FATEC ANTONIO RUSSO

CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS – AMS – PTECH

ANÁLISE DE BANCOS DE DADOS NÃO RELACIONAIS

São Caetano do Sul

2024

FATEC ANTONIO RUSSO

Bianca de Freitas Soares

Bruno Leonardo Martinho

Giovanni de Pita Cicero

Jorge Antonio Terence Novaes de Santana

Ruan Defavari Godoi

Sarah Alves De Melo

FISHNET

São Caetano do Sul

2024

JanusGraph (antigo Titan)

Significado

JanusGraph é um banco de dados distribuído desenvolvido para lidar com quantidades massivas de informações organizadas em formato de grafos. Esse tipo de banco de dados é muito valioso em situações em que as conexões entre os dados têm tanta relevância quanto os dados em si, como em redes sociais, sugestões de conteúdo e análise de redes.

Origem

O Titan foi desenvolvido pela Aurelius e foi lançado pela primeira vez em 2012. Ele foi pensado para superar as limitações dos bancos de dados relacionais e NoSQL convencionais. A empresa responsável pelo banco de dados Apache Cassandra, a DataStax, comprou Aurelius em 2015. O JanusGraph foi uma continuação do desenvolvimento de um banco de dados de grafos distribuído após o fim do projeto Titan. Atualmente, o JanusGraph é mantido por desenvolvedores da Linux Foundation.

Características

Distribuição Horizontal: Como pode ser executado em clusters distribuídos, o Titan é ideal para aplicativos de larga escala. Armazenamento Backend Flexível: Vários sistemas de armazenamento back-end podem ser configurados no Titan, incluindo:

- Cassandra é projetado para escalabilidade e armazenamento de dados distribuídos.
- O HBase é uma ferramenta para armazenamento de dados baseado em colunas.
- BerkeleyDB: destinado ao armazenamento local de discos.

Integração com ElasticSearch: A integração com o motor de busca ElasticSearch permite consultas e indexação de dados, o que facilita a busca e análise de grandes volumes de dados.

Consistência Configurável: O Titan pode fornecer alta consistência (com Cassandra, por exemplo) ou alta disponibilidade (com HBase, por exemplo), permitindo configurações para manter a integridade dos dados. Isso depende do back-end escolhido.

Suporte ao TinkerPop: A consulta e a manipulação de grafos são facilitadas pela API de grafos Apache TinkerPop para interagir com o banco de dados. Transações: O Titan permite conjuntos atômicos de operações em um grafo, ou seja, todas as operações são aplicadas ou nenhuma.

Vantagens

Escalabilidade: adicionar mais nós ao cluster permite a escala horizontal, o que permite lidar com grandes quantidades de dados espalhados em vários servidores.

Alta conectividade: Bancos de dados de grafos, como o Titan, são perfeitos para administrar redes de dados complexas com muitas interconexões entre entidades, como redes sociais e sistemas de recomendação.

Desempenho em consultas relacionais: Consultas que envolvem várias conexões de dados, como "amigos de amigos" nas redes sociais, são muito mais rápidas do que consultas em bancos de dados relacionais.

Suporte a Transações e ACID: garante atomicidade, consistência, isolamento e durabilidade das operações, facilitando o desenvolvimento de aplicativos transacionais.

Flexibilidade com backends de armazenamento: O Titan pode ser configurado para usar vários backends de armazenamento, o que o permite se adaptar a várias necessidades de armazenamento e consistência.

Navegando o Titan usando o Gremlin

Navegar um banco de dados de grafos como o JanusGraph (ou seu predecessor, Titan) envolve a utilização da linguagem de consulta *Gremlin*. Gremlin é baseada em um estilo de travessia de grafos (traversal). Através do Gremlin, você pode consultar, inserir, atualizar e excluir vértices (nós) e arestas (relações) no grafo.

Aqui está uma visão de como navegar e consultar o Titan usando Gremlin.

Exemplo de Estrutura de Dados

Vértices (V): Filmes e Atores.

Arestas (E): Representam a relação entre atores e filmes, como ACTED_IN.

Comandos Básicos para Navegação

Aqui estão os comandos básicos para navegar pelos vértices e arestas.

Encontrar Vértices: `g.V()`

Filtrar por Propriedade: `g.V().hasLabel('Movie').has('title', 'Unbreakable')`

Encontrar Arestas: `g.E().hasLabel('ACTED_IN')`

Navegação Entre Relacionamentos

Navegar de um vértice para outros por relacionamento: `g.V().has('Actor', 'name', 'Bruce Willis').out('ACTED_IN').values('title')`

O caminho inverso: `g.V().has('Movie', 'title', ‘Unbreakable').in( 'ACTED_IN').values('name')`

Implementação práticas

Implementação prática de um catálogo de filmes usando Titan (ou seu sucessor, JanusGraph) e a linguagem de consulta Gremlin. Vamos ver como criar um pequeno sistema de banco de dados de grafos para armazenar filmes e atores e suas relações (como quem atuou em qual filme).

Etapas da Implementação

Instalação do Titan (ou JanusGraph): instale o servidor pelo site oficial https://janusgraph.org. Certifique-se de que o Gremlin Server esteja ativo e rodando no porta padrão 8182.

Conclusão

O Titan ofereceu uma solução inovadora para lidar com grandes volumes de dados distribuídos e escaláveis. No entanto, sua descontinuação e substituição pelo JanusGraph podem tornar o uso do Titan difícil. A escalabilidade e a eficiência das consultas baseadas em relações são suas grandes vantagens, mas suas principais desvantagens são a complexidade e a falta de suporte atual.

MongoDB

O que é?

"É um sistema de gerenciamento de banco de dados não relacional, baseado em software livre, que utiliza documentos flexíveis em vez de tabelas e linhas para processar e armazenar várias formas de dados", *IBM*

As unidades básicas do MongoDB são os documentos e coleções de documentos que imitam os dados e tabelas. São formatados como **Binary JSON** e podem armazenar vários tipos diferentes de dados. Por ser muito dinâmico e fácil de usar, ele acaba sendo perfeito para grandes quantidades de informações e a análise das mesmas.

Características

Vejamos algumas características do MongoDB que diferem ele de outros:

- **Modelo de Dados Flexível:** documentos armazenados em documentos BSON onde podem ter estruturas complexas sem um esquema fixo como em modelos relacionais.
- **Escalabilidade Horizontal:** altamente escalável, o MongoDB pode ser distribuído em vários servidores, aumentando sua capacidade.
- **Alta Performance:** usando indexação para acelerar consultas e tendo uma arquitetura interna eficiente para lidar com uma grande massa de dados.
- **Consultas Avançadas:** o próprio MongoDB oferece uma linguagem flexível para consultas, podendo filtrar, projetar, ordenar, agregar, extrair, manipular dados, ver mais.

Ferramentas

O MongoDB oferece múltiplas ferramentas de comando até gráficos e bibliotecas de terceiros. Essas ferramentas são extremamente úteis caso você queira uma administração completa do seu banco de dados MongoDB. Vejamos:

MongoDB Compass: uma GUI visual que interage com os bancos MongoDB e permite que usuário explore, visualize e análise esses dados.

MongoDB Shell: interface de linha de comando para interagir com o banco. Permite: consultas, scripts e comandos administrativos.

MongoDB Atlas: uma plataforma que permite implantar, gerenciar e escalar clusters na nuvem de maneira simples e eficiente.

MongoDB Ops Manager: uma plataforma de gerenciamento de monitoramento, permitindo backup, automação e segurança.

MongoDB Atlas Data Lake: ferramenta que permite consultar e analisar dados em nuvem, como Amazon S2 ou Azure Data Lake Storage, usando o MongoDB Query Language (MQL).

MongoDB BI Connector: uma ferramenta que faz ponto entre o MongoDB e ferramentas de business inteliigence BI como Power Bi ou Tableau.

Drivers MongoDB: bibliotecas de cliente disponíveis para várias linguagens de programação que permitem que aplicativos se conectem e interajam com o banco de dados MongoDB.

Instalação

Podemos instalar o MongoDB de várias maneiras, sendo local, executáveis, por Docker ou pelo MongoDB Atlas.

Vejamos como instalamos o MongoDB usando os executáveis no Windows:

1. Baixe o MongoDB ZIP: acessando esse link e clique em *"download"*.
2. Extrair o ZIP: após o download do ZIP, extraia o conteúdo para uma pasta de sua escola. Um exemplo: C:\mongodb. Opcional: criar uma pasta para ser um lugar para armazenar os dados do MongoDB. Exemplo: C:\mongodb\data.
3. Iniciar o MongoDB: abra o prompt de comando e navegue até o diretório que criamos no passo anterior onde tudo foi extraído. Exemplo:

```
cd C:\mongodb\bin

mongodb.exe –dbpath C:\mongodb\data

cd C:\mongodb\bin

mongo.exe
```

Comandos

Alguns comandos no MongoDB são essenciais para começar a criar e mexer seu próprio banco de dados. Vamos começar criando um banco fictício da sua família.

1. Vamos criar uma coleção nova, para isso usamos db.createColletion(nome_da_colecao). Exemplo:

   ```
   db.createCollection("family")
   ```

2. Precisamos adicionar pessoas à família. Podemos usar o código db.collection.insert(document) para isso. Exemplo:

   ```
   db.family.insert({nome: "Maria", idade: 50, familyrole: "grandma"})
   db.family.insert({nome: "Joseph", idade: 80, familyrole: "grandpa"})
   ```

3. Temos duas pessoas na família agora: a Maria, de 78 anos que é a avó e Joseph, de 80 anos que é o avô. Podemos filtrar nossa família para achar apenas o avô. Fazemos isso com o código db.collection.find(filter). Exemplo:

```
db.family.find({familyrole: "grandpa"})
```

4. Infelizmente, houve um erro na hora de inserir a avó: ela, na verdade, é a mãe da família. Podemos resolver isso atualizando o documento usando db.colletion.update(filter, update). Exemplo:

```
db.family.update(
   {nome: "Maria"},
   {$set: {familyrole: "mother"}},
)
```

5. Quando querermos remover algum documento, podemos usar db.collection.remove(filter). Vamos remover o avô, que morreu de causas naturais:

```
db.family.remove({nome: "Joseph"})
```

6. Listar coleções do banco: show collections.
7. Remover coleção: db.collection.drop).

Conclusão

O MongoDB é uma solução flexível, acessível e único. Com suas ferramentas e controles, você pode ter um monitoramento e uma administração completa de forma fácil, sendo melhor para uma grande quantidade de dados.

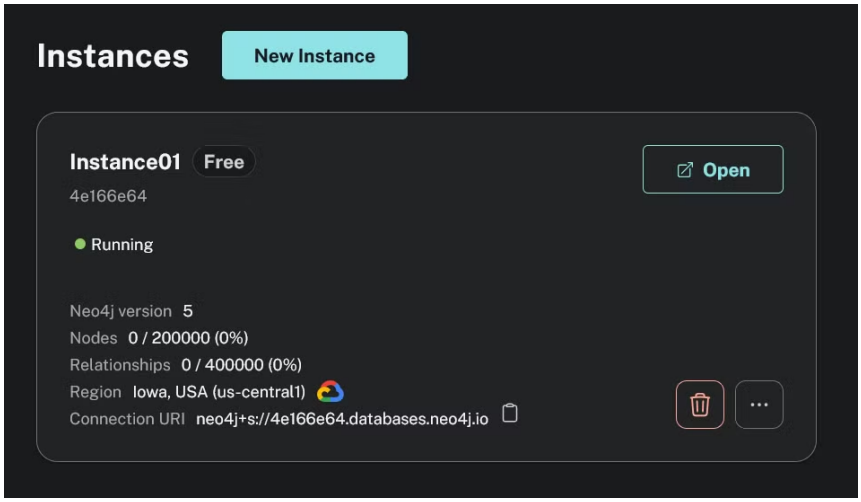
Documentações úteis:

- Documentações gerais
- Guia para iniciantes
- Operações de consulta e projeção
- Operações de atualização
- Estágios de aggregation
- Operações de agregação

Neo4j

Um distribuidor de bancos de dados baseado em grafos. O Neo4j original pode ser implantado em qualquer máquina, processo que é facilitado com a imagem de Docker oficial, permitindo que uma instancia seja iniciada através de um container virtualizado.

A AuraBD é oferecida como SaaS (*Software as a Service*), com uma instância manejada automaticamente. No plano profissional é possível escolher um provedor cloud para a implantação. Na versão gratuita, a instância do banco de dados é alocada na CGP (*Google Cloud Platform*) em um servidor nos Estados Unidos.



O que é uma base de dados baseada em grafos?

Um grafo é um conjunto de ***vértices*** e ***arestas***. Existem vários sinônimos para esses termos: vértices são chamados de nós, pontos e conexões e arestas também são conhecidas como conexões, linhas etc.

Essas entidades-base podem ser interpretadas de diversas maneiras para satisfazer os requisitos de um sistema. Em alguns casos, as arestas têm o papel principal, em outros são os vértices.

Navegando a base de dados

A AuraDB possui o equivalente a um SGBD: o Neo4j Workspace. Ele possui uma interface web, onde é possível criar vértices e arestas e editar o esquema de dados através de uma interface visual. Infelizmente, não conseguimos nos conectar ao serviço, que se mostra extremamente lento.

É possível se conectar à base de métodos de baixo nível:

- Console virtual e CLI
- Drivers para linguagens de programação (Python, Go, Java, JS e C#)
- API REST

- API GraphQL

A linguagem para interagir com a base de dados se chama Cypher.

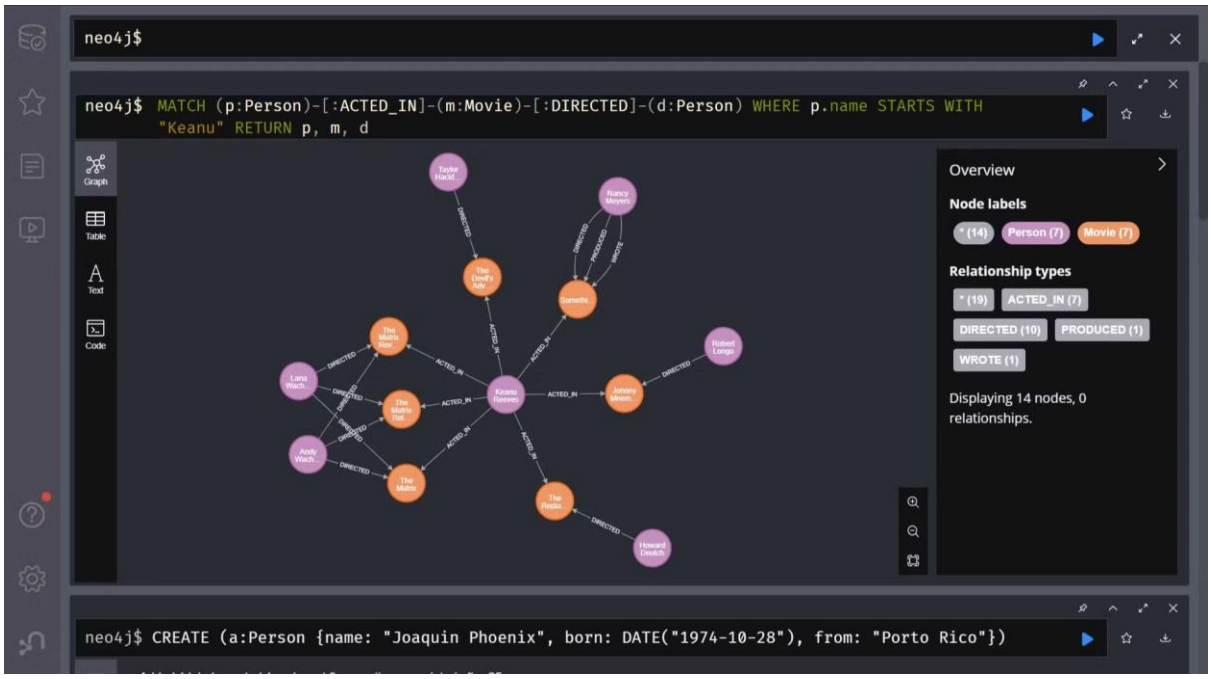
Uma simples seleção de dados se dá pelo operador MATCH

```
MATCH (movie:Movie)
WHERE movie.rating > 7
RETURN movie.title
```

No entanto, relacionamentos são armazenados fisicamente na base dados com as arestas, diferentemente dos JOINs em SQL, que criam uma tabela virtual, encaixando as chaves primária e estrangeira de duas tabelas.

```
MATCH (actor:Actor)-[:ACTED_IN]->(movie:Movie {title: 'Unbreakable'})
RETURN actor.name
```

A Neo4j oferece um SGBD onde é possível interagir com bancos locais e remotos. Nele é possível ter uma visualização dos queries.



Query para retornar os filmes em que Keanu Reeves atuou e seus diretores.

Implementações práticas

Vamos implementar uma conexão simples com a BD local. Como linguagem, usaremos Python para conectar com a BD através do driver oficial.

```
from neo4j import GraphDatabase
```

```
if __name__ == "__main__":
    driver = GraphDatabase.driver("bolt://localhost:7687", auth=("<db_user>",
"<db_password>"))
    with driver.session() as session:
        session.execute_write(
            "CREATE (a:Person {name: $name, from: $country, role: $role}) RETURN p",
            name="Floriano",
            country="Brazil",
            role="Teacher",
        ).single()[0]
        res = session.execute_read(
            "MATCH (a:Person) WHERE role = $role RETURN a LIMIT 20",
            role="student",
        )
        print(res)
    driver.close()
```