

Semifir

CASSANDRA

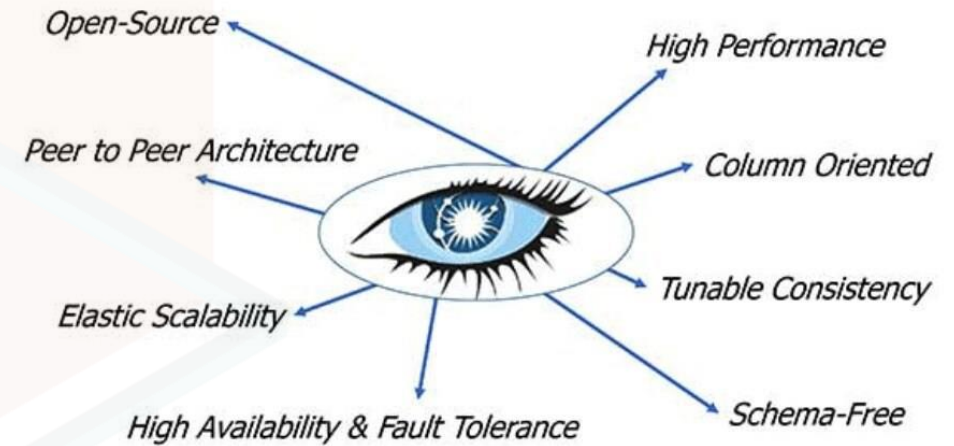
semifir.com
03 20 52 20 68
13 Avenue du Président John F. Kennedy,
59000 Lille.
contact@semifir.com

APACHE CASSANDRA

Cassandra est une base données initialement créée par Facebook:

- Un **SGBD** (Système de gestion de base de données)
- Une base de données Open source
- Axé non relationnel dit "**NoSQL**"
- Conçue pour gérer des quantités colossales de données
- Une technologie Big Data à faible latence.

Cassandra characteristics



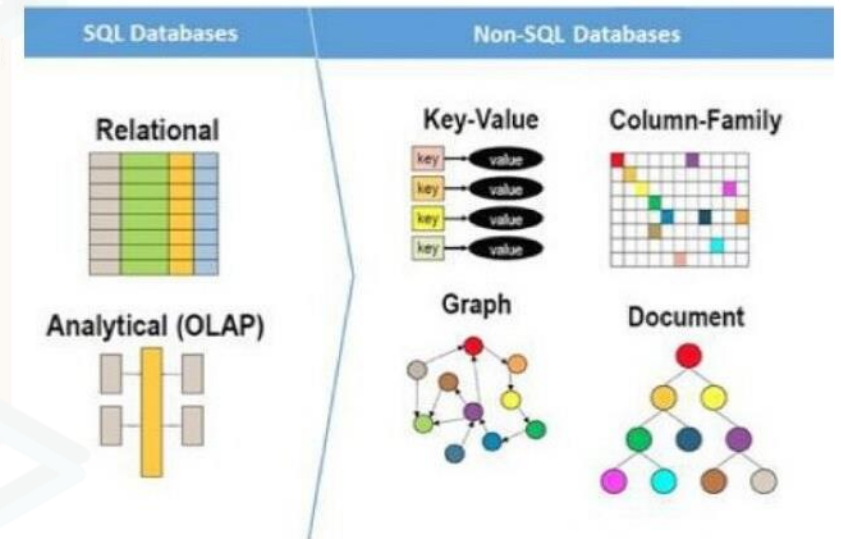
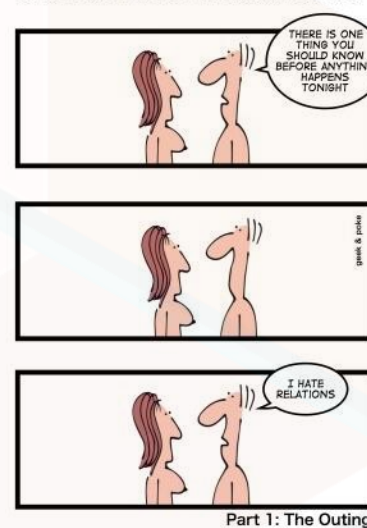
Semifir

LE NoSQL

Les SGBD NoSQL respectent le théorème suivant:

- **Cohérence:** la base de données doit être cohérente après chaque opération.
- **Disponibilité:** Le système doit toujours être disponible, même en cas de panne.
- **Tolérance au partitionnement:** Le système continue de fonctionner même en cas de problème de communication. Les serveurs sont partitionnés en plusieurs groupes.

The Hard Life of a NoSQL Coder



Cassandra en production

Les entreprises choisissent Cassandra lorsque leur projet demandent une grande scalabilité et une haute disponibilité.

Cassandra propose:

- Une réplication complète de la base de données
- Disponibilité globale à faible latence
- Evolutivité sur le matériel de base
- Augmentation du débit linéaire avec chaque node supplémentaire
- Equilibrage de charge

Cassandra in Production



75 000 nodes

10 PB



100 nodes

250 PB

NETFLIX

2 500 nodes

420 PB

1 000 000 000 000 reqs / day

11 500 000 reqs / sec











Semifir

La concurrence

Cassandra est un grand acteur du Big Data et du NoSQL.

Voici quelques SGBD:

| Type | Example | |
|-------------------|---|--|
| Key-Value Store |  redis |  riak |
| Wide Column Store |  HBASE |  cassandra |
| Document Store |  mongoDB |  CouchDB relax |
| Graph Store |  Neo4j |  InfiniteGraph The Distributed Graph Database |



CASSANDRA EN PRODUCTION

Cassandra est basé sur Amazon Dynamo et Google Big Table, des SGBD NoSQL mise à disposition par les deux cloud providers (AWS, GCP).

On retrouve Apache Cassandra sur <https://cassandra.apache.org>

- Cassandra gardera de bonne performances, même avec un nombre conséquent de nodes.
- Assurance de haute disponibilité, les nodes ont tous le même rôle. Absence de point de défaillance.
- Les nodes ayant le même rôles, il devient simple de les remplacer ou ajouter.
- Cassandra s'écrit en CQL (Casssandra Query Language)

Snitch

Maintenant que nous savons que les nodes d'un cluster dans l'architecture Cassandra ont tous le même rôles, il faut bien un moyen pour que ces nodes sachent la topologie du cluster (Où sont les autres nodes, dans quel Datacenter, ...). C'est **Snitch**.

Quelques options pour définir **Snitch** dans un cluster:

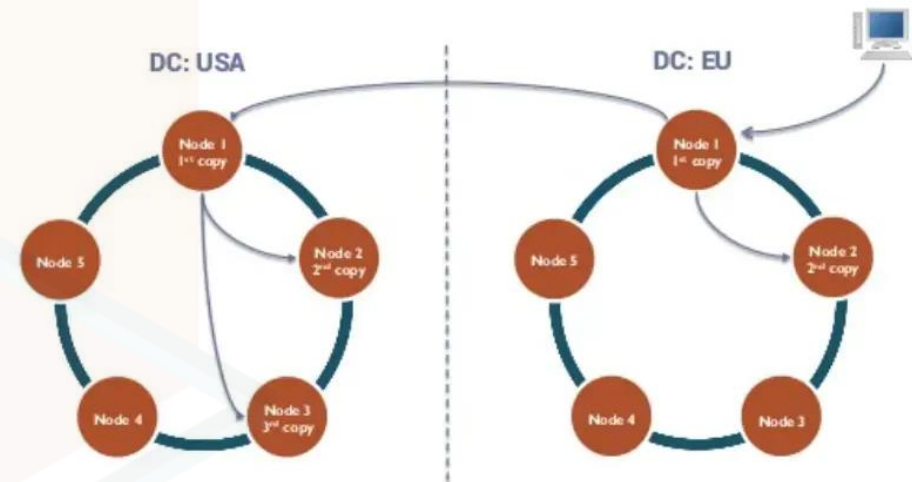
- **Simple Snitch:** Le snitch par défaut. Pour les clusters Cassandra d'un **même Datacenter**.
- **PropertyFileSnitch:** Pour chaque nodes du cluster, on peut spécifier son Datacenter.

Semifir

Gossip

- **Gossip** protocol qui indique la manière dont chaque nodes d'un cluster communiquent entre eux.
- Toutes les secondes, chaque node communique avec d'autres nodes (03 maximum) échangeant des informations sur lui et les nodes dont il a les informations.

```
CREATE KEYSPACE johnny WITH REPLICATION =  
{'class':'NetworkTopologyStrategy', 'USA':3, 'EU': 2};
```



Définitions

Quelques définitions:

- **Cluster:** Ensemble de serveurs liés par leur configuration sur un ou plusieurs Datacenter.
- **Nodes/peer:** serveur membre du cluster.
- **Keyspaces:** Espace logique dans lequel on va stocker une ou plusieurs tables. Equivalent en SQL d'une database ou d'un schéma (ProgreSQL).
- **Tables:** Affectées dans des keyspaces, sont composées de lignes et de colonnes.
- **Users:** Comptes ayant des accès, composé d'un login/password.

Semifir

Définitions

Quelques définitions:

- **Commitlog:** Elements écrits sur disque, requêtes d'écriture.
- **Memtables:** Données écrites en mémoire (Table de mémoire alimentée par la Java virtual Machine, **JVM**)
- **JVM:** Machine virtuelle Java lancée lors de l'initialisation de Cassandra.
- **SSTables:** Datafiles où les données sont stockées

Semifir

Principe de fonctionnement:

- **Coordinateur:** Node ayant reçu la requête en premier et déterminera qui sera le plus apte à traiter cette requête.
- **Replica:** Copie de données distribuées, stratégie de réplication par keyspaces.
- **Consistence:** Cassandra déterminera le nombre de réplicas à écrire pour que la requête aboutisse (tunable consistency).

Semifir

Suivi individuel:

La documentation Cassandra conseille un minimum de :

- 8 Gb à 32 Gb de RAM pour de la production
- 2 CPU

Pour nos tests nous utiliserons une image Docker de Cassandra.

Nous nous baserons sur le github suivant:

<https://github.com/bitnami/bitnami-docker-cassandra>

Semifir

Suivi individuel:

Pré requis:

- **Installation de docker engine**
- **Installation de docker-compose**
- **Bind un volume pour stocker les data de Cassandra**

Semifir

Suivi individuel:

Le docker-compose qu'on aura:

```
cassandra > docker-compose.yml
1  version: '3'
2
3  services:
4    cassandra:
5      image: bitnami/cassandra:4.0 # Image officielle cassandra
6      ports:
7        - 7000:7000 # communication inter-node
8        - 9042:9042 # connexion CQL client (port par défaut)
9        - 9160:9160 # Api client
10       - 7199:7199 # jmx, relevé de metrics
11      volumes:
12        - cassandra_data:/bitnami # Volume bind en local de data.
13      networks:
14        - cassandra_net # Réseau personnalisé de l'image
15      environment:
16        - CASSANDRA_SEEDS=cassandra # Nom du seed, qui est utilisée par Gossip pour amorcer de nouveaux nœuds rejoignant un cluster.
17        - CASSANDRA_PASSWORD_SEEDER=yes # Activer le mot de passe du seed
18        - CASSANDRA_PASSWORD=cassandra # Mot de passe du seed.
19  volumes:
20    cassandra_data: # Nom du volume à créer
21      driver: local # Volume utilisant le driver local
22      driver_opts:
23        o: bind
24        type: none
25        device: /data_docker/cassandra/ # Chemin local du volume bind
26  networks:
27    cassandra_net: # Nom du réseau à créer
28      driver: bridge
29      ipam:
30        config:
31          - subnet: 192.168.10.0/24 # Adresse réseau du bridge
```

Suivi individuel:

Procédure:

- On lance la commande **docker-compose up -d**

On exécute la commande `nodetool status`, qui nous listera les nodes et leur état.

`$ docker exec -it cassandra_cassandra_1 nodetool status`

```
alexandre@alexandre:~/cassandra$ docker exec -it cassandra_cassandra_1 nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address            Load            Tokens      Owns (effective)  Host ID                               Rack
UN  192.168.10.2        602.78 KiB      256         100.0%            9bb6759a-34b6-48ee-8bca-4c3ac9d57b97  rack1
```

Semifir

• Suivi individuel:

```
alexandre@alexandre:~/cassandra$ docker exec -it cassandra_cassandra_1 nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
-- Address            Load            Tokens      Owns (effective)  Host ID                               Rack
UN  192.168.10.2        602.78 KiB      256         100.0%            9bb6759a-34b6-48ee-8bca-4c3ac9d57b97  rack1
```

- Nous avons **01 node**
- Le node est dans le datacenter1
- La colonne **Load** représente la quantité de data contenue dans le **node**.
- La colonne **Address** indique les adresses IP de chaque **node**.
- La colonne **Owns (effective)** détermine la charge de données contenue dans le **node** (exemple, si on avait 04 nodes, on devrait avoir 25% sur chaque node)

Semifir

Suivi individuel:

```
alexandre@alexandre:~/cassandra$ docker exec -it cassandra_cassandra_1 nodetool status
Datacenter: datacenter1
=====
Status=Up/Down
-- State=Normal/Leaving/Joining/Moving
-- Address      Load       Tokens     Owns (effective)  Host ID                               Rack
UN 192.168.10.2   602.78 KiB  256        100.0%            9bb6759a-34b6-48ee-8bca-4c3ac9d57b97 rack1
```

On remarque le symbole "**UN**":

- Le symbole **U** nous permet de savoir que le node est **UP**.
- Le symbole **N** indique que le node est dans un état **normal**.

Semifir

Suivi individuel:

Essayons d'avoir des informations un peu plus spécifiques sur notre node.

```
alexandre@alexandre:~/cassandra/bitnami-docker-cassandra$ docker exec -it cassandra_cassandra_1 nodetool info
ID : 9bb6759a-34b6-48ee-8bca-4c3ac9d57b97
Gossip active : true
Native Transport active: true
Load : 602.78 KiB
Generation No : 1629277244
Uptime (seconds) : 20993
Heap Memory (MB) : 691.82 / 7936.00
Off Heap Memory (MB) : 0.00
Data Center : datacenter1
Rack : rack1
Exceptions : 0
Key Cache : entries 13, size 1.13 KiB, capacity 100 MiB, 134 hits, 151 requests, 0.887 recent hit rate, 14400 save period in seconds
Row Cache : entries 0, size 0 bytes, capacity 0 bytes, 0 hits, 0 requests, NaN recent hit rate, 0 save period in seconds
Counter Cache : entries 0, size 0 bytes, capacity 50 MiB, 0 hits, 0 requests, NaN recent hit rate, 7200 save period in seconds
Percent Repaired : 100.0%
Token : (invoke with -T/--tokens to see all 256 tokens)
```

Semifir

Keyspaces

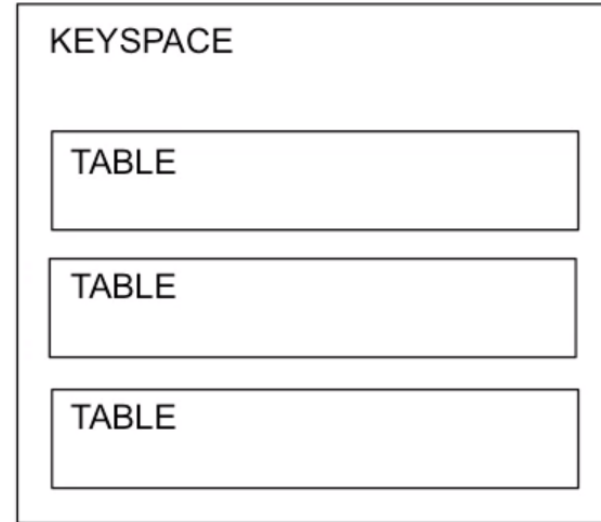
Dans les keyspaces, on peut définir des table.

Dans Cassandra, par défaut, on a quelques Keyspaces, des keyspaces systèmes.

Depuis notre hôte, on peut executer la commande

"docker exec -it cassandra cassandra 1 cqlsh --username cassandra --password cassandra"

Nous nous connectons à la Base de données avec l'utilisateur Cassandra et son mot de passe.



Semifir

Keyspaces

Une fois dans notre Cluster, on peut vérifier les **Keyspaces** qui existent en ce moment.

```
cassandra@cqlsh> DESCRIBE KEYSPACES  
  
system          system_distributed  system_traces  system_virtual_schema  
system_auth     system_schema       system_views  
  
cassandra@cqlsh> █
```

Nous pouvons vérifier par exemple le contenu de la **Keyspace system**

"DESCRIBE KEYSPACE system"

Semifir

Keyspaces

Autres commandes:

- Création d'un Keyspace: "CREATE KEYSPACE"

```
CREATE KEYSPACE my_keyspace  
WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
```

Nous avons un seul node pour le moment dans notre cluster, donc le "WITH REPLICATION" sera assez simple, sur un seul datacenter (le DC1) avec l'option "**replication_factor**" portée à "1".

Semifir

Keyspaces

- Vérification de la présence de notre Keyspace:

```
cassandra@cqlsh> DESCRIBE KEYSPACE my_keyspace  
  
CREATE KEYSPACE my_keyspace WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'} AND durable_writes = true;  
cassandra@cqlsh> █
```

- Suppression d'un Keyspace:

```
DROP KEYSPACE my_keyspace ;
```

Semifir

Keyspaces

- Création d'un KEYSPACE avec réplicat sur plusieurs DataCenter

```
CREATE KEYSPACE my_keyspace  
WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy' , 'dc1' : 3 'dc2' : 2 };
```

Ici, notre keyspace sera répliqué 3 fois sur notre DC1 et 2 fois sur notre DC2, grâce à la stratégie "**NetworkTopologyStrategy**".

Semifir

Keyspaces: TP

- Création d'un keyspace nommé "*home_security*" avec une stratégie de réplication "*SimpleStrategy*" de facteur 1.
- Utilisation de la commande "*DESCRIBE*" pour lister tous les keyspaces définis dans le cluster.
- Description de notre cluster "*home_security*"

Semifir

Keyspaces

- Création d'une table dans un **keyspace**:

```
CREATE TABLE alarm (  
  Home_id text,  
  Datetime timestamp,  
  Event text,  
  Code_used text,  
  ...
```

- Chaque colonnes de la table contient des **Types de données**. Comme en SQL il existe plusieurs types de données disponibles. La doc Cassandra sur le CQL contient des détails sur les data types.

https://docs.datastax.com/en/cql-oss/3.x/cql/cql_reference/cql_data_types_c.html

Semifir

Keyspaces

- Une table est assigné à un Keyspace.
- Pour créer une table, il faut tout d'abord spécifier le Keyspace dans le lequel elle sera créer.
- La commande **Use** permet de spécifier le keyspace dans lequel sera contenu notre Table.

```
Use my_keyspace;
```

- Comme en SQL, chaque table doit avoir une clé primaire, permettant d'identifier chaque enregistrement dans celle-ci, de manière unique. **Chaque enregistrement de cette clé primaire est UNIQUE.**

Semifir

Keyspaces

- Création d'une table dans un **keyspace**:

```
CREATE TABLE alarm (  
  Home_id text,  
  Datetime timestamp,  
  Event text,  
  Code_used text,  
  PRIMARY KEY ( Home_id, datetime )  
);
```

- Ici, notre table (alarm) aura deux colonnes comme primay key (home_id et datetime). Car en effet.
- **On ajoute la key datetime à home_id** car la colonne home_id aura parfois les même valeurs.

Semifir

Distribution des données

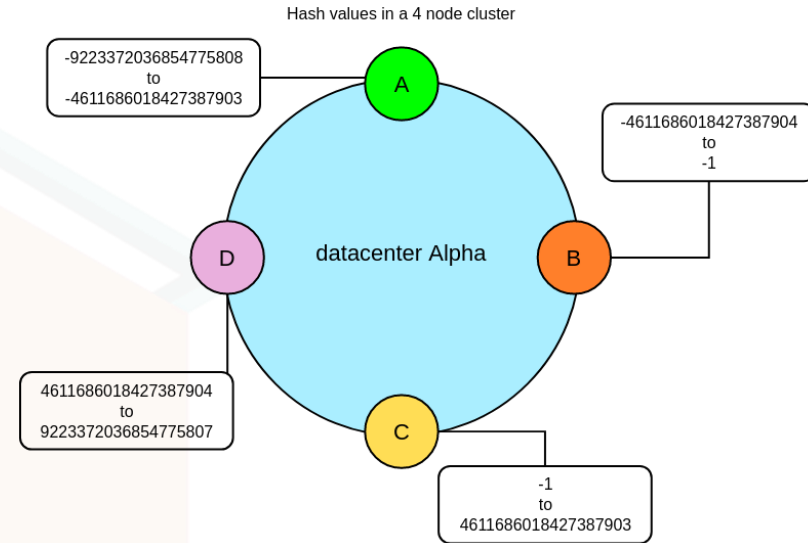
- La distribution de données permet d'avoir une répartition des données dans tout le cluster.
- Cette distribution se fait à travers un système de **hachage des données**.
- Au lieu d'avoir tous les enregistrements sur un seul Node, ils sont tous distribués à travers tout notre cluster.
- Dans les tables, le hachage se fait sur la première colonne de la table.

| home_id | datetime | event | code_used |
|-----------|---------------------|-----------|-----------|
| H01033638 | 2014-05-21 07:55:58 | alarm set | 2121 |
| H01545551 | 2014-05-21 08:30:14 | alarm set | 8889 |
| H00999943 | 2014-05-21 09:05:54 | alarm set | 1245 |

Semifir

Distribution des données

- La distribution des enregistrements à travers le cluster se fait à travers un **partitioner**.
- Le partitioner utilise un algorithme qui détermine le node sur lequel sera enregistrée une donnée précise.
- Le **partitioner** par défaut de Cassandra est **Murmur3**.
- Murmur3** prend la valeur de la première colonne de l'enregistrement et génère un numéro de hach unique entre -2^{63} et 2^{63} .



| home_id | datetime | event | code_used |
|-----------|---------------------|-----------|-----------|
| H01033638 | 2014-05-21 07:55:58 | alarm set | 2121 |
| H01545551 | 2014-05-21 08:30:14 | alarm set | 8889 |
| H00999943 | 2014-05-21 09:05:54 | alarm set | 1245 |

H01033638 -> **-7456322128261119046**
H01545551 -> **-2487391024765843411**
H00999943 -> **6394005945182357732**

Partition Key

- Dans notre table, la **partition key** est la valeur de la clé primaire avant le ",".

```
CREATE TABLE alarm (  
  Home_id text,  
  Datetime timestamp,  
  Event text,  
  Code_used text,  
  PRIMARY KEY ( Home_id, datetime )  
);
```

| home_id | datetime | event | code_used |
|-----------|---------------------|-----------|-----------|
| H01033638 | 2014-05-21 07:55:58 | alarm set | 2121 |
| H01545551 | 2014-05-21 08:30:14 | alarm set | 8889 |
| H00999943 | 2014-05-21 09:05:54 | alarm set | 1245 |

- Les données sont stockées de manière à ce que tous les enregistrements ayant la même clé de partition sont dans la même partition. (ou RowKey).

Stockage de data

- Une table peut afficher des données en ascendant (par défaut) ou en descendant.
- Par exemple, dans notre table (alarm), nous voudrions afficher les évènements les plus récents.

```
CREATE TABLE alarm (  
  Home_id text,  
  Datetime timestamp,  
  Event text,  
  Code_used text,  
  PRIMARY KEY ( Home_id, datetime )  
 ) WITH CLUSTERING ORDER BY  
 (datetime DESC);
```

Semifir

Importation de fichier

- Pour importer des enregistrement dans un table on utilise la commade *INSERT INTO*.
- Cette commande est utile pour écrire un seul enregistrement à la fois dans la table.

```
INSERT INTO alarm (home_id, datetime, event, code_used)  
VALUES ('H0147477', '2014-05-21 07:32:16', 'alarm set', '5599');
```

- La commande *SELECT* permet d'accéder à une table.
- Par exemple on peut utiliser la commande *SELECT FROM * alarm;* pour afficher tous les enregistrement de la table *alarm*.
- Pour limiter le nombre de colonnes à afficher *SELECT home_id, datetime, event FROM alarm;*

Semifir

Importation de fichier

- La copie d'un .csv pour importation de ses enregistrements vers une table se fait par la commande *COPY*.

COPY alarm (home_id, datetime, event, code_used)
FROM 'chemin vers le .csv'
WITH header = true AND delimiter = '|';

- Le délimiter dans le fichier .csv est indiqué
- On indique aussi s'il y'a un header dans notre fichier. Voici le contenu de notre *event.csv*

```
home_id|datetime|event|code_used
H02257222|2014-05-21 05:29:47|alarm set|1566
H01474777|2014-05-21 07:32:16|alarm set|5599
H01033638|2014-05-21 07:50:22|alarm set|2121
H01033638|2014-05-21 07:50:43|alarm turned off|2121
H01033638|2014-05-21 07:55:58|alarm set|2121
H01545551|2014-05-21 08:30:14|alarm set|8889
H00999943|2014-05-21 09:05:54|alarm set|1245
```

Stockage de data

- En CQL, la valeur des clés primaires rend l'enregistrement unique.

| home_id | datetime | event | code_used |
|-----------|---------------------|------------------|-----------|
| H01474777 | 2014-05-22 07:44:13 | alarm set | 5599 |
| H01474777 | 2014-05-21 18:30:33 | alarm turned off | 5599 |
| H01474777 | 2014-05-21 07:32:16 | alarm set | 5599 |

- En interne, la valeur de partition key rend le stockage des enregistrement unique.
- La partition Key contiendra tous les enregistrements qui correspondront à sa valeur.

| | | | |
|-----------|--|---|--|
| H01474777 | 2014-05-22 07:44:13 alarm set 5599 | 2014-05-21 18:30:33 alarm turned off 5599 | 2014-05-21 07:32:16 alarm set 5599 |
|-----------|--|---|--|

Stockage de data

- Vu des données stockées dans Cassandra grâce au cassandra-cli.



Semifir

TP 2

- Création d'un keyspace nommé "*home_security*"
- Création d'une table nommée *home* avec des colonnes nommée *home_id*, *address*, *city*, *state*, *zip*, *contact_name*, *phone*, *phone_password*, *email*, *main_code*, *guest_code*. **Toutes de type text.**
- Notre table aura pour clé primaire et clé de partition *home_id*.
- Utilisé la commande *DESCRIBE TABLE* pour avoir plus d'information sur la table.
- Ajout des données contenues dans le event.csv (fichier TP2) dans la table *alarm*.
- Utilisation de la commande *SELECT* pour afficher le contenu de la table *alarm*.

Semifir