Cassandra

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande

Agenda

Introdução

Cassandra não é:

- não é um banco de dados relacional
- não é um banco para *analytics*
- não há *joins*, não tem *fk's*
- não tem transações (ainda....)
- é difícil de gerenciar
- para melhores resultados, desaprenda os bancos relacionais

Introdução

- **Distribuído**
  - Cada nó no *cluster* tem a mesma função.
  - Não há um ponto único de falha.
  - Os dados são distribuídos pelo *cluster* (para que cada nó contenha dados diferentes), mas não há mestre, pois cada nó pode atender a qualquer solicitação.
- **Escalibilidade (de performance e armazenamento)**
  - Projetado para ter taxa de transferência de leitura e gravação, aumenta linearmente à medida que novas máquinas são adicionadas, com o objetivo de não haver tempo de inatividade ou interrupção nos aplicativos.

Introdução

- **Tolerante a falhas**
  - Os dados são replicados automaticamente para vários nós para tolerância a falhas.
  - A replicação em vários data centers é suportada. Nós com falha podem ser substituídos sem tempo de inatividade.
- **Consistência ajustável**
  - A disponibilidade e a tolerância da partição são, geralmente, consideradas mais importantes que a consistência no Cassandra
- **Linguagem de consulta**
  - Cassandra introduziu a Cassandra Query Language (CQL).
  - O CQL é uma interface simples para acessar o Cassandra, como uma alternativa à tradicional SQL (*Structured Query Language*).

Introdução

Quem usa?

- Netflix
- Facebook
- Spotify
- Instagram
- Nasa

Agenda

Aplicações

Catálogo de Produtos:

- Milhões de produtos e variações, características com seus preços e estoques (que devem ser retornados rapidamente)

Carrinho de Compras

- Alta confiabilidade no registro das compras Velocidade no fechamento da venda Alto volume de campanhas promocionais

Recomendações

- Gera disparos comportamentais/de dados para alimentar modelos de ML (Machine Learning) como também gerar recomendações/sugestões para futuras compras

Aplicações

Transações Financeiras

- Consulta ao extrato de transações do cliente via app ou website

Aplicações de Mídia

- Manter o serviço Global e disponível

Agenda

Modelo de Dados

- Cassandra é armazena os dados em colunas e, como tal, torna-se um híbrido entre um valor-chave e um sistema de gerenciamento de banco de dados tabular.
- Seu modelo de dados é um armazenamento de linha particionado com consistência ajustável.
  - Linhas são organizadas em tabelas; o primeiro componente da chave primária de uma tabela é a chave de partição;
  - Dentro de uma partição, as linhas são agrupadas pelas colunas restantes da chave.
  - Outras colunas podem ser indexadas separadamente da chave primária.

Modelo de Dados

- Cassandra não pode fazer junções ou subconsultas.
- Em vez disso, Cassandra enfatiza a **desnormalização** por meio de recursos como coleções.
- Cada linha é identificada exclusivamente por uma chave de linha.
- Ao contrário de uma tabela em um SGBD relacional, linhas diferentes na mesma família de colunas não precisam compartilhar o mesmo conjunto de colunas, e uma coluna pode ser adicionada a uma ou várias linhas a qualquer momento.

Agenda

Cassandra *Query Language* (CQL)

Criar Banco:

```
CREATE KEYSPACE IF NOT EXISTS exemplo
    WITH REPLICATION = {
        'class': 'SimpleStrategy', 'replication_factor': 1
    };
```

Visualizar *Keyspaces*:

```
DESC KEYSPACES;
```

Selecionar um *Keyspace*:

```
USE exemplo;
```

Listar Tabelas:

```
DESC TABLES;
```

Tipos de Dados

- **ascii**
- **bigint**
- **blob**
- **boolean**
- **counter**: tipo específico para contagem.
  - Devido as características do Cassandra, existe a necessidade de **inteiros contadores** específicos
  - Já que o banco (Cassandra) desconhece valor anterior (original)
  - Desta forma, consultas como *update* my_video set views = views+1
    - não funcionariam corretamente no caso que *views* fosse somente uma coluna inteira

Tipos de Dados

- **date**
- **decimal**
- **double**
- **duration**
- **float**
- **inet**
- **int**
- **smallint**
- **text**
- **time**
- **timestamp**
- **timeuuid**

Tipos de Dados

- **tinyint**
- **uuid**
- **varchar**
- **varint**
- **static**: valor compartilhado para todos os registros da partição.

Exemplo

```
CREATE TABLE cart (
    cartId uuid,
    cartTotal number STATIC,
    productId uuid,
    price number,
    qty number,
    total number,
    primary key (cartId, productId)
)
```

Tipos Avançados

- **Collection Types:**

```
set<int> (conjunto com elementos não-repetidos)
list<text>
map<int, text>
```

- ***Tuples types***:

```
    tuple <uuid, text, decimal>
```

Tipos Avançados

- **Tipos definidos pelo Usuário**
  - *User-defined Types* (UDTs):

```
CREATE TYPE address (
    street text,
    city text,
    state text
);
```

- ***Custom Types***:
  - Sua própria implementação com java (não recomendado)

Exemplos

Criar Tabela

```
CREATE TABLE "my_keyspace"."contacts" (
    contact_id uuid,
    first_name text,
    last_name text,
    phone_number text,
    primary key (contact_id)
);
```

Exemplos

Criar Tabela

```
CREATE TABLE exemplo.pessoa (
    cpf text primary key,
    nome text,
    nascimento date,
    gostos list<text>
);
```

Exemplos

INSERT

```
INSERT INTO pessoa (cpf, nome, nascimento)
    VALUES ('111.111.111-11', 'João', '1990-01-01');
INSERT INTO pessoa (cpf, nome, nascimento)
    VALUES ('222.222.222-22', 'Maria', '2000-01-01');
```

SELECT

```
SELECT * FROM pessoa where cpf = '111.111.111-11';
```

Dá erro buscar registros por outras colunas que não são a PK:

```
SELECT * FROM pessoa where nome = 'João';
```

Agenda

Instalação

Docker

```
-- 1 vez
sudo docker pull cassandra

sudo docker run --name cassandra -p \
127.0.0.1:9042:9042 -p 127.0.0.1:9160:9160 -d cassandra

sudo docker exec -it cassandra cqlsh

-- demais vezes
sudo docker start cassandra
sudo docker exec -it cassandra cqlsh
```

Agenda

Instalação

Maven

```
<dependencies>
        <dependency>
        <groupId>com.datastax.cassandra</groupId>
        <artifactId>cassandra-driver-core</artifactId>
        <version>3.11.0</version>
    </dependency>
    <dependency>
    <groupId>com.datastax.cassandra</groupId>
        <artifactId>cassandra-driver-mapping</artifactId>
        <version>3.11.0</version>
    </dependency>
</dependencies>
```

Como usar?

Observações:

- Classes e Atributos devem respeitar a convenção de nomes do Java;
- Atributos devem ter *getters* e *setters*
- **Quando usar *LocalDate* é preciso usar a *LocalDate* do pacote**

Como usar?

```
CREATE TABLE exemplo.pessoa (cpf text primary key, nome text,
nascimento date, gostos list<text>);
```

```
import com.datastax.driver.core.LocalDate;
@Table (keyspace="exemplo", name="pessoa")
public class Pessoa {
    @PartitionKey
    private String cpf;
    private String nome;
    private LocalDate nascimento;
    private List<String> gostos;
    public Pessoa(){}
    @PartitionKey
    public String getCpf(){
        return this.cpf;
    }
}
```

Como usar?

Conectar (*cluster* e *session*):

```
Cluster cluster = null;
try {
    cluster = Cluster.builder().addContactPoint("127.0.0.1")
                               .build();
    Session session = cluster.connect("exemplo");
} finally {
    if (cluster != null) cluster.close();
}
```

Como usar?

Inserir:

```
Pessoa p = null;
p = new Pessoa("333.333.333-33", "Fulano",
    LocalDate.fromYearMonthDay(1987,01,20),
    List.of("cassandra"));

try (Cluster cluster = Cluster.builder()
        .addContactPoint("localhost")
        .build()) {

    Session session = cluster.connect("exemplo");
    Mapper<Pessoa> pessoaMapper = new MappingManager(session)
                .mapper(Pessoa.class);

    pessoaMapper.save(p);
}
```

Como usar?

Selecionar:

```
try (Cluster cluster = Cluster.builder()
        .addContactPoint("localhost")
        .build()) {

    Session session = cluster.connect("exemplo");

    ResultSet results = session.execute ("select * from pessoa");

    Mapper<Pessoa> pessoaMapper = new MappingManager(session)
                .mapper(Pessoa.class);

    Result<Pessoa> pessoas = pessoaMapper.map(results);
    return pessoas;
}
```

Como usar?

Deletar:

```
try (Cluster cluster = Cluster.builder()
    .addContactPoint("localhost").build()) {

    Session session = cluster.connect("exemplo");

    Mapper<Pessoa> pessoaMapper = new MappingManager(session)
            .mapper(Pessoa.class);

    pessoaMapper.delete("111.111.111-11");
}
```

Como usar?

Atualizar:

```
try (Cluster cluster = Cluster.builder()
    .addContactPoint("localhost").build()) {

    Session session = cluster.connect("exemplo");

    PreparedStatement prepared =
    session.prepare("UPDATE pessoa SET nome = ? WHERE cpf = ?");

    BoundStatement b = prepared.bind(novoNome, cpf);

    session.execute(b);
}
```

Cassandra

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande