BRUNO DONIZETI DA SILVA

ESTUDO DE CASO EM BANCO DE DADOS DE GRAFOS:

SISTEMA DE RECOMENDAÇÃO DE IMÓVEIS SIMILARES

LAVRAS – MG

2019

BRUNO DONIZETI DA SILVA

ESTUDO DE CASO EM BANCO DE DADOS DE GRAFOS: SISTEMA DE RECOMENDAÇÃO DE IMÓVEIS SIMILARES

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do Curso de Sistemas de Informação para obtenção do título de Bacharel em Sistemas de Informação.

Prof. DSc. Ramon Gomes Costa

Orientador

LAVRAS – MG

2019

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Silva, Bruno Donizeti da
Estudo de caso em Banco de Dados de Grafos : Sistema de recomendação de imóveis similares / Bruno Donizeti da Silva. – Lavras : UFLA, 2019.
29 p. : il.

Monografia (graduação)–Universidade Federal de Lavras, 2019.
Orientador: Prof. DSc. Ramon Gomes Costa.
Bibliografia.

1. Banco de dados de grafos. 2. Neo4j. 3. Sistema de recomendação de similares. I. Costa, Ramon Gomes. II. Título.

BRUNO DONIZETI DA SILVA

ESTUDO DE CASO EM BANCO DE DADOS DE GRAFOS: SISTEMA DE RECOMENDAÇÃO DE IMÓVEIS SIMILARES
CASE STUDY IN GRAPH DATABASE: RECOMMENDATION SYSTEM OF SIMILAR HOUSES

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do Curso de Sistemas de Informação para obtenção do título de Bacharel em Sistemas de Informação.

APROVADA em 12 de Junho de 2019.

Prof. DSc. Ramon Gomes Costa	UFLA
Prof. DSc. Dilson Lucas Pereira	UFLA
Prof. DSc. Mayron César de Oliveira Moreira	UFLA

Prof. DSc. Ramon Gomes Costa
Orientador

Prof. Ramon Gomes Costa
DCC / UFLA

LAVRAS – MG
2019

Dedico esse trabalho à minha família que todas as formas possíveis me incetivaram a continuar, me manter firme e não desistir em momento algum.

AGRADECIMENTOS

Agradeço primeiramente a Deus que, com certeza, esteve sempre presente em todos os momentos.

Aos meus pais e minha irmã, que se dedicaram para que eu pudesse ter uma educação de qualidade e sempre me incentivaram a continuar.

À Carol, que esteve comigo durante toda graduação, me ajudando, sofrendo e passando por todas as coisas ao meu lado.

Ao professor Ramon Gomes Costa, que dedicou seu tempo para me auxiliar durante esse trabalho.

E a todas as outras pessoas que me auxiliaram de alguma maneira durante esses anos, meu muito obrigado!

RESUMO

Bancos de dados relacionais têm dominado o mercado desde os anos 80. O modelo de dado relacional é pouco flexível à modificações constantes no esquema. Em contra partida o modelo de dados orientado a grafo é muito flexível (schemaless), seu uso é aconselhável quando a interconectividade dos dados é tão ou mais importante que a organização fixa dos dados. Alguns bancos de dados NoSQL são flexíveis a ponto de não existir a necessidade de sempre ter que adequar os dados para tabelas, realizar a normalização ou criar relacionamentos. Dentre esses Sistemas Gerenciadores de Bancos de Dados podemos destacar o Neo4j, que é um SGBD orientado a grafos. Neste artigo será apresentado um estudo de caso usando o SGDB Neo4j para criação um sistema de recomendação de imóveis similares. O artigo mostra todos os passos desde a tradução de um modelo Relacional para o modelo em Grafos, passando pela migração dos dados e implementação do sistema de recomendação.

Palavras-chave: Banco de dados orientado a grafos. Neo4j. Sistema de recomendação. Cypher. NoSQL

ABSTRACT

Relational databases have been dominating the market since the '80s. The relational data model isn't flexible enough to the constant modifications in the schema. In contrast, the graph-oriented data model is very flexible (schemaless), its use is advisable when the data interconnectivity is at least as important as the fixed organization of the data. Some NoSQL databases are flexible to the point that there is no need to always adapt data to tables, perform normalization, or create relationships. Among these Database Management Systems, we can highlight Neo4j, which is a graph-oriented DBMS. In this paper, we will present a case study using SGDB Neo4j to create a similar houses recommendation system. The article shows all the steps from the translation of a Relational model to a Graph model, through data migration and implementation of the recommendation system.

Keywords: Graph database. Neo4j. Recommendation system. Cypher. NoSQL

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O modelo de dados Relacional obteve um grande uso durante muitos anos na maioria das organizações, mas, atualmente, observa-se casos em que esse modelo de dados pode não ser o único ou o mais adequado. Na área científica e em grandes corporações, alternativas ao modelo de dados Relacional têm sido consideradas. Diante deste contexto, diferentes modelos de dados estão sendo criados e estudados para apoiar as mais diversas abordagens. Devido à alta demanda relacionada ao grande volume de dados, à variabilidade dos dados encontrados na Web e à necessidade de maior velocidade no gerenciamento de dados (BROWN, 2010), várias tecnologias de banco de dados de código aberto, chamadas NoSQL (Not Only SQL) (MONIRUZZAMAN; HOSSAIN, 2013), têm surgido nos últimos anos.

Atualmente, algumas implementações de Sistemas Gerenciadores de bancos de dados (SGBD) NoSQL são alternativas ao SGBDs Relacionais. Estes SGBDs são comumente classificados segundo o modelo de dados implementado. Dentre eles, podem ser citados os bancos de dados: orientado a Documentos, orientado a Objetos, Chave/Valor, e Orientado a Grafos.

Neste trabalho será apresentado um estudo que caso que se utiliza de uma dessas implementações NoSQL. No estudo de caso, foi construído um sistema de recomendação de imóveis similares utilizando um SGDB que utiliza uma implementação orientada a grafos, o Neo4j. E este estudo de caso tem como objetivo mostrar como é possível modelar o problema em nível de banco de dados de uma maneira bem mais próxima ao seu modelo no mundo real, sem a necessidade de adequar os dados ao modelo Relacional.

O Capítulo 2 apresenta um introdução a modelos de banco de dados apresentando os principais bancos de dados NoSQL e um breve resumo da teoria dos grafos. O Capítulo 3 mostra como funciona a estrutura e componentes de um banco de dados orientado a grafos. No Capítulo 4 são apresentadas algumas vantagens de se utilizar uma modelagem orientada a grafos. O Capítulo 5 descreve o estudo de caso proposto. O Capítulo 6 é apresenta a implementação do sistema de recomendação de imóveis e os resultados obtidos. Por fim, no Capítulo 7 temos a conclusão e sugestão de possíveis melhorias no modelo.

2 REFENCIAL TEÓRIOCO

Nos últimos anos, o aumento da capacidade de processamento computacional, a conectividade e o armazenamento de informações vem permitindo que a Tecnologia de Informação (TI) possa prover um avanço em suas soluções. Na década de 1960 e começo dos anos 1970, foram desenvolvidos os primeiros Sistemas de Gerenciamento de Banco de Dados (SGBD) em rede e hierárquicos que atenderam a necessidade do mercado de softwares para mainframes, sendo fortemente utilizados até os anos 1990 (GRAD, 2012).

Na década de 1970, quando foram desenvolvidos os primeiros SGBDs, Edgar Frank Cood publicou um estudo sobre uma nova abordagem envolvendo a criação e posteriormente a aplicação dos conceitos de álgebra relacional e normalizações (CODD, 1970)(CODD, 1971)(CODD, 1979). Em seguida, Codd, junto à empresa International Business Machines (IBM®) [1], desenvolveu um SGBD com enfoque no modelo relacional que foi largamente difundido. Neste modelo, os elementos básicos são as Relações (Tabelas), as quais são compostas de linhas (Tuplas) e colunas (Atributos).

O modelo de dados Relacional desenvolvido por Codd é ainda hoje o mais utilizado para a implementação de SGBDs do mercado. Porém, tanto a complexidade quanto a exigência no que se refere a informação tem mudado rapidamente. Nas últimas décadas, com o desenvolvimento da Web, o volume de dados gerados por aplicações, soluções, recursos e tudo que se refere a sistemas computacionais cresceu de forma expressivamente acelerada, de forma que 90% dos dados existentes até o ano de 2014 no mundo foram produzidos nos últimos três anos (WU X. ZHU, 2014).

Em ambientes onde a complexidade é alta, bancos de dados Relacionais podem não ser a única ou melhor opção. Problemas surgem no contexto de redes sociais, como Twitter [2], Facebook [3], etc. Neste contexto, o uso de bancos de dados Relacionais envolve um grande número de operações de junção que consomem muitos recursos computacionais para serem realizadas (BATRA; TYAGI, 2012). A implementação de bancos de dados capazes de atender as necessidades deste novo cenário passa a conflitar com os conceitos de Codd (GRIER, 2012). As conexões entre vários nós e pontos na Web não é facilmente implementável no modelo Relacional. O modelo de dados Relacional teria sido desenvolvido para uma estrutura que

[1] https://www.ibm.com
[2] https://www.twitter.com
[3] https://www.facebook.com

permitiria se concentrar somente em alguns tipos de relacionamentos (LOPES, 2014). Neste ponto surge a discussão sobre o uso de soluções não relacionais.

2.1 NoSQL

O termo NoSQL já foi utilizado por Carlos Strozzi em 1998, cujo objetivo era apresentar um novo SGBD Relacional não orientado por SQL (NASCIMENTO, 2012). Strozzi afirmava que seu SGBDR se apresentava como uma solução rápida e portátil, cujos limites de escalabilidade não poderiam ser determinados (STROZZI, 1998).Nos últimos anos, Bancos de Dados NoSQL cresceram em utilização, objetivando atender demandas de escalabilidade (LAKSHMAN; MALIK, 2010)). Uma aplicação como o Facebook, por exemplo necessita de um alto índice de escalabilidade, uma vez que são milhares de pessoas gerando um alto fluxo de informações o tempo todo.

NoSQL e, especialmente, bancos de dados orientados a grafos estão constantemente ganhando popularidade entre os desenvolvedores. Isso se deve ao fato desses SGBDs proporem um modelo de gerenciamento de dados com um desempenho superior ao manusear dados altamente interligados em comparação com bancos de dados Relacionais. (HOLZCHUHER; PEINI, 2016).

2.2 Bancos de dados orientados a grafos

Os bancos de dados orientados a grafos são otimizados para aplicações de redes sociais e estrutura de Web links, pois um grafo é uma maneira natural de armazenar conexões entre usuários. Bancos de dados Relacionais podem não ser as melhores opções quando o modelo de dados evolui de acordo com o tempo (Neo4j Blog), o que significa que os bancos Relacionais dependem de um esquema rígido e de pouca flexibilidade na adição de novos relacionamentos entre objetos. Limitações no uso do modelo Relacional levaram à concepção dos bancos de dados orientados a grafos (BATRA; TYAGI, 2012).

Os sistemas de bancos de dados, em que os relacionamentos entre os objetos ou entidades são igualmente importantes quanto os próprios objetos, são denominados Banco de Dados Orientado a Grafos. Neste modelo, os dados são representados por nós, vértices e propriedades. Os nós são representados pelos objetos, os vértices pelos relacionamentos entre os nós e as propriedades pelos valores armazenados nos objetos e relacionamentos. Existem várias

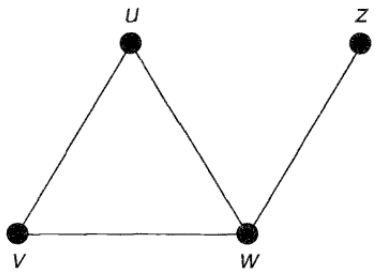
implementações para este tipo de banco de dados. Tanto os nós quanto os vértices podem ter propriedades que descrevem suas características específicas. Os bancos de dados orientados a grafos mais conhecidos são: Infogrid [4], HypergraphDB [5], Jena [6], FlockDB [7] e Neo4j [8].

Sistemas para gerenciamento de bancos de dados orientado a grafos fornecem uma solução para o armazenamento de dados em cenários atuais, nos quais os dados estão cada vez mais conectados, os modelos de grafos são amplamente utilizados, e os sistemas precisam escalar para grandes conjuntos de dados. Neste quadro, a conversão da camada de persistência de uma aplicação de um modelo de dados Relacional para um armazenamento de dados em grafo pode ser conveniente. Mas, geralmente, é uma tarefa difícil para os administradores de banco de dados (DBA) (VIRGILIO; TORLONE, 2013).

2.3 Teoria dos grafos

Um grafo simples G consiste de um conjunto finito não vazio G=(V,E) de elementos chamados vértices (ou nós), e um conjunto finito E de pares não ordenados distintos de elementos distintos de V(G) chamados de arestas. Diz-se que uma aresta *{v, w}* se une aos vértices *v* e *w* e é geralmente abreviada para *vw*. Por exemplo, a Figura 2.1 representa o grafo simples G cujo conjunto de vértices V(G) é *u, v, w, z* e cujo conjunto de arestas E (G) consiste das arestas *uv, uw, vw* e *wz* (WILSON, 1996).

Figura 2.1 – Exemplo de grafo.



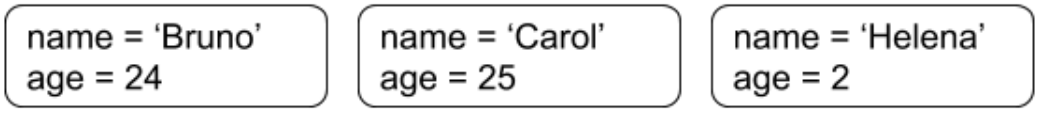
Fonte: (WILSON, 1996)

[4] https://www.infogrid.org
[5] https://www.hypergraphdb.org
[6] https://www.jena.apache.org
[7] https://www.github.com/twitter-archive/flockdb
[8] https://www.neo4j.com

3 BANCOS DE DADOS DE GRAFOS

Um banco de dados orientado a grafos armazena os dados usando um esquema conceitual baseado na teoria dos grafos. Os **nós** são frequentemente usados para representar entidades, mas, dependendo das relações de domínio, podem ser usadas também para representar estas relações. A representação de um nó pode ser visualizado na Figura 3.1.

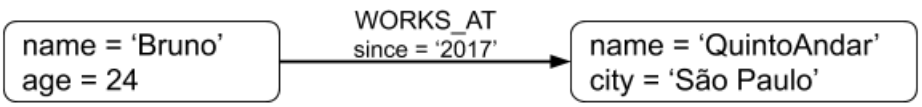
Figura 3.1 – Exemplo de nós em um banco de dados orientado a grafos.



Fonte: Do autor (2019)

O **relacionamento** entre os nós é a característica principal dos bancos de dados de grafos, pois permitem encontrar dados relacionados. Um relacionamento conecta dois nós e é garantido que tenha um nó fonte e um nó de destino válidos. Os relacionamentos organizam nós em estruturas arbitrárias, permitindo que um grafo se assemelhe à uma lista, uma árvore, um mapa ou uma entidade composta de qualquer uma das estruturas as quais pode ser combinadas em estruturas ainda mais complexas e ricamente interligadas. Na Figura 3.2 é apresentada a representação de um relacionamento.

Figura 3.2 – Exemplo de um relacionamento em um banco de dados de grafos.



Fonte: Do autor (2019)

As **propriedades** são valores nomeados em que o nome (ou chave) é uma *string*. Os valores de propriedade suportados são:

(i) Valores numéricos;

(ii) Valores nominais;

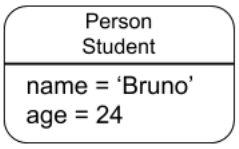
(iii) Valores booleanos; e

(iv) Listas de quaisquer um dos valores acima

Um **rótulo** é uma construção de grafo com nome, que é usada para agrupar nós em conjuntos. Todos os nós marcados com o mesmo rótulo pertencem ao mesmo conjunto. Muitas

consultas de banco de dados podem funcionar com esses conjuntos em vez de todo o grafo, tornando as consultas mais fáceis de escrever e mais eficientes para serem executadas. Um nó pode ser rotulado com qualquer número de rótulos, incluindo nenhum, tornando esta rotulação uma adição opcional ao grafo. Os rótulos também podem ser adicionados e removidos em tempo de execução. Na Figura 3.3 temos a representação de um nó com 2 rótulos.

Figura 3.3 – Exemplo da adição de rótulos ao grafo.



Fonte: Do autor (2019)

Neo4j é um Sistema Gerenciador de Banco de Dados orientado a grafos. Sua arquitetura é projetada para otimizar o gerenciamento, o armazenamento e o percurso de nós e relacionamentos. Gerenciamento de memória dedicado e operações de memória altamente escaláveis e eficientes contribuem para os benefícios (NEO4J, 2016). A abordagem do grafo de propriedades permite o uso consistente do mesmo modelo ao longo da concepção, projeto, implementação, armazenamento e visualização de qualquer domínio ou caso de uso. Com o modelo opcional de esquema, o modelo de domínio pode ser evoluído continuamente à medida que os requisitos mudam.

4 MODELANDO UM GRAFO

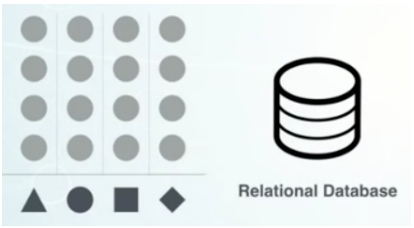
Trocar o modelo de banco de dados Relacional para o orientado a grafos implica em uma mudança na forma como os dados são conceitualmente representados e armazenados. Em algumas situações os dados são esparsos ou não seguem um mesmo padrão ou possuem muito relacionamentos entre si. Um exemplo de representação da organização pode ser dada como na Figura 4.1.

Figura 4.1 – Exemplo de dados.

Fonte: (NEO4J, 2016)

O modelo de dados Relacional faz com que seja necessário organizar os dados para que eles sejam acondicionados em tabelas. Em muitos casos, existe a necessidade de que eles sejam normalizados para evitar a redundância não controlada que causa inconsistências, mas isso faz com que seja difícil uma visualização dos dados em um contexto diferente do transacional, tal como o contexto de uso de Grafos. A Figura 4.2, apresenta um esboço para uma visualização análoga de como os dados devem ser tabulados usando o modelo de Dados Relacional.

Figura 4.2 – Exemplo de tabulação de dados para tabelas.



Fonte: (NEO4J, 2016)

Se os dados possuem muitos relacionamentos, o modelo é difícil de adequar em tabelas ou é necessário que o modelo evolua para junto das regras de negócio, ou seja, para além do uso de um Sistema Gerenciador de Banco de Dados. O modelo de dados orientado a grafos aproxima a representação a como os dados estão relacionados em seu esquema conceitual.

Em uma rede social, por exemplo, existem muitos relacionamentos entre os nós de um mesmo tipo e esses relacionamentos também possuem suas propriedades. Usando um banco

de dados Relacional, pode ser difícil representar e visualizar os dados, pois os relacionamentos e dados estão em torno de uma ou poucas tabelas. No modelo orientado a grafos é possível interligar nós uns com os outros sem a necessidade de fazer com que os dados se encaixem em um esquema de tabela. A essa propriedade, denominamos Schemaless. A Figura 4.3, apresenta como os dados estão organizados de maneira mais próxima a como os dados conceitualmente são representados na Teoria de Grafos.

Figura 4.3 – Exemplo de representação em grafos usando o Neo4J.

Fonte: (NEO4J, 2016)

5 ESTUDO DE CASO

O estudo de caso consiste da implementação de uma pequena parte de um sistema de recomendação de imóveis similares que foi construído para uso comercial na empresa QuintoAndar [1]. Entende-se ainda, que este estudo de caso pode servir como um material didático para iniciantes ao uso de Sistemas de Banco de Dados orientados a grafos. O QuintoAndar é a maior plataforma para aluguel de imóveis do Brasil e seu *site* possui em média 150 mil usuários ativos por dia, 20 mil imóveis publicados e mais de 40 mil visitas à imóveis por semana. Sempre que um usuário entra na página de anúncio de um imóvel, existe uma funcionalidade que mostra imóveis similares àquele no momento.

Anteriormente, esta funcionalidade já havia sido implementada com uma proposta diferente, usando apenas características simples de um imóvel como número de quartos e metragem do apartamento. Desta maneira, a funcionalidade recomendava apartamentos que podiam não ser similares ao que o usuário estava visualizando. Deste problema surgiu a necessidade de se implementar um serviço mais preciso e coerente com a recomendação de imóveis similares.

Para implementação desse novo serviço começamos um teste utilizando o Neo4j como SGBD. Durante os testes, ele se mostrou muito promissor, atendendo às necessidades do sistema de recomendação. A partir disso, o Neo4j foi adotado como SGBD a ser utilizado para a implementação da solução final.

O Neo4j foi escolhido como SGDB, pois, diferente de outros bancos de dados NoSQL, que têm como principal foco melhorar a performance quando trabalhando com grandes volumes de dados, o Neo4j foca na performance ao realizar operações em grafos. Sua implementação têm como foco armazenar os dados o mais próximo possível à modelagem de um grafo, para que as operações realizadas sobre os dados sejam mais performáticas (NEO4J, 2016). O Neo4j é o SGBD orientado a grafos mais utilizado no mercado conforme o ranking DB-Engines [2]. Em seu site existe uma grande quantidade de material disponibilizada, sobre como usar o SGBD, tutoriais da linguagem Cypher e diversos estudos de casos em que o SGDB foi utilizado. Além disso, eles também possuem uma versão do SGDB para empresas com diversas otimizações, recursos para monitoramento e suporte para implementação de novos sistemas. As próximas seções apresentam como um Sistema de Banco de Dados orientado a grafos pode ser usado para criar um sistema de recomendação de imóveis similares.

[1] https://www.quintoandar.com.br
[2] https://db-engines.com/en/ranking/graph+dbms

5.1 Modelagem anterior

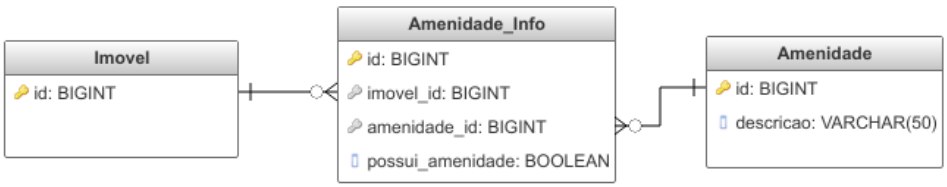
O QuintoAndar possui um banco de dados Relacional com milhares de imóveis que estão publicados no site ou já estiveram publicados em algum momento. Estes imóveis possuem várias características chamadas de amenidades. As **amenidades** de um imóvel podem ser, por exemplo, dados que informam se o imóvel aceita animais de estimação, se possui armários embutidos nos quartos ou se possui piscina no prédio. Cada imóvel possui relacionamento de muitos para muitos com as amenidades, pois um imóvel pode ter várias amenidades e uma amenidade pode estar presente em diversos imóveis, tal como apresentado na Figura 5.1.

Figura 5.1 – Modelagem ER.

Fonte: Do autor (2019)

O banco de dados relacional é então modelado seguindo o Diagrama usando notação IE (Information Engineering) (Figura 5.2), onde há a tabela imóvel, a tabela amenidades e a tabela amenidades_info que representa o relacionamento dos imóveis com suas amenidades. De acordo com a modelagem do banco de dados Relacional, todos os imóveis possuem um relacionamento com todas as amenidades (tabela amenidades_info) e existe o campo *possui_amenidade* que é uma *flag* que pode ser verdadeiro ou falso indicando se o imóvel possui aquela amenidade ou não. Percebe-se aqui, que o modelo Relacional foi adaptado para um contexto diferente do seu uso tradicional. O uso de marcadores(flags) é, por si só, um recurso que deve ser evitado.

Figura 5.2 – Banco de dados relacional.



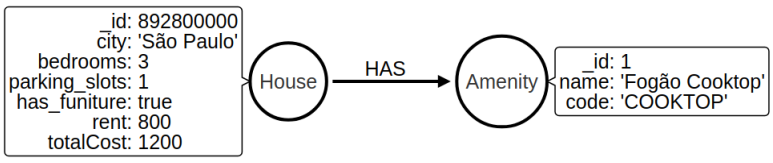
Fonte: Do autor (2019)

6 IMPLEMENTAÇÃO

Para começar a construção do novo sistema de recomendação, primeiramente foi necessário criar a modelagem através do mapeamento do Diagrama ER Conceitual de alto nível (Figura 8) para um modelo de Dados em grafo. A parte modelada é simples, mas atende a finalidade de apresentar um conteúdo didático do uso do modelo de dados de grafos para o problema de recomendação de imóveis similares. Assim, foi usado apenas uma parcela dos campos presentes na tabela imóvel para criar o banco de dados de grafo usando o Neo4j. No nó de Imóvel são modelados os atributos de características principais do imóvel que serão usados para realizar um filtro inicial no conjunto de imóveis. Existem algumas características do imóvel que não são consideradas amenidades, mas também possuem importância na recomendação de similares. Essas características foram modeladas como propriedades do nó Imóvel.

A relação imóvel foi mapeada para nós com o rótulo *House*, as amenidades para nós com o rótulo *Amenity* e a tabela amenidades_info para o relacionamento (aresta) com rótulo *HAS*, conforme apresentado na Figura 6.1. As propriedades foram preenchidas com dados para facilitar na compreensão do leitor.

Figura 6.1 – Modelagem orientada a grafos.



Fonte: Do autor (2019)

6.1 Migração dos dados

Para a migração dos dados, foi realizada a exportação dos dados armazenados em Banco de Dados Relacional para três arquivos com extensão CSV (Comma Separated Values). Cada um dos arquivos foram gerados com os seguintes dados:

1) Tabela imóvel com o subgrupo de campos que seriam utilizados;

2) Tabela amenidades com todas as amenidades;

3) Tabela amenidades_info, somente quando o campo *possui_amenidade* continha valor verdadeiro.

6.2 Populando o banco de Dados de Grafo

O Neo4j utiliza a linguagem Cypher [1] para realizar as operações no banco de dados. A linguagem Cypher foi projetada para ser facilmente legível por pessoas que já possuem algum conhecimento sobre grafos. Seu construto é baseado em prosa e iconografia em inglês para tornar a sintaxe visual e de fácil compreensão (NEO4J, 2016). Ela também contém vários recursos que ajudam a popular o banco de dados, entre eles a importação de dados a partir de arquivo com extensão CSV, conforme apresentado na Figura 6.2. Neste código é apresentada a leitura de dados da primeira linha de um arquivo CSV e a impressão dos dados lidos, apenas como um teste e para visualização da estrutura do CSV. O resultado deste comando é apresentado na Figura 6.3.

Figura 6.2 – Teste carregamento de CSV.

```
LOAD CSV WITH HEADERS FROM "file:///houses.csv" AS line
WITH line
LIMIT 1
RETURN line;
```

Fonte: Do autor (2019)

Figura 6.3 – Retorno da leitura de arquivo CSV.


```
{
  "cidade": "Brasília",
  "aluguel": "500",
  "valorTotal": "504",
  "numeroQuartos": "1",
  "id": "892896797",
  "numeroVagas": "0",
  "mobiliado": "false"
}
```


Fonte: Do autor (2019)

Após o teste de carregamento do arquivo CSV, é necessário criar uma restrição de integridade de unicidade (*unique constraint*) para os nós do tipo House e Amenity. Esta restrição é importante para garantir que os imóveis e amenidades não se repetirão ao importar os dados e quando novos registros forem inseridos após o banco de dados ser populado. O comando

[1] https://neo4j.com/developer/cypher-query-language

utilizado para criar o índice é apresentado na Figura 6.4. Este comando também cria um índice associado à propriedade que possui a restrição de unicidade.

Figura 6.4 – Criando indices.

```
CREATE CONSTRAINT ON (h:House) ASSERT h.id IS UNIQUE;
CREATE CONSTRAINT ON (a:Amenity) ASSERT a.id IS UNIQUE;
```

Fonte: Do autor (2019)

Após criado o índice e a restrição de integridade que garante a não duplicidade dos dados, pode-se começar a importação dos dados a partir dos arquivos CSV. Na Figura 6.5 e Figura 6.6, a importação de imóveis e de amenidades são respectivamente apresentadas. A primeira linha de ambos os comandos contém a instrução *using periodic commit*. Esta instrução faz com que após uma certa quantidade de linhas, seja realizado o *commit* da transação. Isso diminui, o consumo de memória durante a execução. O valor padrão desse comando é 500. Logo, a cada 500 registros inseridos no banco, é realizado um *commit*(Dados são consolidados no Banco de Dados).

Figura 6.5 – Importando imóveis.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///houses.csv" AS line
WITH line
CREATE (h:House{
    id: toInteger(line.id),
    city: line.cidade,
    rent: toInteger(line.aluguel)
    totalCost: toInteger(line.valorTotal),
    bedrooms: toInteger(line.numeroQuartos),
    parking_slots: toInteger(line.numeroVagas),
    has_furniture: toBoolean(line.mobiliado)
});
```

Fonte: Do autor (2019)

O comando *create*, é usado para criar novos nós no banco de dados. Usando este comando e com as *constraints* criadas na Figura 6.4, caso exista um registro duplicado no CSV, a inserção dos dados falharia. É possível trocar o comando create pelo comando *merge*. Assim, caso houvesse um registro duplicado ocorreria a junção do registro já existente no banco com o que está sendo inserido e caso não existisse, ele criaria um novo registro. Porém, como o CSV

que foi gerado não possui nenhum registro duplicado, foi utilizado o *create* e todos os registros foram inseridos sem inconsistência.

Figura 6.6 – Importando amenidades.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///amenities.csv" AS line
WITH line
CREATE (a:Amenity{
    id: toInteger(line.id),
    name: line.nome,
    code: line.code
});
```

Fonte: Do autor (2019)

O primeiro comando de importação de imóveis, presente no código 3, cria 13400 nós do tipo *House* em um tempo de 1350 milissegundos para terminar a execução. Já o comando de importação de amenidades, do código 4, cria 46 nós do tipo *Amenity* em um tempo de 45 milissegundos em sua execução.

A título de comparação dos números de tempo de execução apresentados neste artigo, todos os comandos, inserções e consultas performadas durante este estudo foram realizados em um notebook com um processador Intel i7 7500u 2.7 GHz com 4 núcleos de processamento, 16 GB de memória RAM, utilizando o sistema operacional Ubuntu 18.10 e a versão 3.5.5 Community do Neo4j. Após o estudo de viabilidade a solução foi migrada para a versão Enterprise do Neo4j em um Cluster disponibilizado pela Amazon Web Services [2].

Após criados todos os imóveis e amenidades no banco de dados, foi importado o terceiro e último arquivo CSV que contém os relacionamentos entre os Imóveis e Amenidades. Na Figura 6.7 é apresentado o comando executado para a criação das arestas. A linha 4 apresenta o comando *match*, que tem um funcionamento semelhante ao *select* de bancos de dados SQL. Este comando seleciona os nós que serão utilizados para criar uma aresta. A linha 5, apresenta o comando *create* com uma sintaxe usando setas para indicar a aresta criada. A aresta é associada ao rótulo *HAS* e é direcionada.

O comando apresentado na Figura 6.7 cria 199500 arestas *HAS* entre House e Amenity em um tempo de 6100 milissegundos para terminar a execução. Para visualizar o grafo criado com as arestas entre imóveis e amenidades pode-se executar uma consulta como a presente no

[2] https://aws.amazon.com

Figura 6.7 – Criando relacionamentos.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///house_amenity.csv" AS line
WITH line
MATCH (h:House { id: toInteger(line.imovel_id)}),
      (a:Amenity { id: toInteger(line.amenidades_id)})
CREATE (h)-[:HAS]->(a);
```

Fonte: Do autor (2019)

Código 6. Um imóvel que possui o *id* igual a *892896797* é selecionado com todas as suas amenidades. O resultado deste comando pode ser visualizado na Figura 6.9. Este grafo foi gerado através da ferramente Neo4j Browser[3] que já é instalada juntamente com o SGBD.

Figura 6.8 – Consulta de um imóvel.

```
MATCH (h:House{id: 892896797})-[HAS]->(a:Amenity)
RETURN h, a;
```

Fonte: Do autor (2019)

É possível inverter o comando executado no Código 6, que busca as amenidades de um imóvel específico, e buscar os imóveis que possuem uma amenidade. A Figura 6.10 contém um exemplo de como buscar todos os imóveis que possuem *banheira*.

6.3 Busca de similares

Com os passos realizados na sessão 6.2, obtém-se o banco de dados populado com todos os dados necessários para o sistema de recomendação de imóveis similares. Com isso, podemos construir o comando que seleciona esses imóveis. A primeira versão testada para a consulta é a que está presente na Figura 6.11. Os passos executados por esta Query são os seguintes:

- Linha 1: Seleciona em *h2* todos os imóveis que têm alguma amenidade em comum com o imóvel *h1*, que seria o imóvel visualizado pelo usuário;

- Linha 2: Seleciona todas as amenidades do imóvel *h1* e todas as amenidades do imóvel *h2*;

- Linha 3: Conta todas amenidades dos imóveis e armazena em *a1Count* e *a2Count*;

[3] https://neo4j.com/developer/guide-neo4j-browser

Figura 6.9 – Resultado da consulta de imóvel.

Fonte: Do autor (2019)

Figura 6.10 – Consulta de uma amenidade.

```
MATCH (h:House)-[HAS]->(a:Amenity{name: 'Banheira'})
RETURN h, a;
```

Fonte: Do autor (2019)

- Linha 4: Seleciona as amenidades em comum entre dois imóveis e armazena em *a*;

- Linha 5: Conta as amenidades presentes em *a*;

- Linha 6, 7, 8: Filtra os imóvel que possuem pelo menos 90% das amenidades em comum;

- Linha 9: Retorna o resultado com os imóveis que satisfazem o filtro;

- Linha 10: Ordena o resultado de maneira decrescente usando o número amenidades em comum.

- Parâmetro "0.9": Estudos realizados pelo QuintoAndar, com base nos imóveis alugados que foram alugados, indicaram que dentre todos os imóveis que um usuário visualiza e demonstra algum interesse, seja marcando visita ou fazendo uma proposta, existem em média 90% de compatibilidade nas amenidades, por isso o valor 0.9 é usado pra filtrar os resultados.

Figura 6.11 – Consulta de similaridades.

```
MATCH (h1:House{id: 892896797})-[:HAS]->(:Amenity)<-[:HAS]-(h2:House)
MATCH (h1)-[:HAS]->(a1:Amenity),(h2)-[:HAS]->(a2:Amenity)
WITH h1, h2, COUNT(DISTINCT a1) AS a1Count,
    COUNT(DISTINCT a2) AS a2Count
MATCH (h1)-[:HAS]->(a:Amenity)<-[:HAS]-(h2)
WITH h1, h2, a1Count, a2Count, COUNT(a) AS commonAmenityCount
WHERE
    commonAmenityCount >= a1Count * 0.9 AND
    commonAmenityCount >= a2Count * 0.9
RETURN h1, h2, commonAmenityCount
ORDER BY commonAmenityCount DESC
```

Fonte: Do autor (2019)

A Query foi executada no tempo de 101000 milissegundos. Entende-se que este é um tempo muito grande e não aceitável para nosso caso de uso. O primeiro problema identificado nesse comando é que ele realiza uma comparação com todos os nós do banco de dados e não somente os da cidade em que o imóvel inicial se encontra. A partir disso, foi alterada a primeira linha para adicionar um filtro por cidade, conforme apresentado na Figura 6.12.

Figura 6.12 – Adicionando filtro de cidade.

```
MATCH (h1:House{id: 892896797})
MATCH (h1)-[:HAS]->(:Amenity)<-[:HAS]-(h2:House{city: h1.city })
```

Fonte: Do autor (2019)

Com esta alteração o tempo de execução reduziu consideravelmente. Em cidades com poucos imóveis, o comando é executado em um tempo médio de 15000 milissegundos, mas em São Paulo, onde existe a maior quantidade de imóveis, ainda demora em torno de 54000 milissegundos, um tempo considerado ainda muito alto. A partir disso, foram adicionados mais filtros com base nas propriedades do imóvel comparado com o imóvel visualizado. Foi incluída uma cláusula *where* entre a primeira e a segunda linha do Código 8, conforme apresentado no Código 10. A quantidade de filtros usados na Query pode mudar de acordo com o filtro realizado pelo usuário no Sistema Web e com as características do imóvel visualizado. A Figura 6.13 apresenta um filtro que inclui o número de vagas no estacionamento, número de quartos, se o imóvel é mobiliado, e limita o valor do aluguel a até 50% maior do que no anúncio atual.

A decisão de filtrar imóveis com o valor de aluguel em 1.5 vezes o valor do imóvel sendo visualizado tem como base estudos realizados pelo QuintoAndar, que demonstraram que um usuário que visualiza um imóvel em uma certa faixa de preço pode alugar um outro imóvel que possui no máximo 50% a mais no valor de aluguel do primeiro imóvel acessado.

Figura 6.13 – Adicionando outros filtros.

```
MATCH (h1:House{id: 892896797})
MATCH (h1)-[:HAS]->(:Amenity)<-[:HAS]-(h2:House)
WHERE h2.parking_slots >= h1.parking_slots AND
    h2.bedrooms >= h1.bedrooms AND
    h2.has_furniture = h1.has_furniture AND
    h2.rent < h1.rent * 1.5
MATCH (h1)-[:HAS]->(a1:Amenity),(h2)-[:HAS]->(a2:Amenity)
```

Fonte: Do autor (2019)

Após a adição dos filtros, o tempo de execução da Query reduziu e na cidade de São Paulo, que antes era o maior problema referente ao tempo de execução, passou a levar em média 700 milissegundos. Em outras cidades, com menos imóveis, a média do tempo de execução passou para 400 milissegundos, um tempo aceitável considerando a configuração da máquina onde os testes foram realizados.

7 CONCLUSÃO

Os bancos de dados NoSQL estão ganhando espaço nas organizações, mas isso não quer dizer que os bancos Relacionais deixarão de existir. Estes bancos de dados que existem desde os anos 70 têm atendido muito bem diversos ramos de aplicações. Há casos em que os dados se adéquam totalmente à tabelas e sua utilização é eficiente.

O motivo do surgimento e popularização dos bancos de dados NoSQL é atender situações em que o uso do modelo de dados Relacional não se adéquam bem ao formato dos dados. Para saber qual o modelo de dados a ser usado é necessário analisar os dados utilizados pelas aplicações e entender como os dados se relacionam, verificando se é vantajoso mudar de um modelo.

No contexto deste trabalho (Recomendação de Imóveis Silimares), o modelo orientado a grafos apresenta-se como uma alternativa de modelagem e armazenamento dos dados. Com ele é possível representar os relacionamentos entre os dados, trabalhar com um pequeno conjunto de dados, com todos os dados disponíveis, entre outros recursos. Entre os recursos podemos destacar que é possível também alterar a forma física na qual os dados são armazenados usando somente uma máquina ou dividir os dados em conjuntos próximos e armazená-los em máquinas diferentes. No modelo de dados Relacional, é difícil criar uma implementação que divide diversas tabelas usando localidade dos dados e armazenar em máquinas diferentes. E a maioria das soluções se encontram em SGBDs Relacionais com valores altos de licença de uso. Essas são algumas vantagens que o modelo orientado a grafos (em especial utilizando o Neo4j) agrega para as aplicações. Usando corretamente e sabendo se beneficiar dos recursos disponíveis, eles podem trazer um grande aumento de desempenho quando comparado aos bancos de dados relacionais.

No estudo de caso apresentado neste artigo apresentou-se um desses casos onde a modelagem orientada a grafo torna a representação dos dados muito mais próxima do que é no mundo real. Uma melhoria futura que pode ser feita na modelagem aqui apresentada é utilizar do recurso de localidade dos dados, modelando os dados usando a cidade do imóvel como um parâmetro e criar arestas entre os imóveis para armazenar a distância entre eles. Estas modificações serão importantes para definir em que nó do Cluster o imóvel será armazenado para melhorar ainda mais o desempenho das consultas realizadas para realizar a recomendação.

Além de melhorias neste estudo de caso, conforme mencionado acima, a implementação de um banco de dados orientados a grafos também abre caminho para criação de diversos outros

sistemas que podem se beneficiar do uso de modelos gráficos. Podemos destacar, por exemplo, a otimização da agenda dos corretores, este modelo poderia auxiliar a encontrar qual o melhor conjunto de visitas para que o corretor possa se deslocar a menor distância possível entre as visitas, e outras aplicações.

REFERÊNCIAS

BATRA, S.; TYAGI, C. Comparative analysis of relational and graph databases. In: **International Journal of Soft Computing and Engineering (IJSCE), v. 2, iss. 2**. [S.l.: s.n.], 2012. p. 509–512.

BROWN, P. G. Overview of scidb: Large scale array storage, processing and analysis. In: **Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD '10**. [S.l.]: New York, NY, USA, 2010, 2010. p. 963–968.

CODD, E. F. A relational model of data for large shared data banks. In: **Comm. ACM, v. 13**. [S.l.: s.n.], 1970. p. 377–387.

CODD, E. F. Normalized database structure: A brief tutorial. In: **SIGFIDET '71 Proceedings of the 1971 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control**. [S.l.: s.n.], 1971. p. 1–17.

CODD, E. F. Extending the database relational model to capture more meaning. In: **ACM Transactions on Database Systems (TODS), v. 4, iss. 4**. [S.l.: s.n.], 1979. p. 397–434.

GRAD, B. Relational database management systems: The formative years. In: **IEEE Annals of the History of Computing**. v. 34, n. 4. [S.l.: s.n.], 2012. p. 7–8.

GRIER, D. A. The relational database and the concept of the information system. In: **IEEE Annals of the History of Computing**. v. 34, n. 4. [S.l.: s.n.], 2012. p. 9–16.

HOLZCHUHER, F.; PEINI, R. Querying a graph database: Language selection and performance considerations. In: **Journal of Computer and System Sciences**. v. 82, iss. 1. [S.l.: s.n.], 2016. p. 45–68.

LAKSHMAN, A.; MALIK, P. Cassandra: A decentralized structured storage system. In: **ACM SIGOPS Operating Systems Review**. v. 44, n. 2. [S.l.: s.n.], 2010.

LOPES, J. M. Um estudo comparativo entre banco de dados considerando as abordagens relacional e orientada a grafo. In: . [S.l.]: Universidade Federal de Santa Catarina Campus Araranguá. Araranguá, SC, 2014.

MONIRUZZAMAN, A. B. M.; HOSSAIN, S. A. Nosql database: New era of databases for big data analytics – classification, characteristics and comparison. In: **IEEE Transactions on Knowledge and Data Engineering**. [S.l.]: Disponível em: http://arxiv.org/abs/1307.0191, 2013.

NASCIMENTO, J. R. A. Nosql: Conceitos e evolução. mundoj - nosql: um novo paradigma ara persistência distribuída e escalável. In: **Rio de Janeiro**. n. 51. [S.l.: s.n.], 2012. p. 6–9.

NEO4J. Graph database concepts introduction. In: . [S.l.]: Disponível em: https://neo4j.com/docs/developer-manual/current/introduction/, 2016.

STROZZI, C. Nosql: a non-sql rdbms.1998. In: . [S.l.]: Disponível em: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en$_US/NoSQL/Home$20$Page$, 1998.

VIRGILIO, A. M. R. de; TORLONE, R. Converting relational to graph databases. In: **First International Workshop on Graph Data Management Experiences and Systems (GRADES'13)**. [S.l.]: New York, York, USA, ACM Press, 2013. p. 1–6.

WILSON, R. J. Introduction to graph theory. In: . 4th. ed. [S.l.]: Longman Group Ltd., 1996. p. 8.

WU X. ZHU, G. Q. W. W. D. X. Data mining with big data. In: **IEEE Transactions on Knowledge and Data Engineering**. v. 26, n. 1. [S.l.: s.n.], 2014. p. 97–207.