cassandra

Banco NoSql Colunar

Carlos Eduardo Rossi Cubas da Silva

O banco de dados Apache Cassandra é um banco de dados NoSql escalável e de alta disponibilidade. Possui escalabilidade linear e tolerância a falhas em servidoresy ou infraestrutura em nuvem. Indicado para o uso em missões críticas.

https://goo.gl/nHQADK

Tolerante a falhas

Os dados são replicados automaticamente para vários nós para tolerância a falhas. A replicação em vários datacenters é suportada. Nós com falha podem ser substituídos sem tempo de inatividade.

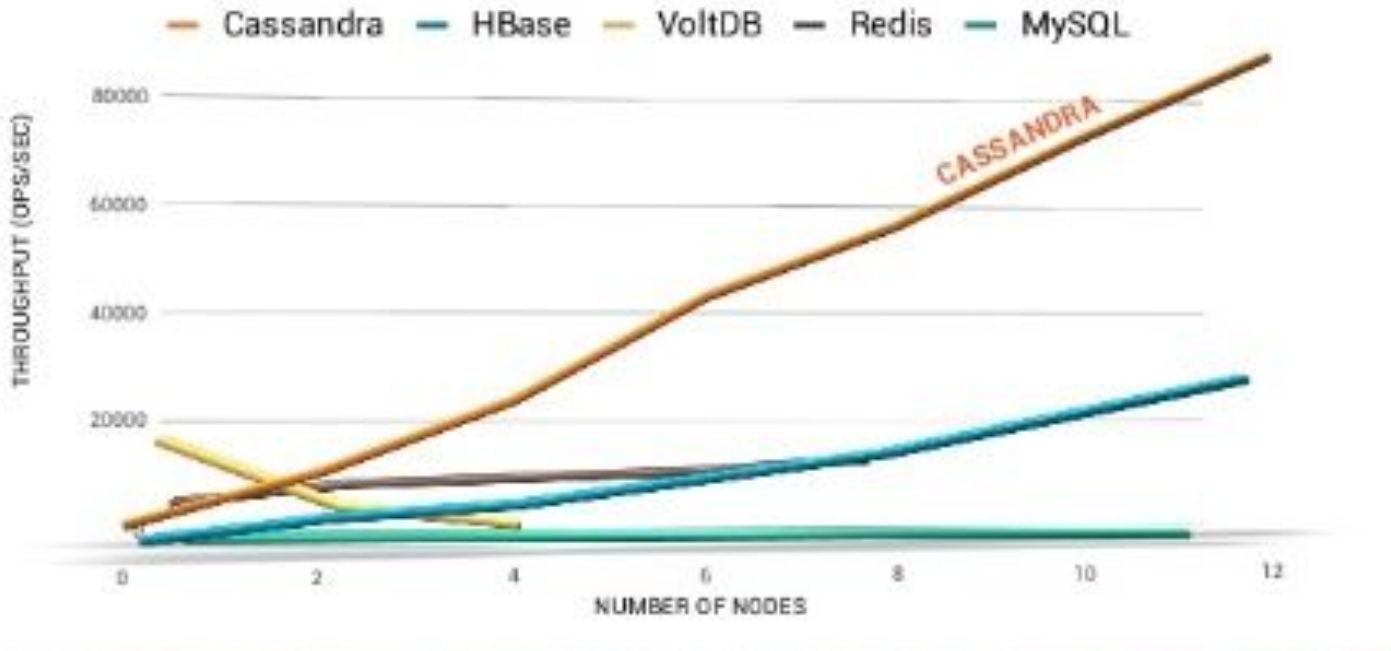
Descentralizado

Não há pontos únicos de falha. Não há gargalos de rede. Cada nó no cluster é idêntico.

Não existe normalização no Cassandra

A normalização se tornou muito popular em 1970, quando o armazenamento era muito caro, assim o desafio era conter a informação de forma econômica e não repeti-la. Atualmente o armazenamento está ficando cada vez mais barato e o desafio mudou: agora é lidar, por exemplo, com um número de requisição cada vez maior (na casa dos milhões, talvez bilhões).

Apache CASSANDRA™



O Cassandra possui uma escalabilidade linear, ou seja, quanto mais nós em seu datacenter, maior será o número de requisições por segundo.

Escalável

Algumas das maiores implantações de produção incluem:

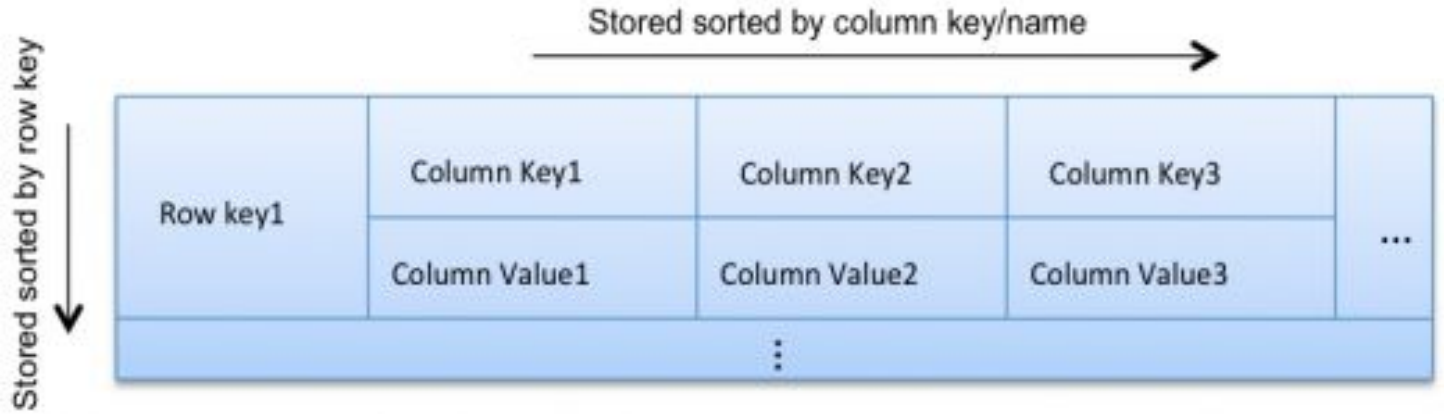
- **Apple** - com mais de 75.000 nós armazenando mais de 10 PB de dados,
- **Netflix** - 2.500 nós, 420 TB, mais de 1 trilhão de solicitações por dia,
- **Easou** (mecanismo de busca chinês) - 270 nós, 300 TB, mais de 800 milhões solicitações por dia,
- **eBay** - (mais de 100 nós, 250 TB).

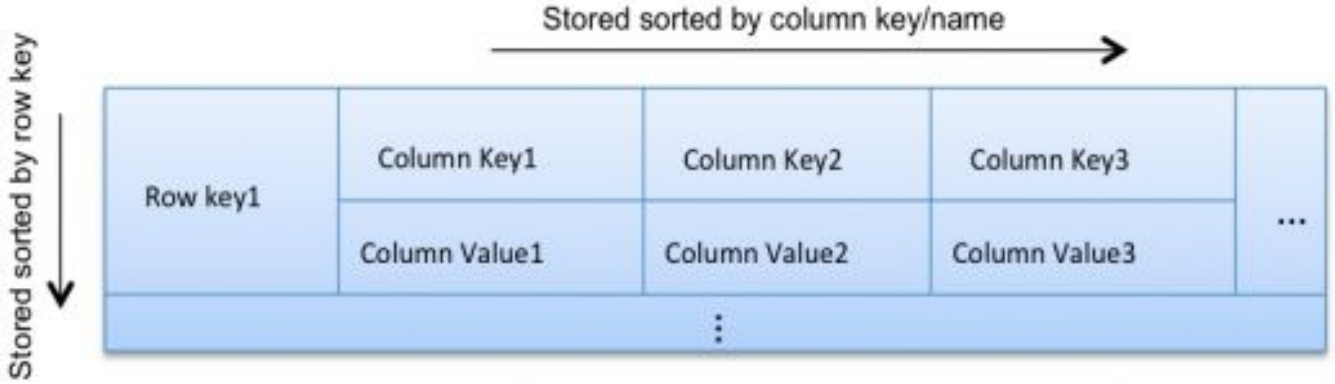
Analogias de termos com um banco relacional

Relational Model	Cassandra Model
Database	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

Exemplo de tabela colunar

As tabelas do Cassandra são como um mapa de arrays que possuem mapas de arrays: um mapa externo com chave de linha e um mapa interno com uma chave de coluna.



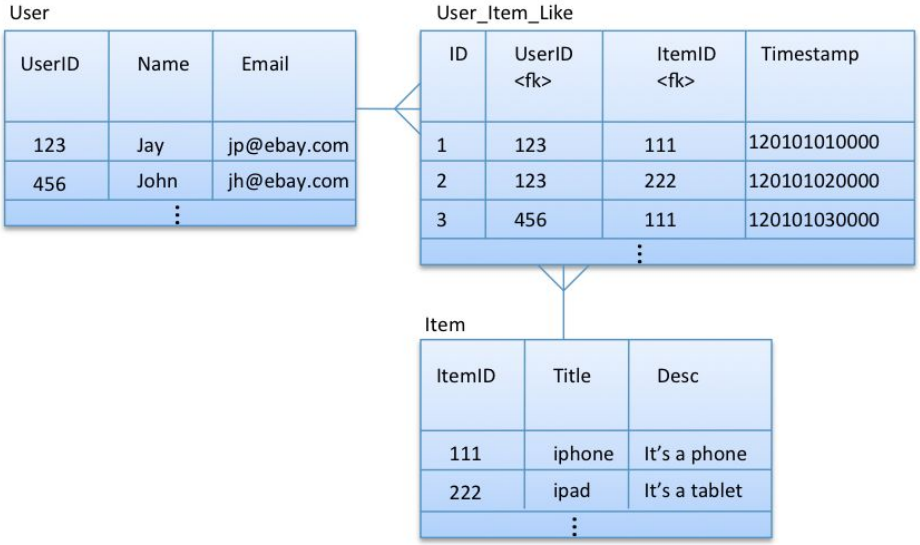


- O mapa fornece pesquisa de chave eficiente e a natureza ordenada fornece verificações eficientes.
- Podemos usar chaves de linha e chaves de coluna para fazer pesquisas eficientes e varreduras de intervalo.
- O número de chaves de coluna é ilimitado permitindo linhas largas.
- Coluna sem valor é permitida.

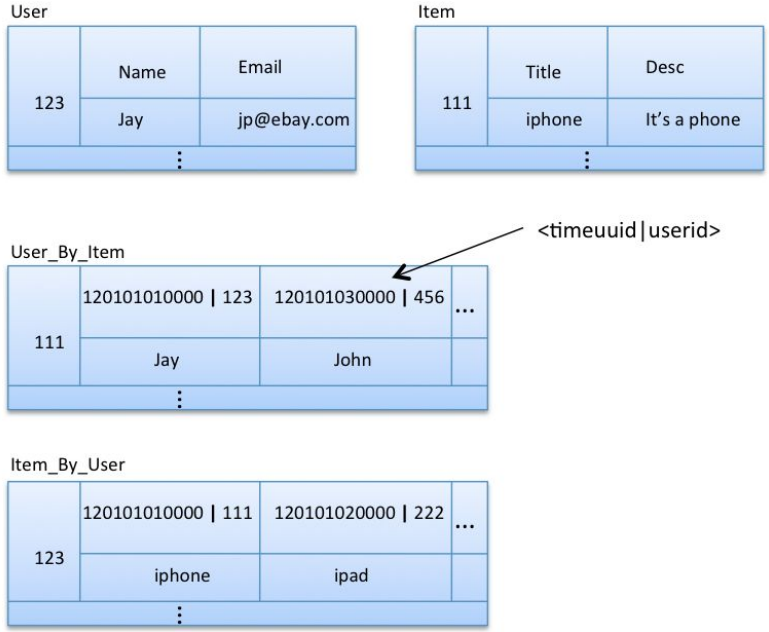
Apache CASSANDRA™

Diferença entre modelagens

Relacional



Colunas

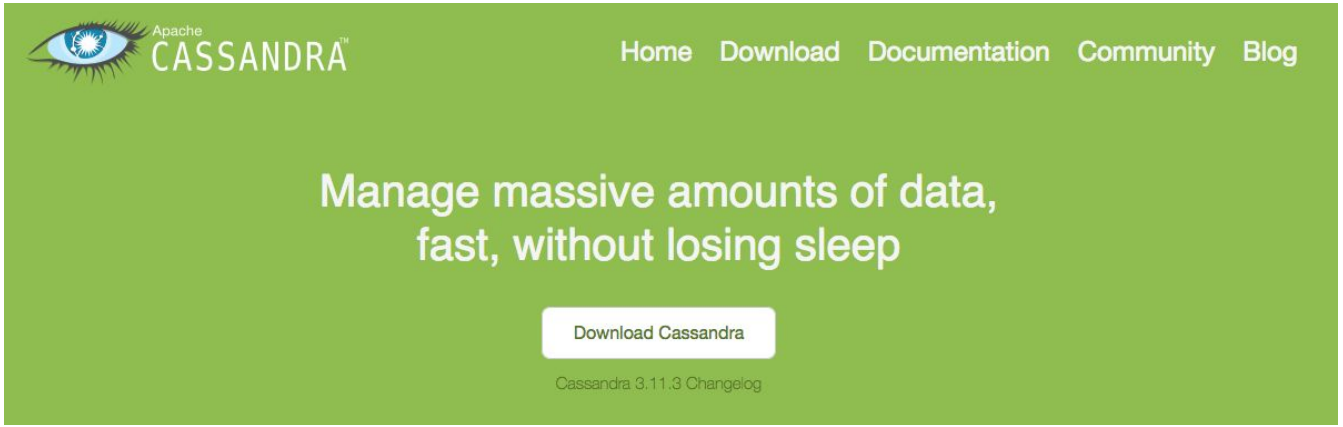


Vantagens em utilizar o Cassandra

- Alta disponibilidade;
- Performance;
- Extremamente tolerante a falhas;
- Escalabilidade linear: se o banco atende 100K de requisições, para atender 200K basta dobrar a infraestrutura;
- Sem nenhum ponto único de falha;
- Altamente distribuído;
- Suporta N datacenters nativamente.

Quando não utilizar o Cassandra

- Se precisar de muita consistência, a aplicação terá que garantir;
- Se o volume de dados ou o throughput da aplicação for muito pequeno;
- É necessário analisar se o modelo da aplicação suporta o paradigma colunar.



Instalando o Cassandra: http://downloads.datastax.com/community/
O Cassandra possui pacotes para distribuições Linux e Mac, e uma versão com instalador para Windows. O Cassandra depende apenas da Java Virtual Machine (JVM) para rodar, então é preciso ter o Java instalado e configurado antes de instalá-lo. Além disso, tendo a JVM instalada, é possível baixar a versão compilada do Cassandra e executá-la sem a necessidade de instalar via pacote.

Apache CASSANDRA™

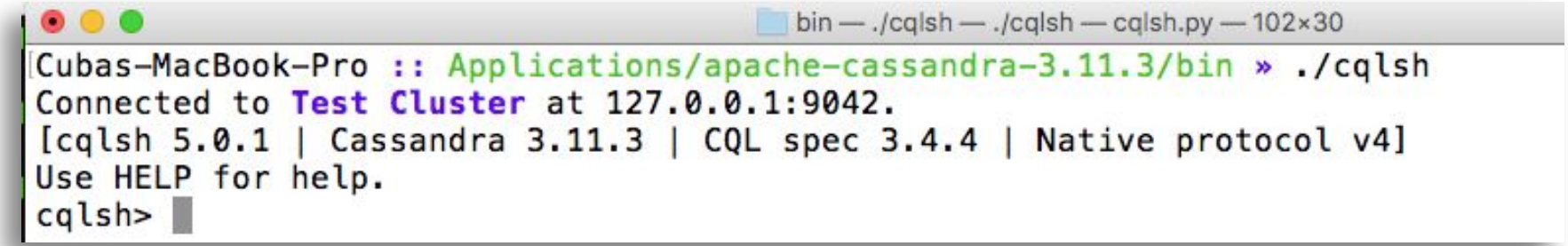
Instalando o Serviço do Banco de Dados

Para iniciar o Cassandra, devemos, depois da instalação, ir até a pasta bin e executar o comando **./cassandra -f**

```
[Cubas-MacBook-Pro :: Applications/apache-cassandra-3.11.3/bin » ./cassandra -f                ]
objc[1272]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVirtualMachines/jd
k1.8.0_11.jdk/Contents/Home/bin/java (0x100fd94c0) and /Library/Java/JavaVirtualMachines/jdk1.
8.0_11.jdk/Contents/Home/jre/lib/libinstrument.dylib (0x1010a14e0). One of the two will be use
d. Which one is undefined.
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.deserializeLargeSubset (
Lorg/apache/cassandra/io/util/DataInputPlus;Lorg/apache/cassandra/db/Columns;I)Lorg/apache/cas
sandra/db/Columns;
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubset (Lj
ava/util/Collection;ILorg/apache/cassandra/db/Columns;ILorg/apache/cassandra/io/util/DataOutpu
tPlus;)V
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubsetSize
 (Ljava/util/Collection;ILorg/apache/cassandra/db/Columns;I)I
CompilerOracle: dontinline org/apache/cassandra/db/commitlog/AbstractCommitLogSegmentManager.a
dvanceAllocatingFrom (Lorg/apache/cassandra/db/commitlog/CommitLogSegment;)V
CompilerOracle: dontinline org/apache/cassandra/db/transform/BaseIterator.tryGetMoreContents (
)Z
CompilerOracle: dontinline org/apache/cassandra/db/transform/StoppingTransformation.stop ()V
```

Acessando o Cassandra

Para interagir com o banco de dados, usaremos o novo *Cassandra Query Language (CQL)*, que pode ser usado pelo comando **cqlsh**



Criando Keyspace

Antes de criar as Column Families que são como tabelas dos bancos relacionais, temos que criar uma Keyspace. Dentro de um Keyspace, criamos as Column Families.

DESCRIBE KEYSPACES: Lista os keyspaces

CREATE KEYSPACE [nome] WITH REPLICATION = [estratégia de replicação]: Cria um novo keyspace.

Estratégia de replicação:
SimpleStrategy - estratégia mais simples que replica os dados nos servidores próximos

NetworkTopologyStrategy - mais complexa, permite replicação em réplicas por rack ou zonas diferentes de rede

Criando Keyspace

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

```
Cubas-MacBook-Pro :: Applications/apache-cassandra-3.11.3/bin » ./cqlsh     2 ↵
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.3 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> DESCRIBE KEYSPACES

system_schema  system                system_traces  ligado
system_auth    system_distributed    listamusicas

cqlsh>
```

Criando Tabelas

No Cassandra só existiam os Column Families, mas nas versões mais novas é usando o nome tabela para designar a estrutura de dados.

```
CREATE TABLE musicas (
  id uuid PRIMARY KEY,
  nome text,
  album text,
  artista text
);
```

```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.3 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> use listamusicas;
cqlsh:listamusicas> DESCRIBE TABLE musicas;

CREATE TABLE listamusicas.musicas (
    id uuid PRIMARY KEY,
    album text,
    artista text,
    nome text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTier
 '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cas
```

Manipulando registros

A sintaxe de manipulação dos dados com o Cassandra parede muito com o SQL dos bancos de dados relacionais

```
INSERT INTO nome_da_tabela(col1) VALUES (val1);

UPDATE nome_da_tabela SET col1 = value WHERE conditional;

DELETE FROM nome_da_tabela WHERE conditional;

SELECT * FROM nome_da_tabela WHERE conditional;
```

Manipulando registros

INSERT : `INSERT INTO musicas (id, nome, album, artista)`
`VALUES (blobAsUuid(timeuuidAsBlob(now())), 'Help', 'Help', 'Beatles');`

UPDATE : `UPDATE musicas SET nome='Help!', album='Help!'`
`WHERE id = e1f9b630-2997-11e9-8b09-c3d826fcb7e1;`

DELETE : `DELETE from musicas WHERE id =`
`a70ca7ff-6d57-4f89-be89-08421c432bb7;`

SELECT : `SELECT * FROM musicas`

Listando registros

```
INSERT INTO musicas (id, nome, album, artista)
VALUES (blobAsUuid(timeuuidAsBlob(now())), 'Help!', 'Help!', 'Beatles');
INSERT INTO musicas (id, nome, album, artista)
VALUES (blobAsUuid(timeuuidAsBlob(now())), 'Yesterday', 'Help!', 'Beatles');
INSERT INTO musicas (id, nome, album, artista)
VALUES (blobAsUuid(timeuuidAsBlob(now())),'Something', 'Abbey Road',
'Beatles');
INSERT INTO musicas (id, nome, album, artista)
VALUES (blobAsUuid(timeuuidAsBlob(now())),'Blackbird', 'The Beatles',
'Beatles');
```

```
cqlsh:listamusicas> select * from musicas;

 id                                   | album       | artista | nome
--------------------------------------+-------------+---------+-----------
 2c8157c0-2999-11e9-8b09-c3d826fcb7e1 |       Help! | Beatles |     Help!
 e1f9b630-2997-11e9-8b09-c3d826fcb7e1 |       Help! | Beatles |     Help!
 3959eac0-2999-11e9-8b09-c3d826fcb7e1 |  Abbey Road | Beatles | Something
 3d87b5f0-2999-11e9-8b09-c3d826fcb7e1 | The Beatles | Beatles | Blackbird
 33883390-2999-11e9-8b09-c3d826fcb7e1 |       Help! | Beatles | Yesterday
```

Listando registros

Buscando as músicas por artistas

```
SELECT * FROM musicas WHERE artista='Beatles';
```

```
cqlsh:listamusicas> SELECT * FROM musicas WHERE artista='Beatles';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involv
e data filtering and thus may have unpredictable performance. If you want to execute this query despite the perfor
mance unpredictability, use ALLOW FILTERING"
cqlsh:listamusicas>
```

Apesar de a sintaxe da busca ser idêntica ao SQL dos bancos relacionais, diferente deles, o Cassandra não faz buscas em campos que não possuem índices.

```
CREATE INDEX ON musicas (artista);
```

Listando registros

Buscando pela cláusula Like (case sensitive)

```
CREATE CUSTOM INDEX fn_nome ON musicas (nome) USING
'org.apache.cassandra.index.sasi.SASIIndex' WITH OPTIONS =
{'mode': 'CONTAINS', 'analyzer_class':
'org.apache.cassandra.index.sasi.analyzer.NonTokenizingAnalyzer',
'case_sensitive': 'false'};
```

```
SELECT * FROM musicas WHERE nome LIKE 'y%';
```

Armazenando playlists

Se optarmos por criar uma tabela para armazenar as playlists, poderíamos ter algo como uma tabela de ligação que teria o ***id da música***, ***id da playlist*** e ***posição***.

id_playlist	id_musica	posicao
1	1	1
1	2	2
1	4	3
1	3	4

Armazenando playlists

Um dos problemas nesta abordagem é que o Cassandra não possui **join** e com isso não teríamos com juntar informações de outra tabelas. Nesse caso, devemos partir para a **desnormalização**.

id_playlist	id_musica	posicao	nome_musica	album
1	2c8157c0-2999-11e9-8b09-c3d826fcb7e1	1	Something	Abbey Road
1	2c8157c0-2999-11e9-8b09-c3d826fcb7e1	2	Blackbird	The Beatles

Armazenando playlists

```
CREATE TABLE playlist_atual (
      id_playlist uuid PRIMARY KEY, posicao int,
      id_musica uuid, nome text, album text, artista text);
```

```
INSERT INTO playlist_atual (id_playlist, posicao, id_musica, nome, album,
    artista) VALUES (c4f408dd-00f3-488e-8800-050d2775bbc7, 1,
04b57c98-33df-11e5-a151-feff819cdc9f, 'Help!', 'Help!', 'Beatles');

INSERT INTO playlist_atual (id_playlist, posicao, id_musica, nome, album,
    artista) VALUES (c4f408dd-00f3-488e-8800-050d2775bbc7, 2,
1a8d6a80-33df-11e5-a151-feff819cdc9f, 'Yesterday', 'Help!', 'Beatles');
```

Armazenando playlists

Curiosamente, em vez de dois registros, temos apenas um. No Cassandra, tanto o INSERT quanto o UPDATE executam a operação upsert. Se já existe o registro, ele altera; caso contrário, ele cria.

```
cqlsh:listamusicas> INSERT INTO playlist_atual (id_playlist, posicao, id_musica, nome, album, artista) VALUES (c4f408dd-0
0f3-488e-8800-050d2775bbc7, 1,
                ... 04b57c98-33df-11e5-a151-feff819cdc9f, 'Help!', 'Help!', 'Beatles');
cqlsh:listamusicas> INSERT INTO playlist_atual (id_playlist, posicao, id_musica, nome, album, artista) VALUES (c4f408dd-0
0f3-488e-8800-050d2775bbc7, 2,
                ... 1a8d6a80-33df-11e5-a151-feff819cdc9f, 'Yesterday', 'Help!', 'Beatles');
cqlsh:listamusicas>
cqlsh:listamusicas> select * from playlist_atual;

 id_playlist                          | album | artista | id_musica                            | nome      | posicao
--------------------------------------+-------+---------+--------------------------------------+-----------+---------
 c4f408dd-00f3-488e-8800-050d2775bbc7 | Help! | Beatles | 1a8d6a80-33df-11e5-a151-feff819cdc9f | Yesterday |       2

(1 rows)
cqlsh:listamusicas>
```

Armazenando playlists

A solução será utilizar como chave primária da nossa tabela uma chave composta, com o id da playlist e a posição da música.

```
CREATE TABLE playlist_atual (
  id_playlist uuid,
  posicao int,
  id_musica uuid,
  nome text,
  album text,
  artista text,
  PRIMARY KEY (id_playlist, posicao)
 );
```

Armazenando playlists

```
INSERT INTO playlist_atual (id_playlist, posicao, id_musica, nome, album, artista) VALUES
(c4f408dd-00f3-488e-8800-050d2775bbc7, 1,04b57c98-33df-11e5-a151-feff819cdc9f, 'Help!',
'Help!', 'Beatles');

INSERT INTO playlist_atual (id_playlist, posicao, id_musica, nome, album, artista) VALUES
(c4f408dd-00f3-488e-8800-050d2775bbc7, 2,  1a8d6a80-33df-11e5-a151-feff819cdc9f,
'Yesterday', 'Help!', 'Beatles');
```

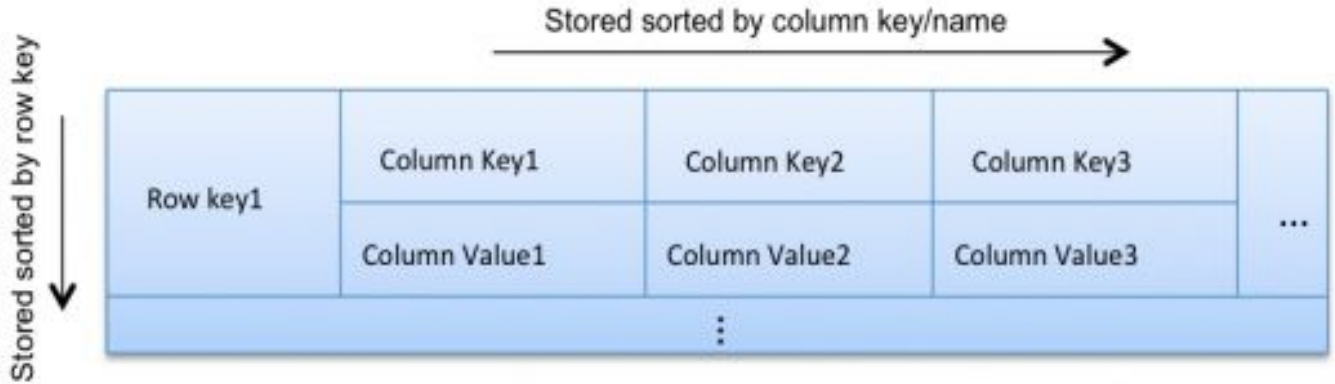
```
cqlsh:listamusicas> select * from playlist_atual;

 id_playlist                          | posicao | album | artista | id_musica                            | nome
--------------------------------------+---------+-------+---------+--------------------------------------+-----------
 c4f408dd-00f3-488e-8800-050d2775bbc7 |       1 | Help! | Beatles | 04b57c98-33df-11e5-a151-feff819cdc9f |     Help!
 c4f408dd-00f3-488e-8800-050d2775bbc7 |       2 | Help! | Beatles | 1a8d6a80-33df-11e5-a151-feff819cdc9f | Yesterday

(2 rows)
cqlsh:listamusicas>
```

Tabelas orientadas a colunas

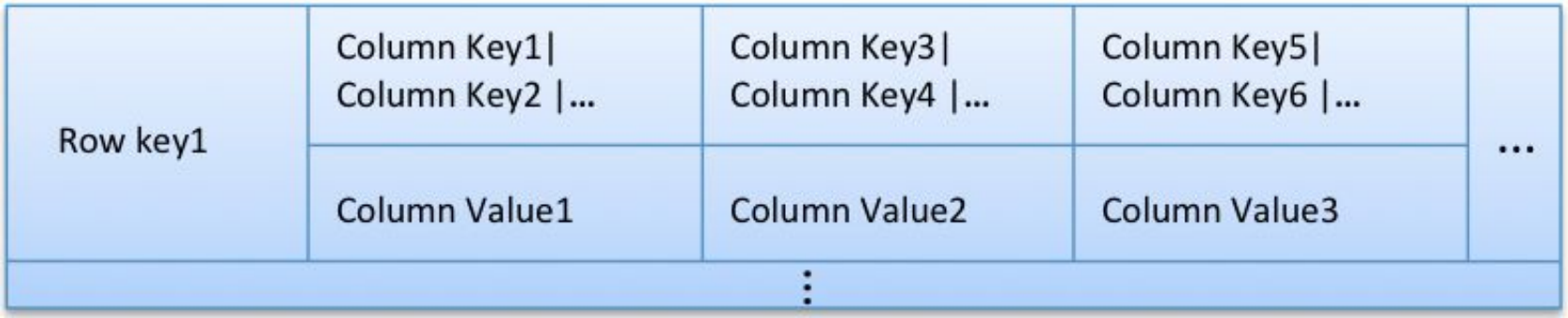
Embora os bancos colunares também utilizem tabelas, os registros são orientados a colunas, e não linhas. As tabelas são *schemaless* (sem esquema)



Um modelo de dado amplamente utilizado em bancos colunares são as *wide rows,* ou *dynamic columns*.

Tabelas orientadas a colunas

Nestas tabelas o que nós estamos acostumados a chamar de linha (row) é, na verdade, chamado de partição (partition)



Cada partição possui sua chave única (partition key) e, a partir desta chave, podemos adicionar vários pares chave/valor

Criando uma tabela orientada a colunas

```
CREATE TABLE playlist_versionada (
     id_playlist uuid,
     versao int,
     modificacao text,
     PRIMARY KEY (id_playlist, versao)
) WITH COMPACT STORAGE;
```

Criando uma tabela orientada a colunas

```
  INSERT INTO playlist_versionada (id_playlist, versao, modificacao)
VALUES (f449a4d0-2e4c-11e9-bca6-db70916459ff, 1, 'ADD(Help!)');

INSERT INTO playlist_versionada (id_playlist, versao, modificacao)
VALUES (f449a4d0-2e4c-11e9-bca6-db70916459ff, 2, 'ADD(Yesterday)');

INSERT INTO playlist_versionada (id_playlist, versao, modificacao)
VALUES (f449a4d0-2e4c-11e9-bca6-db70916459ff, 3, 'ADD(Blackbird)');

INSERT INTO playlist_versionada (id_playlist, versao, modificacao)
VALUES (f449a4d0-2e4c-11e9-bca6-db70916459ff, 4, 'ADD(Something)');

INSERT INTO playlist_versionada (id_playlist, versao, modificacao)
VALUES (f449a4d0-2e4c-11e9-bca6-db70916459ff, 5, 'ADD(Blackbird)');
```

Criando uma tabela orientada a colunas

Embora o resultado da consulta seja muito parecido com uma tabela de um banco relacional, os dados estão orientados a colunas internamente.

```
cqlsh:listamusicas> select * from playlist_versionada;

 id_playlist                          | versao | modificacao
--------------------------------------+--------+-----------------
 f449a4d0-2e4c-11e9-bca6-db70916459ff |      1 |     ADD(Help!)
 f449a4d0-2e4c-11e9-bca6-db70916459ff |      2 | ADD(Yesterday)
 f449a4d0-2e4c-11e9-bca6-db70916459ff |      3 | ADD(Blackbird)
 f449a4d0-2e4c-11e9-bca6-db70916459ff |      4 | ADD(Something)
 f449a4d0-2e4c-11e9-bca6-db70916459ff |      5 | ADD(Blackbird)
```

Como são armazenado os dados de uma tabela orientada a colunas

Existe apenas uma partição por isso o resultado abaixo mostra apenas 1

$ cassandra-cli: list playlist_versionada;

```
RowKey: f449a4d0-2e4c-11e9-bca6-db70916459ff
=> (name=1, value=ADD(Help!), timestamp=1438818548125646)
=> (name=2, value=ADD(Yesterday), timestamp=1438818553747085)
=> (name=3, value=ADD(Blackbird), timestamp=1438818571767466)
=> (name=4, value=ADD(Something), timestamp=1438818576933964)
=> (name=5, value=ADD(Blackbird), timestamp=1438818586229982)
                                                1 Row Returned.
```

Tipos especiais no Cassandra

No Cassandra existe a possibilidade de trabalhar com outros tipos de dados como listas, conjuntos e mapas. O tipo especial para conjuntos é o **set** e, quando vamos usá-lo, precisamos definir o que podemos armazenar neste conjunto.

```
ALTER TABLE musicas ADD tags set<text>;

UPDATE musicas SET tags = {'Beatles', '60s'} WHERE id =
2c8157c0-2999-11e9-8b09-c3d826fcb7e1;
```

```
 id                                   | album       | artista | nome      | tags
--------------------------------------+-------------+---------+-----------+--------------------
 2c8157c0-2999-11e9-8b09-c3d826fcb7e1 |       Help! | Beatles |     Help! | {'60s', 'Beatles'}
 e1f9b630-2997-11e9-8b09-c3d826fcb7e1 |       Help! | Beatles |     Help! |               null
 3959eac0-2999-11e9-8b09-c3d826fcb7e1 |  Abbey Road | Beatles | Something |               null
 3d87b5f0-2999-11e9-8b09-c3d826fcb7e1 | The Beatles | Beatles | Blackbird |               null
 33883390-2999-11e9-8b09-c3d826fcb7e1 |       Help! | Beatles | Yesterday |               null

(5 rows)
```

Tipos especiais no Cassandra

Para fazer uma busca pelos campos das tags, devemos criar um índice

```
CREATE INDEX ON musicas (tags);
```

Com o índice criado, podemos fazer a busca. A diferença é que, para filtrar dentro da coleção, temos de usar a palavra-chave CONTAINS.

```
SELECT * FROM musicas WHERE tags CONTAINS '60s';
```

```
cqlsh:listamusicas> SELECT * FROM musicas WHERE tags CONTAINS '60s';

 id                                   | album | artista | nome  | tags
--------------------------------------+-------+---------+-------+--------------------
 2c8157c0-2999-11e9-8b09-c3d826fcb7e1 | Help! | Beatles | Help! | {'60s', 'Beatles'}
```

Tipos especiais no Cassandra

Como em bancos colunares é comum utilizar desnormalização pode-se usar a abordagem de criar a tabela tag e incluir os ids das músicas.

```
CREATE TABLE tags (nome text PRIMARY KEY, musicas set<uuid>);

INSERT INTO tags (nome, musicas) VALUES ( '60s',
    {2c8157c0-2999-11e9-8b09-c3d826fcb7e1});
```

```
 nome | musicas
------+--------------------------------------
  60s | {2c8157c0-2999-11e9-8b09-c3d826fcb7e1}

(1 rows)
cqlsh:listamusicas>
```

Tipos especiais no Cassandra

Um detalhe importante de como trabalhar com as coleções é como adicionar um tag a uma lista quando não sabemos se esta já possui tags. Se usarmos

```
UPDATE musicas SET tags = {'classic rock'} WHERE id =
2c8157c0-2999-11e9-8b09-c3d826fcb7e1;
```

vamos sobrescrever a coleção toda. Para modificar uma coleção, devemos usar o estado atual e somar os valores que vamos adicionar.

```
UPDATE musicas SET tags = tags +  {'Classic Rock'}
 WHERE id = 2c8157c0-2999-11e9-8b09-c3d826fcb7e1;
```

```
id                                   | album      | artista | nome       | tags
-------------------------------------+------------+---------+------------+-----------------------------------
2c8157c0-2999-11e9-8b09-c3d826fcb7e1 |      Help! | Beatles |      Help! | {'60s', 'Beatles', 'Classic Rock'}
e1f9b630-2997-11e9-8b09-c3d826fcb7e1 |      Help! | Beatles |      Help! |                               null
```

Agrupando dados com o cassandra

A partir da versão 3.10 do Cassandra, é possível criar agrupamentos no nível da partição ou no nível da coluna. A sintaxe geral pode ser a seguinte:

```
SELECT partitionKey, max(value) FROM myTable GROUP BY
partitionKey;

SELECT partitionKey, clustering0, clustering1, max(value)
FROM myTable GROUP BY partitionKey, clustering0, clustering1;
```

Agrupando dados com o cassandra

```
CREATE TABLE temperature_by_day (
        weatherstation_id text,
        date text,
        event_time timestamp,
        temperature float,
        PRIMARY KEY
    ((weatherstation_id,date),event_time)
  );
```

Agrupando dados com o cassandra

```
INSERT INTO
temperature_by_day(weatherstation_id,date,event_time,temperature)
    VALUES ('1234WXYZ','2016-04-03','2016-04-03 07:01:00',73);

INSERT INTO
temperature_by_day(weatherstation_id,date,event_time,temperature)
    VALUES ('1234WXYZ','2016-04-03','2016-04-03 07:02:00',70);

INSERT INTO
temperature_by_day(weatherstation_id,date,event_time,temperature)
    VALUES ('1234WXYZ','2016-04-04','2016-04-04 07:01:00',73);

INSERT INTO
temperature_by_day(weatherstation_id,date,event_time,temperature)
    VALUES ('1234WXYZ','2016-04-04','2016-04-04 07:02:00',74);
```

Agrupando dados com o cassandra

```
select * from temperature_by_day;
```

weatherstation_id	date	event_time	temperature
1234WXYZ	2016-04-03	2016-04-03 10:01:00.000000+0000	73
1234WXYZ	2016-04-03	2016-04-03 10:02:00.000000+0000	70
1234WXYZ	2016-04-04	2016-04-04 10:01:00.000000+0000	73
1234WXYZ	2016-04-04	2016-04-04 10:02:00.000000+0000	74

```
SELECT weatherstation_id, date, MAX(temperature) FROM temperature_by_day
                    GROUP BY weatherstation_id, date;
```

weatherstation_id	date	system.max(temperature)
1234WXYZ	2016-04-03	73
1234WXYZ	2016-04-04	74

(2 rows)

Agrupando dados com o cassandra

```
SELECT weatherstation_id, date, MAX(temperature) FROM temperature_by_day
                GROUP BY weatherstation_id, date, event_time;
```

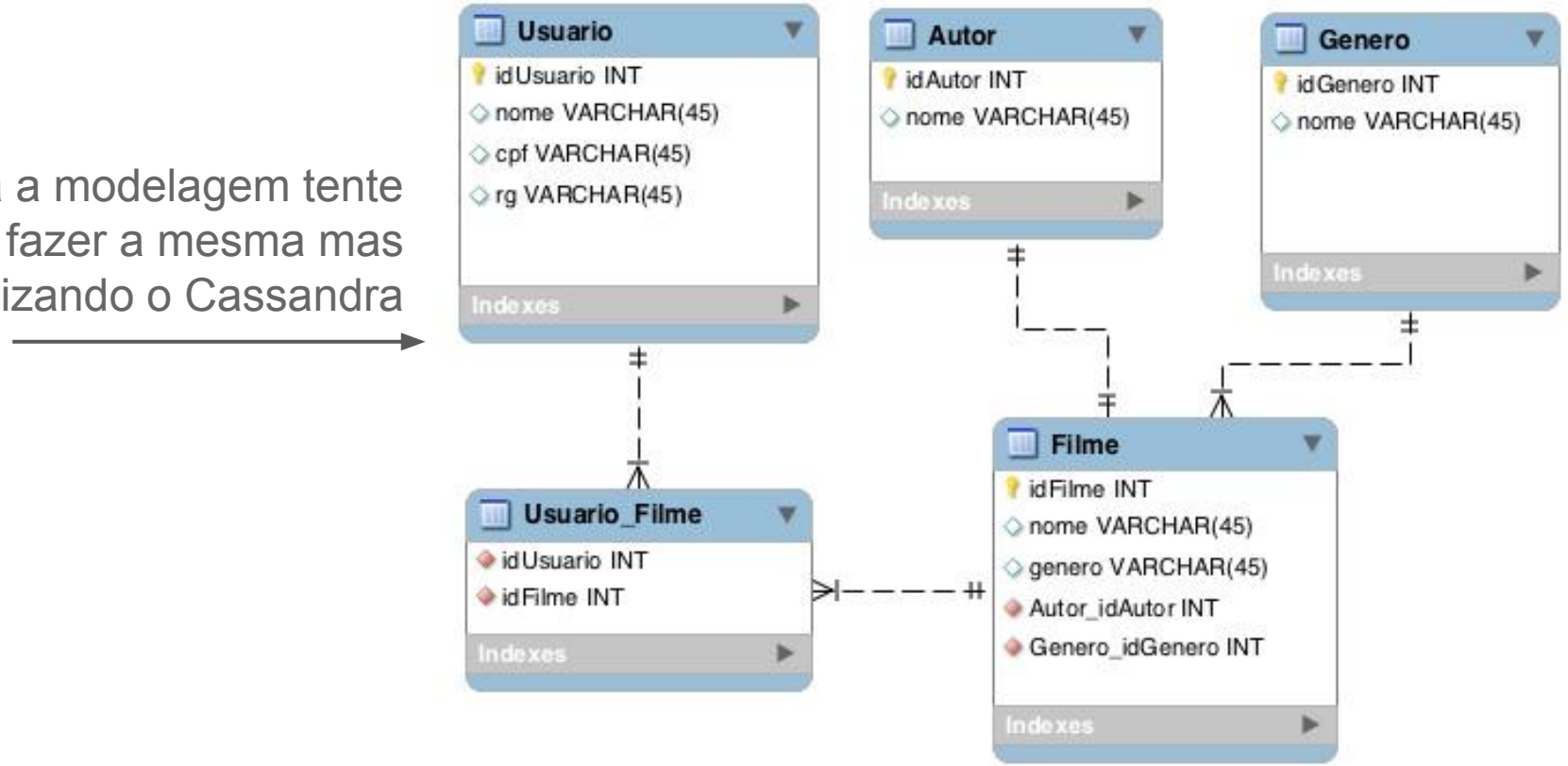
```
cqlsh:listamusicas> SELECT weatherstation_id, date, MAX(temperature) FROM temperature_by_day
                ...  GROUP BY weatherstation_id, date, event_time;

 weatherstation_id | date       | system.max(temperature)
-------------------+------------+-------------------------
          1234WXYZ | 2016-04-03 |                      73
          1234WXYZ | 2016-04-03 |                      70
          1234WXYZ | 2016-04-04 |                      73
          1234WXYZ | 2016-04-04 |                      74

(4 rows)
```

Apache CASSANDRA™

Dada a modelagem tente fazer a mesma mas utilizando o Cassandra



Estudos de caso

Ebay:
https://www.datastax.com/resources/casestudies/ebay

NetFlix:
https://www.datastax.com/personalization/netflix

Spotify:
https://www.datastax.com/resources/casestudies/case-study-spotify

Link Para a Apresentação

https://goo.gl/nHQADK

Referências

https://www.ebayinc.com/stories/blogs/tech/cassandra-data-modeling-best-practices-part-1/
https://www.ebayinc.com/stories/blogs/tech/cassandra-data-modeling-best-practices-part-2/
http://cassandra.apache.org/
https://razorsql.com/download_win.html
https://medium.com/nstech/apache-cassandra-8250e9f30942
https://www.oreilly.com/library/view/cassandra-the-definitive/9781491933657/ch04.html
https://howtoprogram.xyz/2017/02/18/using-group-apache-cassandara/
Paniz, D. NoSQL - *Como armazenar os dados de uma aplicação moderna* - Casa do Código