Universidade Federal de Campina Grande

COMPUTAÇÃO UFCG

Cassandra Database

No-SQL

Jayne de Morais Silva
Programa de Pós-Graduação em Ciência da Computação (Mestrado)
Disciplina: **Banco de Dados -** Prof. **Dr. Carlos Eduardo**
Novembro, 2020

ÍNDICE

I. Introdução

II. Estrutura

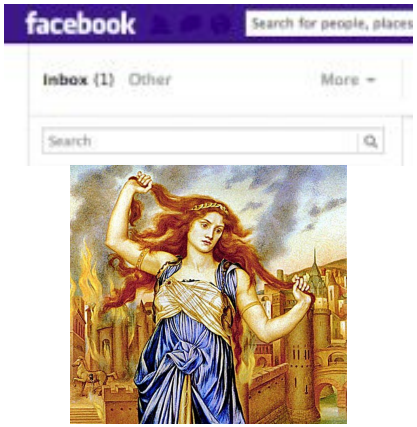
III. Demonstração

IV. Vantagens e Desvantagens

V. Aplicações Notáveis

VI. Referências

I. INTRODUÇÃO



2008

Motor de Busca

Caixa de Entrada de Mensagens do Facebook

2008

Open-Source

Código aberto que pode ser adaptado para diversos fins.

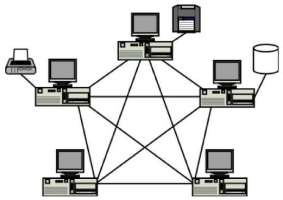
2009

Apache Foundation

*Licença Apache 2.0

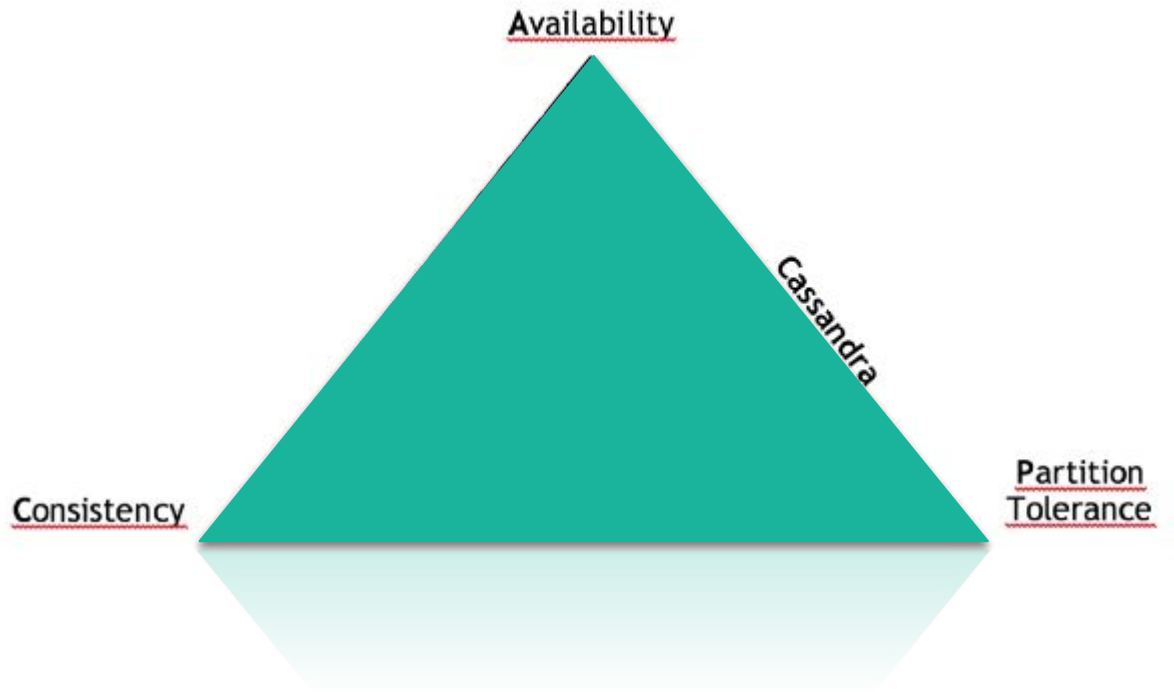
DATASTAX

ARQUITETURA



- Arquitetura **Distribuída** e **Descentralizada**
    - Modelo de **Distribuição do Sistema**
- Escalabilidade Horizontal
- Forma de **Organização** dos **Dados**

TEOREMA CAP



*É possível ter todas as três garantias no Cassandra, mas *não ao mesmo tempo.*

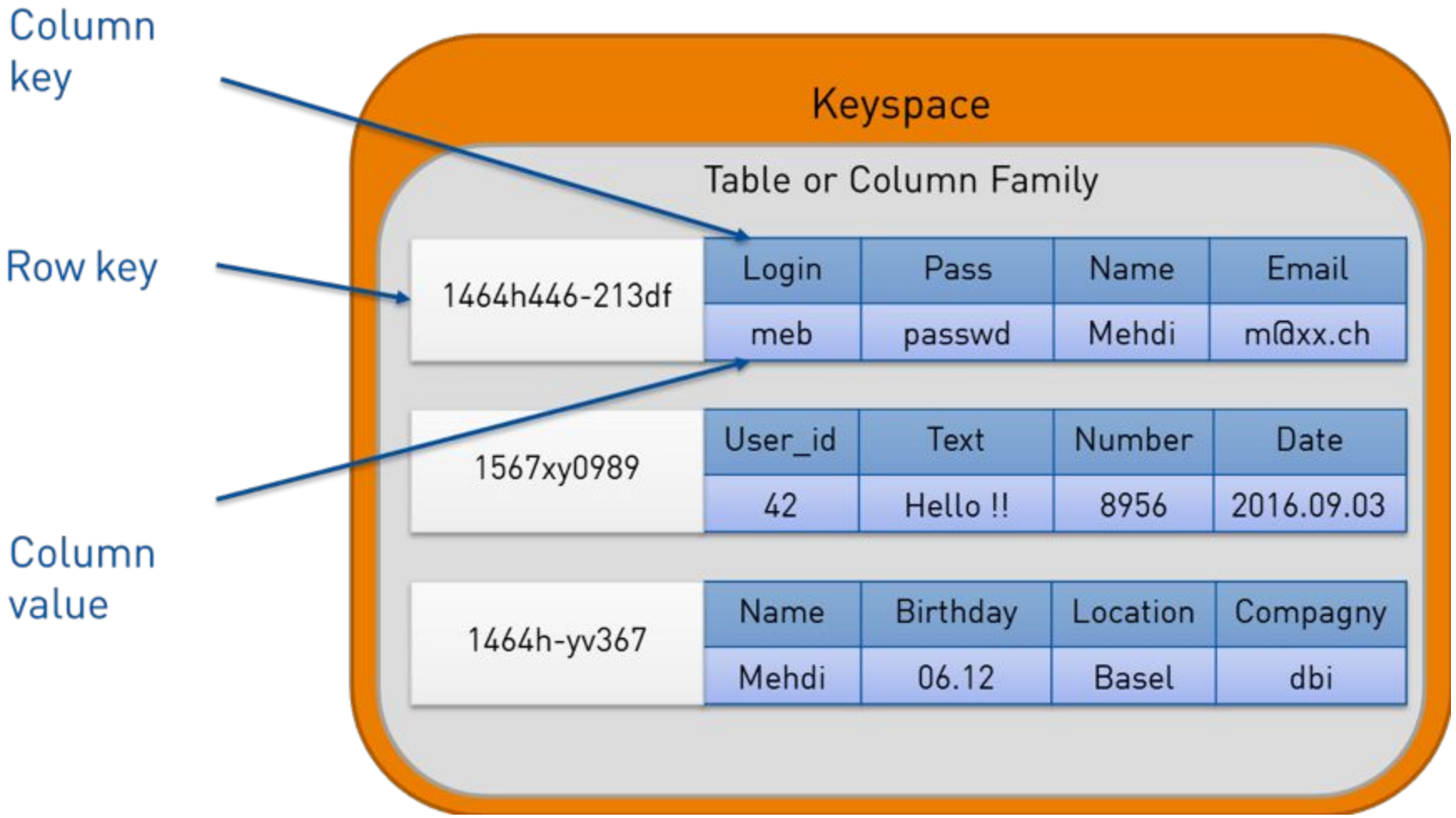
II. ESTRUTURA

MODELO DE DADOS

- **Tipo Colunar**
  - Estrutura agregada de duas camadas, onde existe um **identificador** de linha e um **mapa** com valores mais detalhado;
  - Dados **desnormalizados** e **amplamente consultados**;
  - **Linhas diferentes** na mesma tabela não precisam compartilhar o mesmo conjunto de **colunas**.

ESTRUTURA

- **Keyspace**: Conjunto de tabelas;
- **Famílias de Colunas**: Contêiner para um conjunto de linhas ;
- **Linhas**: Contém uma uma chave primária e **um conjunto de colunas relacionadas;**
- **Colunas**: representam um conjunto de **chave/valor**:
  - **Column key**: Nome da coluna;
  - **Column value**: É o valor que está sendo persistido;
  - **Timestamp**: Esse campo é utilizado para resolver conflitos e determinar qual é o valor mais atual.


Column key
Row key
Column value
Keyspace
Table or Column Family
1464h446-213df
Login
Pass
Name
Email
meb
passwd
Mehdi
m@xx.ch
1567xy0989
User_id
Text
Number
Date
42
Hello !!
8956
2016.09.03
1464h-yv367
Name
Birthday
Location
Compagny
Mehdi
06.12
Basel
dbi

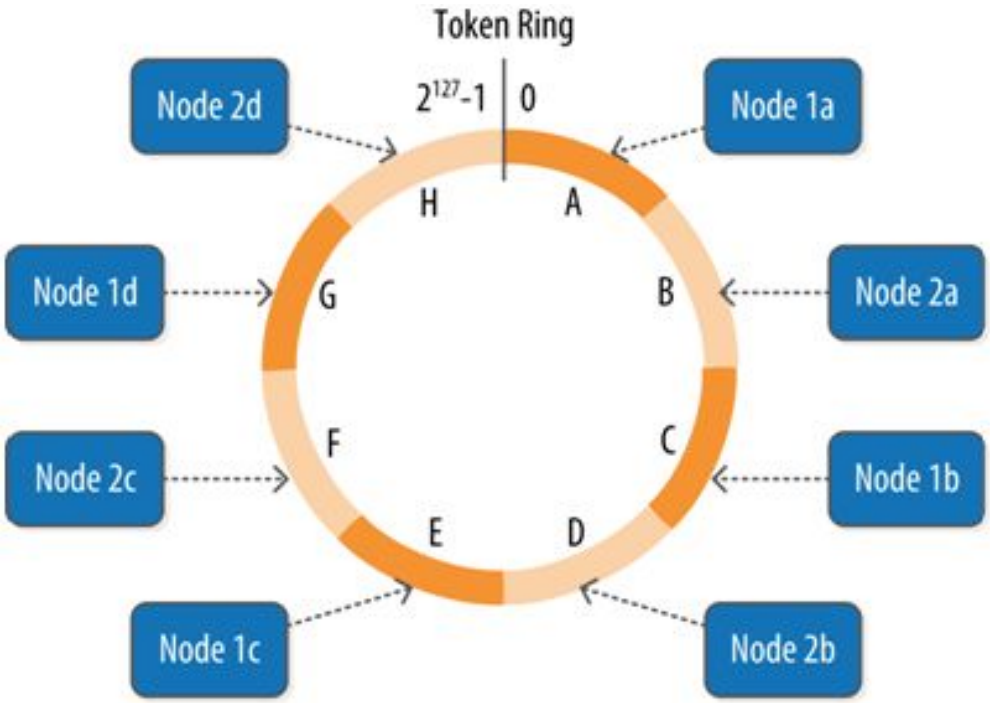
FUNCIONAMENTO I

- Partition Key Table
- *Data Sharding*
  - Distribuição das partições entre nós do cluster e então **replicadas** em múltiplos nós;
  - Alta Disponibilidade e Tolerância a Falhas

- **Em que nó do cluster cada partição será gravada?**

ARQUITETURA EM ANEL

- Faixas (tokens) baseadas na Hash de chave
  - Inteiro de 64 bits (–263 até 263–1)



FUNCIONAMENTO II

- **Detecção** de nós com indícios de **falha;**
  - *Gossip Protocol.*

- **Atualização** de Dados e **Inserção** de Dados;

III. DEMONSTRAÇÃO

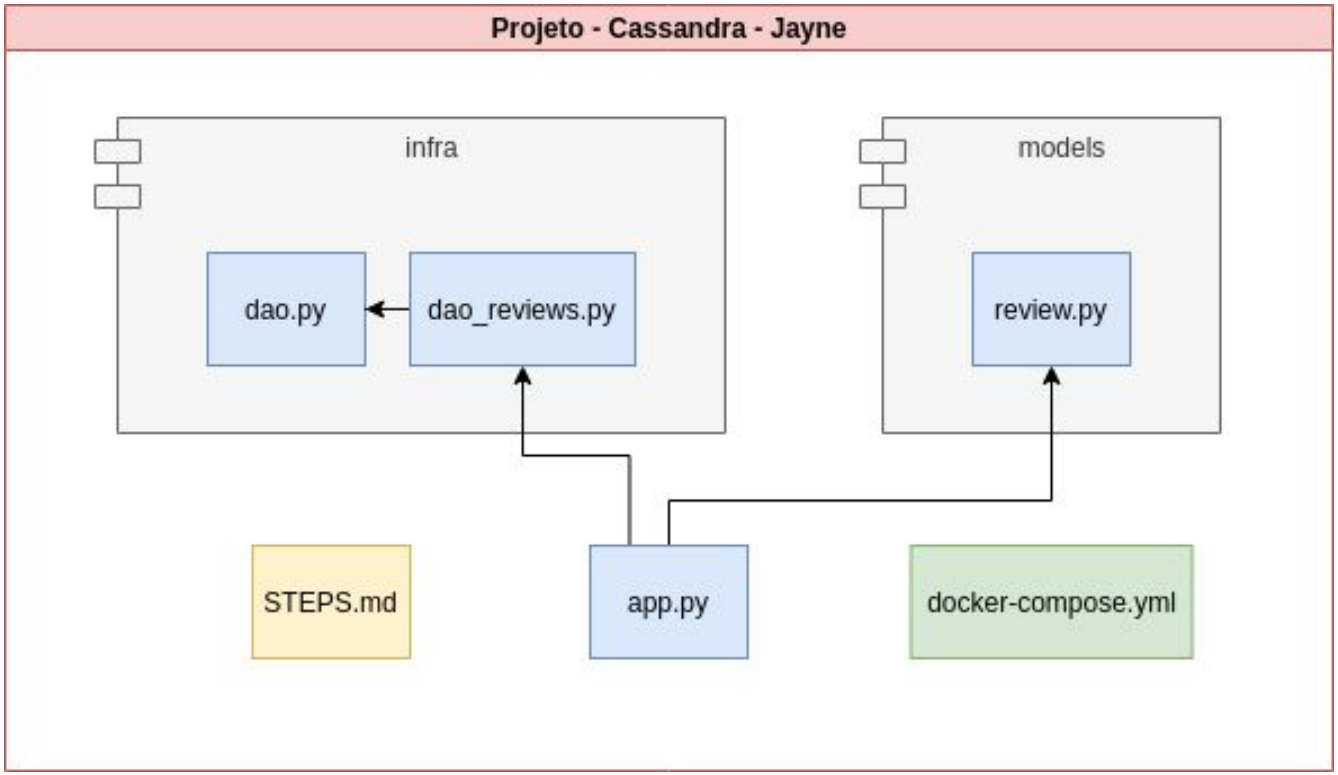
CASO DE USO

Reviews de usuários sobre Filmes

- *Objetivo e Justificativa: Utilização do banco NoSQL Cassandra para otimização sob as consulta sobre reviews, visto que existem muitos dados sobre reviews e os mesmos são utilizados para aplicações específicas, como em modelos de Ciência de Dados.*

*Caso de Uso Fictício

ARQUITETURA



Instalação e Configuração

- https://github.com/JayneMorais/CassandraDatabase
- *Cassandra Query Language*

KEYSPACE

```
22
23      def create_keyspace(self):
24
25          query = """
26                  CREATE KEYSPACE IF NOT EXISTS reviews_db
27                  WITH REPLICATION = {
28                  'class' : 'SimpleStrategy',
29                  'replication_factor' : 2
30                  };
31                  """
32          self.__session.execute(query)
33          self.__session.set_keyspace("reviews_db")
```

TABLE

```
38
39      def create_table(self):
40          query = """
41                  CREATE TABLE IF NOT EXISTS reviews_db.reviews (
42                  id UUID,
43                  gostou INT,
44                  usuario_id TEXT,
45                  filme_id TEXT,
46                  comentario TEXT,
47                  criado_em TIMESTAMP,
48                  atualizado_em TIMESTAMP,
49                  PRIMARY KEY (id));
50          """
51          self.__session.execute(query)
```

CREATE

```
80
81      def cria_review(self, review):
82          try:
83              rows = self.__session.execute_async(
84                  """
85                  INSERT INTO reviews_db.reviews (id, usuario_id, filme_id, comentario, gostou, criado_em, atualizado_em)
86                  VALUES (%s, %s, %s, %s, %s, %s, %s)
87                  """,
88                  (
89                      uuid.uuid1(),
90                      review.usuario_id,
91                      review.filme_id,
92                      review.comentario,
93                      review.gostou,
94                      unix_time_millis(datetime.datetime.now()),
95                      unix_time_millis(datetime.datetime.now())
96                  ),
97              )
98
99              response = to_dict_list(rows)
100             return {"code": 201, "response": "criado", "count": len(response), "data": response }
101         except Exception as e:
102             return {"code": 500, "response": "erro " + str(e)}
103
```

READ

```
52
53      def listar_reviews(self):
54          try:
55              rows = self.__session.execute_async(
56                  "SELECT * FROM reviews_db.reviews",
57              )
58
59              response = to_dict_list(rows)
60
61              return {"code": 200, "response": "ok", "count": len(response), "data": response }
62          except Exception as e:
63              return {"code": 500, "response": "erro "+str(e)}
64
65      def busca_review(self, review_id):
66
67          review_id = uuid.UUID(review_id)
68
69          try:
70              rows = self.__session.execute_async(
71                  """
72                  SELECT * FROM reviews_db.reviews WHERE id=%s
73                  """,
74                  [review_id]
75              )
76
77              return {"code": 200, "response": "ok", "data": to_dict_list(rows)}
78          except Exception as e:
79              return {"code": 500, "response": "erro "+str(e)}
80
```

UPDATE

```python

    def atualiza_review(self, review_id, novo_review):

        review_id = uuid.UUID(review_id)

        try:
            rows = self.__session.execute_async(
                """
                UPDATE reviews_db.reviews
                SET comentario = %s,
                gostou = %s,
                atualizado_em = %s
                WHERE id = %s
                """,
                (novo_review.comentario, novo_review.gostou, unix_time_millis(datetime.datetime.now()), review_id),
            )

            response = to_dict_list(rows)

            return {"code": 200, "response": "atualizado", "count": len(response), "data": response }
        except Exception as e:
            return {"code": 500, "response": "erro"}
```

DELETE

```python

    def remove_review(self, review_id):

        review_id = uuid.UUID(review_id)

        try:
            rows = self.__session.execute_async(
                """
                DELETE FROM reviews_db.reviews
                WHERE id=%s
                """,
                [review_id],
            )

            response = to_dict_list(rows)

            return {"code": 200, "response": "removido", "count": len(response), "data": response }
        except Exception as e:
            return {"code": 500, "response": "erro"}
```

EXTRA: GET REVIEWS POR FILME

```
145
146     def busca_reviews_filme(self, filme_id):
147
148         try:
149             rows = self.__session.execute_async(
150                 """
151                 SELECT * FROM reviews_db.reviews WHERE filme_id=%s ALLOW FILTERING
152                 """,
153                 [filme_id]
154             )
155
156             return {"code": 200, "response": "ok", "data": to_dict_list(rows)}
157         except Exception as e:
158             return {"code": 500, "response": "erro "+str(e)}
```

VI. VANTAGENS E DESVANTAGENS

VANTAGENS

- Alta **disponibilidade**;
- **Performance**;
- **Tolerante a falhas**;
- **Escalabilidade linear**: se o banco atende 100K de requisições, para atender 200K basta dobrar a infraestrutura;
- Altamente **distribuído**.

DESVANTAGENS

- 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.

V. APLICAÇÕES NOTÁVEIS

APLICAÇÕES NOTÁVEIS

- O **WebEx** da Cisco usa o Cassandra para armazenar feeds e atividades do usuário quase em tempo real;

- O **Discord** mudou para Cassandra para armazenar bilhões de mensagens do MongoDB em novembro de 2015;

- O **Globo.com** usa o Cassandra como um banco de dados de back-end para seus serviços de streaming;

APLICAÇÕES NOTÁVEIS

- A **Netflix** usa Cassandra como seu banco de dados de back-end para seus serviços de streaming;

- O **Uber** usa o Cassandra para armazenar cerca de 10.000 recursos em seu Feature Store atualizado diariamente em toda a empresa para acesso de baixa latência durante previsões de modelos ao vivo;

VI. REFERÊNCIAS

REFERÊNCIAS

- https://www.devmedia.com.br/introducao-ao-cassandra/38377
- https://www.devmedia.com.br/introducao-ao-banco-de-dados-nosql-cassandra/30533
- https://medium.com/nstech/apache-cassandra-8250e9f30942
- http://db4beginners.com/blog/cassandra/
- https://pt.wikipedia.org/wiki/Apache_Cassandra
- https://flask.palletsprojects.com/en/1.1.x/
- https://cassandra.apache.org/doc/latest/
- https://docs.python.org/3/
- https://docs.docker.com/
- https://developer.ibm.com/br/technologies/databases/tutorials/ba-set-up-apache-cassandra-architecture/
- https://imasters.com.br/banco-de-dados/bancos-de-dados-nosql-uma-visao-geral
- https://micreiros.com/tipos-de-bancos-de-dados-nosql/
- http://db4beginners.com/blog/escolha-o-tipo-bd/

DÚVIDAS?

OBRIGADA!

Jayne Morais

jayne@copin.ufcg.edu.br

"Feliz daquele que consegue enxergar na alegria do outro como resultado das suas ações, do seu investimento, da sua dedicação e do seu empenho."

Clóvis de Barros Filho

Universidade Federal de Campina Grande

COMPUTAÇÃO UFCG

Cassandra Database

No-SQL

Jayne de Morais Silva
Programa de Pós-Graduação em Ciência da Computação (Mestrado)
Disciplina: **Banco de Dados -** Prof. **Dr. Carlos Eduardo**
Novembro, 2020