



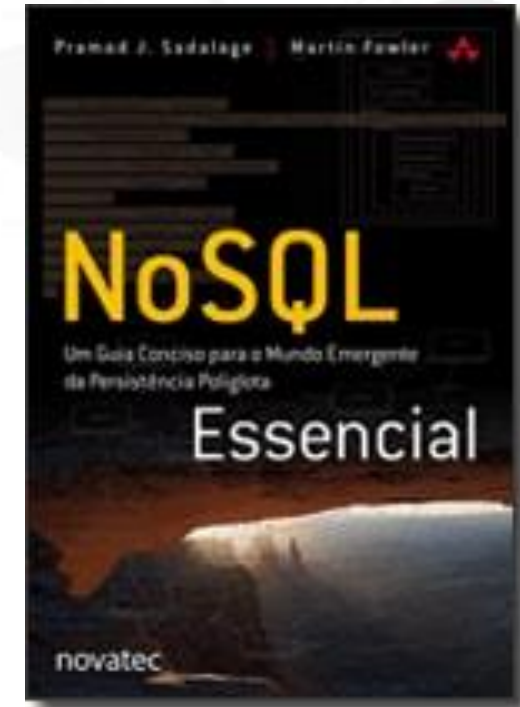
Banco de Dados não Relacionais - NoSQL

PUC Minas Virtual

Banco de dados de família de colunas

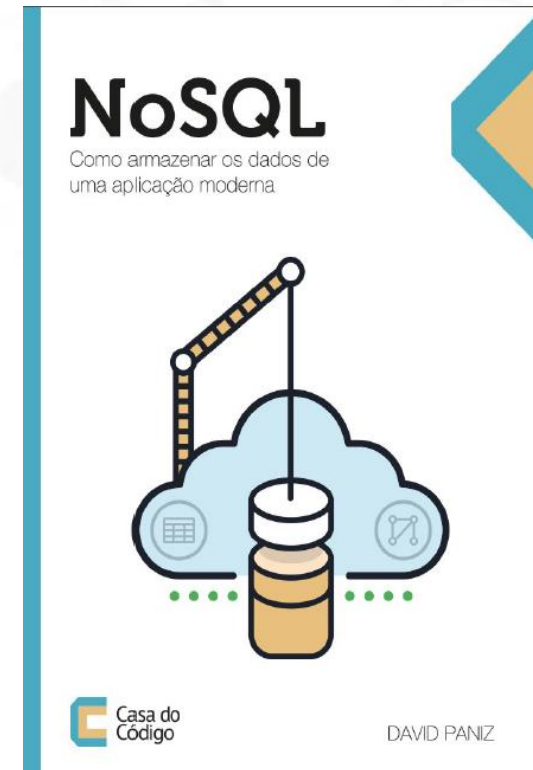
Principais Referências

Pramod J.; Sadalage, Martin Fowler.
**NoSQL Essencial: Um Guia Conciso
para o Mundo Emergente da
Persistência Poliglota.** Novatec
Editora, 2013.



Principais Referências

Paniz, David. NoSQL: **Como armazenar os dados de uma aplicação moderna**. Casa do Código, 2017.



Família de colunas

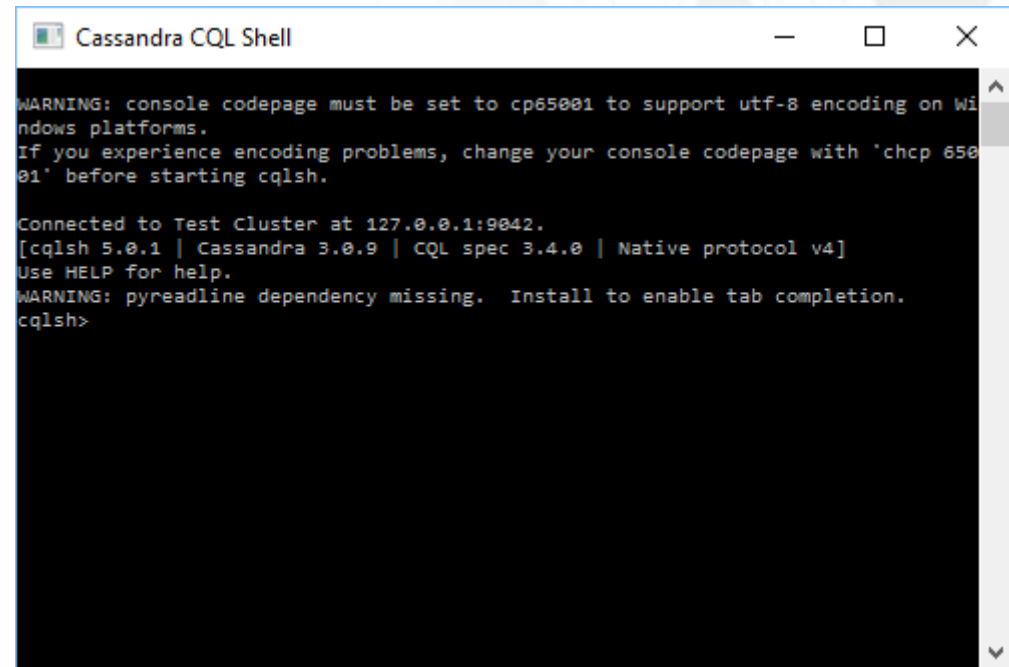
Há versões para Linux, Mac e Windows

É necessário ter a Java Virtual Machine (JVM) instalada

<http://cassandra.apache.org/download/>

Família de colunas

Após a instalação, há um programa Shell para acessar o Cassandra por meio de linha de comando.

A screenshot of a terminal window titled "Cassandra CQL Shell". The window has a black background with white text. The text inside the terminal shows a warning about console codepage settings, connection details to a test cluster at 127.0.0.1:9042, and version information for cqlsh 5.0.1, Cassandra 3.0.9, and CQL spec 3.4.0. It also mentions a missing pyreadline dependency for tab completion. The prompt "cqlsh>" is visible at the bottom.

```
Cassandra CQL Shell

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.0.9 | CQL spec 3.4.0 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing. Install to enable tab completion.
cqlsh>
```

Família de colunas

Abra o console do Cassandra. Utilizando o Cassandra Query Language (CQL), podemos inserir comando pelo console.

Podemos utilizar o seguinte comando para visualizar os keyspaces existentes:

```
Describe keyspaces
```

Família de colunas

Para criar um novo Keyspace, utilize o comando:

Use NetworkTopologyStrategy
para mais de um datacenter

Nome do nosso banco

```
CREATE KEYSPACE Aula WITH REPLICATION = {'class':  
'SimpleStrategy', 'replication_factor': '3'};
```

Replicação de dados (sem levar
em consideração a topologia)

Quando uma linha é gravada, 3 cópia
serão armazenadas em outros nós

Família de colunas

Para criar uma nova tabela. O Cassandra utiliza o nome tabela para referenciá-la.

Semelhante ao banco relacional, temos que definir o nome e tipo de dados de cada coluna.

É necessário também definir a coluna que será a chave primária.


Família de colunas

Antes, devemos selecionar o banco criado: `use aula;`

Criaremos uma tabela chamada “musicas”

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

Definição da Row Key do
tipo `uuid` (universal user
id) para evitar colisão



Obs.: Para ver as tabelas criadas: `DESCRIBE TABLES`
ou `DESCRIBE TABLE NomeDaTabela` para detalhes da mesma

Família de colunas

Para excluir uma família de colunas, basta utilizar o comando DROP TABLE:

```
DROP TABLE IF EXISTS musicas
```

Optional

Manipulando dados no Cassandra

Inserindo registros

A sintaxe do Cassandra para inserção de dados também é muito semelhante ao banco relacional.

```
INSERT INTO musicas (id, nome, album, artista)  
VALUES (uuid(), 'Help', 'Help', 'Beatles');
```

Função que gera um universal user id

Recuperando registros

Para recuperar e visualizar os dados, a sintaxe no Cassandra também é muito semelhante ao banco relacional.

```
SELECT * FROM musicas;
```

id	album	artista	nome
63dbffdc-2d96-4314-8297-2bc7ba678d88	Help	Beatles	Help

Atualizando registros

Para atualizar registros, usamos UPDATE.

```
UPDATE musicas  
SET nome='Help!', album='Help!'  
WHERE id = e4a46e87-fa5a-4488-949a-d230630d2f23;
```

Repita do comando SELECT para visualizar o resultado:

```
SELECT * FROM musicas;
```

Apagando registros

Da mesma forma, podemos utilizar o comando **DELETE** para apagar registros na tabela.

```
DELETE FROM musicas  
WHERE id = 63dbffdc-2d96-4314-8297-2bc7ba678d88;
```


Inserindo registros com TTL

No Cassandra é possível inserir registros que expiram após determinado tempo utilizando o comando TTL.

Insira o registro abaixo e faça um consulta.

Tempo de vida
do registro em
segundos

```
INSERT INTO musicas (id, nome, album, artista)  
VALUES (uuid(), 'The Night Before', 'Help', 'Beatles')  
USING TTL 30;
```

Inserindo registros com TTL

Faça uma consulta SELECT para visualizar o resultado e aguarde um tempo:

```
SELECT * FROM musicas;
```

Repita do comando SELECT para visualizar o resultado:

```
SELECT * FROM musicas;
```

Inserindo registros com TTL

O uso do TTL (Time To Live) é adequado para funcionalidades para expirar por tempo de uso. Revogar acesso de usuário ou expirar o tempo de uma propaganda (banner) no site, por exemplo. Se não for especificado, o valor default é nunca expirar.

Busca no Cassandra

Vamos inserir vários registros dentro da família de colunas "musicas".

```
INSERT INTO musicas (id, nome, album, artista)  
VALUES (uuid(), 'Help!', 'Help!', 'Beatles');
```

```
INSERT INTO musicas (id, nome, album, artista)  
VALUES (uuid(), 'Yesterday', 'Help!', 'Beatles');
```

Busca no Cassandra

```
INSERT INTO musicas (id, nome, album, artista)  
VALUES (uuid(), 'Something', 'Abbey Road', 'Beatles');
```

```
INSERT INTO musicas (id, nome, album, artista)  
VALUES (uuid(), 'Blackbird', 'The Beatles', 'Beatles');
```

Repita do comando SELECT para visualizar o resultado:

```
SELECT * FROM musicas;
```

Busca no Cassandra

Após o comando SELECT:

id	album	artista	nome
a441aca9-b267-475b-855c-3eba38535c88	Help!	Beatles	Yesterday
ae55446d-4ad2-41cf-8173-2ef034b08a75	Abbey Road	Beatles	Something
24174b4b-0cc1-46e7-b4c4-4c4d93abfba1	The Beatles	Beatles	Blackbird
7650a863-2b98-4547-86f5-2ab436419189	Help!	Beatles	Help!

Busca no Cassandra

Agora tente buscar as informações dos artista "Beatles". Execute a seguinte consulta:

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

Busca no Cassandra

Agora tente buscar as informações dos artista "Beatles". Execute a seguinte consulta:

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

O Cassandra não irá permitir que esta consulta seja executada. Por que?

Busca no Cassandra

O Cassandra não permite que a consulta seja feita: a coluna para filtro não é indexada:

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



Coluna não é a *Row Key* e portanto, não há índice para esta coluna

Consulta por campos que não possuem índices são, geralmente, muito lentas (mesmo em um banco relacional apesar de ser permitido).

Busca no Cassandra

Se ainda sim você desejar executar a consulta, utilize o comando **ALLOW FILTERING**:

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



O Cassandra irá permitir a execução da consulta usando filtros, mesmo que ela seja pouco performática. Mas **não** é a estratégia aconselhada.

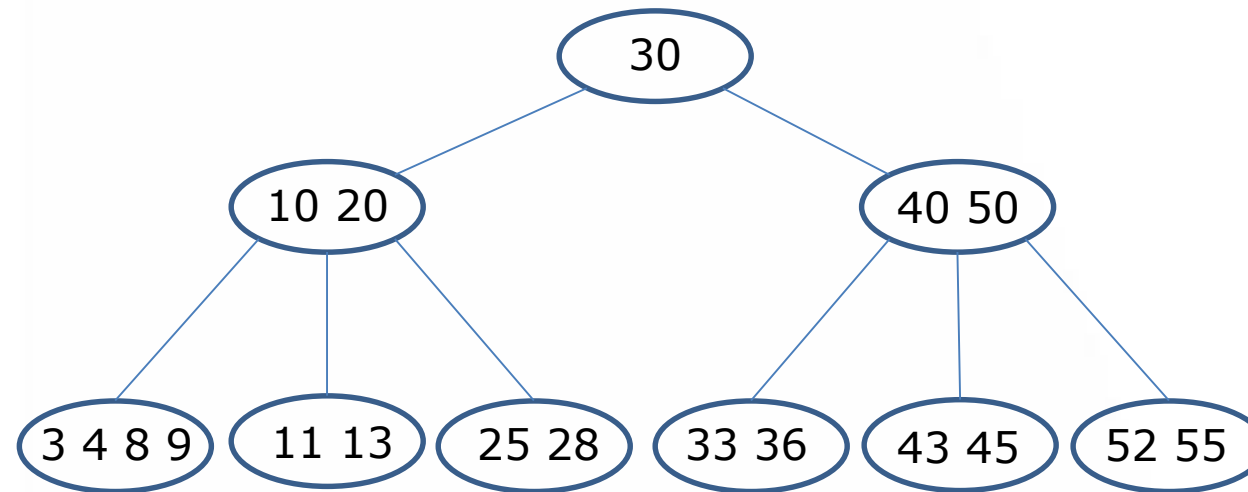
Busca no Cassandra

Como um índice funciona?

Um índice é geralmente uma estrutura em árvore ou hash que acelera a consulta em um banco.

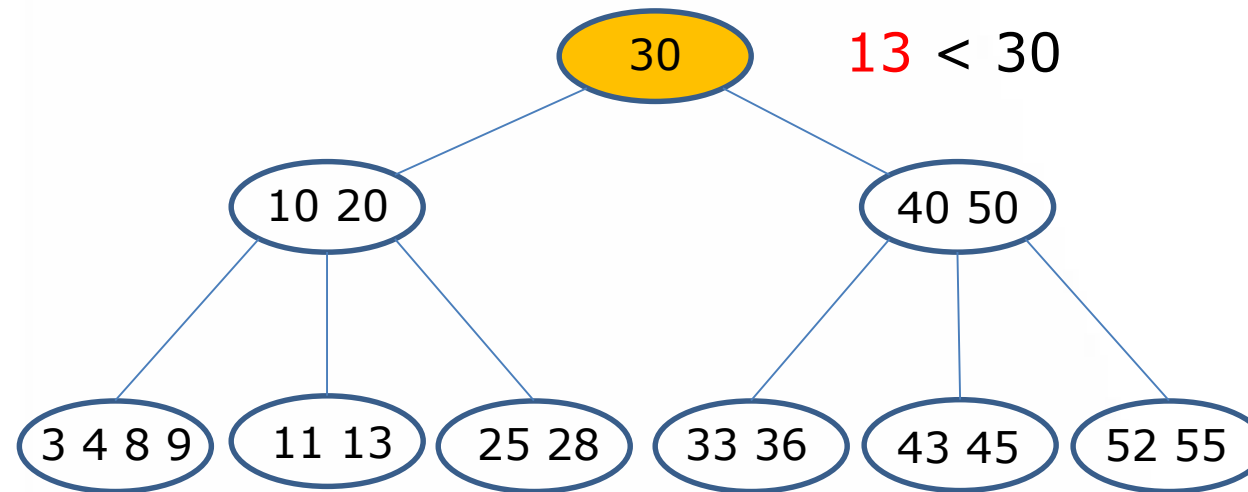
Exemplo de estrutura em índice

Estrutura em árvore para índices: consulta = 13



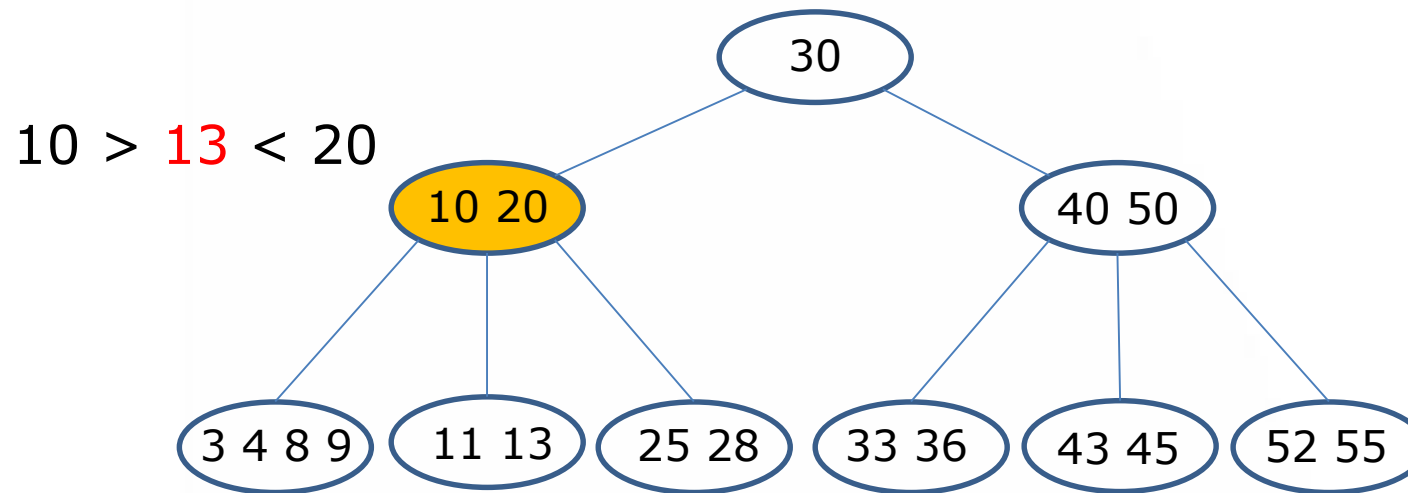
Exemplo de estrutura em índice

Estrutura em árvore para índices: consulta = 13



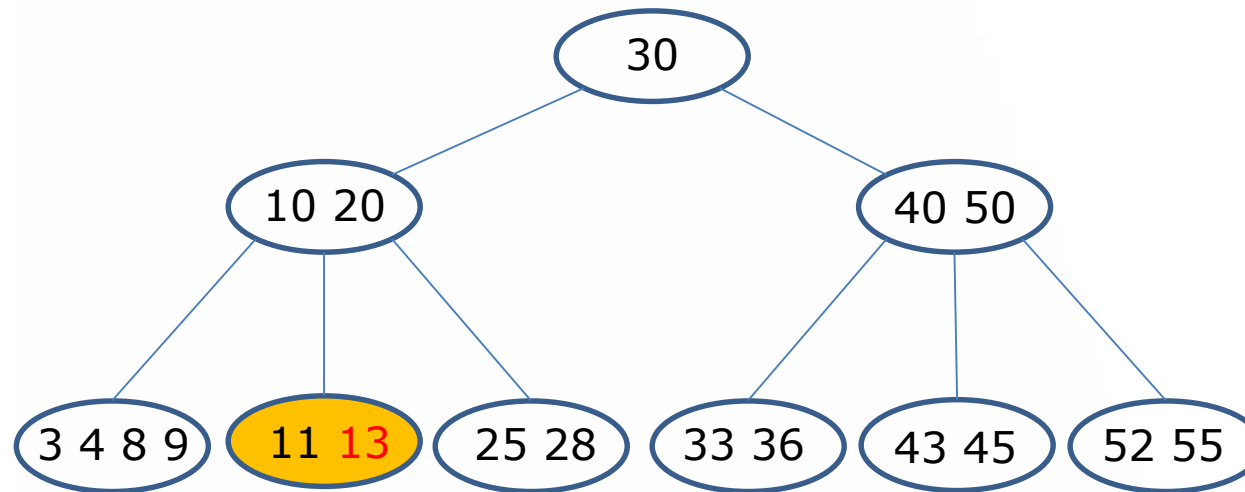
Exemplo de estrutura em índice

Estrutura em árvore para índices: consulta = 13



Exemplo de estrutura em índice

Estrutura em árvore para índices: consulta = 13



Tempo de busca em dados ordenados: $\log (n)$

Busca no Cassandra

O índice evita que todos as linhas sejam consultadas para atender ao filtro especificado na consulta.

Para melhorar, podemos criar índices dos campos em cada tabela (o campo Row Key contém índice criado pelo próprio Cassandra).

Criando índices

Podemos utilizar o comando `CREATE INDEX ON` para criar um índice no campo “artista” na tabela “musicas”, por exemplo. Assim será possível filtrar as consulta pelo artista.

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

Ou desta forma:

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

Busca no Cassandra

Agora repita a consulta e veja o resultado:

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

As consulta agora podem ser filtradas pelo **id** ou pelo **artista** (os dois possuem índices)

id	album	artista	nome
a441aca9-b267-475b-855c-3eba38535c88	Help!	Beatles	Yesterday
ae55446d-4ad2-41cf-8173-2ef034b08a75	Abbey Road	Beatles	Something
24174b4b-0cc1-46e7-b4c4-4c4d93abfba1	The Beatles	Beatles	Blackbird
7650a863-2b98-4547-86f5-2ab436419189	Help!	Beatles	Help!

Excluindo índices

Para excluir um índice, basta utilizar o comando **DROP INDEX**:

```
DROP INDEX IF EXISTS nome_do_índice;
```

Optional

Ou desta forma:

```
DROP INDEX IF EXISTS musicas_artista_idx;
```

Optional

Esta estrutura é usada quando não foi especificado um nome para o índice na sua criação, assim:
<table_name>_<column_name>_idx

Alguns detalhes sobre o uso de índices

Evite criar índices em colunas que podem sofrer muitas alterações (acessar índice é rápido, mas criar e modificar pode ser muito lento).

Alguns detalhes sobre o uso de índices

Evite criar índices em colunas com alta cardinalidade (coluna com vários valores). Ex.: ao buscar uma músicas pode ser mais eficiente consultas por “artistas” do que por “compositor”.

《 Implementando a Playlist 》

Implementando a playlist

Conforme vimos anteriormente, toda vez que um cliente alterar a playlist, gravamos um registro na família de colunas (tabela) de controle de versão.

A tabela terá o id da playlist, o id da música e a posição.

Implementando a playlist

Modelo da playlist

PlaylistFinal

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

Estratégia adequada para o Cassandra, uma vez que possui alta performance na escrita



Observe que o banco de família de colunas não possui “joins” (como no relacional), por isso trabalhamos com tabelas desnormalizadas. Aumenta a complexidade na escrita, mas a leitura será mais rápida.

Implementando a playlist

Criando as tabela playlist:

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

Obs.: Para ver as tabelas criadas: DESCRIBE TABLES

Implementando a playlist

Adicionando as playlists

```
INSERT INTO playlist_atual (id_playlist, posicao, id_musica, nome, album, artista)  
VALUES (1, 1, 62d77d6f-d6d7-46ca-a1a1-6476f1914812, 'Help!', 'Help!', 'Beatles');
```

```
INSERT INTO playlist_atual (id_playlist, posicao, id_musica, nome, album, artista)  
VALUES (1, 2, 26203678-0fa0-43a3-a02d-4c3f10ab66f5, 'Yesterday', 'Help!',  
'Beatles');
```

Implementando a playlist

Adicionando as playlists

```
INSERT INTO playlist_atual (id_playlist, posicao, id_musica, nome, album, artista)  
VALUES (1, 3, eca2f815-d29a-424f-a0d0-4682d356521d, 'Something', 'Abbey Road',  
'Beatles');
```

```
INSERT INTO playlist_atual (id_playlist, posicao, id_musica, nome, album, artista)  
VALUES (1, 4, 54665e70-32d8-46e0-8673-9fed563e872a, 'Blackbird', 'The Beatles',  
'Beatles');
```

Implementando a playlist

Faça uma consulta `SELECT` para visualizar o resultado:

```
SELECT * FROM playlist_atual;
```

<code>id_playlist</code>	<code>posicao</code>	<code>album</code>	<code>artista</code>	<code>id_musica</code>	<code>nome</code>
1	1	Help!	Beatles	7650a863...	Help!
1	2	Help!	Beatles	a441aca9...	Yesterday
1	3	Abbey Road	Beatles	ae55446d...	Something
1	4	The Beatles	Beatles	24174b4b...	Blackbird

Controle de versão



Implementando a playlist

Faça uma consulta `SELECT` para visualizar a `playlist_atual`:

```
SELECT * FROM playlist_atual;
```

<code>id_playlist</code>	<code>posicao</code>	<code>album</code>	<code>artista</code>	<code>id_musica</code>	<code>nome</code>
1	1	Help!	Beatles	7650a863...	Help!
1	2	Help!	Beatles	a441aca9...	Yesterday
1	3	Abbey Road	Beatles	ae55446d...	Something
1	4	The Beatles	Beatles	24174b4b...	Blackbird

Implementando o controle de versão

O modelo de dados mais amplamente utilizado em bancos colunares são chamados “dynamic columns”.

Estas tabelas não possuem colunas predefinidas.

Implementando a o controle de versão

As linhas são chamadas de **partição** com uma chave única, e a partir dela, adicionamos os pares chave-valor (coluna-linha).

Cada par chave-valor (coluna-linha) é chamado de **célula**.

Implementando a o controle de versão

Este modelo (colunas dinâmicas) será usado para implementar a tabela de controle de versão. No modelo anterior, tínhamos:

PlaylistVersionada (Servidor)

idPlaylist	versão	acao	posicao	para	idMusica
1	1	adiciona	1		Help!
1	2	adiciona	2		Yesterday
1	3	adiciona	3		Blackbird
1	4	adiciona	4		Something
1	5	troca	3	2	

Implementando a o controle de versão

Veja que agora linhas passaram a ser colunas (o campo “versao” é o nome da coluna), os demais campos formam o valor das células.

id_playlist	1	2	3	4	5
1	ADICIONA(Help!)	ADICIONA(Yesterday)	ADICIONA(Blackbird)	ADICIONA(Something)	TROCA (3,2)

Partição
(com um id)

Célula com chave e valor
(coluna e linha)

Implementando a o controle de versão

Neste modelo, precisamos criar uma tabela com três campos: (1) chave da partição; (2) nome das colunas; e (3) valor da linha.

A chave primária é composta pela chave da partição (1) e nome das colunas (2).

Implementando a o controle de versão

Comando para criação da tabela:

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

Especificação
para criar tabelas
com colunas
dinâmicas

Implementando a o controle de versão

Agora, inserindo dados na tabela dinâmica:

```
INSERT INTO playlist_versionada (id_playlist, versao, modificacao)  
VALUES (1, 1, 'ADI(Help!));
```

```
INSERT INTO playlist_versionada (id_playlist, versao, modificacao)  
VALUES (1, 2, 'ADI(Yesterday));
```

```
INSERT INTO playlist_versionada (id_playlist, versao, modificacao)  
VALUES (1, 3, 'ADI(Blackbird));
```

Implementando a o controle de versão

Agora, inserindo dados na tabela dinâmica:

```
INSERT INTO playlist_versionada (id_playlist, versao, modificacao)  
VALUES (1, 4, 'ADI(Something)');
```

```
INSERT INTO playlist_versionada (id_playlist, versao, modificacao)  
VALUES (1, 5, 'TROCA(3,2)');
```

Implementando a o controle de versão

Faça uma consulta SELECT para visualizar o resultado:

```
SELECT * FROM playlist_versionada;
```

id_playlist	versao	modificacao
1	1	ADI(Help!)
1	2	ADI(Yesterday)
1	3	ADI(Blackbird)
1	4	ADI(Something)
1	5	TROCA(3,2)

Implementando a o controle de versão

Apesar do comando SELECT retornar uma tabela muito semelhante ao modelo relacional, os dados no Cassandra não são armazenados neste formato. Veja abaixo:

RowKey: 1

```
=> (name=1, value=ADI(Help!), timestamp=1438818548125646)
=> (name=2, value=ADI(Yesterday), timestamp=1438818553747085)
=> (name=3, value=ADI(Blackbird), timestamp=1438818571767466)
=> (name=4, value=ADI(Something), timestamp=1438818576933964)
=> (name=5, value=TROCA(3,1,2), timestamp=1438818586229982)
```


Criando a tabela playlist

Para finalizar, criamos a tabela playlist:

```
CREATE TABLE playlist (  
    id int PRIMARY KEY,  
    nome text  
);
```

```
INSERT INTO playlist (id, nome)  
VALUES (1, 'Beatles forever');
```

Criando a tabela playlist

Faça uma consulta `SELECT` para visualizar o resultado:

```
SELECT * FROM playlist;
```

id	nome
1	Beatles forever

Cassandra Driver para Python

Cassandra Driver para Python

Biblioteca para acesso e manipulação do Cassandra em Python

Após a instalação do Python 3.6.1, abra o prompt de comando e execute o comando: "python". O console do Python será executado, e então execute o comando abaixo para instalação do Pymongo.

```
pip install cassandra-driver
```

Referência:

<https://academy.datastax.com/resources/getting-started-apache-cassandra-and-python-part-i>

Cassandra Driver para Python

```
from cassandra.cluster import Cluster
cluster = Cluster()
session = cluster.connect('aula')

session.execute ("""

CREATE TABLE users (
    id uuid PRIMARY KEY,
    lastname text,
    age text,
    city text,
    email text,
    firstname text
);

""")
```

Salve o código ao lado em um arquivo (scriptCassandra.py). Script em python usado para criar tabelas no Cassandra.

Cassandra Driver para Python

```
from cassandra.cluster import Cluster
cluster = Cluster()
session = cluster.connect('aula')
```

Conecta no Cassandra na
keyspaces chamado
"banco" (deve ser
previamente criada).

```
session.execute("""
```

```
CREATE TABLE users (
    id uuid PRIMARY KEY,
    lastname text,
    age text,
    city text,
    email text,
    firstname text
);

""")
```

← Todo comando em CQL deve ser
executado usado session.execute().
Observe que o comando deve estar
entre 3 aspas duplas.

Cassandra Driver para Python

```
from cassandra.cluster import Cluster
cluster = Cluster()
session = cluster.connect('aula')

session.execute("""

insert into users (id, lastname, age, city,
email, firstname) values (uuid(), 'Jones',
'35', 'Austin', 'bob@example.com', 'Bob')

""")

result = session.execute("select * from users
where lastname='Jones' ALLOW FILTERING; ")[0]
print (result.age, result.lastname)
```

Salve o código ao lado em um arquivo (script2.py). Script em python usado para inserir dados na tabelas do Cassandra e consultar (exibindo o resultado na tela).

Cassandra Driver para Python

```
from cassandra.cluster import Cluster
cluster = Cluster()
session = cluster.connect('aula')
```

```
session.execute("""
```

```
insert into users (id, lastname, age, city,
email, firstname) values (uuid(), 'Jones',
'35', 'Austin', 'bob@example.com', 'Bob')
```

```
""")
```

```
result = session.execute("select * from users
where lastname='Jones' ALLOW FILTERING; ")[0]
print (result.age, result.lastname)
```

← Comando(s) para inserir dados na tabela "users"

← Retorna todos os dados do usuário com " lastname = 'Jones' " e exibe na tela apenas sua idade e último nome.

Quando usar e não usar

Situações apropriadas para o uso

Registro de eventos: São muitos bons para armazenar informações sobre eventos ou estado do aplicativo, por lidar com qualquer tipo de dado.

Situações apropriadas para o uso

Registro de eventos: Cada aplicação na empresa pode salvar seus eventos no Cassandra, cada um com sua coluna. O id da linha pode ser o evento e a coluna uma aplicação diferente.

Situações apropriadas para o uso

Controle de tempo de uso: Muito útil para armazenar informações com tempo de expiração.

Um banner em um blog, por exemplo, pode ter um tempo para expirar.

Situações para não usar

Quando funções agregadas são importantes (soma, média) entre os dados do banco, será necessário fazer isso do lado do cliente.

Situações para não usar

Inicialmente, quando não se conhece o padrão de consultas do banco, pode ser melhor aguardar antes de criar o banco, pois ao conhecer as consultas, pode ser necessário definir o padrão das colunas no Cassandra.



Dicas!

Para saber mais: Consulte o site dos desenvolvedores

<http://docs.datastax.com/en/archived/cassandra/2.0/index.html>



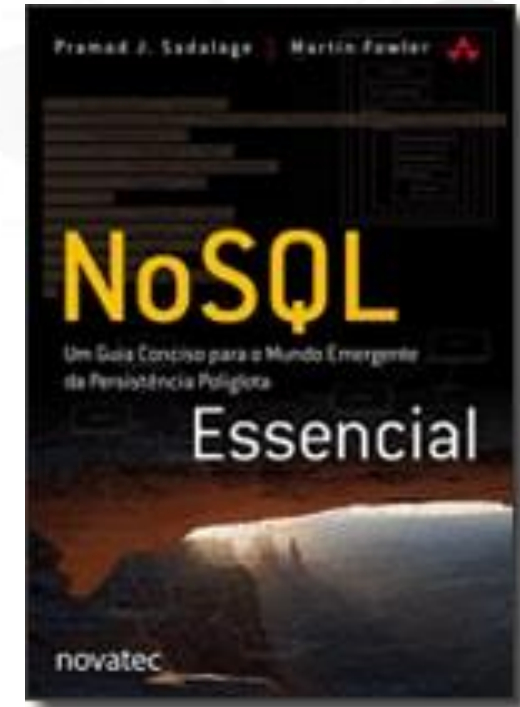
Dicas!

**Para aprender mais sobre a linguagem CQL,
consulte:**

<http://www.w3ii.com/pt/cassandra/default.html>

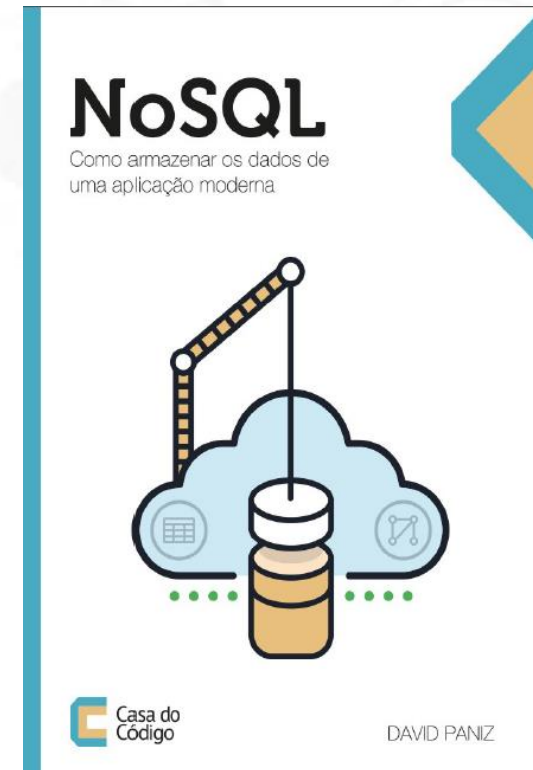
Principais Referências

Pramod J.; Sadalage, Martin Fowler.
**NoSQL Essencial: Um Guia Conciso
para o Mundo Emergente da
Persistência Poliglota.** Novatec
Editora, 2013.



Principais Referências

Paniz, David. NoSQL: **Como armazenar os dados de uma aplicação moderna**. Casa do Código, 2017.





PUC Minas
Virtual

© PUC Minas • Todos os direitos reservados, de acordo com o art. 184 do Código Penal e com a lei 9.610 de 19 de fevereiro de 1998.

Proibidas a reprodução, a distribuição, a difusão, a execução pública, a locação e quaisquer outras modalidades de utilização sem a devida autorização da Pontifícia Universidade Católica de Minas Gerais.