Cassandra

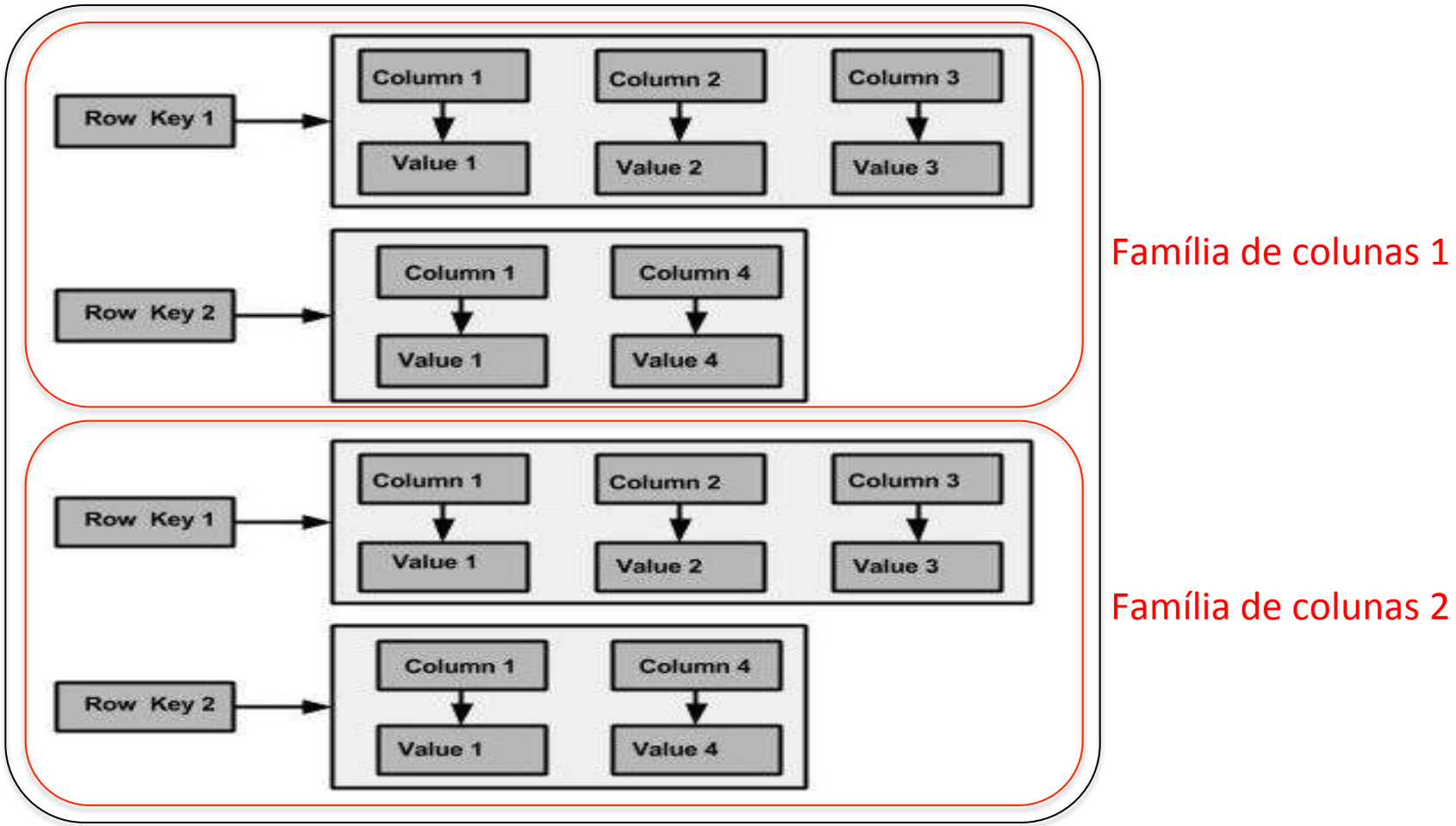
Profa. Andrêza Leite

Modelo de dados

- bancos de dados = Cluster
- Keyspace - uma estrutura semelhante a um Banco de Dados
- Dentro de um Keyspace, criamos as Column Families, o equivalente às tabelas.
- CQL – Cassandra Query Language
  - Comandos do tipo SQL

Modelo de dados

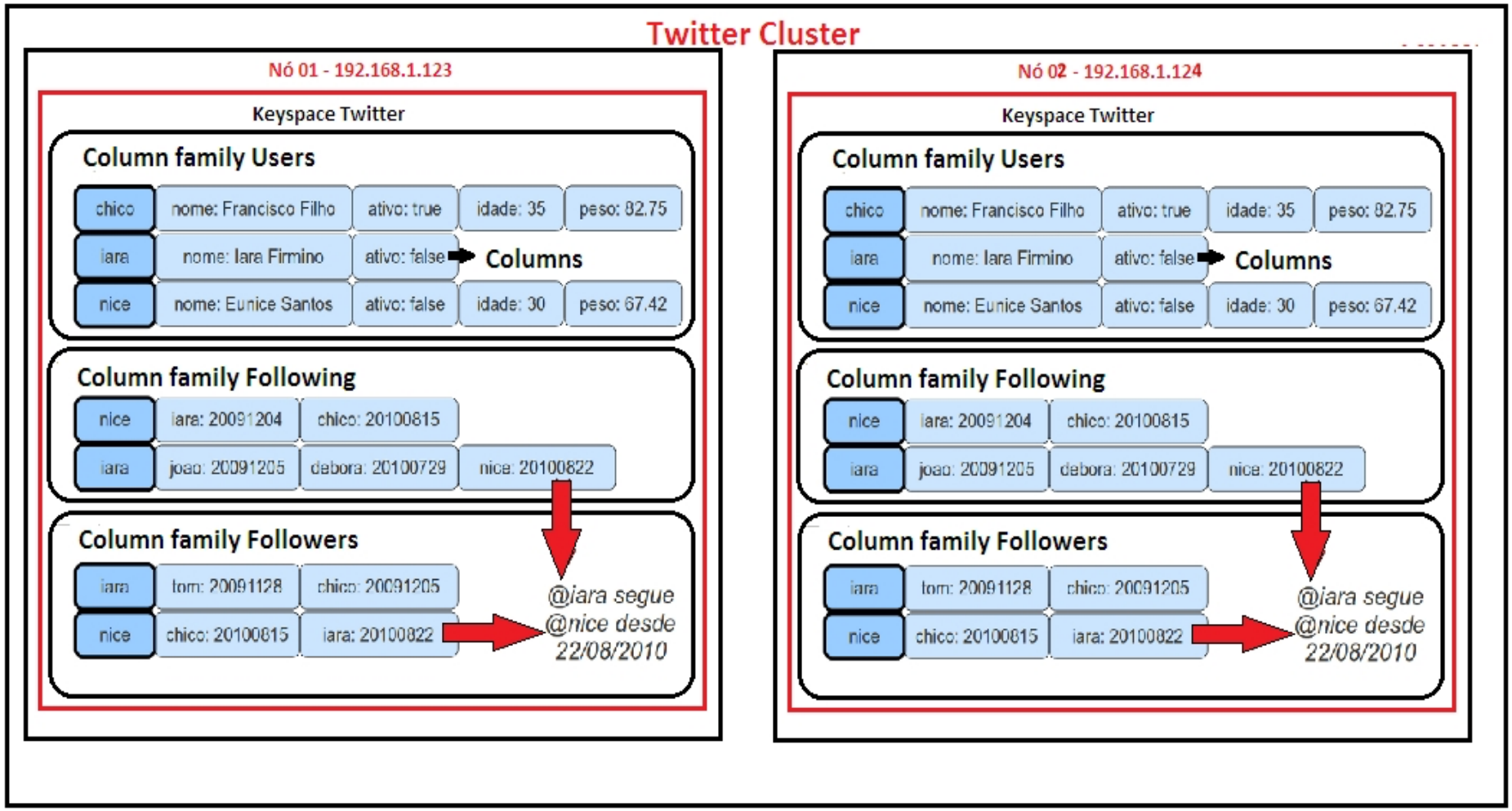
Keyspace



Família de colunas 1

Família de colunas 2

Exemplo Twitter



Relacional X Cassandra

Modelo relacional	Modelo Cassandra
Database	Keyspace
Table	Column Family (CF)
Primary key	Row key
Column name	Column name/key
Column value	Column value

Características e Recursos

- Disponibilidade
  - Todos os nodos são tidos como mestre
    - Reduzir o nível de consistência para aumentar a disponibilidade
    - Balanceamento (R+W)>N
      - W: número mínimo de nodos a gravar,
      - R: mínimo de nodos que devem responder uma leitura com sucesso
      - N: número de nodos que participam da replicação

Características e Recursos

- Disponibilidade
  - Balanceamento (R+W)>N
    - Cluster 10 nodos (2+2)>3 quando um nó cair a disponibilidade não é afetada uma vez que os dados podem ser recuperados dos outros dois(R=2)
    - W=2 e R=1, não disponível p/ gravação(mínimo 2) , mas p/ leitura
    - R=2 e W=1, não disponível p/ leitura (mínimo 2) , mas p/ gravação

Características e Recursos

- Consistência
  - Gravações efetuadas primeiro no registro de operações (*commit log*) depois na estrutura de memória (*memtable*)
    - Gravação bem sucedida = gravada no *commit log* e na *memtable*
  - Consistência = ONE Cassandra retorna os dados da primeira réplica, mesmo se dados antigos, leituras subsequentes obterão dados mais recentes (reparação de leitura).
    - Usar quando tiver requisitos de desempenho altos.
    - Pode perder gravações se um nó cair antes de a gravação ser replicada.

Características e Recursos

- Consistência
  - Consistência = QUORUM p/ R assegura que a maioria dos nodos responderá a leitura. P/ W a gravação será propagada para a maioria dos nodos.
  - Consistência ALL todos os nós terão que responder = cluster tolerante a falhas. Se for apenas 1 operações serão bloqueadas e informadas como falhas.

Características e Recursos

- Transações
  - Não possui transações no sentido tradicional
  - Uma gravação é atômica
    - bem sucedida(commit log e memtable) ou falha
    - Pode-se usar bibliotecas externas de transações como a Zookeeper para sincronizar gravações e leituras

Características e Recursos

- Escalabilidade
  - Horizontal
    - Adicionar mais nodos no cluster
  - Como nenhum é mestre, isso melhora a capacidade de suportar mais operações (R/W)
  - Permite obter o máximo de uptime (disponibilidade)

Casos Apropriados

- Registros de eventos (log)
- Sistemas de Gerenciamento de Conteúdo, blog
- Contadores
  - App precisa contar e categorizar visitantes
- Expirando o uso
  - Colunas que expiram (excluídas após um tempo determinaxo, TTL=***Time To Live***)
  - Acesso demonstrativo ou exibição de banners de propaganda

Casos Inapropriados

- Sistemas que requerem transações ACID para leituras e gravações
- Se precisar de agregações em consultas (SUM ou AVG), terá de fazer do lado cliente
- Primeiros protótipos ou ações iniciais
  - Padrões de consultas poderão mudar = alterar formato das famílias de colunas

Prática

Iniciando Cassandra

$ cqlsh

Connected to Test Cluster at 127.0.0.1:9042.

[cqlsh 5.0.1 | Cassandra 2.1.7 | CQL spec 3.2.0 | Native protocol

v3]

Use HELP for help.

cqlsh> **DESCRIBE KEYSPACES;**

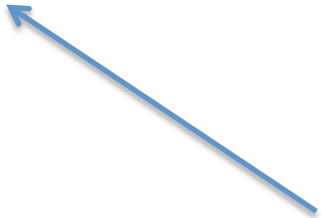
system_traces system

Criando Keyspaces

Sintaxe:

CREATE KEYSPACE [nome] WITH REPLICATION = [estratégia de replicação]

- *SimpleStrategy* - replica os dados nos próximos servidores sem levar em conta a topologia da rede,
- *NetworkTopologyStrategy* - muito mais complexo, pois permite configurar réplicas por Rack, Data Center etc.



durabilidade e alta disponibilidade

Criando Keyspaces

cqlsh> **CREATE KEYSPACE ligado WITH replication = {'class': 'SimpleStrategy', 'replication_factor ': '3'};**

cqlsh> **DESCRIBE KEYSPACES;**

system_traces system ligado

Criando tabelas

<column family>

https://notepad.pw/wxfsx1pz

```
cqlsh> use ligado;
cqlsh:ligado> CREATE TABLE musica (
      id int PRIMARY KEY,
      nome text,
      album text,
      artista text
);

cqlsh:ligado> DESCRIBE TABLE musicas;
```

Manipulando registros

cqlsh:ligado>

INSERT INTO musica (id, nome, album, artista) VALUES (now(), 'Help', 'Help', 'Beatles’);

a70ca7ff-6d57-4f89-be89-08421c432bb7

cqlsh:ligado> **SELECT * FROM musicas;**

Atualizando registros

```
cqlsh:ligado>
UPDATE musica SET
    nome='Help!',
    album='Help!'
WHERE id = a70ca7ff-6d57-4f89-be89-08421c432bb7;

cqlsh:ligado> SELECT * FROM musicas;
```

Apagando registros

cqlsh:ligado>

DELETE from musicas

WHERE id = a70ca7ff-6d57-4f89-be89-08421c432bb7;

cqlsh:ligado> **SELECT * FROM musicas;**

Inserindo mais dados

cqlsh:ligado> **INSERT INTO musicas (id, nome, album, artista)**
VALUES (04b57c98-33df-11e5-a151-feff819cdc9f,
'Help!', 'Help!', 'Beatles');
cqlsh:ligado> **INSERT INTO musicas (id, nome, album, artista)**
VALUES (1a8d6a80-33df-11e5-a151-feff819cdc9f,
'Yestarday', 'Help!', 'Beatles');
cqlsh:ligado> **INSERT INTO musicas (id, nome, album, artista)**
VALUES (04b57f0e-33df-11e5-a151-feff819cdc9f,
'Something', 'Abbey Road', 'Beatles');
cqlsh:ligado> **INSERT INTO musicas (id, nome, album, artista)**
VALUES (1a8d649a-33df-11e5-a151-feff819cdc9f,
'Blackbird', 'The Beatles', 'Beatles');

Buscando Dados

cqlsh:ligado> **SELECT * FROM musicas**
WHERE artista='Beatles';

InvalidRequest: code=2200 [Invalid query]
message="No secondary indexes on the restricted columns support the provided operators: "

Cassandra não faz buscas em campos que não possuem índices

Criando Índices

Sintaxe: CREATE INDEX ON nome_da_tabela (nome_do_campo)

cqlsh:ligado> **CREATE INDEX ON musicas (artista);**

cqlsh:ligado> **SELECT ***
FROM musicas
WHERE artista='Beatles';

Excluindo Keyspaces

```
cqlsh> drop keyspace ligado;
```

Armazenando Playlists

- Precisarmos suportar o versionamento das listas
  - Ter de calcular quais as músicas e a ordem em que elas devem ser tocadas toda vez que alguém quiser ver, ou escutar uma playlist, **não é nada performático**.
  - tabela auxiliar com uma espécie de "fotografia" de como a playlist está nesse momento, cache da versão calculada.
- Alteração na playlist = inserir um novo registro do controle de versão da playlist

Armazenando Playlists

PlaylistFinal

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

```
SELECT m.nome, m.artista, p.posicao
FROM playlistFinal p
JOIN musicas m
ON p.idMusica = m.id
WHERE p.idPlaylist = 1;
```

chave/valor, orientados a documentos, e colunares não suportam JOIN

=

N+1 queries ou desnormalização

=

manter a complexidade na escrita para facilitar ao máximo a leitura

Armazenando Playlists

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

Armazenando Playlists

```
cqlsh:ligado> 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’);

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

Armazenando Playlists

```
cqlsh:ligado> SELECT * FROM playlist_atual ;

 id| album | artista | id_musica  | nome      | posicao
---+-------+---------+------------+-----------+---------
 1 | Help! | Beatles |      2     | Yesterday |       2

(1 rows)
```

Tanto o INSERT quanto o UPDATE executam a operação **upsert** . Se já existe o registro, ele altera; caso contrário, ele cria.

Armazenando Playlists

- Como nós podemos ter várias músicas na mesma playlist, vamos precisar criar uma chave composta usando algum outro campo junto.
  - utilizar o id da música impediria de ter a mesma música na mesma playlist. se usarmos a posição como parte da chave, teremos um modelo mais flexível.
  - PK = id da playlist + posição da música

Armazenando Playlists

```
cqlsh:ligado> DROP TABLE playlist_atual;
cqlsh:ligado> 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

```
cqlsh:ligado> 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');

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