Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

MATHEUS FILIPE SIEIRO VARGAS

MONOGRAFIA DE PROJETO ORIENTADO EM COMPUTAÇÃO I

INTEGRAÇÃO DE BANCOS DE DADOS ORIENTADOS A GRAFOS AO AMBIENTE JUPYTER NOTEBOOK

Belo Horizonte
2019/2º semestre

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

INTEGRAÇÃO DE BANCOS DE DADOS ORIENTADOS A GRAFOS AO AMBIENTE JUPYTER NOTEBOOK

por

Matheus Filipe Sieiro Vargas

Monografia de Projeto Orientado em Computação I
Apresentado como requisito da disciplina de Projeto Orientado em Computação I do Curso de Bacharelado em Ciência da Computação da UFMG

Prof. Dr. Mirella Moura Moro
Orientador(a)

Belo Horizonte
2019/2º semestre

RESUMO

O projeto Jupyter nasceu como extensão ao IPython com intuito de prover ambiente de computação interativa para exposição de dados e computação científica. A ferramenta tem sido amplamente utilizada devido a facilidade de integração com bibliotecas para as mais específicas finalidades, como por exemplo big data, machine learning e visualização de dados. Por possuir código aberto, apresenta forte relacionamento com a comunidade e facilidade de integração.

Por outro lado, objetivando extrair melhor desempenho em persistência e manipulação de dados, os bancos de dados orientados a grafos surgiram em alternativa para os modelos relacionais tradicionais, quando o tipo de relacionamento entre as entidades é fundamental para o escopo em questão.

O objetivo do projeto é apresentar as formas de relacionar as duas ferramentas a fim de extrair as vantagens do uso de cada elemento envolvido. Para tanto, apresenta um aprofundamento no estudo de algumas implementações disponíveis no mercado e expõe para cada uma delas as formas de comunicação com o ambiente em questão.

Palavras-chave:.Projeto Jupyter, Jupyter Notebook, Bancos de Dados, Bancos de Dados Orientados a Grafos, Grafos

ABSTRACT

The Jupyter project was born as an extension for IPython in order to provide interactive computing environment for data exposure and scientific computing. The tool has been widely used due to ease of integration with libraries for the most specific purposes such as big data, machine learning and data visualization. Because it has open source, it has a strong relationship with the community and ease integration.

On the other hand, aiming to extract better performance in data persistence and manipulation, graph-oriented databases have emerged as an alternative to traditional relational models, when the type of relationship between entities is fundamental to the scope in question.

The goal of the project is to present ways of relating the two tools in order to extract the advantages of using each element involved. To do so, it presents a deeper study of some implementations available in the market and exposes for each one of them the ways of communication with the environment in question.

Keywords*:*
Jupyter Project, Jupyter Notebook, Databases, Graph Driven Databases, Graphs

LISTA DE FIGURAS

LISTA DE SIGLAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Desenvolvido para dar suporte à computação científica e a ciência de dados, o projeto Jupyter se tornou uma ferramenta muito útil e amplamente utilizada. Por se tratar de um software de código aberto e com muitas facilidades, ganhou notoriedade pelas soluções desenvolvidas por parte da comunidade. Suas aplicações, nos dias atuais, superam os limites inicialmente planejados.

A persistência de dados por sua vez é uma prática que precede a computação: seja em pinturas na parede representando histórias, escrituras antigas ou em arquivos organizados em gavetas, invariavelmente a intenção de quem produz a informação é armazená-la.

Com a evolução dos recursos computacionais e o aumento do volume de dados, novas estratégias para organização e armazenamento surgiram como forma de otimizar, prover segurança e consistência.

A maneira mais generalista de armazenamento: a distribuição de informações de forma estruturada em tabelas, imperou e ainda representa grande parte dos tipos de disposição de dados ao redor do mundo. Parte deste sucesso consiste na robustez como contorna os obstáculos que fazem esta abordagem não ser ótima para todas as soluções, seja através do uso de sistemas de gestão de base de dados (SGDBs), modelos ou esquemas específicos.

Novas estratégias porém, surgiram a fim de suprir as necessidades de gerenciar volumes crescentes de informação e otimizar a organização destas para as mais diversas aplicações.

Para o escopo deste projeto, serão exploradas as vantagens de uma das abordagens alternativas ao método de tabelas, o banco de dados baseado em grafos, e mais especificamente sua integração com o ambiente de desenvolvimento para softwares de código aberto, o Jupyter.

.

Justificado pela complexidade em correlacionar dados no modelo tradicional, o banco de dados orientado a grafos também tem como forte incentivo a pesquisa científica, sendo aplicado nos mais diversos campos, de dados geográficos a bioinformática.

A abordagem herda dos bancos de dados não relacionais (NoSQL) o foco em velocidade na recuperação da informação, uma vez que ao indexar os vértices do grafo e utilizar as ferramentas corretas de chaveamento, a consulta pode ser feita de maneira ótima. A organização em vértices e arestas possibilita também uma melhor visualização na correlação entre as informações, motivo pelo qual tem sido amplamente utilizada pela comunidade científica.

Uma vez reconhecidas as vantagens deste tipo de banco de dados, adicionados aos benefícios de um ambiente colaborativo e de ampla gama de recursos, o objetivo deste projeto é fornecer embasamento teórico e posteriormente prático para utilização e integração do melhor dos dois cenários.

2 TRABALHOS RELACIONADOS

2.1 Bancos de dados em grafos

A abstração do modelo relacional para os mais diversos cenários ao longo do tempo, resultou em diversas implementações de soluções escaláveis horizontalmente, não relacionais, distribuídas e de código aberto, definindo assim o termo Not Only SQL [NOSQL 2010]. Dentre elas, estão: bancos de dados (BDs) de famílias de colunas, chave e valor, orientados a documentos, grafos e implementações de multimodelo. Cada uma dessas abordagens apresenta inúmeras justificativas para o emprego em determinadas situações.

Em especial, a abordagem em grafos trata a modelagem dos dados como um grafo dirigido. As operações efetuadas sobre esta abstração representam transformações no grafo propriamente dito e em suas estruturas fundamentais. É ainda uma alternativa natural a relação *muitos para muitos*, ou seja, quando a organização dos dados tem importância fundamental [DE DIANA and GEROSA 2010].

Considerando a estrutura de um grafo como G = (V, A), onde V representa os vértices e A os arcos, podemos aproximar uma situação cotidiana em que os vértices assumam o papel de entidades, enquanto os arcos representam a relação entre elas.
Em um exemplo simples, o grafo G, representa as relações (possui sede em) entre postos de combustível (entidade representada em vértice), com as unidades federativas (outra entidade representada em um vértice).

Figura 1 Exemplo de relação em um grafo

A forma como estes dados serão persistidos nos permite classificar a implementação do banco de dados em grafos de forma nativa ou não-nativa [PENTEADO et al. 2014].

2.1.1 Não-nativos

Apesar de apresentar os dados em forma de grafos, os modelos não-nativos armazenam a informação presente em cada estrutura de forma separada; tabular, chave e valor dentre outros [PENTEADO et al. 2014].

2.1.2 Nativos

Por outro lado, as abordagens nativas, além de apresentarem os elementos persistidos em formato de grafo, em geral, armazenam as informações utilizando a aproximação computacional do grafo como uma matriz de adjacência [PENTEADO et al. 2014].
A explicação da forma como apresentar um grafo como uma lista de adjacência é considerada fora do escopo deste projeto, e portanto não será abordada.

Os conceitos definidos nesta sessão, serão utilizados como fundamentos para as seções seguintes.

2.2 Projeto Jupyter

O Projeto Jupyter é um projeto de código aberto e sem fins lucrativos, com o objetivo de fornecer um ambiente para desenvolvimento científico [JUPYTER 2019]. Nasceu da evolução do IPython, interface com ferramentas para uso interativo da linguagem Python [IPython 2007] com intuito de suportar implementações em diversas linguagens.

O ambiente tem sido utilizado para exportar os resultados de pesquisas e como fundamentação para publicações científicas. Como ferramenta open-source, possui diversas extensões publicadas pela comunidade que permitem as mais amplas aplicações em inúmeras áreas de conhecimento [KLUYVER et al. 2016].

Baseado na premissa de permitir a computação interativa, o projeto fornece a capacidade de extração de informação e conhecimento dos códigos gerados por usuários leigos em computação, ou que dominam apenas uma parte restrita do código apresentado.
As informações são apresentadas em *notebooks* contendo células que podem ser executadas em separado e podem ser independentes entre si [PEREZ and GRANGER 2015].

As inúmeras capacidades do projeto podem ser exploradas através dos diversos estudos publicados, além da utilização da ferramenta propriamente dita. Para o escopo deste projeto, apenas a introdução do conceito do projeto é suficiente para prosseguir.

3 DESENVOLVIMENTO DO TRABALHO

Por se tratar de ferramentas utilizadas no cotidiano da computação, inicialmente a ideia do projeto se baseava em:

1. Identificar as soluções de bancos de dados em grafos existentes no mercado.
2. Avaliar o desempenho de cada solução para um número de dados variável, utilizando como base, dados públicos da receita federal.
3. Avaliar a estratégia e interface utilizada para persistência e recuperação de dados.
4. Identificar possíveis integrações com as demais ferramentas disponíveis para o ambiente.

Ao longo do período de consolidação do conhecimento adquirido a partir de pesquisas e manuseio das ferramentas, observou-se que a metodologia proposta inicialmente, teria como resultados uma comparação entre as versões de mercado, objetivo este, fora da proposta deste projeto.

Observou-se porém que a etapa de reconhecimento das estratégias de mercado para implementação de banco de dados em grafos, gerou base de conhecimento técnico de grande relevância para o estudo da integração entre o ambiente Jupyter notebook e cada uma das soluções.

3.1 Metodologia

Uma vez efetuados os ajustes que permitem atingir embasamento teórico para a integração, a seguinte metodologia foi utilizada:

1. Identificar as soluções de BDs existentes no mercado com possibilidade de integração com o ambiente Jupyter Notebook.
2. Investigar os elementos estruturais envolvidos na implementação desta solução que possibilitam a integração.
3. Avaliar outros recursos relevantes de cada abordagem.

3.1.1 Soluções em Banco de dados orientado a grafos

Basta pesquisar por soluções de BDs em grafos para encontrar inúmeras soluções de mercado. Dentre os resultados, os filtros devem ser aplicados às implementações que possuem interfaces de programação de aplicativos (APIs) para Python ou mesmo transferência representacional de estado (REST). Estas foram as formas de integração identificadas nos estudos realizados na primeira etapa da metodologia empregada anteriormente.

3.1.2 Sistemas analisados

Aplicando os filtros definidos no item anterior, e com o objetivo de explorar os elementos estruturais envolvidos na implementação que possibilitam a integração com o Jupyter, os

seguintes sistemas foram escolhidos de forma arbitrária por exemplificar diferentes aspectos estruturais.

Sistema	API	Query Method
Neo4j [NEO4J 2019]	Várias linguagens, incluindo Python	Cypher, nativeJavaAPI, JRuby
Titan [TITAN 2019]	Java, Blueprints, Gremlin, Python, Clojure	Gremlin, SPARQL

Tabela 1 - Soluções de mercado para bancos de dados em grafos

3.1.2.1 Neo4j

Segundo o site de pesquisas sobre BDs, DB-Engines [DB-ENGINES 2019], dentre as implementações em grafos, o Neo4j é o mais popular.
O ranking leva em consideração o número de citações em vagas de emprego, redes sociais, uso em repositórios de código dentre outros.
Por ser a implementação mais utilizada, possui amplo suporte da comunidade além de inúmeras análises de desempenho e estrutura.

Para nosso escopo, analisaremos os elementos principais de sua implementação que permitem a comunicação com o ambiente pretendido.

Dentre as APIs disponíveis para utilização, o Neo4j fornece suporte nativo a linguagem Python, que por sua vez, pode ser uma porta de entrada para o Jupyter Notebook. Como mencionado anteriormente, os notebooks implementam a interface de computação interativa IPython e, portanto, pacotes podem ser instalados através dos gerenciadores de pacotes disponíveis para o mesmo.

Para promover a comunicação entre as ferramentas, basta então expor os drivers de comunicação do Neo4j para o notebook, através por exemplo, do gerenciador de pacotes python (PIP).

```
pip install neo4j
```

Figura 2 Instalação de drivers neo4j

Uma vez que os drivers estiverem devidamente adicionados aos pacotes no ambiente do notebook, basta importar as dependências e utilizar os objetos expostos pela API.
O servidor do banco de dados deve ser inicializado como um serviço a parte, prática comum em se tratando de outros tipos de bancos de dados.
Uma vez que o serviço do BD tenha sido inicializado corretamente e os drivers importados para o projeto, o programador possuirá acesso ao banco.

O método de consultas exposto pelos drivers nativos do Neo4j implementa a linguagem Cypher Query Language:

Descrição	Query
Atribuição de uma entidade do tipo posto de combustível com nome e preço	p:Posto de combustível {name : 'Posto Ale', preco : 4,55}
Descrição da relação entre duas entidades	(a:Posto de combustível)-[:SEDE]-(b:Unidade Federativa)
Alteração de valores	MATCH (p:Posto) WHERE p.name = "Posto Ale" SET p.preco = 4,65

Tabela 2 - Exemplos de consultas em Cypher Query Language

A visualização dos dados do grafo nativa é provida pela interface do cliente do Neo4j. Para visualizar parte do grafo no notebook é necessário utilizar bibliotecas de terceiros como por exemplo a D3.js.

Assim como definido anteriormente, uma vez que esta abordagem implementa uma matriz de adjacência como persistência dos dados, se trata de uma implementação nativa.

3.1.2.2 TitanGraph

Por sua vez, o TitanGraph é uma implementação que merece destaque por oferecer opções na etapa de persistência e na estratégia utilizada para consultas.

A interface com o Python e, consequentemente com o Jupyter Notebook, pode ser implementada através do uso das linguagens de consulta: Gremlin Query Language, ou SPARQL.

Gremlin Query Language é subproduto da solução da Apache [APACHE 2019] para grafos, enquanto SPARQL consiste em uma linguagem padronizada para consulta de grafos RDF (Resource Description Framework).

A importação dos drivers de ambas as implementações, assim como no caso anterior, pode ser feita através do gerenciador de pacotes PIP.

Novamente uma vez que os drivers estiverem devidamente adicionados aos pacotes no ambiente do notebook, basta importar as dependências e utilizar os objetos expostos pela API.

A seguir, as tabelas exemplificam algumas consultas nas linguagens implementadas pela solução:

Descrição	Query
Retorna o menor valor do combustível dentre os postos	g.V().hasLabel('posto').values('preco').min()
Emite uma relação do número de postos de combustível por Unidade Federativa	g.V().hasLabel('uf').groupCount()
Retorna a média do preço do combustível no posto Ale	g.V().has('name','Posto Ale').values('preco').mean()

Tabela 3 - Exemplos de consultas em Gremlin Query Language

Descrição	Query
Retorna a listagem de nomes de postos de gasolina	PREFIX dc: <url> SELECT ?name WHERE { <url> dc:name ?name }

Retorna os Postos Ale onde os preços são menores que 4,40	PREFIX ex: <url>
	
	SELECT ?name ?preco
	WHERE{
	?x ex:name ?name .
	?x ex:preco ?preco
	FILTER (?preco<4,40 || ?name =’Posto Ale’)
	}

Tabela 4 - Exemplos de consultas em SPARQL Query Language

Para a camada de persistência dos dados, o TitanGraph, fornece ao usuário as opções pelos BDs: Cassandra, HBase, MapR M7 Tables, BDB, Persistit e Hazelcast. Cabe ao programador identificar as vantagens de cada opção.

4 CONCLUSÕES

Através dos resultados expostos pelas implementações descritas no item anterior, podemos verificar que, uma vez que o projeto Jupyter nasceu da evolução do IPython e este, por sua vez é uma exposição da linguagem python para um ambiente de computação interativa, as opções de integração se relacionam ao suporte à linguagem.

No quesito persistência de dados, verificamos a existência de diferentes formas de armazenamento (nativos e não-nativos), bem como diferentes soluções de backend para a camada de mais baixo nível.

A camada de interface entre os dados e o código pode ser abstraída por diferentes linguagens, nas seções anteriores foram abordadas algumas destas linguagens, cada uma com suas características e limitações.

As soluções de mercado consistem em implementações de uma pilha formada pela camada de persistência (podendo esta variar mesmo dentro de uma solução), pela camada de consultas de dados, e pela exposição dos dados.

A camada de exposição de dados pode ser implementada pela própria solução, como é o caso do Neo4j, ou pode não ser implementada, deixando a cargo do usuário a implementação de tal recurso.

Muitas vezes, apesar de organizar os dados em forma de grafo, o usuário não manifesta a necessidade de visualizar os dados como um todo, ou mesmo de pequenas partes, basta apenas recuperar a informação objetivo. Em uma companhia com um cenário de armazenamento de milhares de dados, por exemplo, não existe sentido em visualizar o grafo em sua totalidade, basta apenas, recuperar a informação de uma relação e seus nós.

Como possíveis estudos futuros, a visualização de dados de forma generalista pode ser explorada como forma de solução para os cenários em que este seja um desejo do usuário, e uma biblioteca pode ser proposta para demonstração gráfica dentro da ferramenta Jupyter notebook.

5 REFERÊNCIAS

DE DIANA, Mauricio; GEROSA, Marco Aurélio. Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0. In: IX Workshop de Teses e Dissertaç oes em Banco de Dados. 2010.

PENTEADO, Raqueline RM et al. Um estudo sobre bancos de dados em grafos nativos. X ERBD-Escola Regional de Banco de Dados, 2014.

ANGLES, R. and GUTIERREZ, C. (2008). Survey of Graph Database Models. ACM Computing Surveys, 40(1):1–39.

NOSQL (2010). NOSQL Databases. http://nosql-database.org/. Acesso em: 20 ago. 2019.

JUPYTER (2019). Projeto Jupyter. https://jupyter.org/ Acesso em: 13 set. 2019

IPython (2007) PEREZ, Fernando; GRANGER, Brian E. IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53. URL: https://ipython.org

KLUYVER, Thomas et al. Jupyter Notebooks-a publishing format for reproducible computational workflows. In: ELPUB. 2016. p. 87-90.

PEREZ, Fernando; GRANGER, Brian E. Project Jupyter: Computational narratives as the engine of collaborative data science. Retrieved September, v. 11, n. 207, p. 108, 2015.

NEO4J (2019). Neo4j Database. https://neo4j.com/ Acesso em: 19 set. 2019

TITAN (2019). TitanGraph Database. https://titan.thinkaurelius.com/ Acesso em: 19 set. 2019

DB-ENGINES (2019). DB-Engines Ranking. https://db-engines.com/en/ranking Acesso em: 20 out. 2019

APACHE (2019). https://tinkerpop.apache.org/gremlin.html Acesso em: 20 out. 2019