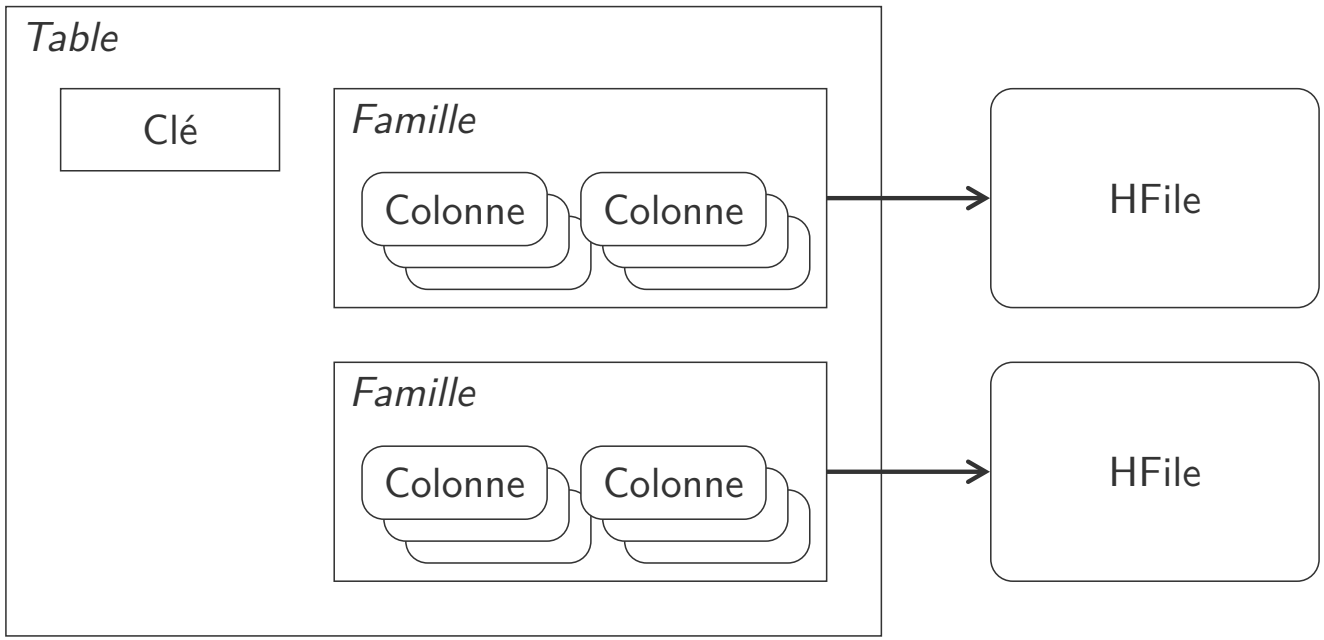
- Une base peut servir d'input/output de MapReduce (Hadoop)

  Couche SQL possible grâce à Apache Phoenix

Modèle de données

- Ensemble de familles de colonnes versionnées

  Colonnes d'une famille stockées ensemble dans un HFile



Chemin pour trouver une valeur : Table → Clé → Famille → Colonne → Timestamp

Architecture (1)

- Basé sur Hadoop et HDFS pour distribuer le stockage

  Combinaison de sharding et de réplication

- Sharding réalisé par des serveurs de région

  Découpe en plusieurs régions lorsqu'une table devient trop grosse

- Réplication assurée automatiquement par HDFS

  Fichier découpé en blocs répliqués avec un certain facteur

Architecture (2)

- Les données écrites passent par plusieurs étapes
  - Première gestion dans un WAL (*Write-Ahead Log*)
  - Placement des données dans un buffer nommé *memstore*
- Memstore écrit dans un HFile sur le HDFS lorsque trop gros

  Ensemble trié de clé-valeur sérialisé sur disque et immuable
- Suppression gérées à l'aide d'un marqueur *tombstone*

  Suppression effective au moment d'un compactage

Installation de HBase

- HBase est un programme développé en Java
- Plusieurs programmes proposés après installation
  - `start-hbase` est un script de lancement d'un serveur HBase
  - `stop-hbase` est un script de terminaison d'un serveur HBase
  - `hbase` permet de lancer plusieurs commandes de gestion
    - `hbase shell` propose un client en ligne de commande
    - `hbase thrift` démarre la passerelle Thrift

Lancement du serveur

- Lancement du serveur et vérification de la connexion

  Utilisation de `status` pour vérifier que tout va bien

```
& start-hbase.sh
```

```
& hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.2, r3f671c1ead70d249ea4598f1bbcc5151322b3a13, Fri Jul
  1 08:28:55 CDT 2016

hbase(main):001:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 2.0000
average load
```

Création d'une table

- Création d'une nouvelle table avec la commande `create`

  Spécification des familles de colonnes avec le nombre de versions

```
hbase(main):002:0> create 'students', {NAME => 'infos', VERSIONS
=> 1}, {NAME => 'registrations', VERSIONS => 2}
0 row(s) in 1.2230 seconds

=> Hbase::Table - students

hbase(main):003:0> list
TABLE
students
1 row(s) in 0.0630 seconds

=> ["students"]
```

Ajout d'une ligne

- Ajout des valeurs des différentes colonnes avec `put`

  En précisant chaque fois la famille de colonnes

```
hbase(main):004:0> put 'students', '16139', 'infos:firstname', '
Alexis'
0 row(s) in 0.1350 seconds

hbase(main):005:0> put 'students', '16139', 'infos:age', '22'
0 row(s) in 0.0120 seconds

hbase(main):006:0> put 'students', '16139', 'registrations:class
', '3BE'
0 row(s) in 0.0110 seconds

hbase(main):007:0> get 'students', '16139'
COLUMN                          CELL
 infos:age                      timestamp=1477172359150, value=22
 infos:firstname                timestamp=1477172339414, value=Alexis
 registrations:class            timestamp=1477172463762, value=3BE
3 row(s) in 0.0750 seconds
```

Nouvelle version d'une colonne

- On peut récupérer les différentes versions d'une colonne

  Utilisation de paramètres de la commande `get`

```
hbase(main):008:0> put 'students', '16139', 'registrations:note',
 'Pasterelle 4M'
0 row(s) in 0.0030 seconds

hbase(main):009:0> put 'students', '16139', 'registrations:note',
 'Passerelle 4M'
0 row(s) in 0.0030 seconds

hbase(main):010:0> get 'students', '16139', {COLUMN => '
registrations:note', VERSIONS => 2}
COLUMN                          CELL
 registrations:note             timestamp=1477173105470, value=
 Passerelle 4M
 registrations:note             timestamp=1477173102196, value=
 Pasterelle 4M
2 row(s) in 0.0110 seconds
```

Module Python happybase

- Module Python `happybase` pour interroger la base

  Passerelle `thrift` à démarrer avec `hbase thrift start`

```
1  import happybase
2
3  connection = happybase.Connection('localhost')
4  print(connection.tables())
5
6  table = connection.table('students')
7  print(table)
```

```
[b'students']
<happybase.table.Table name=b'students'>
```

Insertion de colonnes

- Insertion de colonnes avec la méthode `put` de la table

  Les différentes colonnes sont fournies par un dictionnaire

- Récupération des colonnes d'une ligne avec la méthode `row`

```
1 table.put('10003', {
2     'infos:firstname': 'Damien',
3     'infos:sex': 'M',
4     'registrations:class': '5MIN'
5 })
6 print(table.row('10003'))
```

```
{b'infos:sex': b'M', b'infos:firstname': b'Damien', b'
registrations:class': b'5MIN'}
```

Récupération de colonnes

- Récupération d'une ligne avec `row` et plusieurs avec `rows`

On peut filter uniquement les colonnes que l'on désire

```
1 users = [b'16139', b'10003']
2 classes = {}
3 rows = table.rows(users, columns=[b'infos:firstname', b'
  registrations:class'])
4 for key, value in rows:
5     students = classes.setdefault(value[b'registrations:class'],
      set())
6     students.add(value[b'infos:firstname'])
7 print(classes)
```

```
{b'5MIN': {b'Damien'}, b'3BE': {b'Alexis'}}
```


Cassandra

Cassandra

- Développé à l'origine par Facebook et libéré en 2008

  Fait maintenant partie du giron de Apache

- Base rapide et scalable, réplication peer-to-peer sur le cluster

  Serveurs puissance moyenne, pas d'unique point de défaillance

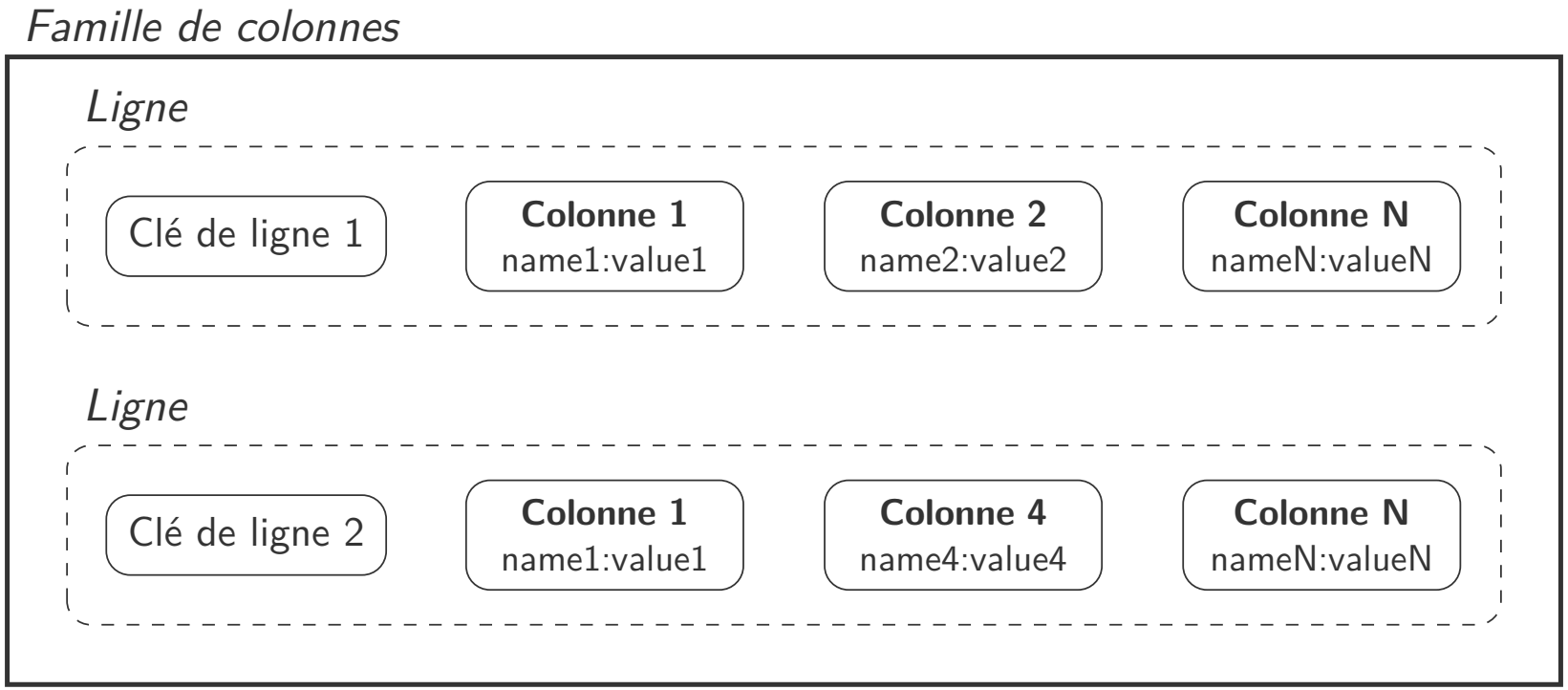
- Langage d'interrogation Cassandra Query Language (CQL)

  Variante du SQL pour interroger les keyspaces Cassandra

Modèle de données

- Ensemble de familles de colonnes contenant des lignes

  Les lignes peuvent contenir différentes colonnes de la famille



Colonne

- Une colonne est une paire nom-valeur avec un timestamp

  Le nom de la colonne joue également le rôle de clé

- Le timestamp définit la durée de vie de la colonne

  Et résolution de conflits d'écriture, données périmées...

```
{
  name: "Firstname",
  value: "Alexis",
  timestamp: 1234567890
}
```

```
{
  name: "Class",
  value: "3BE",
  timestamp: 1234567890
}
```

Famille de colonnes standard

- Une ligne est une collection de colonnes

  Une clé est attachée à cette collection de colonnes

- Une famille de colonnes est une collection de lignes similaires

  Les colonne sont simples, juste un nom et une valeur

```
{
  alexis: {                    # ligne avec 3 colonnes, clé "alexis"
    Firstname: "Alexis",
    Class: "3BE",
    Age: 22
  },
  damien: {                    # ligne avec 3 colonnes, clé "damien"
    Firstname: "Damien",
    Class: "5MIN",
    Sex: "M"
  }
}
```

Super colonne

- La valeur d'une super colonne est un map

  « Plusieurs colonnes » comme valeur d'une colonne

- Une super colonne est un conteneur de colonnes

  Chaque colonne contenue possède un timestamp

```
{
  name: "E3060",
  value: {
    name: "Electronique analogique",
    coordinator: "Gueuning",
    credits: 5
  },
  timestamp: 1234567890
}
```

Famille de super colonnes

- Une famille de super colonnes rassemble des super colonnes

Attention que Cassandra rapatrie tout, pas toujours optimal

```
{
  3BE: {
    E3060: {
      name: "Electronique analogique",
      coordinator: "Gueuning",
      credits: 5
    },
    E3090: {
      name: "Electromécanique appliquée",
      coordinator: "Kimplaire",
      credits: 5
    }
  },
  5MIN: {
    I4020: {
      name: "Architecture logicielle",
      credits: 3
    }
  }
}
```

Keyspace

- Cassandra organise les familles de colonnes en keyspaces

  Agit comme un espace de nom pour les familles de colonnes

- Similaire à la notion de base des moteurs relationnels

  Rassemblement des familles liées à une même application

Installation de Cassandra

- Cassandra est un programme développé en Java
- Plusieurs programmes proposés après installation
  - `cassandra` permet de démarrer un serveur Cassandra
  - `cqlsh` est un client en ligne de commande
  - `nodetool` donne des informations sur le serveur Cassandra

Lancement du serveur

- Lancement du serveur et vérification de la connexion

  Indication immédiate de si un serveur a été trouvé

```
& cassandra
```

```
& cqlsh
Connected to Test Cluster at localhost:9042.
[cqlsh 5.0.1 | Cassandra 3.7 | CQL spec 3.4.2 | Native protocol
v4]
Use HELP for help.
cqlsh>
```

Exécution d'une requête

- Obtention d'informations sur le cluster avec une requête CQL

  Récupération d'informations à partir de la table `system.local`

- Similarité très grande avec les requêtes SQL

```
cqlsh> SELECT cluster_name, listen_address FROM system.local;

 cluster_name | listen_address
--------------+----------------
 Test Cluster |      127.0.0.1

(1 rows)
```

Information sur la base

- Obtention d'informations avec la commande `DESCRIBE`

  Description du `cluster`, des `keyspaces`, `tables`...

```
cqlsh> DESCRIBE CLUSTER;

Cluster: Test Cluster
Partitioner: Murmur3Partitioner

cqlsh> DESCRIBE KEYSPACES;

system_traces  system_schema  system_auth  system
system_distributed

cqlsh> DESCRIBE TABLES;

Keyspace system_traces
----------------------
events  sessions
[...]
```

Création d'un keyspace

- Création d'un nouveau keyspace avec `CREATE KEYSPACE`

  Configuration des propriétés du keyspace, par exemple réplication

- Exemple avec simple réplication avec un facteur donné

```
cqlsh> CREATE KEYSPACE myschool
   ... WITH replication={'class': 'SimpleStrategy', '
   replication_factor': 3};

cqlsh> DESCRIBE keyspaces;

myschool  system_schema  system_auth  system  system_distributed
 system_traces

cqlsh> USE myschool;
cqlsh:myschool>
```

Création d'une table

- Création d'une nouvelle table avec `CREATE TABLE`

  Définition des différentes colonnes de la table

- Clé primaire pour identifier les lignes de manière unique

```
cqlsh:myschool> CREATE TABLE students (
           ... serial int PRIMARY KEY,
           ... firstname text,
           ... class text,
           ... age int,
           ... sesque text
           ... );

cqlsh:myschool> SELECT * FROM students;

 serial | age | class | firstname | sesque
--------+-----+-------+-----------+--------

(0 rows)
```

Ajout et suppression de colonnes

- La structure d'une table est modifiable avec `ALTER TABLE`

  Possibilité d'ajouter et de supprimer des colonnes

- Exemple de correction de la colonne `sesque` en `sex`

```
cqlsh:myschool> ALTER TABLE students DROP sesque;

cqlsh:myschool> ALTER TABLE students ADD sex text;

cqlsh:myschool> SELECT * FROM students;

 serial | age | class | firstname | sex
--------+-----+-------+-----------+-----

(0 rows)
```

Ajout d'une ligne

- Ajout d'une ligne dans la table avec `INSERT INTO`

  Spécification des colonnes pour lesquelles on a une valeur

- Exemple d'ajout d'Alexis dans la table `students`

```
cqlsh:myschool> INSERT INTO students (serial, firstname, class,
age)
            ... VALUES (16139, 'Alexis', '3BE', 22);
cqlsh:myschool> SELECT * FROM students;

 serial | age | class | firstname | sex
--------+-----+-------+-----------+------
  16139 |  22 |   3BE |    Alexis | null

(1 rows)
```

Autres opérations CRUD

- Trois autres opérations CRUD comme en SQL
  - **Mise à jour** de lignes

    `UPDATE table SET n1=v1, n2=v2... WHERE cond`
  - **Lecture** de lignes

    `SELECT c1, c2... FROM table WHERE cond`
  - **Suppression** de lignes

    `DELETE c1, c2... FROM table WHERE cond`
- Opération sur une seule ligne avec une condition sur sa clé

  Ne pas spécifier `c1, c2...` agit sur toute une colonne

Module Python cassandra

- Module Python `cassandra` pour interroger la base

  Création d'un cluster et connexion sur un keyspace

```
1  from cassandra.cluster import Cluster
2
3  cluster = Cluster(['127.0.0.1'])
4  session = cluster.connect('myschool')
5
6  print(cluster)
7  print(session)
```

```
<cassandra.cluster.Cluster object at 0x1096af240>
<cassandra.cluster.Session object at 0x10a6bed30>
```

Exécution d'une requête

- Utilisation de la méthode `execute` sur la session

  Exécuter une requête CQL et récupérer un tuple nommé

- La colonne `class` ne sera pas accessible comme un champ

  Car en conflit avec la propriété `class` de Python

```
1 rows = session.execute('SELECT * FROM students')
2 for row in rows:
3     print(row)
4     print('=> {} ({} ans)'.format(row.firstname, row.age))
```

```
Row(serial=16139, age=22, field_2_='3BE', firstname='Alexis', sex
=None)
=> Alexis (22 ans)
```

Construction d'une requête

- Requête en insérant des valeurs dans une chaine de caractères

  Similaire à l'utilisation des chaines formatées

```
1  session.execute(
2      '''
3      INSERT INTO students (serial, firstname, class, sex)
4      VALUES (%s, %s, %s, %s)
5      ''',
6      (10003, 'Damien', '5MIN', 'M')
7  )
8
9  rows = session.execute('SELECT * FROM students')
10 for row in rows:
11     print(row)
```

```
Row(serial=10003, age=None, field_2_='5MIN', firstname='Damien',
sex='M')
Row(serial=16139, age=22, field_2_='3BE', firstname='Alexis', sex
=None)
```

Requête préparée

- Construction d'une requête préparée avec la méthode `prepare`

  Exécution ensuite avec la méthode `execute`

- Autoriser la recherche sur une colonne avec `ALLOW FILTERING`

```
1  search_class = session.prepare('SELECT class FROM students WHERE
   firstname=? ALLOW FILTERING')
2
3  users = ['Sylvain', 'Alexis', 'Charles', 'Damien']
4  classes = {}
5  for user in users:
6      rows = session.execute(search_class, [user])
7      for row in rows:
8          students = classes.setdefault(row[0], set())
9          students.add(user)
10 print(classes)
```

```
{'3BE': {'Alexis'}, '5MIN': {'Damien'}}
```

Crédits

- Photos des logos depuis Wikipédia
- https://www.flickr.com/photos/zolakoma/2847597889
- https://www.flickr.com/photos/balusss/14004726607
- https://www.flickr.com/photos/110777427@N06/14184365994