

BANCOS DE DADOS II

Cap.4 - Bancos de Dados Orientados a
Colunas



Prof. MSc. Renzo P. Mesquita

Objetivos

- Compreendermos características de BDs NoSQL Orientados a Colunas;
- Trabalhar este conceito por meio do BD Cassandra e sua ferramenta para visualização, busca e manipulação de dados: o DataStax Astra;
- Aprender recursos fundamentais da linguagem de programação Python para conectarmos aplicações feitas nesta linguagem a BDs NoSQL;



Capítulo 4

Bancos de Dados Orientados a Colunas

- 4.1. Bancos Orientados a Colunas;**
 - 4.1.1. O Banco de Dados Cassandra;**
 - 4.1.2. Arquitetura do Cassandra;**
 - 4.1.3. Estruturas de Armazenamento do Cassandra;**
- 4.2. Cassandra Query Language (CQL);**
 - 4.2.1. Criando Keyspaces e Tabelas;**
 - 4.2.2. Inserindo, Atualizando e Deletando Dados;**
 - 4.2.3. Buscando Dados;**
- 4.3. Conexão Python + Cassandra;**
 - 4.3.1. Configurando uma Conexão;**
 - 4.3.2. Exemplo de Client no Cassandra;**

4.1. Bancos Orientados a Colunas

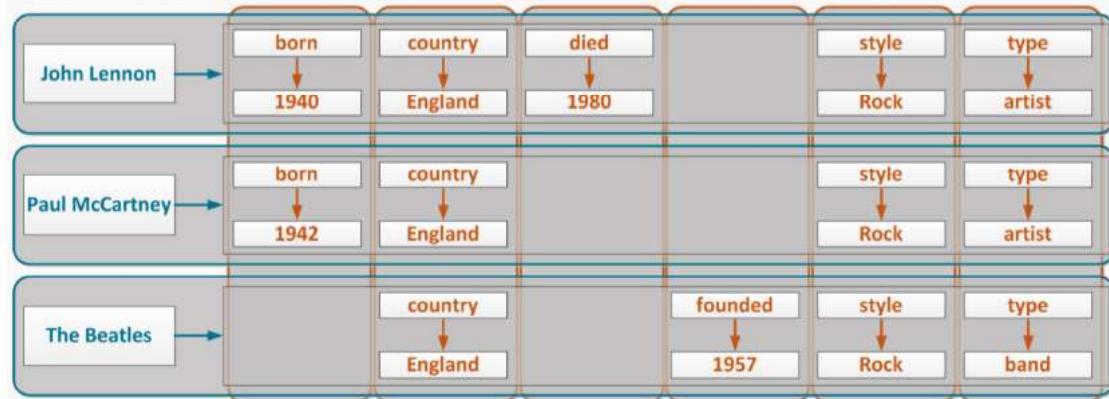
Enquanto BDs Relacionais são otimizados para lidarem com linhas de dados, BDs Orientados a Colunas são otimizados para lidarem com Colunas de Dados. Isso significa que, cada linha de um BD Orientado a Colunas, não precisa ter necessariamente os mesmos campos.

Exemplo:

performer	born	country	died	founded	style	type
John Lennon	1940	England	1980		Rock	artist
Paul McCartney	1942	England			Rock	artist
The Beatles		England		1957	Rock	band

Relacional

Colunas



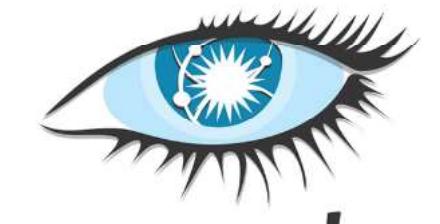
Quais seriam as possíveis vantagens e desvantagens desta abordagem?
Veremos ao longo deste Capítulo ;)

4.1. Bancos Orientados a Colunas

4.1.1. O Banco de Dados Cassandra

Banco NoSQL open source criado originalmente pelo Facebook e que atualmente é mantido pela Apache e outras empresas.

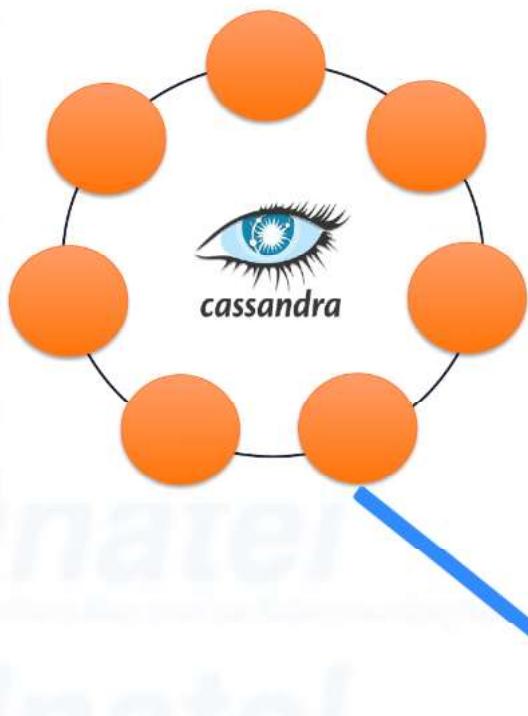
- É um dos BDs Orientados a Colunas mais populares do mercado;
- Recomendado para empresas que precisam de desempenho ao lidarem com um volume de dados "que tende ao infinito". Ex: Petabytes de dados;
- Tem como características ser naturalmente distribuído e não possuir um ponto único de falha;
- Oferece mecanismos simplificados para replicação de dados entre data centers distribuídos;
- Utiliza do Cassandra Query Language (CQL) para definição, manipulação e busca dos dados (lembra muito o SQL, porém, com as particularidades dos BDs Orientados a Colunas que veremos mais em detalhes em breve).



4.1. Bancos Orientados a Colunas

4.1.2. Arquitetura do Cassandra

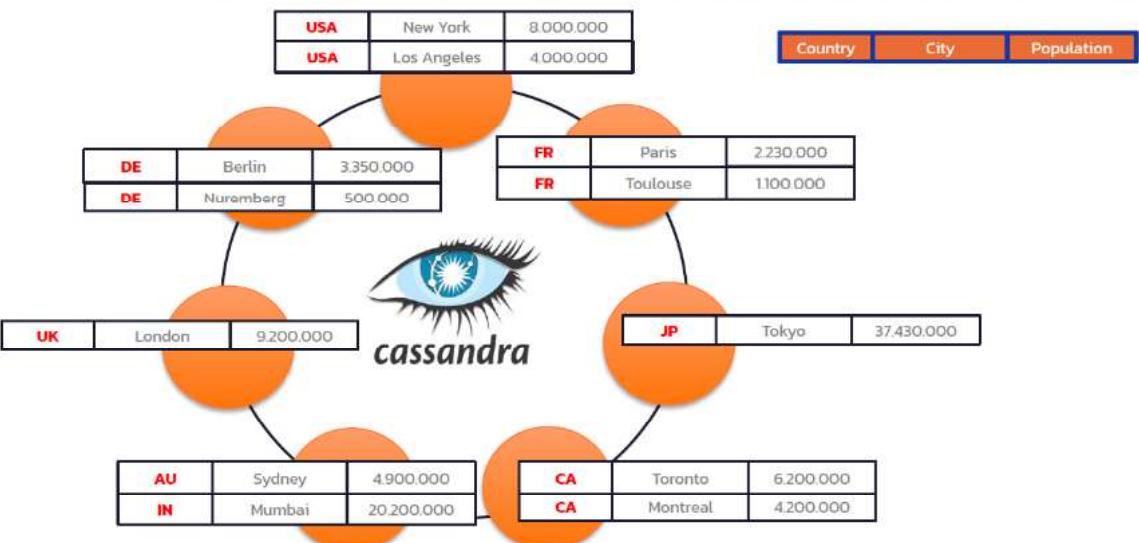
O Cassandra já foi feito para que seus dados sejam naturalmente distribuídos em diferentes Nós (computadores geralmente distribuídos geograficamente).



Country	City	Population
USA	New York	8.000.000
USA	Los Angeles	4.000.000
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

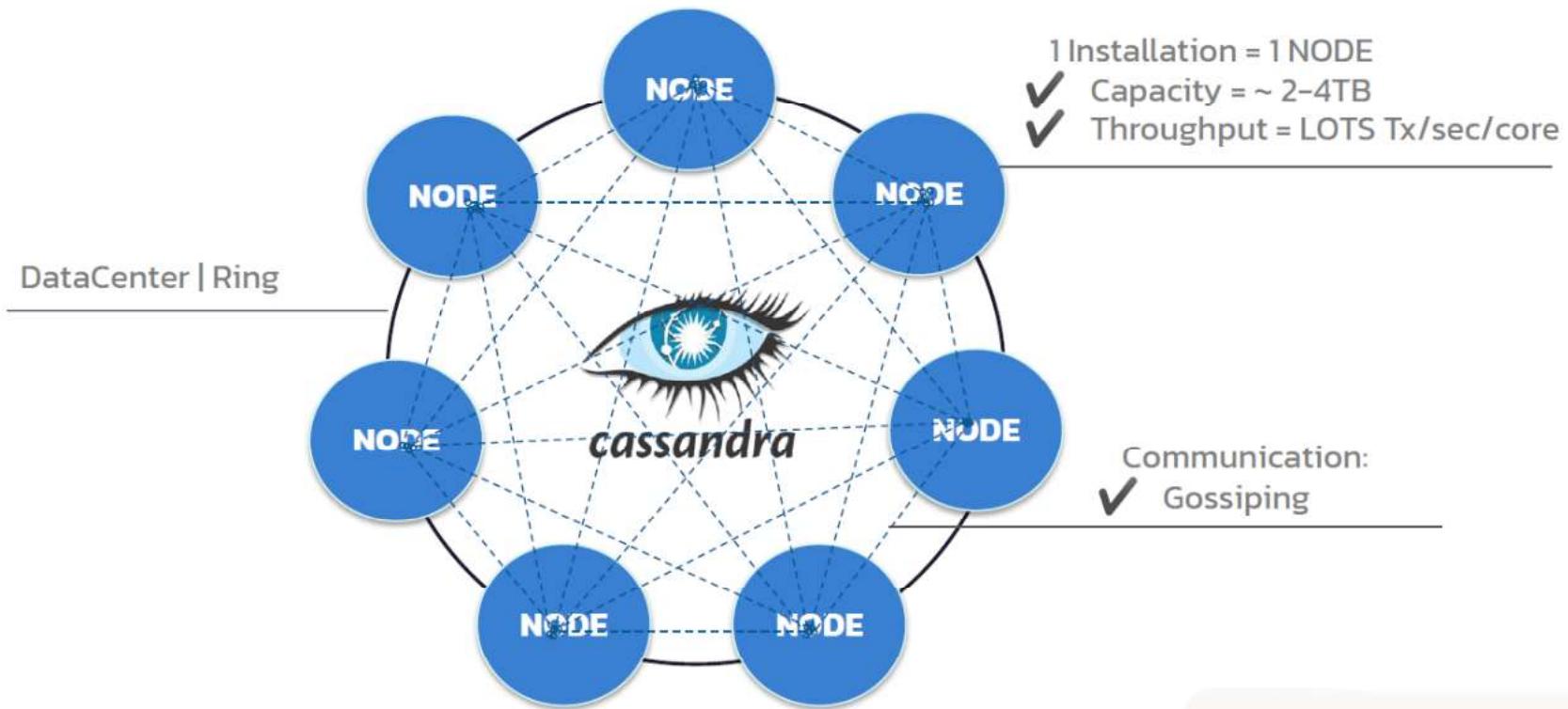
Partition Key

Observe os dados da Tabela sendo distribuídos em N Nós, se baseando no conceito de Partition Key.



4.1. Bancos Orientados a Colunas

Um conjunto de Nós (Nodes) pertencentes a uma empresa ou instituição que usa de um BD Orientado a Colunas se chama Ring ou Datacenter.

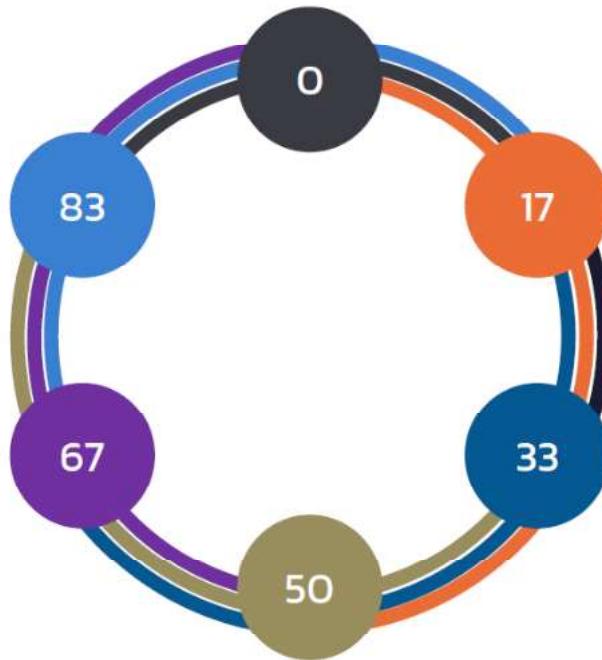


- Os Nodes são organizados em uma arquitetura Peer-to-Peer (P2P), ou seja, não existe um Node centralizado (este conceito se chama **Masterless**);
- Os Nodes se comunicam por meio de uma técnica chamada de **Gossiping** (em português, fofoca), para que todos sempre estejam atualizados do que está acontecendo no Anel de Nós (Ring);

4.1. Bancos Orientados a Colunas

No Cassandra, existe naturalmente o conceito de Replication Factor (RF), que mostra a quantos Nós um novo dado adicionado no BD pode ser replicado.

Exemplo:

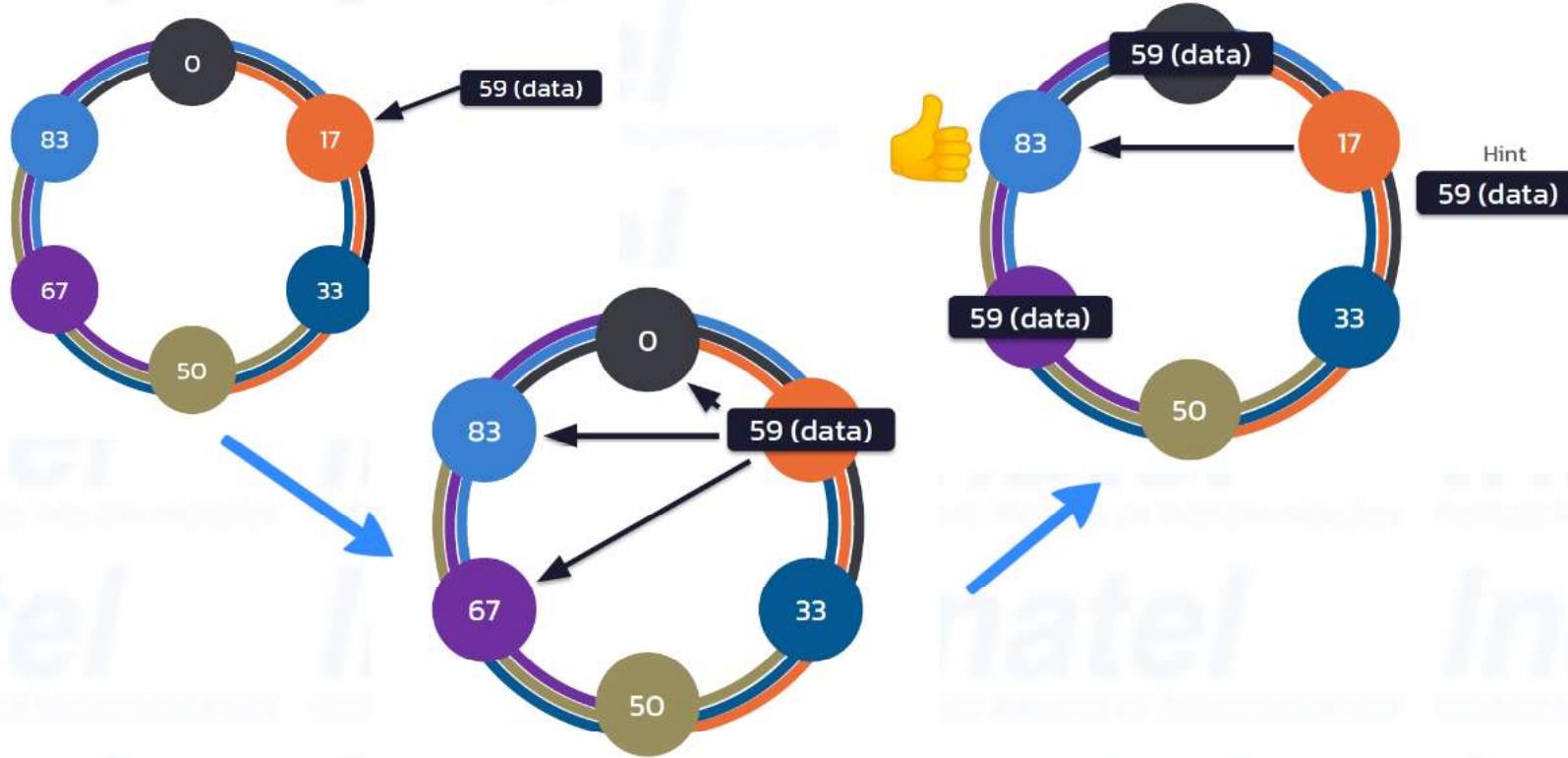


Neste exemplo está sendo ilustrado um processo de replicação com RF = 3;

- Nos bastidores, o Cassandra transforma cada Partition Key em um Token (que é um número inteiro), tornando o Ring como se fosse uma grande Tabela Hash Distribuída;
- No exemplo, o primeiro Nό poderia guardar Tokens de 84 a 0, o segundo de 1 a 17, e assim por diante (além dos dados replicados);

4.1. Bancos Orientados a Colunas

Ao se buscar um dado, observe como a Replicação pode agilizar o processo.

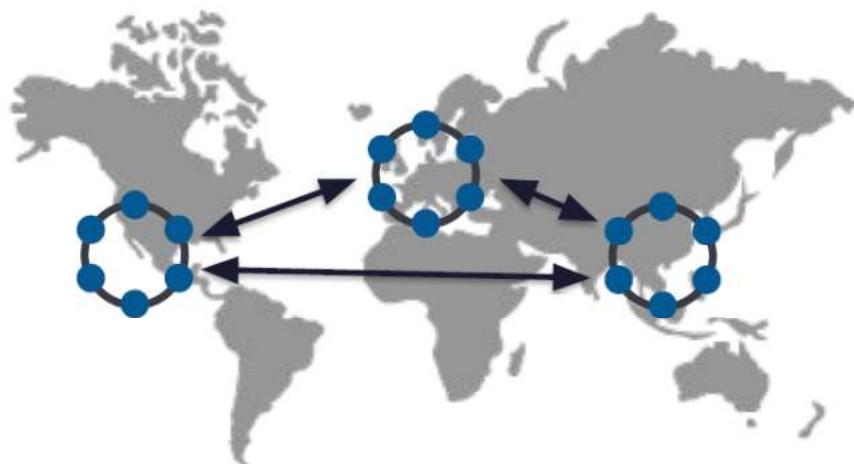


- Observe que quando um Nô não é responsável pelo dado, outros Nôs que possam possuí-lo são consultados;
- Veja que mesmo se um Nô em algum momento estiver fora do ar, outros poderão atender a requisição;
- A replicação diminui a necessidade de backups em um BD, pois vários Nôs poderão ter um dado específico;

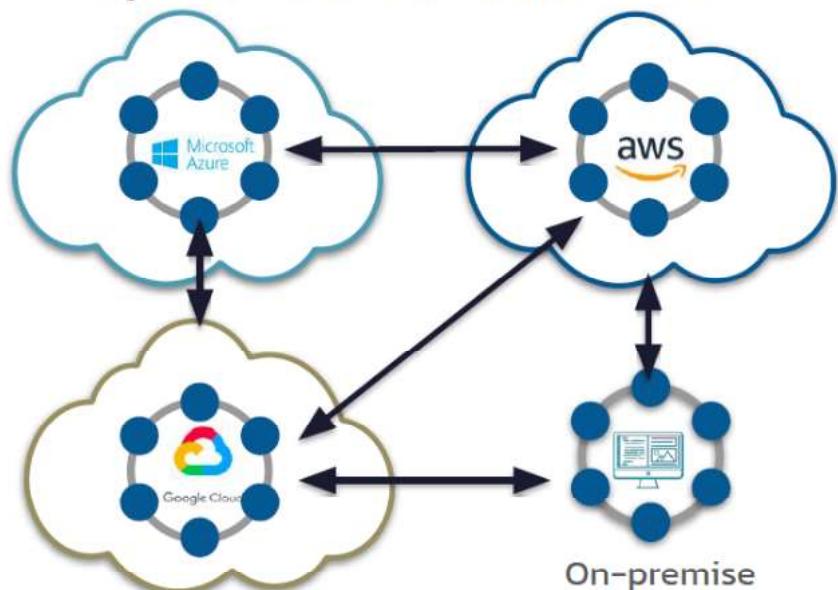
4.1. Bancos Orientados a Colunas

Apesar de um Ring já ser formado por vários Nós (máquinas), uma grande empresa pode possuir N Rings de dados.

- Geographic Distribution



- Hybrid-Cloud and Multi-Cloud



- Rings podem ser completamente proprietários ou serem formados em plataformas de serviços em nuvem como Google Cloud, AWS, Microsoft Azure, etc.
- Apesar do Cassandra ter uma arquitetura moderna e naturalmente distribuída, veremos que também possui algumas restrições que devemos ficar atentos.

4.1. Bancos Orientados a Colunas

4.1.3. Estruturas de Armazenamento do Cassandra

Em alguns momentos teremos o sentimento que estaremos trabalhando com um BD SQL, porém, não se engane! O Cassandra tem suas particularidades, fique atento!

1. Tudo começa com uma Célula, que nada mais é que uma interseção entre uma linha e uma coluna de uma Tabela para se armazenar um dado;

John

2. Em seguida temos uma LINHA, que nada mais é que uma VISUALIZAÇÃO de um conjunto de valores (registro) pertencentes a algo e que podem ser retornados juntos;

1	John	Doe	Wizardry
---	------	-----	----------

3. E agora um dos "Pulos do Gato!" do Cassandra, que é o conceito de PARTIÇÃO (PARTITION): um conjunto de linhas que possuem a mesma Chave de Partição (PARTITION KEY).

ps: observe no exemplo a coluna Departamento funcionando como uma PARTITION KEY.

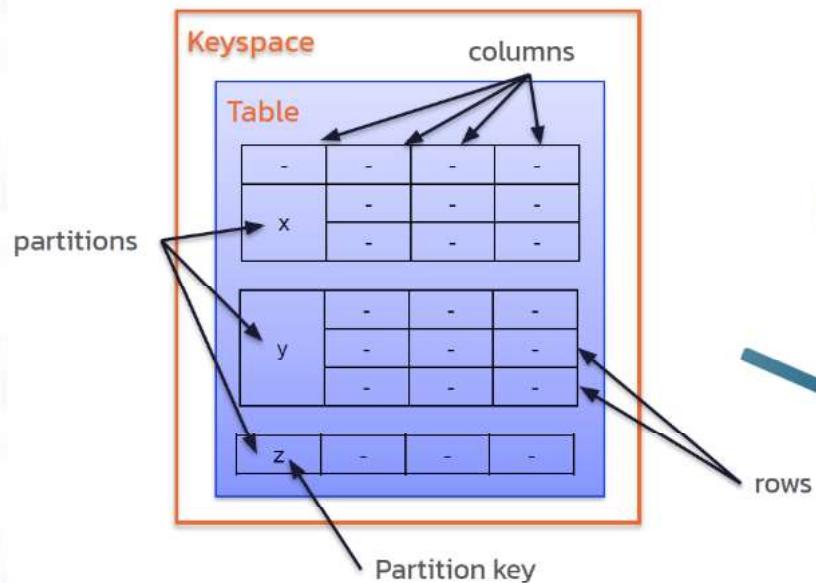
ID	First Name	Last Name	Department
1	John	Doe	Wizardry
399	Marisha	Chapez	Wizardry
415	Maximus	Flavius	Wizardry

4.1. Bancos Orientados a Colunas

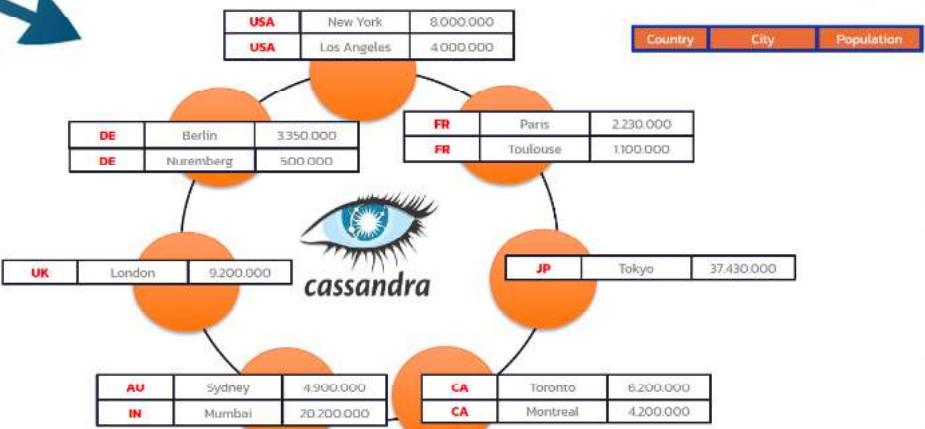
No Cassandra, dados que possuem uma mesma Partition Key são armazenados JUNTOS, em um MESMO NÓ e já de forma ordenada.

("Store together what you retrieve together!")

Uma KEYSPACE é como se fosse um Esquema de um BD Relacional, ou seja, onde armazenamos um conjunto de Tabelas.



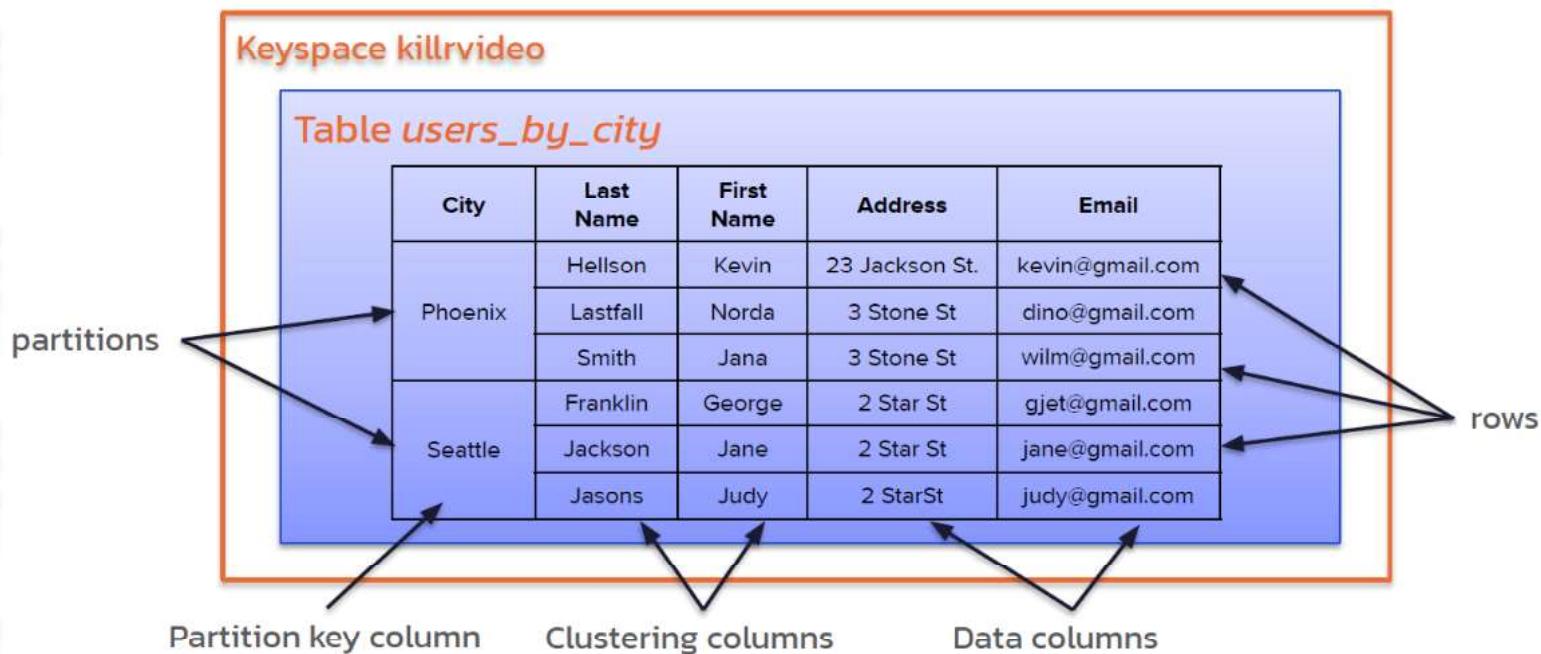
Estrutura Geral de Organização



Como os dados ficam depois de armazenados.

4.1. Bancos Orientados a Colunas

A Partition Key ajuda o Cassandra a identificar em qual N o um dado ser  armazenado, j a as Clustering Columns ajudam o Cassandra a definir o crit rio de ordena o destes dados dentro do N o.



- Professor ent o quer dizer que a minha Partition Key   como se fosse a minha Chave Prim ria (Primary Key) da Tabela?
- Infelizmente N O!
- Ah, ent o essas Clustering Keys formam a minha Chave Prim ria?
- Tamb m N O jovem!"



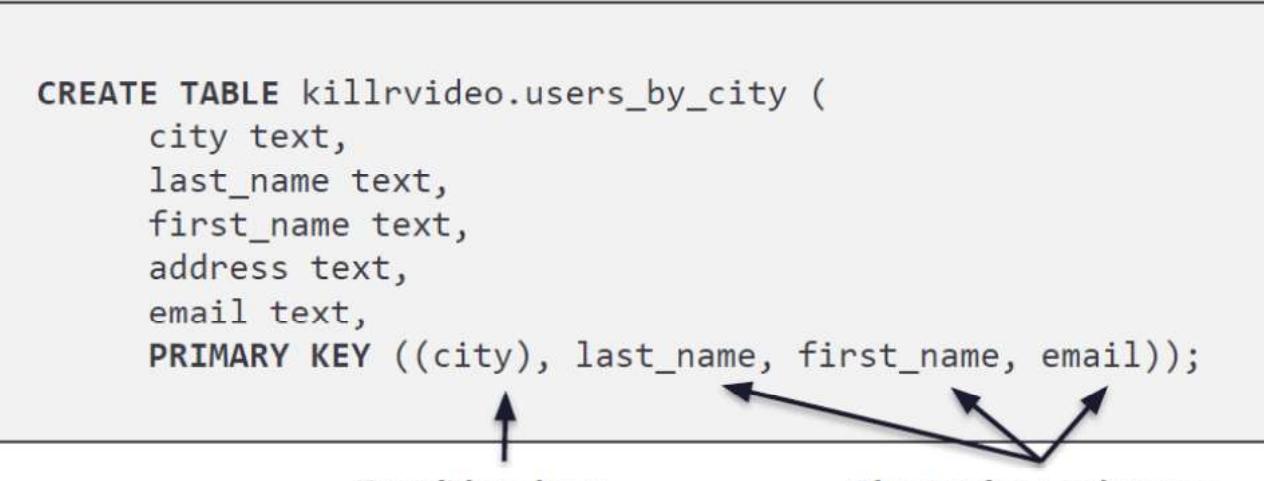
4.1. Bancos Orientados a Colunas

No Cassandra, uma Chave Primária (Primary Key) consiste de pelo menos uma Partition Key e zero ou mais Clustering Columns.

Lembre-se que o objetivo de uma Primary Key é oferecer UNICIDADE a uma linha de dados em um BD, por isso, deve ser bem planejada, da mesma forma que acontecia nos BDs Relacionais.

Observe a formação
de uma PK no
Cassandra.

```
CREATE TABLE killrvideo.users_by_city (
    city text,
    last_name text,
    first_name text,
    address text,
    email text,
    PRIMARY KEY ((city), last_name, first_name, email));
```



Good Examples:

```
PRIMARY KEY ((city), last_name, first_name, email);
```

```
PRIMARY KEY (user_id);
```

Bad Example:

```
PRIMARY KEY ((city), last_name, first_name);
```

Exemplos de Planejamentos Bons e Ruins
para uma Primary Key.

4.1. Bancos Orientados a Colunas

No Cassandra, devemos planejar nosso BD para evitar partições muito grandes, para que isso não atrapalhe o que o BD tem de melhor: a rápida leitura e escrita de dados.

Exemplo:

Imagine uma infraestrutura de IoT, com Nós espalhados por todo o mundo e milhares de sensores enviando seus status para o Cassandra a cada 10s.

1)

`PRIMARY KEY ((sensor_id), reported_at);`



2)

`PRIMARY KEY ((sensor_id, month_year), reported_at);`



- Veja que se a Primary Key fosse planejada como na situação 1, teríamos todos os dados de um sensor na mesma Partição, mas poderíamos formar partições gigantescas;
- Já na situação 2 também teríamos dados de um sensor em uma mesma partição, mas com maior modularização. No caso do exemplo, é como se a cada mês criássemos automaticamente uma nova partição para guardar os dados de um respectivo sensor (este conceito é chamado de Bucketing);

4.1. Bancos Orientados a Colunas

A forma mais prática de se explorar e manipular os dados de um BD Cassandra é por meio de um Client online e oficial distribuído pela empresa DataStax. Estamos falando do Astra.

- É um serviço nativo na nuvem chamado **Cassandra-as-a-service** que oferece acesso ao BD Cassandra de forma simplificada;
- Elimina as complexidades e atrasos para instalar, operar e escalar um BD Cassandra;
- Permite armazenar os dados diretamente na nuvem, evitando que o desenvolvedor se preocupe com questões de performance, disponibilidade e acessibilidade dos dados;
- Nos fornece 25GB de armazenamento gratuito;



Fully managed Cassandra
Without the ops!

Acesse: <https://www.datastax.com/cassandra>

e faça seu cadastro/acesso ao ASTRA em

[Try For Free](#)

4.2. Cassandra Query Language (CQL)

Bancos de Dados Relacionais utilizam do SQL como linguagem para estruturação, manipulação e consulta de dados. O Cassandra usa do Cassandra Query Language (CQL).

Antes de começarmos a trabalhar com o CQL, vamos criar um novo Database no DataStax Astra.

Create a Database

Create Database



Enter the Basic Details

Database Name *	dbiot	Keyspace Name *	ksiot
Give it a memorable name - this can't be changed later.		Name your keyspace to reflect your data model.	

- O nome do Database precisa ser um nome original que resume bem o que vai ser armazenado dentro dele;
- A Keyspace é como se fosse nosso Esquema, onde partes das nossas Tabelas serão armazenadas em diferentes Nós;
- Depois de criado o novo BD, clique no nome dele e em seguida selecione a aba CQL Console.

Dashboard / dbiot

Overview Health Connect CQL Console Settings

```
Connected as renzokuken31@gmail.com.
Connected to cndb at cassandra.ingress:9042.
[cqlsh 6.8.0 | DSE DB 4.0.0.6811 | CQL spec 3.4.5 | Native protocol v4]
Use HELP for help.
token@cqlsh> 
```

4.2. Cassandra Query Language (CQL)

4.2.1. Criando Keyspaces e Tabelas

Cassandra é Schemaless no sentido de, quando uma coluna de um registro não possui um dado preenchido, não se aloca armazenamento pra ela de forma desnecessária (como acontecia nos BDs SQL). Porém, diferente de outros BDs NoSQL, o Cassandra exige uma estrutura inicial.

Queremos que nossos dados sejam armazenados na Keyspace que criamos. Por isso, precisamos setá-la por meio do comando:

```
> USE ksiot;
```

Para criarmos uma estrutura de Tabela inicial dentro deste Keyspace:

```
> CREATE TABLE sensor(  
    id BIGINT,  
    ano INT,  
    mes INT,  
    dia INT,  
    hora TEXT,  
    leitura FLOAT,  
    PRIMARY KEY((id,ano,mes),dia,hora)  
)
```

Observe a semelhança com o SQL, porém, em breve veremos suas particularidades ;)

Para verificar se a Tabela foi criada com sucesso, execute o comando:

```
> DESCRIBE TABLES;
```

4.2. Cassandra Query Language (CQL)

Uma vez que esta Tabela tenha sido criada, o Cassandra já vai prepará-la para que suas partes (Partitions) sejam distribuídas para os diferentes Nós de um Ring.

Isso traz um ganho avassalador de performance para buscas individuais, pois os dados não ficarão mais dentro de um único Nós, que em algum momento possa estar sobrecarregado.

Para excluirmos uma tabela, podemos executar o comando:

```
> DROP TABLE sensor;
```

Para adicionarmos mais campos a nossa tabela:

```
> ALTER TABLE sensor ADD local TEXT;
```

```
> ALTER TABLE sensor ADD modelo TEXT;
```

Para ver como está nossa tabela até o momento:

```
> DESCRIBE sensor;
```



4.2. Cassandra Query Language (CQL)

Mas qual o critério que o Cassandra usa de base para quebrar uma tabela em Partitions (Partições)?

O detalhe está no que colocamos na nossa PRIMARY KEY (PK). Diferente do BD Relacional, aqui ela é subdividida em DUAS PARTES:

PRIMARY KEY((id,ano,mes),dia,hora)

Parte 1 Parte 2

Parte 1 : É a parte da Primary Key chamada de Partition Key. É ela que é usada de base pelo Cassandra para identificar em qual Partition um registro vai se encontrar;

Parte 2 : É a parte da Primary Key chamada de Clustering Column(s). Representa um ou mais campos que são utilizados pelo Cassandra para ordenar os dados dentro de uma Partition;

O grande "Pulo do Gato!" desta abordagem é que os dados com uma mesma Partition Key SEMPRE serão armazenados em um mesmo Nô e já estarão automaticamente ordenados se baseando na Clustering Columns. Isso faz a busca de um dado ser MUITO RÁPIDA!

4.2. Cassandra Query Language (CQL)

4.2.2. Inserindo, Atualizando e Deletando Dados

Ao inserir um novo registro no Cassandra, nem todos os campos precisam ser preenchidos. O Null que aparece quando realizamos um SELECT aparece mais para fins de apresentação, mas um dado não preenchido no Cassandra nunca gasta armazenamento de forma desnecessária.

Para inserirmos novos registros em uma Tabela:

Inserção Completa:

```
> INSERT INTO sensor(id,ano,mes,dia,hora,leitura,local,modelo)  
VALUES(25,2021,5,10,'18:30',35,'SRS','TexasTemp1');
```

Inserção Parcial:

```
> INSERT INTO sensor(id,ano,mes,dia,hora,leitura)  
VALUES(72,2021,5,10,'19:00',27);
```

Se tentarmos inserir um novo registro com a mesma chave, teremos problemas?

Faça o teste e veja que não! O Cassandra simplesmente vai sobrescrever o registro que ele já possuia ;)

4.2. Cassandra Query Language (CQL)

É importante ressaltar que o Cassandra, assim como a grande maioria dos BDs NoSQL, é um banco que busca a DENORMALIZAÇÃO dos dados. Isso significa não realizar relacionamentos entre Tabelas, mas já prepará-las para conter tudo que for necessário de uma só vez.

Mas então como posso criar estes registros aninhados para evitarmos os relacionamentos?

Uma ótima alternativa, por exemplo, é usar do conceitos de LISTAS (Lists) e MAPAS (Maps):

LISTAS (Lists):

```
> ALTER TABLE sensor ADD obs LIST<text>;  
  
> INSERT INTO sensor(id,ano,mes,dia,hora,leitura,obs)  
VALUES(72,2021,5,10,'19:00',29,['Valor estavel','Sem delay na leitura']);
```

MAPAS (Maps):

```
> ALTER TABLE sensor ADD limites MAP<text,decimal>;  
  
> INSERT INTO sensor(id,ano,mes,dia,hora,leitura,obs,limites)  
VALUES(72,2021,5,10,'19:00',29,['Valor estavel','Sem delay na leitura'],{'Superior':38,'Inferior':25});
```

4.2. Cassandra Query Language (CQL)

No Cassandra não existe UPDATE e DELETE sem WHERE! Pode parecer uma desvantagem, mas isso exigiria do BD vasculhar TODOS os Nós do Ring, e isso poderia matar o que ele oferece de melhor: DESEMPENHO nas operações.

No Cassandra todo UPDATE deve ser feito passando-se a PRIMARY KEY, que é formada pelas PARTITION KEYS + CLUSTERING COLUMNS.

> UPDATE sensor

SET modelo = 'BuffaloMoist1', local = 'PA'

WHERE id = 72 AND ano = 2021 AND mes = 5;



> UPDATE sensor

SET modelo = 'BuffaloMoist1', local = 'PA'

WHERE id = 72 AND ano = 2021 AND mes = 5 AND dia = 10 AND hora = '19:00';



- No UPDATE atenção! Se uma Clustering Column for digitada errada ou não existir, um novo registro será criado;
- No WHERE também poderá ser usado outros campos que não sejam os da PK, desde que eles sejam indexados como ÍNDICES SECUNDÁRIOS (será visto no laboratório);
- Para se atualizar colunas em mais de um registro de uma só vez, podemos usar da palavra-chave IN na última coluna da Chave Primária para especificarmos quem vai receber a alteração, desde que as Partition Keys sejam iguais para estes registros.

Ex: ...mes = 5 AND dia = 10 AND hora IN ('19:00', '19:30', '20:00');

4.2. Cassandra Query Language (CQL)

4.2.3. Buscando Dados

A mesma regra do UPDATE também serve para o SELECT: sempre adicione uma Cláusula WHERE passando a PK do registro.

```
> SELECT local,leitura  
    FROM sensor  
    WHERE leitura > 28;
```



```
> SELECT local,leitura  
    FROM sensor  
    WHERE id = 72 and ano = 2021 and mes = 5 and dia = 10 and hora = '19:00';
```



Podemos "forçar" o Cassandra a fazer buscas condicionais e sem usar uma PK, por meio do ALLOW FILTERING:

```
> SELECT local,leitura  
    FROM sensor  
    WHERE leitura > 28 ALLOW FILTERING;
```



- Apesar deste recurso existir, CUIDADO com ele! Dependendo do número de Nós que tiver em seu Ring, a busca pode se tornar MUITO custosa;
- Usar o comando LIMIT n no final de buscas deste tipo pode ajudar a melhorar o desempenho da operação.

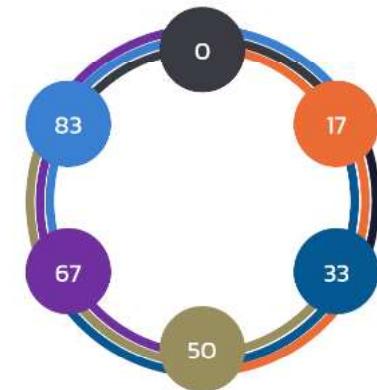
4.2. Cassandra Query Language (CQL)

Como estamos lidando com um BD naturalmente distribuído, um conceito importante a ser discutido é o de **CONSISTENCY** no Cassandra. Quando inserimos ou buscamos um registro, em quantos Nós eu quero que ele seja inserido/buscado quando o comando for executado?

Basta executarmos os seguintes comandos no Console CQL:

- > **consistency all** -> o dado será inserido ou buscado em todos os Nodes;
- > **consistency one** -> o dado será inserido ou buscado em um único Node;
- > **consistency quorum** -> o dado será inserido ou buscado na maioria dos Nodes;

- A vantagem do all ou quorum é que sempre o registro mais atualizado dos Nodes serão retornados. Isso ajuda as inserções ou buscas terem mais Consistência;
- Mas claro, isso é uma balança! Se você está buscando Consistência, é natural que se gaste mais poder computacional (ou seja, se perde performance).



O Cassandra apesar de ser um BD moderno, ele não é uma verdade absoluta! Você pode chegar a usá-lo e ter desempenho pior que o SQL, pois você pode não ter compreendido sua filosofia para usá-lo da melhor forma possível.

4.2. Cassandra Query Language (CQL)

Exercício Proposto

Baseado no que vimos até o momento de CQL, crie comandos que respondam às seguintes questões:

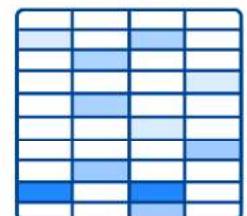
1. Para o sensor de id 25, no ano de 2021, mes 5 e dia 10, adicione mais uma leitura de 37 no horário das 19:00 de forma completa (ou seja, inserindo todos os campos);
2. Para o sensor de id 25, no ano de 2021, mes 5 e dia 11, adicione mais uma leitura de 36 no horário das 18:30 de forma parcial (ou seja, preenchendo apenas os campos necessários);
3. Para o último registro inserido, preencha seus campos Nulos por meio do comando UPDATE;

4. Para o sensor de id 25, no ano de 2021 e mes 5, busque o modelo, dia, hora e leitura dos registros dos dias 10 e 11;

(Observe que aqui como a busca vai acontecer dentro de uma mesma Partition, você pode utilizar do operador IN para selecionar os dias específicos)

5. Busque todos os registros para o sensor de id 25, no ano de 2021, mes 5, dia 10 e hora >= '18:30';

(Observe que aqui também não precisaremos do ALLOW FILTERING, e mesmo que tivéssemos centenas de registros, a busca seria rápida pois se encontram na mesma Partition. Observe a última parte da Chave Primária permitindo realizarmos uma condição para busca de múltiplos registros.



4.3. Conexão Python + Cassandra

4.3.1. Configurando uma Conexão

A fim de criarmos Clients poderosos capazes de se conectarem remotamente ao Cassandra, devemos utilizar das orientações fornecidas pela aba "Connect" do Astra.

Dentro de um BD específico no Astra e na aba "Connect", serão apresentadas várias possibilidade de conexão. Aqui utilizaremos a forma mais popular, que é a "Connect using a driver" com Python.

1. Primeiramente, faça download do arquivo .zip "[Download Bundle](#)" no Astra. Ele é um pacote de arquivos responsável por nos conectar remotamente e com segurança ao Cassandra;
2. Em seguida, é necessário gerar e baixar um "[Application Token](#)", que nos fornecerá um nome de usuário e senha para nos conectarmos ao Cassandra;

[Download Bundle](#) ▾

Application Tokens

Generate a new token

Select which role you want to attach to your token. The permissions for that role will be displayed before generating your token to ensure you give your application the right permissions.

Select Role

Admin User

Generate Token

Selecione o papel de Admin User, gere o Token em "Generate Token" e faça download do arquivo .CSV com as credenciais.

4.3. Conexão Python + Cassandra

3. Agora dentro do PyCharm, vamos criar um novo projeto para realizarmos o processo de conexão (Sugestão de nome de projeto: ProjetoClientCassandra);
4. Coloque dentro da pasta do seu projeto o arquivo de Bundle que fizemos download (Neste caso, como estamos trabalhando com o dbiot, é o arquivo [secure-connect-dbiot.zip](#));
5. Instale a biblioteca [cassandra-driver](#) por meio do módulo de instalações do PyCharm;
6. Utilize no projeto o código gerado automaticamente pelo Astra (parecido com o da figura ao lado) para testarmos uma conexão;
7. Por fim, como o Bundle já está dentro da pasta do nosso projeto, apague o <>/PATH/TO/>> e baseando nas informações do .CSV que geramos, preencha os dados <><>CLIENT ID<>> e <><>CLIENT SECRET<>>.

```
from cassandra.cluster import Cluster
from cassandra.auth import PlainTextAuthProvider

cloud_config = {
    'secure_connect_bundle': '<>/PATH/TO/>>secure-connect-dbiot.zip'
}
auth_provider = PlainTextAuthProvider('<><>CLIENT ID<>>', '<><>CLIENT SECRET<>>')
cluster = Cluster(cloud=cloud_config, auth_provider=auth_provider)
session = cluster.connect()

row = session.execute("select release_version from system.local").one()
if row:
    print(row[0])
else:
    print("An error occurred.")
```

8. Rode seu código e observe se recebeu como retorno a versão do Cassandra que está se conectando remotamente ;).

4.3. Conexão Python + Cassandra

4.3.2. Exemplo de Client no Cassandra

Se baseando no código de conexão gerado pelo Astra, vamos adaptá-lo para criarmos um Client exemplo de acesso ao Cassandra.

Este Client terá como objetivo ler um conjunto de valores de um sensor específico para traçar um gráfico de como estão suas leituras.

1. Primeiramente, vamos setar qual a keyspace que queremos acessar:

```
session.set_keyspace('ksiot')
```

2. Vamos excluir os registros que temos até o momento em nossa Tabela:

```
result = session.execute('DELETE FROM sensor WHERE id IN (25,72) and ano = 2021  
and mes = 5;')
```

```
if result is not None:
```

```
    print("Sucesso!")
```

ps: observe o uso do operador IN para exclusão de múltiplos registros. Isso é possível pois os registros devem ser inseridos explicitamente, evitando que um acesso em todo o Ring aconteça.



4.3. Conexão Python + Cassandra

3. Vamos inserir novos registros para um sensor:

```
result = session.execute(  
    'BEGIN BATCH '+  
    'INSERT INTO sensor(id,ano,mes,dia,hora,leitura,limites,local,modelo) VALUES(25,2021,5,10,\'18:45\',33,{\'Inferior\':30,\'Superior\':38},\'SRS\',\'NestTemp1\');'+  
    'INSERT INTO sensor(id,ano,mes,dia,hora,leitura,limites,local,modelo) VALUES(25,2021,5,10,\'19:00\',34,{\'Inferior\':30,\'Superior\':38},\'SRS\',\'NestTemp1\');'+  
    'INSERT INTO sensor(id,ano,mes,dia,hora,leitura,limites,local,modelo) VALUES(25,2021,5,10,\'19:15\',37,{\'Inferior\':30,\'Superior\':38},\'SRS\',\'NestTemp1\');'+  
    'INSERT INTO sensor(id,ano,mes,dia,hora,leitura,limites,local,modelo) VALUES(25,2021,5,10,\'19:30\',38,{\'Inferior\':30,\'Superior\':38},\'SRS\',\'NestTemp1\');'+  
    'INSERT INTO sensor(id,ano,mes,dia,hora,leitura,limites,local,modelo) VALUES(25,2021,5,10,\'19:45\',37,{\'Inferior\':30,\'Superior\':38},\'SRS\',\'NestTemp1\');'+  
    'APPLY BATCH;'  
)  
if result is not None:  
    print("Sucesso!")
```

ps: observe o uso do BATCH para realização de múltiplos comandos de uma só vez;

4. Agora vamos buscar os registros do sensor por meio do:

```
result = session.execute(  
    "SELECT * FROM sensor "+  
    "WHERE id=25 AND ano = 2021 AND mes = 5 AND dia = 10 AND hora >= '18:45'"  
).all()  
  
if result is not None:  
    for r in result:  
        print(r)
```

5. Instale a biblioteca **matplotlib** por meio do módulo de instalações do PyCharm e a importe no seu projeto com o código `import matplotlib.pyplot as plt;`

4.3. Conexão Python + Cassandra

6. Hora de extrairmos as informações dos registros para plotarmos o gráfico:

```
if result is not None:
```

```
    # Listas que receberao os valores para os eixos X e Y
    eixoX = []
    eixoY = []
    # Dicionario que guardara a cor de cada ponto no grafico
    corPontos = {}
    # Extraindo do resultado o local e nome do sensor pesquisado
    local = result[0][7]
    sensor = result[0][8]
```

```
    # EixoX guardara as horas das leituras
```

```
    # EixoY guardara as leituras
```

```
    for r in result:
```

```
        eixoX.append(r[4])
```

```
        eixoY.append(r[5])
```

```
        # Se a leitura estiver dentro dos limites, bolinha verde
```

```
        # Senao, bolinha vermelha
```

```
        if r[6]['Inferior'] < r[5] < r[6]['Superior']:
```

```
            corPontos[r[4]] = 'green'
```

```
        else:
```

```
            corPontos[r[4]] = 'red'
```

```
# Customizando informacoes do grafico
```

```
plt.title('Sensor ' + sensor + ' em ' + local)
```

```
plt.xlabel('Horarios'), plt.ylabel('Leituras')
```

```
# Plotando as bolinhas
```

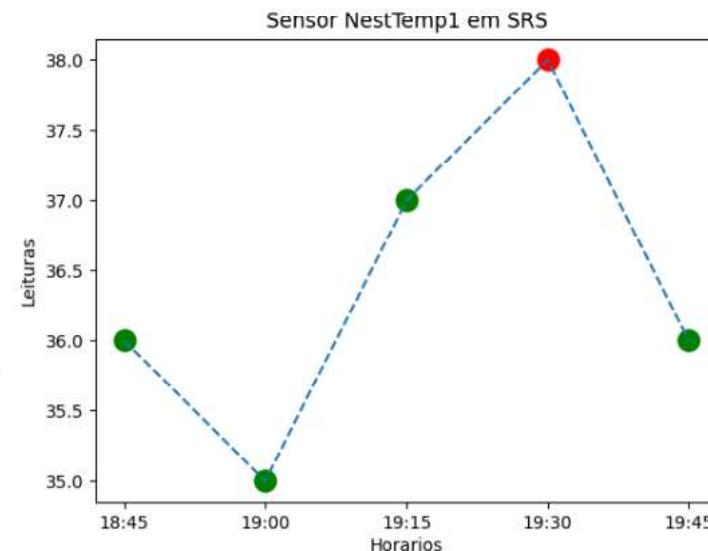
```
for i in range(0, len(eixoX)):
    plt.scatter(eixoX[i], eixoY[i],
                color=corPontos[eixoX[i]], s=150)
```

```
# Plotando os tracos conectando as bolinhas
```

```
plt.plot(eixoX, eixoY, '--')
```

```
# Mostrando o grafico
```

```
plt.show()
```



4.3. Conexão Python + Cassandra

Vimos a filosofia e recursos fundamentais do Cassandra. Porém, para mais detalhes e recursos avançados, sempre usem como base a Documentação Oficial!

Links importantes:

1. Documentação do Cassandra fornecida pela DataStax:

<https://docs.datastax.com/en/cql-oss/3.3/>

2. Documentação oficial do Cassandra fornecida pela Apache:

<https://cassandra.apache.org/doc/latest/>



**FIM
DO
CAPÍTULO 4**



EXERCÍCIOS