Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise de Desempenho de SGBD não Relacionais com Dados Geográficos do OpenStreetMap

Victor Fernandes Uriarte

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientadora
Prof. ª Dr. ª Maristela Terto de Holanda

Brasília
2018

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise de Desempenho de SGBD não Relacionais com Dados Geográficos do OpenStreetMap

Victor Fernandes Uriarte

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Prof. ª Dr. ª Maristela Terto de Holanda (Orientadora)
CIC/UnB

Prof. ª Iasmini Virgínia Oliveira Lima
UNIEURO

Prof. Dr. Vinicius Ruela Pereira Borges
CIC/UnB

Prof. Dr. José Edil Guimarães de Medeiros
Coordenador do Curso de Engenharia da Computação

Brasília, 19 de Dezembro de 2018

Dedicatória

Dedico este trabalho em especial à minha família, que me apoiou ao longo de toda minha jornada acadêmica de maneira incondicional.

Agradecimentos

Agradeço à minha família, em especial aos meus pais, Marcelo Uriarte e Ana Paula Fernandes Uriarte; e ao meu irmão, Leonardo Fernandes Uriarte. Agradeço a Kelly Cristina Nascimento Amancio, pelo companheirismo e apoio incondicional. Agradeço aos colegas de curso Murilo Zaffalon Marra, Victor Zaffalon Marra, André Barbosa Carneiro da Cunha Bauer e Ana Paula Almeida. Agradeço a orientadora Maristela Terto de Holanda, pelo apoio e paciência.

Resumo

Com a popularização da internet e o aumento de fluxo e complexidade de dados, surgem os chamados bancos não relacionais, ou NoSQL, uma alternativa ao já consagrado modelo relacional, buscando maior flexibilidade de dados e desempenho. Aliado a isto, há o aumento de dados fornecidos por usuários, dando origem às plataformas colaborativas, como o OpenStreetMap, que visa a confecção de um modelo de dados do globo terrestre para uso comunitário. Esse serviço, logicamente, movimenta uma grande quantidade de dados de diferentes formatos. Nesse contexto, este trabalho visa investigar a viabilidade de adoção de três bancos de dados, dois oriundos dessa categoria, o MongoDB e o Neo4j, e um relacional, o PortgreSQL, mensurando o desempenho destes SGBD na inserção e em consultas utilizando dados espaciais provenientes do OpenStreetMap.

Palavras-chave: NoSQl, OpenStreetMap, PostgreSQL, PostGIS, MongoDB, Neo4j, Neo4j Spatial.

Abstract

The popularization of Internet and the increasing data complexity and flow made the non relational databases, or NoSQL, emerge as an alternative to the old and well-established relational model, seeking greater data flexibility and performance. In adition to this, there is an increase in data provided by users, giving birth to collaborative platforms such as OpenStreetMap, that aims to build an accurate Earth dataset model for community use. It is evident that this service drives massive volumes of data of different formats. In this context, this project aims to investigate the feasibility of adopting three databases, two provenient from this category, the MongoDB and Neo4j, and one relational, the PostgreSQL, measuring their performances inserting and querying spatial data from OpenStreetMap.

Keywords: NoSQl, OpenStreetMap, PostgreSQL, PostGIS, MongoDB, Neo4j, Neo4j Spatial.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Desde seus primórdios, nos anos 90, a *World Wide Web*, também conhecida como *Web 1.0*, passou por diversas evoluções e aperfeiçoamentos, se tornando uma importante fonte de dados e informações. Estas evoluções resultaram em uma quebra de paradigma no começo do século 21, onde os usuários não mais utilizavam a Web apenas para o consumo de dados pré-disponibilizados, mas também para a criação e disponibilização de seus próprios dados, marcando assim o nascimento da *Web 2.0* [67].

Aliado a este fato, a popularização de dispositivos com conectividade GPS (*Global Positioning System*, ou Sistema de Posicionamento Global) e acesso à *Internet* tornou o mapeamento da superfície terrestre, até então reservada para profissionais, uma tarefa palpável para um usuário comum. Houve, desse modo, uma democratização na disponibilização de dados geográficos, onde a geração e o compartilhamento dos mesmos se dá de maneira colaborativa por vários usuários [49].

Atualmente, a plataforma colaborativa OpenStreetMap (OSM) é a principal representante deste panorama. O OpenStreetMap é um sistema de informação geográfica voluntária desenvolvido no ano de 2004 pelo estudante da *University College London* (UCL) Steve Coast, que possui como objetivo principal a criação e disponibilização de uma base de dados geográficos que possa ser utilizado pelos usuários de forma gratuita nas mais diversas aplicações [72].

Para o armazenamento e manipulação de dados, os bancos de dados relacionais atualmente são as principais ferramentas utilizadas. Nos últimos anos, porém, houve um crescimento na adoção dos bancos não relacionais, chamados NoSQL, devido principalmente à necessidade de um modelo de dados mais flexível, com grande escalabilidade, disponibilidade e capacidade de lidar com um volume cada vez maior de informações. A sigla NoSQL significa *Not Only SQL* e diz respeito aos novos sistemas gerenciadores de banco de dados (SGBD) que procuram abordagens diferentes do tradicional modelo relacional, como modelo de dados baseados em grafos, documentos, chave-valor e colunas,

além de buscar um desempenho escalável [64].

De acordo com o site *Db-Engines*[1], o SGBD não relacional mais popular atualmente é o MongoDB, um banco de dados orientado a documentos utilizado por diversas empresas como *Facebook*, *Cisco*, *EA Games*, *Ebay*, *Adobe* entre outras. Há, porém, uma forte tendência de aumento no uso de SGBD orientados a grafo, sendo o representante mais popular da categoria o Neo4j, também utilizado por companhias de grande porte como *Microsoft*, *Walmart*, *IBM*, *Airbnb* etc.

Nesse contexto, a proposta deste trabalho é utilizar amostras de dados de diferentes tamanhos proveniente da plataforma OpenStreetMap em conjunto com os SGBD não relacionais MongoDB e Neo4j, além do tradicional banco relacional PostgreSQL, realizando testes de desempenho e investigando a usabilidade das ferramentas de tratamento de dados espaciais presentes nos três SGBD. Este trabalho visa contribuir na escolha de um SGBD com bom desempenho para utilizar em conjunto com dados espaciais do OpenStreetMap, por meio da realização de testes e comparações entre abordagens já consagradas no mercado e abordagens emergentes no cenário dos dados geoespaciais.

1.1 Objetivo

O objetivo deste trabalho é realizar uma análise de desempenho de três bancos de dados, dois não relacionais, o MongoDB e o Neo4j, e um relacional, o PostgreSQL, para o armazenamento e manipulação de dados geográficos provenientes da plataforma colaborativa OpenStreetMap (OSM).

1.1.1 Objetivos Específicos

Para alcançar o objetivo geral, os seguintes objetivos específicos foram definidos:

- Coletar amostras de dados geográficos de diferentes tamanhos provenientes da plataforma OpenStreetMap;
- Realizar testes de inserção das amostras nos três SGBD, medir o tempo gasto na operação e traçar uma comparação entre os tempos decorridos nas inserções em cada SGBD;
- Realizar consultas espaciais equivalentes para as três plataformas, tomando o tempo decorrido da operação para posterior comparação;

[1] `https://db-engines.com`

- Com base nos testes, traçar uma conclusão acerca da maturidade e viabilidade do emprego dos SGBDs não relacionais utilizados neste trabalho para tratamento de dados espaciais provenientes do OpenStreetMap.

1.2 Estrutura do Trabalho

Os seguintes capítulos fazem parte desta monografia:

- O Capítulo 2 apresenta o referencial teórico envolvido neste trabalho. São apresentados os conceitos de Bancos de dados Geográficos, OpenStreetMap e Bancos de Dados Relacionais, em especial o PostgreSQL. Com um pouco mais de profundidade, também são apresentados os Bancos não relacionais, ou NoSQL, assim como suas características, seus modelos, sua popularidade e os sistemas gerenciadores de bancos de dados MongoDB e Neo4j, alvos deste trabalho;
- O Capítulo 3 traz a metodologia relativa à coleta, inserção e manipulação dos dados espaciais provenientes do OpenStreetMap nos três SGBD: PostgreSQL, MongoDB e Neo4j. Além disso, é nesse capítulo que estão demonstrados todos os resultados, as inferências que podem ser feitas a partir deles e algumas considerações finais;
- O Capítulo 4 consiste na conclusão propriamente dita do trabalho, resumindo o que foi realizado no trabalho e explicitando o cumprimento ou não dos objetivos e os possíveis trabalhos futuros.

Capítulo 2

Referencial Teórico

O objetivo deste capítulo é apresentar alguns dos principais conceitos envolvidos na realização deste trabalho, tais como Banco de Dados Geográficos e NoSQL, além de explicar com mais detalhes algumas das ferramentas envolvidas, como o MongoDB, OpenStreetMap, Neo4j e PostgreSQL.

2.1 Banco de Dados Geográficos

Bancos de Dados Geográficos, também chamados de Banco de Dados Espaciais, são bancos que realizam o armazenamento e manipulação de objetos que possuem características espaciais que os descrevem e que possuem relacionamentos espaciais entre eles em um espaço multidimensional [47].

Ao se empregar um Banco de Dados Geográfico, se faz necessário a representação do espaço propriamente dito e de objetos no espaço, de modo a possibilitar a modelagem de um país, região ou área definida e de seus componentes, como ruas, rios ou florestas. A união de ambas as representações constitui uma coleção de objetos espacialmente relacionados entre si [50].

As estruturas de dados presentes em um banco de dados geográficos são classificadas em [39]:

- Estrutura Vetorial: São dados baseados em coordenadas geográficas, onde cada objeto geográfico é representado por um par de coordenadas cartesianas;
- Estrutura Matricial (*raster*): O espaço é representado como uma estrutura plana por meio de um conjunto de linhas horizontais e verticais, formando uma matriz. Geralmente, os dados são provenientes de imagens geradas por dispositivos eletrônicos, satélites e dispositivos de sensoriamento.

Nas estruturas vetoriais, existem três abstrações fundamentais [39]:

- Ponto: Representa a localização de um objeto sem considerar a área do mesmo;
- Linha: Curva que representa o caminho entre pontos, usada para abstrair rios, pontes, estradas, dentre outros;
- Polígono: Representação de um objeto levando em consideração suas dimensões, demarcando uma região bem definida. O polígono é formado por um conjunto de vértices onde cada um deles é uma coordenada geográfica, representado por um par ordenado XY. A Figura 2.1 exemplifica um polígono representado em estrutura vetorial.

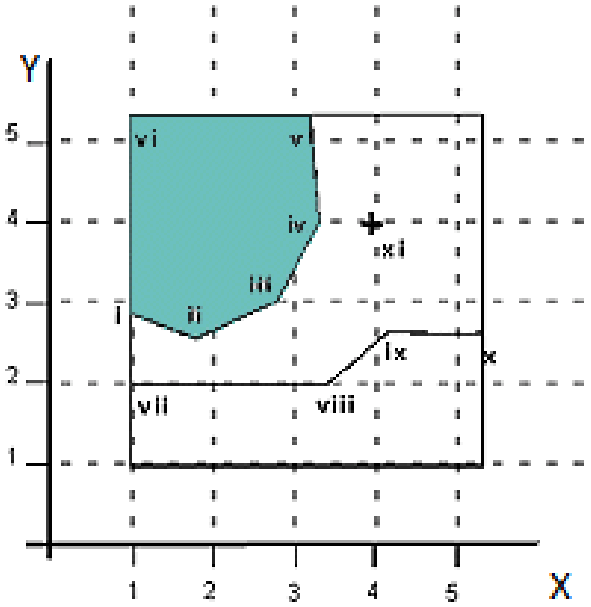

Figura 2.1: Polígono em um espaço vetorial. Adaptado de [26].

Na estrutura matricial, cada pixel da imagem possui dados geocodificados, possibilitando a representação de espaços com grande riqueza de detalhes. Nessa estrutura, um polígono é definido por um conjunto de células da matriz, conforme a Figura 2.2 [38].

2.1.1 Funções e Propriedades Espaciais

De acordo com [70], existem algumas funções e atributos desejáveis para um bom sistema de banco de dados espacial. Primeiramente, é interessante que o SGBD possua suporte a diferentes sistemas de referenciamento espacial para que os requisitos das aplicações possam ser atendidos.

É necessário que se saiba que tipo de dados são suportados pelo banco de dados, como pontos, multipontos, linhas, polígonos, entre outros. Saber como se dá a indexação

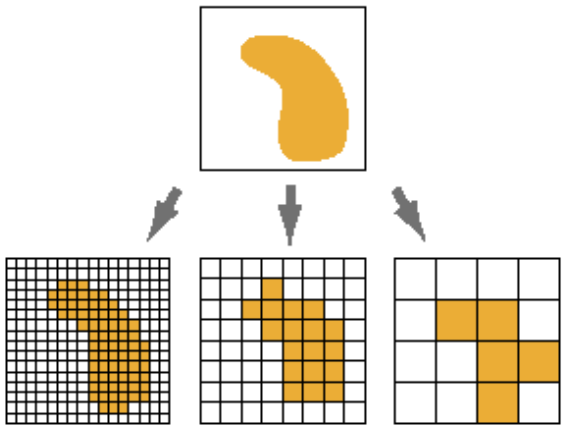

Figura 2.2: Um objeto e sua representação matricial em diferentes níveis de detalhamento, adaptado de [28].

dos dados espaciais também é importante, visto que existem certas técnicas que reduzem consideravelmente o tempo de processamento.

Para que se possa processar os dados espaciais de forma relevante, há três funções de tratamento de dados espaciais específicas que podem ser empregadas: funções de análise métrica, funções de conjunto e funções topológicas.

Funções de análise métrica consistem em funções que realizam algum tipo de análise espacial nos dados. Algumas delas são [70]:

- *length()*: Processa e retorna o tamanho total da geometria;
- *area()*: Processa e retorna a área total da geometria;
- *buffer(distance)*: Retorna a geometria referente aos pontos que se encontram a uma distancia menor ou igual a *distance*.

Funções de conjunto consistem em funções que são executadas sobre um conjunto de dados espaciais. Algumas delas são [70]:

- *union(Geometria)*: Processa um conjunto de dados espaciais e retorna a geometria formada pela união dos mesmos;
- *intersection(Geometria)*: Processa um conjunto de dados espaciais e retorna a geometria formada pela interseção dos mesmos;
- *symDierence(Geometria)*: Processa um conjunto de dados espaciais e retorna a geometria formada pela diferença dos mesmos;

Funções espaciais consistem em funções de tratamento topológico, onde a escala, a rotação e o *zoom* não fazem diferença. Algumas delas são [70]:

- *equals(Geometria)*: Compara duas geometrias e verifica se elas são iguais ou não;
- *touches(Geometria)*: Dadas duas geometrias, indica se há o contato de suas fronteiras sem que haja contato de seus interiores;
- *within(Geometria)*: Dadas duas geometrias, indica se a segunda encontra-se por completo no interior da primeira;
- *disjoint(Geometria)*: Dadas duas geometrias, indica se as duas estão totalmente separadas uma da outra.

2.1.2 Entrada e Saída de Dados

Diferentes SGBD podem utilizar diferentes formatos de entrada e saída de dados. Por esse motivo, é importante conhecer alguns dos principais formatos utilizados [70]:

- *JSON (JavaScript Object Notation)*: É um formato aberto de transferência de dados baseado em uma coleção de pares nome-valor. Possui uma variação chamada de *GeoJSON*, cujo objetivo é representar dados geográficos em conjunto com dados não espaciais;
- *SHP (Shapefile)*: É um vetor de dados geoespaciais criado e mantido pela ESRI[1]. É o mais popular dentre os formatos, sendo considerado praticamente o padrão dos formatos geoespaciais;
- *WKT (Well-Known Text)*: É um formato que usa texto para representar dados geográficos. É um formato desenvolvido pela *Open Geospatial Consortium*[2] (OGC).
- *WKB (Well-Known Binary)*: Similar ao WKT, porém representa os dados através de binários. Também desenvolvida pela *Open Geospatial Consortium* (OGC).
- *SVG (Scalable Vector Graphics)*: É um formato que utiliza um vetor baseado em XML para gráficos em duas dimensões. É proposto pelo W3C[3].
- *GML (Geography Markup Language)*: É um formato que usa XML para expressar dados geográficos. É um formato padrão mantido pela *Open Geospatial Consortium* (OGC).

[1] http://www.esri.com/
[2] http://www.opengeospatial.org/
[3] http://www.w3.org/

- *OSM (Open Street Map)*: É um formato que usa XML para codificar dados espaciais de modo ordenado e estruturado. Formato padrão mantido pela plataforma colaborativa OpenStreetMap, projetado para enviar dados espaciais de maneira fácil e padronizada.

Para este trabalho, o formato OSM será utilizado para a entrada dos dados nos SGBD. Os dados serão providos pelo OpenStreetMap, plataforma explicada na sessão a seguir.

2.2 OpenStreetMap

O OpenStreetMap é um Sistema de Informação Geográfica Voluntária (SIGV) colaborativo que tem como principal objetivo a criação e disponibilização de uma base de dados de informações geográficas sob licença aberta que pode ser utilizado para mapeamento, navegação e em aplicações gerais. O projeto foi criado em 2004 pela *The OpenStreetMap Foundation*, uma organização sem fins lucrativos que promove o crescimento, desenvolvimento e a distribuição de dados geoespaciais de forma gratuita. O projeto foi, em grande parte, motivado pelo sucesso de outros sistemas colaborativos como a *Wikipedia*[4]. Atualmente, seu banco de dados central e seus principais *web services* estão hospedados em diversos servidores na *University College London*[5]. A Figura 2.3 ilustra a *homepage* do projeto [81, 54].

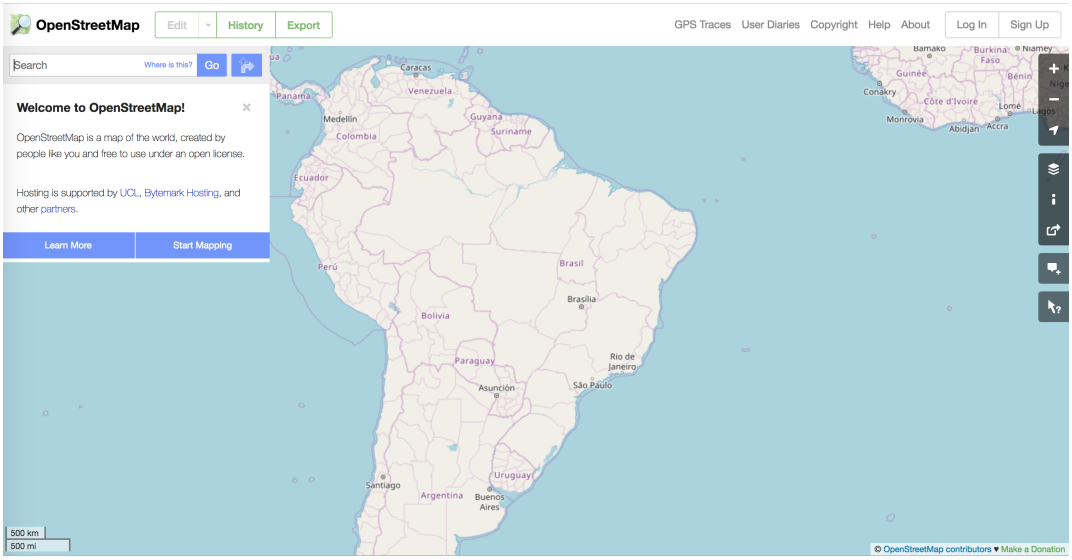

Figura 2.3: Interface principal do OpenStreetMap.

Dois fatores foram primordiais para o sucesso da plataforma [81]:

[4] https://pt.wikipedia.org/
[5] https://www.ucl.ac.uk/

1. A flexibilização das restrições de uso e acesso aos dados geográficos mundiais;
2. A popularização dos dispositivos GPS.

Os dados disponibilizados pelo OpenStreetMap são gratuitos e estão em constante atualização pelos usuários da plataforma, que possuem a liberdade de adicionar os pontos que lhe interessarem. Isso faz com que seja possível o mapeamento das mais diversas regiões do globo, até mesmo aquelas onde muitas das empresas de mapeamento comercial não conseguem mapear ou não o julgam necessário ou rentável [79].

Atualmente, a plataforma conta com 4.3 milhões de usuários registrados, 4.2 bilhões de nós inseridos e cerca de 3 milhões de modificações diárias [25]. Nos últimos anos, grandes empresas como *Foursquare*[6] e *Wikipédia*[7] migraram suas soluções para o OpenStreetMap em detrimento de outras soluções de mapeamento como o *Google Maps*, pois este último passou a cobrar pela utilização de seus dados. A Figura 2.4 demonstra a utilização do OpenStreetMap em um artigo da *Wikipédia* referente ao Museu de Arte da Pampulha [79].

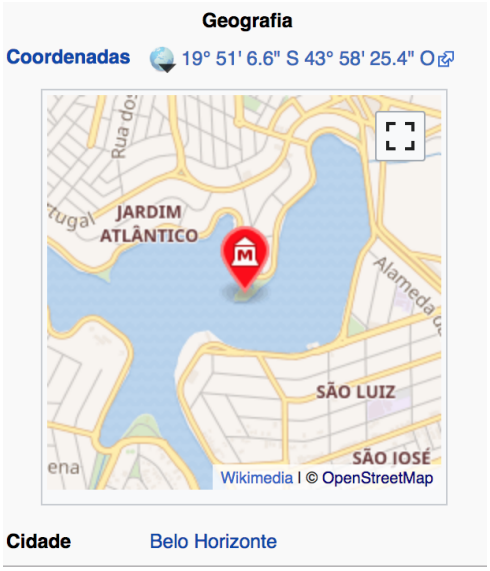

Figura 2.4: Utilização do OpenStreetMap em artigos da *Wikipédia*. Retirado de [16].

Embora o projeto do OpenStreetMap ser em grande parte colaborativo, conforme dito anteriormente, é importante ressaltar que houve a participação de empresas privadas e órgãos governamentais de diversos países na composição da base de dados. A subseção a seguir tem como objetivo detalhar um pouco melhor de onde os dados presentes na plataforma OSM se originam.

[6] `https://foursquare.com/`
[7] `https://www.wikipedia.org/`

2.2.1 Origem dos Dados

Os dados geográficos disponíveis na plataforma OSM são, em sua grande maioria, fruto de um trabalho colaborativo de profissionais e entusiastas da área de sistemas de informação geográfica, assim como de voluntários munidos de dispositivos móveis com GPS. Contudo, há doações de dados geográficos por parte de empresas privadas e de orgãos governamentais em diversos países, além da incorporação ao projeto de dados abertos disponibilizados por estes.

Em 2006, o *Yahoo!*[8] doou imagens digitais gratuitamente para a comunidade OSM, para que o mapeamento pudesse ser feito diretamente pelas imagens. Posteriormente, o OSM obteve dados geográficos gratuitos de empresas e governos, tais como toda a malha viária da Holanda. Outros países que cooperaram com o projeto foram os EUA, Áustria, França e Espanha [54, 65].

Em novembro de 2010, similarmente com o que ocorreu com o *Yahoo!* em 2006, um acordo foi firmado entre o OSM e a *Microsoft Bing Aerial Imagery*. Estes dois acordos tiveram um grande impacto na quantidade de objetos disponíveis na plataforma, principalmente em relação às informações referentes às construções [65].

Durante os últimos anos, houve uma mudança no padrão dos dados fornecidos pelos voluntários. Havia um esforço grande para o mapeamento das ruas e rodovias em detrimento de outros elementos como transporte público, construções e paisagens, o que vem mudando gradativamente [54].

2.3 Banco de Dados Relacional

O banco de dados relacional foi desenvolvido em 1970 pelo pesquisador da *IBM* Edgar Codd, utilizando o modelo relacional como estrutura de armazenamento das informações. Ainda na década de 70, surgiu na *IBM* uma linguagem de pesquisa declarativa baseada na álgebra relacional chamada SQL (*Structured Query Language*), desenvolvida para dar suporte aos sistemas da empresa que já utilizavam o modelo relacional [42, 46].

O modelo relacional estrutura o banco de dados como um conjunto de tabelas com linhas e colunas, onde cada tabela e cada coluna possui um nome para ajudar na compreensão do significado dos valores presentes em cada uma das linhas. Cada linha é chamada de tupla, o nome designado à coluna é chamado de atributo, a tabela é chamada de relação e os tipos de dados que descrevem os valores é representado pelo domínio de valores possíveis [47].

[8] `https://br.yahoo.com`

2.3.1 Transações

Transação é uma unidade lógica de processamento no SGBD que inclui uma ou mais operações de acesso (leitura e escrita) de maneira atômica, isto é, como um bloco único e indivisível. Ao final da transação, o banco de dados deve ser deixado em um estado válido ou coerente. No modelo relacional, o SGBD deve garantir que as chamadas propriedades ACID sejam seguidas. São elas [47, 32]:

- Atomicidade: Uma transação só é completada quando todas as operações são completadas, caso contrário ocorrerá o retorno a um estado válido anterior à execução da transação (*rollback*);
- Consistência: Uma transação não pode acarretar em inconsistências ou em operações ilegais, caso contrário ocorrerá um *rollback*;
- Isolamento: Todas as transações são independentes e uma não poderá afetar a outra;
- Durabilidade: Quando é realizado um *commit*, as transações realizadas devem persistir no banco de dados, sem que haja perda das mesmas em caso de falha.

2.3.2 Linguagem de Consulta

Como já mencionado anteriormente, a linguagem de consulta declarativa padrão dos bancos de dados relacionais é a SQL (*Structured Query Language*), criada com o intuito de facilitar a extração de informações do banco por parte dos usuários [47].

A linguagem SQL foi projetada para ser abrangente, possuindo instruções para definição de dados, consultas e atualizações. Pode-se agrupar as operações realizáveis no SQL de acordo com o tipo da mesma [80]:

- DML (*Data Manipulation Language*): Linguagem de manipulação de dados, utilizada para permitir consultas, inserções, atualizações e deleções de dados;
- DDL (*Data Description Language*): Linguagem de descrição de dados, utilizada para especificar o esquema de um banco de dados;
- DTL (*Data Transaction Language*): Linguagem de transação de dados, utilizada para controle das transações;
- DCL (*Data Control Language*): Linguagem de controle de dados, utilizada pra controlar as permissões de acesso ao banco de dados.

Por ser um padrão, a migração de dados entre diferentes SGBD relacionais não é uma tarefa difícil, visto que ambos usarão SQL, o que foi um dos principais motivos para o sucesso dos bancos relacionais [47].

Neste trabalho, o SGBD relacional utilizado é o PostgreSQL, o qual é brevemente introduzido na seção seguinte.

2.4 PostgreSQL

O PostgreSQL é um sistema gerenciador de banco de dados do modelo relacional de código aberto. Ele foi concebido com base no código fonte do chamado *Postgres*, um SGBD primitivo sem suporte a linguagem SQL desenvolvido na Universidade de Califórnia Berkeley em 1986 [61].

Com mais de 15 anos de desenvolvimento ativo, o PostgreSQL é um dos SGBD mais avançados disponíveis, possuindo características como [45, 59, 44]:

- Portabilidade: Suporta diferentes plataformas como *MacOS*, *Linux* e *Windows*;
- Replicação: Permite a replicação entre servidores de modo gratuito;
- Extensibilidade: Por ser de código aberto, novas funcionalidades podem ser implementadas e adicionadas ao PostgreSQL por qualquer usuário;
- Segurança: Suporte nativo a SSL e algoritmos de criptografia como MD5, além de extensibilidade para outros, como SHA1;
- Conectividade: Utiliza uma arquitetura *process-per-user*, onde um processo mestre cria processos filhos para cada cliente requisitando conexão ao banco de dados;
- Objeto-relacional: Os dados são tratados por um modelo objeto-relacional, capaz de suportar regras e rotinas complexas, como suporte multi-usuário, suporte concorrente multi-versão, transações, otimização de *querys*, *arrays* e herança;
- API flexível: Possui uma API que permite o suporte ao desenvolvimento em conjunto com as mais variadas linguagens, como *Perl*, *Java*, *Python*, *Ruby*, *C/C++*, dentre diversas outras.
- Capacidade de armazenamento: Suporte eficiente a grandes volumes de informações, o que possibilita tabelas com 32TB, linhas com 1.6TB, campos de 1GB e tamanho total ilimitado para o banco.

Por meio de sua extensibilidade, o PostgreSQL foi o primeiro banco de dados de código aberto a possuir um módulo específico para trabalhar com dados geográficos vetoriais, o chamado PostGIS. Esse módulo foi desenvolvido pela empresa *Refractions Research* sob licença aberta e adiciona suporte a funções, indexação e novos tipos espaciais, o que

permite a transformação do PostgreSQL em um banco de dados geográfico com mais de 1000 funções e operadores para esse fim [83, 23, 21].

Desde seu lançamento, por possuir código aberto, está em constante manutenção pela comunidade. É por esse motivo, também, que o PostGIS se firma como a mais poderosa e popular extensão geoespacial livre, sendo a preferida entre organizações sem fins lucrativos, agências de pesquisa, universidades e agências governamentais ao redor do globo [66].

2.5 Banco de Dados NoSQL

A crescente popularização da internet trouxe consigo a necessidade de um gerenciamento de volumes cada vez maiores de dados que, por sua vez, tornaram-se mais complexos, menos estruturados e mais dinâmicos. Estes fatores, juntamente com a grande quantidade de usuários realizando operações simultâneas de leitura e escrita, levaram a uma dificuldade na adoção dos tradicionais bancos relacionais, seja pela impossibilidade na adequação dos dados a um modelo rígido ou por restrições na escalabilidade necessária para lidar com a enorme quantidade de dados [64, 37].

Desse modo, novas ferramentas foram criadas para lidar com estes desafios, dentre elas os bancos de dados NoSQL. O termo NoSQL, abreviação de *Not only SQL*, refere-se aos cada vez mais populares sistemas de gerenciamento de banco de dados (SGBD) não relacionais, onde não há a necessidade de um modelo de dados fixo e da exclusiva utilização da linguagem SQL para consulta e manipulação dos dados [64, 37, 52].

Por volta do ano 2000, surgiram diversos novos projetos de SGBD alternativos ao modelo relacional. Dentre diversos que surgiram nesta época, pode-se destacar o BigTable da *Google* em 2004, o Dynamo pela *Amazon*, o inicio do MongoDB e o lançamento do Neo4j em 2007 e o começo do Cassandra pelo *Facebook* em 2008 [41, 43, 85, 71].

2.6 Características dos Bancos de Dados NoSQL

Atualmente, não há uma definição amplamente aceita para o que são os NoSQL, porém há algumas características que podem ser identificadas em grande parte destes bancos. São elas [64, 68, 53]:

- Escalabilidade horizontal e vertical;
- Processamento distribuído;
- O modelo de dados não possui um esquema definido, permitindo maior flexibilidade quanto ao tipo dos dados armazenados;

- Grande parte dos NoSQL são *open source*;
- As transações possuem maior flexibilidade em relação às propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), sacrifício necessário para que se possa obter um processamento rápido proveniente da distribuição do sistema;
- Suporta o armazenamento de uma grande quantidade de dados e permite operações simultâneas de leitura e escrita;
- Suporte a outras linguagens de consultas.

Além dessas características, os bancos de dados NoSQL, em sua maioria, são projetados com base no teorema CAP, que diz que, para um sistema de banco de dados distribuídos, só é possível garantir duas das três principais propriedades independentes. As três propriedades são [86, 69]:

- Consistência (*Consistency*): Sempre que novos dados forem escritos no banco, todos os usuários que realizarem operação de leitura neste banco irão receber a versão mais recente dos dados;
- Disponibilidade (*Availability*): Significa que os usuários sempre poderão esperar que cada operação realizada resulte em uma resposta esperada. Alta disponibilidade é alcançada através de um grande número de servidores físicos agindo como um único banco de dados através de compartilhamento e replicação de dados;
- Tolerância à partição (*Partition tolerance*): Significa que a base de dados pode ser lida ou pode receber escrita mesmo que algumas partes dela estejam indisponíveis. Isso é alcançado salvando os dados que eram destinados à uma das partes indisponíveis em uma parte disponível para que, posteriormente, haja a sincronização entre as duas partes.

A todo momento, diversas operações são realizadas em um SGBD podendo, inclusive, ser realizadas de modo concorrente, fazendo-se necessário algum mecanismo para garantir que não ocorram inconsistências no banco de dados. No âmbito relacional, os sistemas gerenciadores de bancos de dados garantem a utilização das propriedades ACID afim de manter a consistência das informações. Deve-se notar, porém, que nem sempre é necessário que se tenha uma consistência total dos dados, como exemplificado no próprio teorema CAP, podendo esta ser sacrificada em partes para dar lugar a uma consistência eventual com uma maior disponibilidade e escalabilidade horizontal do banco de dados [69].

Visando esse cenário, o modelo transacional BASE foi proposto. BASE é uma abreviação para [32]:

- Basicamente disponível (*Basically available*): Todos os dados são distribuídos, mesmo quando há uma falha o sistema continua em funcionamento;
- Estado leve (*Soft-state*): Não há garantia de consistência;
- Eventualmente consistente (*Eventual consistency*): O sistema garante que caso não ocorra novas mudanças nos dados, eventualmente haverá consistência dos dados.

Pode-se dizer que o teorema BASE é mais flexível que o ACID, fornecendo consistência eventual e alto particionamento. Em resumo, é interessante a adoção do modelo BASE quando a aplicação necessita de um sistema distribuído com vários *clusters*, devendo-se privilegiar a capacidade de escalar o sistema e a *performance* sob a consistência dos dados [32, 86, 69].

2.6.1 Tipos e Categoria dos Bancos de Dados NoSQL

De acordo com a classificação em [7], existem atualmente mais de 225 bancos de dados NoSQL, subdivididos em categorias de acordo com o modelo e a estrutura de dados empregado. Dentre outras, pode-se destacar quatro principais categorias: as modelagens orientadas a grafo, documentos, chave-valor e colunas. As três últimas categorias são as mais populares e possuem uma característica em comum em seus modelos de dados: a orientação agregada, em que os dados são trabalhados na forma de unidades, com estruturas mais complexas que um conjunto de tuplas [53, 69, 40].

Bancos de Dados Orientados a Colunas

Esta abordagem emprega uma estrutura de dados distribuída orientada a colunas capaz de acomodar múltiplos atributos por chave. Pode-se dizer que este modelo é o que mais se aproxima do tradicional modelo relacional. Basicamente, a organização do banco se dá do seguinte modo [64, 32]:

- Colunas: Representa uma unidade do dado identificado por uma chave e um valor;
- Super Colunas: Agrupa as informações da coluna;
- Família de colunas: Conjunto de dados estruturados similar à uma tabela de um banco relacional, constituida por uma variedade de super colunas.

O acesso aos dados é realizado informando, respectivamente, a família de colunas seguido da chave e por fim a coluna desejada. Pode-se adicionar novas colunas sempre que necessário [32].

Chave
1
2
3
4
5

Primeiro nome
João
José
Maria
Matheus
Pedro

Sobrenome
Silva
Silveira
Parreira
Fernandes
Cunha

Cep
88015-396
88016-234
88015-205
88012-222
88025-345

Idade
20
25
30
22
27

Figura 2.5: Exemplo de registros em um banco de dados orientado a colunas. Cada coluna é armazenada em um arquivo diferente. Adaptado de [30].

A Figura 2.5 apresenta, de forma simplificada, registros em um banco de dados orientado a colunas.

Bancos orientados a coluna trabalham muito bem com armazenamento e processamento de larga escala distribuído de dados, além de análise de dados estatísticos. Entre suas características, pode-se destacar o particionamento, indexação e alta compressão dos dados apresentando, porém, uma baixa velocidade de inserção [31].

Comercialmente, o *Google* foi pioneiro no emprego desse modelo através do chamado Bigtable [5], que veio a inspirar o desenvolvimento de diversas outras soluções como o Cassandra [27] pelo *Facebook*, que é utilizado também pela rede social *Twitter* [52, 32].

Bancos de Dados Orientados a Chave-Valor

No modelo chave-valor, os registros são compostos por um identificador alfanumérico único, chamado de chave, que possui um valor associado em uma grande tabela, chamada de tabela *Hash*. Esses valores nada mais são que as informações contidas nos dados em um formato primitivo, como *strings*, inteiros e vetores, o que elimina a necessidade de um modelo de dados rígido e fixo [64, 52]. A Figura 2.6 exemplifica o armazenamento de um objeto em um banco de dados chave-valor.

Este modelo é considerado o mais simples, sendo a manipulação de seus dados dada por operações elementares de inserção, deleção, atualização e busca utilizando o campo chave, não sendo suportado chaves secundárias, além dos resultados estarem limitados a combinações exatas [52].

Devido a sua simplicidade, esta abordagem torna-se ideal para aplicações onde é necessário uma grande escalabilidade e velocidade na recuperação de valores no banco por uma tarefa, tal como o gerenciamento de perfis ou a recuperação de nomes de produtos. Como exemplos do emprego dessa abordagem na indústria, pode-se destacar o uso do DynamoDB [2] pela *Amazon* em seus carrinhos de compras e o emprego do *Project Voldemort* [22], um banco *Open Source* baseado no próprio DynamoDB, pela rede social *LinkedIn* [64].

Chave	Valor
1	Montadora: Nissan Modelo: Pathfinder Cor: Verde Ano: 2003
2	Montadora: Nissan Modelo: Pathfinder Cor: Azul Cor: Verde Ano: 2005 Transmissão: Auto

Figura 2.6: Armazenamento chave-valor em um banco de dados NoSQL, adaptado de [64].

Bancos de Dados Orientados a Grafos

De acordo com Van Bruggen em [84], grafos são uma representação abstrata de duas ou mais entidades que se conectam ou se relacionam de alguma forma. Grafos são compostos de 3 elementos [56]:

- Nós ou vértices: Utilizados geralmente para representar objetos independentes;
- Arestas ou ligações: Utilizadas para estabelecer relações entre os diferentes nós;
- Propriedades: Descrevem os atributos de nós e arestas.

Grafos recebem esse nome pois são estruturas que podem ser representados graficamente, sendo que esta representação é o que torna possível o entendimento de muitas das suas propriedades. Não existe um modo único de desenhar um grafo e, conforme novas propriedades vão sendo adicionadas, novos tipos de grafos com diferentes significados vão surgindo. Entre outros tipos, pode-se destacar [75]:

- Grafos Simples: Quando dois vértices são conectados entre si por apenas uma aresta, não permitindo laços;
- Multigrafos: Quando dois nós possuem mais de uma aresta ligando um ao outro;
- Grafos Dirigidos: Quando o relacionamento entre dois nós se dá exclusivamente em uma direção;
- Grafos Valorados: Quando uma ou mais arestas possuem um peso associado.

Conforme é possível observar na Figura 2.7, estes diferentes tipos de grafos podem ser utilizados em conjunto. Em 1, pode-se observar um multigrafo; em 2 um grafo valorado; em 3 um grafo dirigido e em 4 pode-se observar um grafo simples [75].

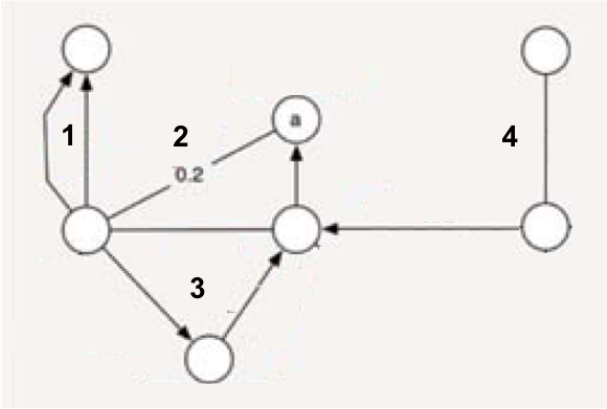

Figura 2.7: Exemplo de tipos de grafos. Adaptado de [75].

Os bancos de dados orientados a grafos são projetados tendo como base a teoria dos grafos, utilizando seus componentes e propriedades para o armazenamento e análise de dados altamente conectados. Desse modo, é possível traçar um paralelo entre os componentes originais de um grafo com sua representação nos bancos orientados a grafo [60]:

- Nós: Serão os registros presentes no banco;
- Arestas: São armazenadas no banco como relações entre os registros;
- Propriedade: Especificam características tanto dos registros como do relacionamento entre estes.

Diferentemente da maioria dos bancos NoSQL, este modelo possui suporte nativo a transações ACID, de modo que operações como inserção e deleção são feitas de forma segura e consistente. A utilização deste modelo é recomendada quando o principal foco do estudo é a relação entre os objetos, como por exemplo em uma rede social ou em sistemas com objetivo de detectar fraudes. Isso ocorre pois, diferentemente das outras abordagens relacionais e não relacionais, as relações entre os objetos são armazenadas no banco de forma explícita. Além disso, o fato destes bancos utilizarem a teoria dos grafos como base faz com que seja possível manipular os dados com base nas estruturas e propriedades presentes em um grafo, tornando possível operações como caminhos, vizinhança ou conectividade [60].

Embora os bancos de dados baseados em grafos possam ser utilizados de maneira distribuída em vários servidores distintos conforme outros bancos NoSQL, deve-se observar

que, devido a alta conectividade entre os objetos presentes no banco, pode-se encontrar certas dificuldades como a complexidade de consultas envolvendo múltiplos nós ou até mesmo baixo desempenho [58].

Atualmente, de acordo com o *DB-Engines*, os principais bancos de dados em grafos ou que suportam grafos são o Neo4j[9], com aproximadamente 43% de popularidade, e o Cosmos DB[10], com 22% de popularidade. A Figura 2.8 mostra popularidade dos principais bancos orientados a grafos.

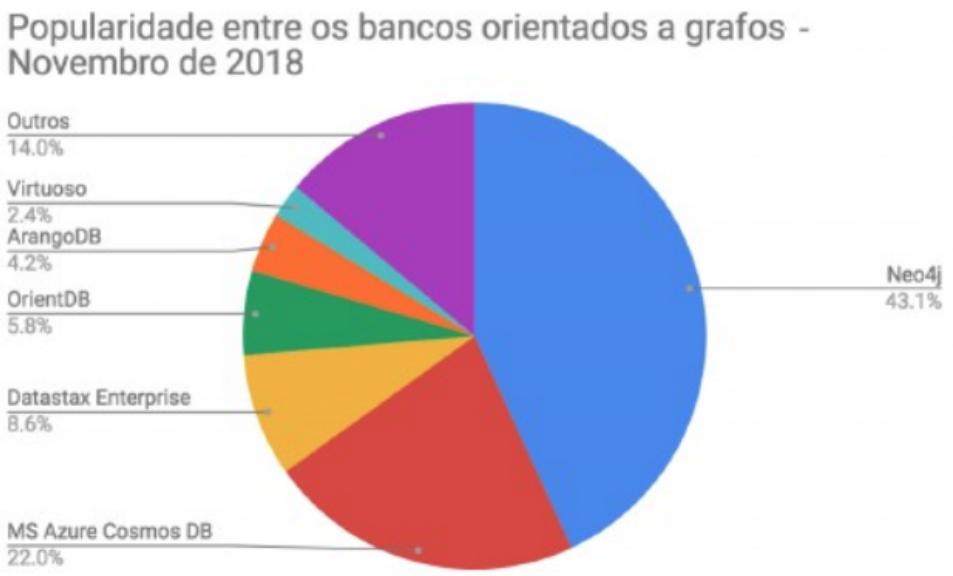

Figura 2.8: Distribuição da popularidade dos bancos em grafo. A porcentagem foi calculada com base na pontuação referente a novembro de 2018, disponível em [9].

[9] `https://neo4j.com/`
[10] `https://azure.microsoft.com/pt-br/services/cosmos-db/`

Bancos de Dados Orientados a Documentos

Os bancos de dados orientados a documentos armazenam e organizam os dados como coleções de documentos. Estes documentos podem ser arquivos *XML*, *JSON* ou qualquer outro formato de documentos, sendo eles estruturados, semiestruturados ou não estruturados. Isso, juntamente com o fato não ser necessário a definição de um esquema rígido e da possibilidade de adicionar um novo arquivo sem que se tenha conhecimento prévio da estrutura do mesmo, fazem com que esse modelo seja considerado o mais flexível entre os bancos não relacionais. Além disso, a abordagem orientada a documentos permite que se faça consultas sobre qualquer valor ou chave presente no banco, ao contrário dos bancos de dados orientados a chave-valor, onde só é possível realizar operações de busca sobre as chaves [53, 32, 52, 56].

Traçando um paralelo com os bancos relacionais, pode-se dizer que uma coleção de arquivos de um banco de dados orientado a arquivos equivale a uma tabela, enquanto um documento da coleção equivale a um registro da tabela. Um documento pode ser referenciado a outro, porém sem a necessidade da utilização de chaves estrangeiras e campos equivalentes, como ocorre no modelo relacional. Contudo, é importante notar que, por mais que haja semelhanças entre os dois modelos, os bancos de dados orientados a documentos não carregam consigo a rigidez presente no modelo relacional, o que permite ao documento, inclusive, apresentar uma estrutura mais complexa que um registo de uma tabela [56].

Quando armazenados, os documentos são codificados em um formato padrão de troca de dados, tal como *XML*, *JSON* ou *BSON* [64].

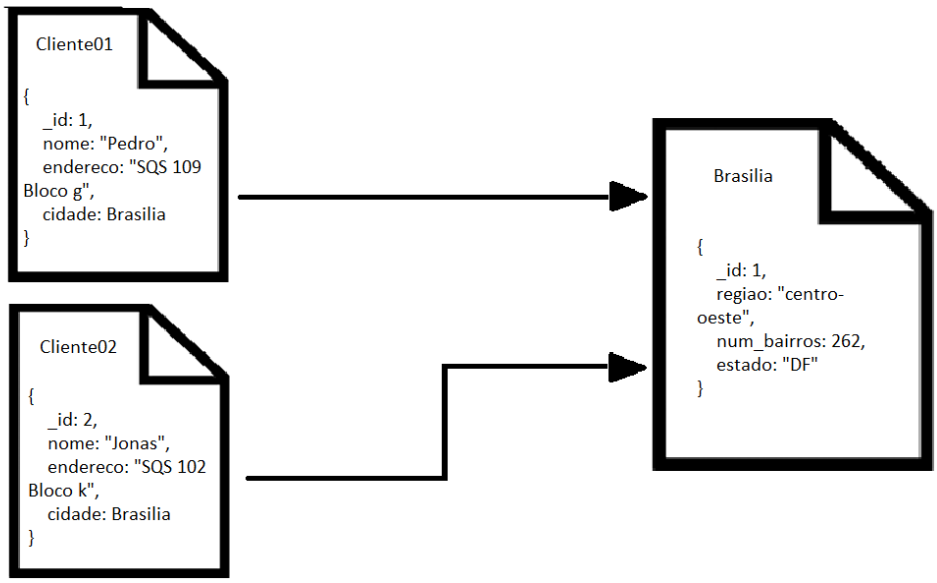

Figura 2.9: Exemplo de objeto armazenado em um banco de dados orientado a objetos, utilizando *JSON*.

Os bancos de dados baseados em documentos são projetados visando a aplicação em ambientes onde é necessário o armazenamento e a manipulação de grandes volumes de dados. Devido a sua flexibilidade, é interessante o uso de bancos orientados a documentos para armazenamento dados esparsos ou semiestruturados [64].

De acordo com [7], os bancos orientados a documentos mais populares atualmente são o MongoDB, com 65.1% de popularidade, e o *Amazon* DynamoDB, com 9.5% de popularidade. A Figura 2.10 mostra popularidade dos principais bancos orientados a documento.

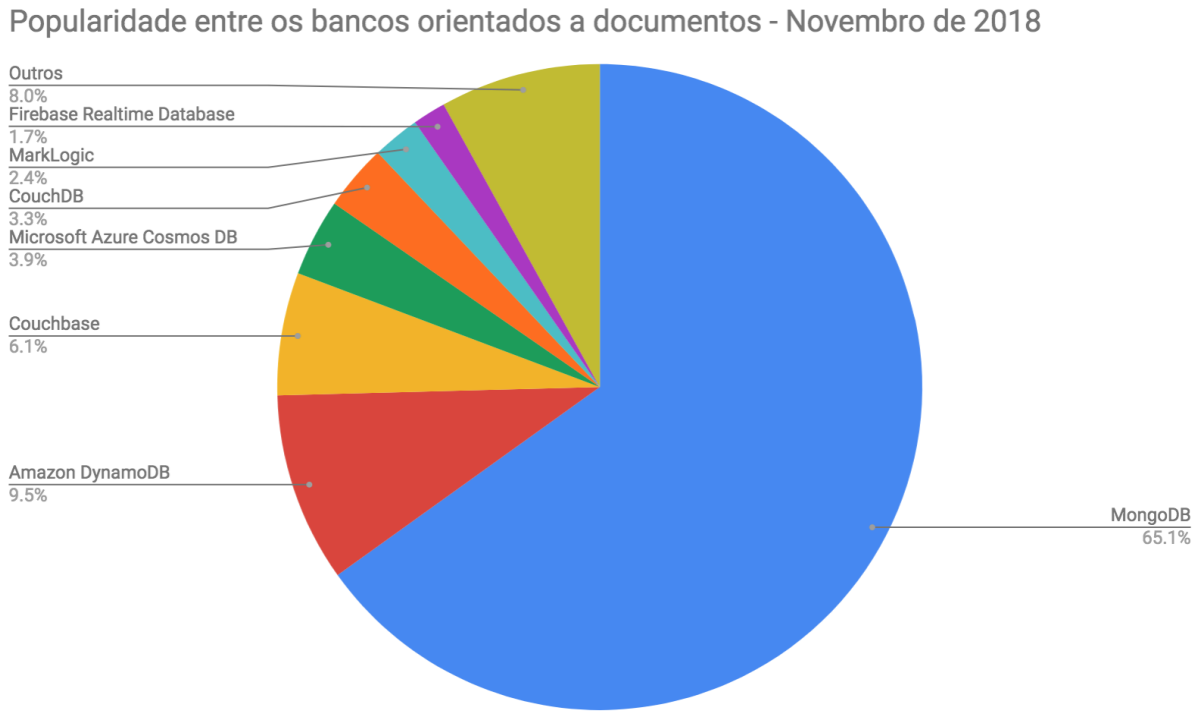

Figura 2.10: Distribuição da popularidade dos bancos orientados a documentos. A porcentagem foi calculada com base na pontuação referente a novembro de 2018, disponível em [9].

2.6.2 Popularidade dos Bancos NoSQL

A Figura 2.11 mostra o gráfico de popularidade e adoção tanto dos SGBD relacionais como dos NoSQL em novembro de 2018 de acordo com [9]. Em um primeiro momento, percebe-se que os bancos relacionais de fato seguem dominando o mercado, com 81%, enquanto os NoSQL abordados nesta monografia somam aproximadamente 14.1%.

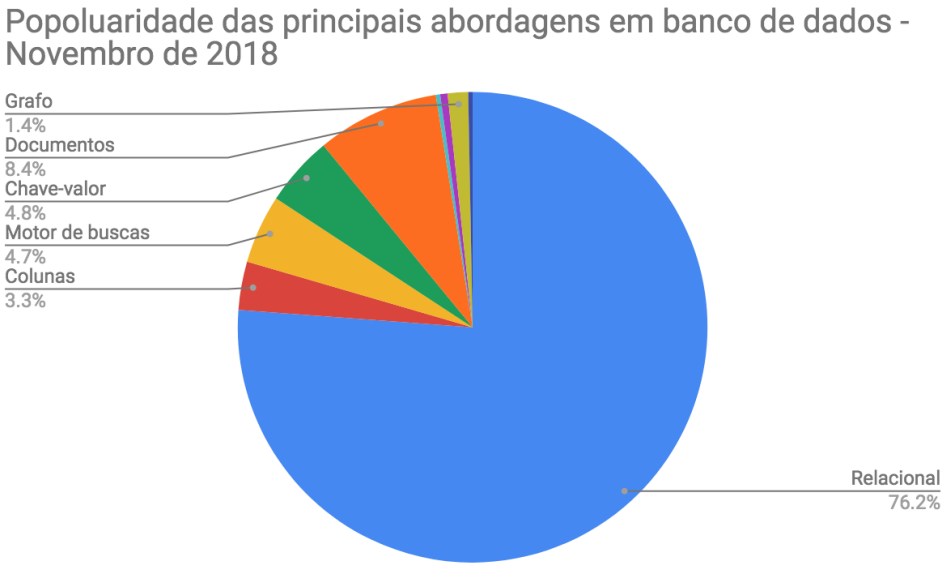

Figura 2.11: Gráfico que mostra a popularidade das principais abordagens em bancos de dados, com dados de Novembro de 2018. Adaptado de [9].

Outro gráfico disponível em [8], porém, ilustra uma tendência interessante. A Figura 2.12 mostra a mudança na popularidade das principais abordagens de SGBD utilizadas de Janeiro de 2013 até Novembro de 2018. Por esse gráfico, é possível notar uma forte tendência ao crescimento na popularidade dos NoSQL, principalmente os orientados a grafo, abordados nesta monografia, enquanto o modelo relacional se mantém estável e até mesmo com uma leve queda.

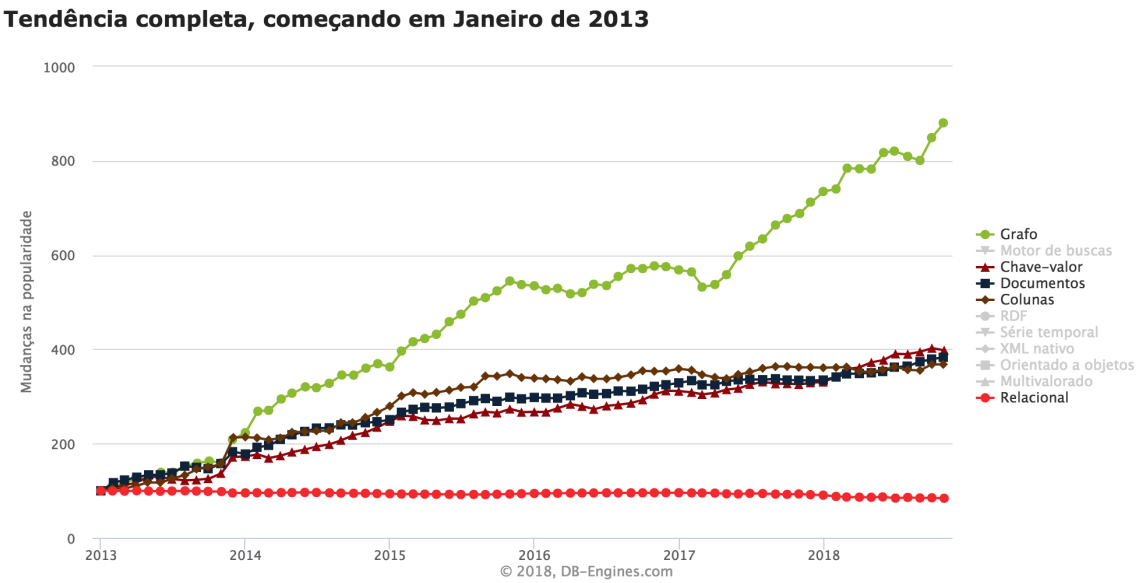

Figura 2.12: Gráfico referente à mudança de popularidade das abordagens de SGBDs de Junho de 2013 até Novembro de 2018. Adaptado de [8].

2.7 MongoDB

O MongoDB é um banco de dados de código aberto orientado a documentos que proporciona alta performance, grande disponibilidade e escalabilidade automática. O projeto foi fundado em 2007 pela mesma equipe responsável pela companhia de propaganda digital *DoubleClick* (hoje, propriedade da *Google*), empresa que, na época, servia cerca de 400 mil propagandas por segundo na *internet*, o que ocasionava eventuais problemas de escalabilidade e lentidão. O MongoDB foi a solução desenvolvida pela equipe do *DoubleClick* para solucionar esses problemas [13, 62].

O projeto foi desenvolvido em *C++* inspirado no *BigTable*, da *Google*, e nasceu da experiência pessoal em desenvolvimento de sistemas robustos, de larga escala e de grande disponibilidade por parte de seus desenvolvedores. A filosofia do projeto é de combinar as capacidades críticas dos tradicionais banco de dados relacionais com as inovações presentes nas tecnologias NoSQL [68, 62].

O MongoDB não foi concebido com o intuito de substituir os bancos relacionais, mas sim utilizá-los como ponto de partida para suprir as necessidades das aplicações modernas. Ele alia funcionalidades já existentes nos tradicionais bancos relacionais, como a presença de indexação, linguagem de consulta e forte consistência, com novos elementos não presentes na abordagem relacional, como modelo de dados flexível, escalabilidade e clusterização [62, 63].

O *design* multimodelo adotado pelo MongoDB reduz significativamente a complexidade de desenvolvimento e operação, quando comparado à utilização de mais de um SGBD

para suprir funcionalidades distintas para a aplicação. Isso ocorre pois o usuário pode utilizar a linguagem de consulta, o modelo de dados, a escalabilidade e as ferramentas operacionais presentes no MongoDB em diversas áreas da aplicação, ajustando o modelo de armazenamento de dados conforme a necessidade de cada área. O fluxo de dados entre os diversos modelos de armazenamento é gerenciado automaticamente pelo MongoDB usando replicações nativas [68, 62].

2.7.1 Modelo de Dados

O armazenamento de dados no MongoDB se dá por meio de um formato de representação binária chamado *BSON*[11], ou *Binary JSON*. O formato *BSON* estende o amplamente adotado JSON[12], incluindo tipos adicionais como inteiros, *long*, *date*, *floating point* e decimal de 128 *bits*. Os documentos *BSON* contém um ou mais campos de um determinado tipo de dado, e cada um desses campos contém um valor do respectivo tipo de dado, podendo ser até mesmo *arrays*, binários e subdocumentos. A Figura 2.13 demonstra a estrutura de um documento *JSON* [52, 68].

```
{
    '_id' : 1,
    'Cidade' : { 'nome' : 'Brasilia' },
    'habitantes' : [
        {
            'nome' : 'Joao'
        }, {
            'nome' : 'Maria'
        }, {
            'nome' : 'Marcelo'
        }
    ]
}
```

Figura 2.13: Exemplo ilustrando como seria uma representação em formato *JSON*.

[11] http://bsonspec.org/
[12] http://www.json.org/

Cada documento possui um campo *_id*, que é análogo às chaves primárias dos bancos relacionais, servindo como identificador único de um documento dentro de uma coleção. A Figura 2.14 mostra uma comparação entre as coleções e o modelo relacional [52, 40].

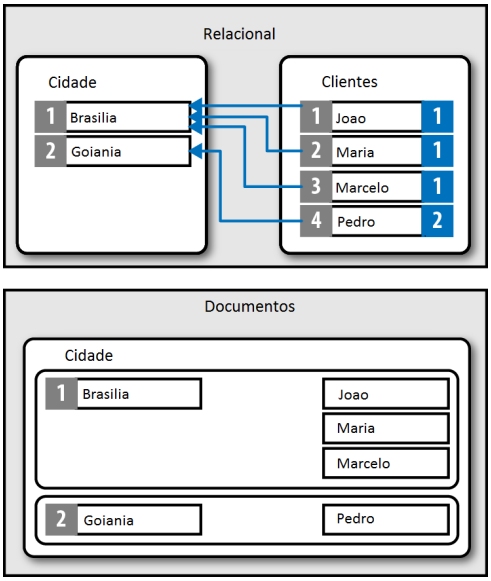

Figura 2.14: Exemplo mostrando como seria uma representação da organização dos registros em modelo relacional e em documentos. Adaptado de [48].

2.7.2 Consultas e Manipulação de Dados

O MongoDB possui *drivers* nativos para diversas linguagens e *frameworks* de programação, onde cada *driver* é projetado de modo idiomático para a linguagem utilizada. A principal diferença em relação às consultas ao banco entre o MongoDB e os bancos relacionais é que o modelo de consulta do MongoDB é implementado como métodos ou funções da *API* da linguagem de programação sendo utilizada pelo usuário, enquanto os bancos relacionais fazem uso da linguagem SQL. A *API* envia a consulta ao MongoDB como um objeto *BSON* [62].

O MongoDB não se limita apenas a operações envolvendo um par chave-valor, o que permite a realização de diversos tipos de operações de consulta. As consultas, por sua vez, podem retornar desde um documento até uma agregação ou transformação de diferentes documentos. Dentre os diversos tipos de consultas, destacam-se [62]:

- Consultas chave-valor: Retorna resultados com base em qualquer campo de qualquer documento presente no banco;
- Consultas de alcance: Retorna resultados baseados em valores definidos com desigualdades, tais como maior que ou menor que, dentre outras;

- Consultas Geoespaciais: Retorna resultados baseados em proximidade, interseção e inclusão especificados para um ponto, linha ou polígono;
- Buscas: As consultas retornam resultados em ordem de relevância e em grupos determinados baseados em argumentos textuais usando operadores booleanos (*and*, *not*, *or*), operações de agrupamento e operações de contagem de resultados;
- Agregação: As consultas retornam agregações e transformações dos valores retornados, similarmente ao operador *group by* da linguagem SQL;
- Operações *Join*: Documentos de coleções separadas podem ser combinados através de uma operação *left outer join*;
- Consultas *MapReduce*: Executa processamento complexo de dados expressos em *JavaScript* através de toda a base de dados.

A manipulação dos documentos no MongoDB se dá por meio de operações de busca, inserção, modificação e deleção. Um documento pode ser completamente substituído, preservando seu identificador, ou ter apenas modificações em certos campos, sendo a operação de modificação de um único documento atômica, o que não é verdade para modificações em mais de um documento [52, 40].

Além disso, o MongoDB otimiza a consulta de forma automática para fazer com que as mesmas sejam processadas da maneira mais eficiente possível. Esse módulo otimizador seleciona os melhores índices (ou *indexes*) possíveis através de uma rodagem periódica de planos de consulta e posterior análise de tempo de resposta das operações. Os resultados são armazenados em cache e atualizados periodicamente, sendo possível aos desenvolvedores visualizar e otimizar os planos através do método *Explain* e filtros de *index* [62].

2.7.3 Suporte a Dados Espaciais

O MongoDB conta com suporte nativo a dados espaciais através da utilização de dois tipos de indexação: *2d* e *2dsphere*. O primeiro é utilizado para operar com dados em um plano, enquanto o segundo tem como função a realização de operações com geometrias em uma esfera [77].

Há a presença de algumas funções espaciais implementadas, embora não possua uma lista tão extensa quanto o PostgreSQL. São elas [12]:

- *geoIntersects*: Seleciona polígonos que possuem interseção com uma geometria em *GeoJSON*. Só há suporte para o índice esférico, ou seja, *o 2dsphere*;
- *geoWithin*: Seleciona geometrias dentro de um limite definido por um geometria em *GeoJSON*. Há suporte para ambos os índices espaciais;

- *near*: Retorna objetos geoespaciais na proximidade de um dado ponto. Requer a utilização de qualquer um dos dois índices espaciais;
- *nearSphere*: Retorna objetos geoespaciais na proximidade de um ponto no interior de uma esfera. Requer a utilização de qualquer um dos dois índices espaciais.

Para a inserção de dados provenientes do OpenStreetMap, porém, é necessário o tratamento prévio dos dados ou a utilização de *scripts* de inserção desenvolvidos pela comunidade de usuários. O OpenStreetMap conta, em sua *wiki* oficial, com uma página dedicada a fornecer informações sobre a utilização do projeto em conjunto com diversos bancos de dados e *APIs* de acesso. Nesta página, há uma lista de alguns *scripts* de inserção de dados provenientes do OpenStreetMap no MongoDB [6].

O MongOSM é um conjunto de *scripts* na linguagem *Python* desenvolvido por Ian Dees e disponibilizado sob licença aberta no *GitHub*. O conjunto de *scripts* realiza, entre outras funções, a conversão e inserção dos dados provenientes do OpenStreetMap, no formato OSM, no MongoDB. Os *scripts* são de fácil utilização e são multiplataforma [15].

2.8 Neo4j

O Neo4j é um banco de dados de código aberto desenvolvido em *Java* pela empresa *Neo Technology*, hoje conhecida como *Neo4j Inc*, e construído nativamente sob os fundamentos da teoria dos grafos, projetado para ser escalável e para tratar não somente os dados, mas também as suas relações. O projeto teve início em 2000, quando os desenvolvedores se depararam com problemas de performance ao utilizar os tradicionais bancos relacionais, tendo sua primeira versão completada em 2002. Atualmente, oferece suporte oficial a várias linguagens populares como *Python* e *Ruby* [82, 76].

O projeto foi desenvolvido para otimizar o armazenamento e gerenciamento rápido de nós e relacionamentos. Nos bancos relacionais são realizadas operações de relacionamento entre tabelas utilizando *Joins*, que possuem uma degradação exponencial de performance em relação ao número de relacionamentos utilizados na consulta, ao passo que, no Neo4j, operações semelhantes são realizadas como sendo navegação de um nó até outro, o que consiste numa operação com performance linear [82].

2.8.1 Modelo de Dados

No Neo4j, os dados são salvos em grafos compostos por nós, relacionamentos e propriedades. O nó é a unidade fundamental formadora do grafo, e são frequentemente utilizados para representar entidades. Os nós podem conter relacionamentos e propriedades, além de ser capaz de possuir um ou mais rótulos que o identificam [51, 76].

Nós que possuem o mesmo rótulo são agrupados como pertencentes ao mesmo conjunto, ou subgrafo. Esse agrupamento de nós por rótulos faz com que seja possível trabalhar apenas com um conjunto reduzido de nós, e não com o grafo inteiro, o que resulta em consultas mais fáceis de se escrever, além de mais eficientes na execução [51, 82].

Os relacionamentos entre nós são a parte chave dos bancos orientados a grafo, eles permitem que seja realizado buscas em dados relacionados. Os relacionamentos conectam dois nós e, necessariamente, possuem um nó inicial e um nó final. Assim como nos nós, os relacionamentos podem possuir propriedades [51, 82].

Além disso, os relacionamentos organizam os nós em estruturas arbitrárias, o que possibilita que o grafo tenha um aspecto de lista, árvore, entidade composta, entre outros. É importante ressaltar que é possível combinar diferentes estruturas para formar uma estrutura maior, mais complexa e ainda mais conectada [82].

As propriedades, por sua vez, são valores que possuem um identificador *string*, ou seja, um nome. As propriedades podem ser valores numéricos, *strings*, valores booleanos ou uma lista de qualquer outro tipo de valor [51].

O Neo4j não torna obrigatório o uso de um esquema, porém a sua adoção normalmente resulta em melhores modelagens e ganho de performance [82].

2.8.2 Consultas e Manipulação de Dados

O Neo4j possui uma linguagem de consulta e manipulação de dados própria, chamada de *Cypher*. O *Cypher* é uma linguagem de consulta declarativa e puramente textual e é, de acordo com seus desenvolvedores, uma linguagem humana, otimizada e de fácil entendimento. Os seus construtores são baseados na língua inglesa e numa iconografia simples, o que contribui para tornar as consultas ainda mais autoexplicativas. Diferentemente das linguagens imperativas de consultas existentes, o *Cypher* enfatiza o que retornar do grafo, e não como retornar, permitindo que seja declarado o padrão que se deseja obter da consulta sem que seja necessário especificar para o banco [84].

A linguagem foi inspirada em diversas outras linguagens pré-existentes, como a linguagem SQL para sua estrutura e algumas palavras reservadas (*Where, Order by*), e *Haskell* e *Python* para a semântica [62].

Dentre as diversas cláusulas possíveis na linguagem *Cypher*, destacam-se [74]:

- *Start*: Ponto inicial do grafo, recuperado via índices ou identificadores;
- *Match*: Padrão de combinação que se deseja retornar. É o jeito mais comum de recuperar dados do grafo;
- *Where*: Parte de outras cláusulas para filtrar ou restringir o que se deseja retornar;

- *Return*: O que retornar;
- *Create*: Cláusula de criação de nós e relacionamentos;
- *Remove*: Cláusula de remoção de nós e relacionamentos;
- *Set*: Definidor de valores para as propriedades;
- *Foreach*: Realiza as modificações em uma lista, iterando elemento a elemento;
- *With*: Divisor de consultas em múltiplas partes.

Na Figura 2.15 é possível observar um grafo e sua descrição na linguagem *Cypher*. Cada nó do grafo é um objeto do tipo pessoa, onde seu nome corresponde ao rótulo do nó. Cada relacionamento é do tipo "Conhece", e significa que a pessoa no nó de início conhece a pessoa no nó de destino. Ao observar a representação em linguagem *Cypher*, nota-se que a sintaxe é praticamente uma tentativa de desenhar o grafo em si através de setas, semelhantes a *ASCII art*.

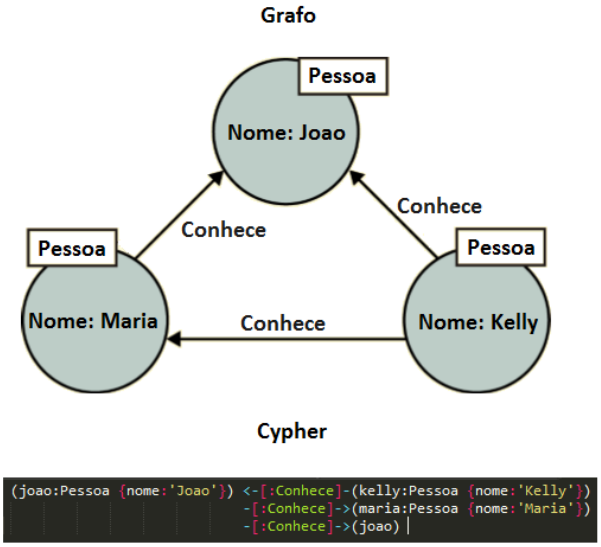

Figura 2.15: Um grafo e sua representação em linguagem *Cypher*. Adaptado de [74].

Por se tratar de um banco de dados construído utilizando grafo como principal estrutura, o Neo4j já implementa nativamente alguns algoritmos de busca em grafos [55]:

- Caminho mínimo;
- Todos os caminhos;
- Todos os caminhos simples;
- *Dijkstra*;

- A*.

Além disso, o Neo4j suporta o chamado *Traversal*. *Traversal* é como se realiza uma consulta num grafo, navegando de um primeiro nó para seus relacionados seguindo regras. O objetivo do *Traversal* é responder a perguntas do tipo "Que amigo dos meus amigos que ainda não é meu amigo", ou "Qual música eu gosto que meus amigos não gostam", ou seja, identificar padrões dentro de um grafo [82].

Por fim, é importante ressaltar que, diferentemente de grande parte dos bancos não relacionais, o Neo4j possui suporte a transações ACID [60].

A linguagem *Cypher* permite a criação de *indexes* sobre propriedades de nós que possuem um certo rótulo. A criação desse resulta em um ganho considerável de *performance* na procura de nós no banco em detrimento de espaço em disco e velocidade de escrita [82].

Uma vez que o *index* é criado, o banco se encarrega de mantê-lo sempre atualizado à medida que ocorre o crescimento do grafo, de modo que todas as operações que busquem por nós indexados tenham um grande aumento de *performance* [82].

O Neo4j possui disponibilidade eventual, o que significa que a criação de um *index* retorna resultado imediato, porém o mesmo não fica disponível de imediato para consultas. Isso acontece pois, em *background*, está ocorrendo a população desse no banco [82].

2.8.3 Suporte a Dados Espaciais

Em relação a dados espaciais, há duas formas de suporte por parte dos SGBD: nativa e por meio de extensões. Até a versão 3.2.12, de maio de 2018, o Neo4j não possuía nenhum suporte a funcionalidades espaciais de forma nativa. Dessa versão em diante, foram implementadas algumas funções nativamente, são elas [1]:

- *distance()* : Função que retorna a distância geodésica entre dois pontos utilizando o mesmo sistema de coordenadas referenciais, seja ele em duas ou três dimensões;
- *points()* : Função que retorna as coordenadas geográficas de um dado objeto. As coordenadas podem ser retornadas em duas dimensões, como um par ordenado longitude e latitude, ou em três dimensões, com coordenadas longitude x, latitude y e altura z.

Este trabalho, porém, foi realizado na versão 3.2.6, ou seja, em uma versão anterior à citada acima. Desse modo, o suporte a dados espaciais se dava exclusivamente por meio de uma extensão espacial chamada *Neo4j Spatial* [17]. O *Neo4j Spatial* é uma biblioteca de utilitários desenvolvida com o intuito de permitir e facilitar a utilização do Neo4j em

conjunto com dados geoespaciais. Trata-se de uma extensão de código aberto desenvolvida inicialmente em linguagem *Java* por engenheiros de *software* da própria *Neo4j Inc.* Atualmente, a extensão é mantida pela comunidade de desenvolvedores que a utiliza [29]. O *Neo4j Spatial* apresenta funcionalidades mais avançadas que as implementadas de forma nativa, contendo classes *Java* específicas para importação de arquivos com a extensão osm, particular do OpenStreetMap, e possibilidade de se trabalhar com os tipos geométricos mais comuns. Além disso, há um total de 11 funções espaciais implementadas, dentre as quais pode-se destacar [18]:

- *Contains*: Verifica se uma segunda geometria é totalmente contida por uma primeira geometria;
- *Intersect*: Retorna os pontos ou partes de duas geometrias que possuem uma interseção em comum;
- *Within*: Verifica se uma primeira geometria é totalmente contida por uma segunda geometria. É o oposto da operação *Contains*;
- *Within Distance*: Verifica se há pontos ou polígonos num dado raio de distância, a partir de um ponto definido pelo usuário.

Por fim, o *Neo4j Spatial* possui um fórum de dúvidas e problemas no seu repositório oficial no *Github* e um manual oficial que são a principal fonte de informação acerca da utilização e do funcionamento da biblioteca.

2.9 Trabalhos Relacionados

A literatura conta com artigos e teses referentes à utilização de bancos não relacionais em conjunto com dados geográficos. Em Anderson *et al* [35], há a implementação de um *framework* para analisar o comportamento das edições colaborativas de usuários do OpenStreetMap em situações de crises naturais ou humanitárias, utilizando o MongoDB como banco de dados. Esse trabalho, porém não realiza testes de desempenho em comparação com outros SGBD.

Também utilizando o MongoDB e o OpenStreetMap, Amat *et al* [33] implementaram um cenário de cidade inteligente mapeando, inclusive, ambientes internos da U-tad, *University Center for Technology and Digital Art*, em Madrid. Novamente, o intuito do trabalho não foi de realizar comparações acerca do desempenho do MongoDB.

Ainda com dados do OpenStreetMap, Amirian *et al* [34] utilizaram bancos orientados a grafos para desenvolver uma aplicação para navegação na *Maynooth University*, utilizando

interfaces textuais, cartográficas e imagens com realidade aumentada para achar pontos de interesse.

Utilizando dados espaciais provenientes do Governo Brasileiro, Roberto *et al* [73] realizou uma análise comparativa entre SGBD orientados a grafos, no caso o Neo4j, e SGBD relacionais, o PostgreSQL em conjunto com a extensão PostGIS. Outra análise comparativa entre SGBD orientados a grafos foi realizada por Martinez e Ariel [57], onde foram utilizados dados da plataforma *openflights*, com informações sobre linhas aéreas e rotas entre aeroportos. Dessa vez, porém, os SGBD comparados foram o Neo4j e o ArangoDB.

Mais próximo a esta monografia, Schmid *et al* [78] compararam o desempenho do MongoDB e do PostgreSQL utilizando três amostras de dados do OpenStreetMap, referentes à região baixa da Bavária, a Bavária completa e a Alemanha. Foram, inclusive, realizadas consultas *within*. Não foram, porém, realizado testes para as demais consultas espaciais. Além disso, a maior das amostras de dados possui um tamanho aproximado de 2.1GB, o que ainda é considerado uma amostra de dados pequena em se tratando de dados espaciais de um país completo.

Baas [36] também realizou uma comparação de desempenho entre um SGBD não relacional, o Neo4j, e um relacional, o PostgreSQL. Nessa tese de mestrado, foram utilizadas três amostras de dados, Medemblik, um município na região norte da Holanda, Amsterdã e a região norte da Holanda propriamente dita. O autor, porém, realizou mudanças nas rotinas de inserção do Neo4j Spatial, o que altera os resultados dos testes.

Diferentemente dos trabalhos citados anteriormente, a contribuição da presente monografia reside nos testes de desempenho de três SGBD, o MongoDB, o Neo4j e o PostgreSQL, utilizando as mesmas amostras de dados. Os trabalhos anteriores apenas utilizavam os SGBD para aplicações ou realizavam testes de desempenho de um representante do modelo relacional com um representante do modelo não relacional. Desse modo, o presente trabalho busca demonstrar como se comportam os três SGBD quando submetidos a condições iguais de operação com amostras de tamanho crescente, iniciando com aproximadamente 50MB até 8GB. Além de inserções de dados, serão realizadas três tipos de consultas espaciais, número superior ao encontrado na literatura até então.

Capítulo 3

Desenvolvimento e Análise de Resultados

Este capítulo tem como objetivo apresentar o processo de desenvolvimento do trabalho, detalhando a metodologia utilizada, ferramentas necessárias, observações e resultados experimentais.

3.1 Ambiente Computacional

Para o desenvolvimento deste trabalho, foi utilizado um *Macbook Pro* com as seguintes especificações:

- Processador *Intel Core i7-4870HQ @ 2.5GHz*;
- *16GB RAM DDR3L @ 1600MHz*;
- *SSD PCIe 500GB*;
- *macOS Sierra version 10.12.6*;
- *MongoDB Community v3.4.6*;
- *PostgreSQL v10*;
- *Neo4j Community v3.2.6*.

3.2 Metodologia

O desenvolvimento desse trabalho se deu basicamente em três etapas: coleta, inserção e manipulação dos dados, conforme a estrutura:

1. Coleta dos dados;
2. Inserção dos dados:
    - Inserções no PostgreSQL;
    - Inserções no MongoDB;
    - Inserções no Neo4j.
3. Consultas espaciais:
    - Consultas *Within*;
    - Consultas *Intersects*;
    - Consultas *Near*.

A primeira etapa, mais simples, se deu unicamente com a aquisição dos dados a serem utilizados nas próximas análises. A segunda etapa consistiu na realização da importação dos dados coletados para os SGBD analisados, tomando medições de desempenho acerca do tempo de inserção para cada SGBD. A última etapa consistiu em realizar consultas geográficas equivalentes nas amostras de dados em cada um dos SGBD, também tomando medições de desempenho.

As etapas são explicadas de modo mais detalhado nas subseções que se seguem.

3.2.1 Coleta dos Dados

No início do projeto, esperava-se utilizar diferentes amostras de dados com aumento gradual de tamanho até atingir o arquivo conhecido como *planet.osm*, ou seja, o conjunto do globo terrestre inteiro disponibilizado pelo OpenStreetMap. Este arquivo pode ser localizado na página de *download* [1] do OpenStreetMap, e, na época de realização dos primeiros testes, consistia em um arquivo disponibilizado nos formatos PBF ou OSM comprimido, com respectivamente 38Gb e 56Gb. Ao ser descompactado, porém, o arquivo crescia 722Gb, impraticável para o hardware disponível, impossibilitado a utilização do *dataset* completo para este experimento.

Nesse contexto, recorreu-se ao site *Geofrabrik*[10], um repositório de dados provenientes do OpenStreetMap que disponibiliza amostras dos dados separados por países, continentes e, em alguns casos, estados. A seleção das amostras de dados, inicialmente, ficaria restrita a países da América do Sul, porém não havia uma amostra de tamanho mais elevado que fosse compatível com as limitações de *hardware* do ambiente computacional utilizado. Além disso, era necessária uma amostra de dados de tamanho reduzido

[1] `https://planet.openstreetmap.org/`

porém significativo, o que também não foi encontrado na América do Sul. Por esse motivo, optou-se pela América Central, localizada próximo à América do Sul, e por Liechtenstein, por possuir o tamanho que se desejava. Dessa maneira, as quatro amostras selecionadas foram Liechtenstein, Colômbia, Chile e América Central, em ordem crescente de tamanho. A Tabela 3.1 apresenta o tamanho das amostras de dados utilizadas.

Tabela 3.1: Tamanho e data de atualização dos arquivos utilizados.

País	Arquivo	Tamanho	Data da última atualização
Liechtenstein	liechtenstein-latest.osm	50,9 MB	19/08/2017
Colômbia	colombia-latest.osm	2,54 GB	16/10/2017
Chile	chile-latest.osm	3,18 GB	07/11/2018
América Central	central-america-latest.osm	8.07 GB	03/11/2018

Os arquivos no formato .osm apresentam três tipos de objetos fundamentais, são eles:

1. Nós (*Nodes*): Representa um ponto definido por um par de coordenadas latitude e longitude na superfície terrestre;
2. Caminhos (*Ways*): Representa uma lista de 2 a 2000 nós ordenados;
3. Relações (*Relations*): É uma estrutura utilizada para documentar o relacionamento entre elementos, podendo eles ser nós, caminhos ou uma outra relação.

Para analisar a quantidade de nós, caminhos e relações existentes em cada uma das amostras, foi utilizado o *OsmConvert*, um utilitário executado via linha de comando desenvolvido para processar e converter arquivos disponibilizados nos formatos do OpenStreetMap. A Tabela 3.2 apresenta os resultados.

Tabela 3.2: Número de nós, caminhos e relações existentes em cada amostra.

Arquivo	Nós	Caminhos	Relações
liechtenstein-latest.osm	251384	25457	547
colombia-latest.osm	12447701	1038852	10511
chile-latest.osm	19780888	1202708	13133
central-america-latest.osm	46472572	6621393	74313

Com base nessa análise inicial de tamanho e quantidade de objetos, é possível perceber que há um aumento acentuado de uma amostra para outra. Essa seleção se deu de modo que seja possível observar como se comporta a degradação de desempenho conforme o aumento da complexidade da amostra.

Com esses quatro *datasets* devidamente adquiridos, pôde-se passar para a etapa de inserção dos dados nos três SGBD estudados nesse trabalho.

3.2.2 Inserção dos Dados

Essa etapa consistiu no armazenamento dos dados obtidos na etapa de coleta nos três SGBD. É importante notar que o modo de inserção das informações difere em cada um dos bancos de dados. Dessa forma, pode-se subdividir essa etapa em três subetapas para melhor entendimento: inserção no PostgreSQL, inserção no MongoDB e Inserção no Neo4j.

Inserção no PostgreSQL

Para começar a inserção dos dados no PostgreSQL, foi necessário criar uma nova base de dados e adicionar as extensões necessárias para trabalhar em conjunto com dados geográficos. As extensões necessárias foram o PostGIS e HStore. A extensão HStore consiste em um armazenamento do tipo chave-valor interno ao PostgreSQL, semelhante a um dicionário na Linguagem *Python*. No contexto desse trabalho, esta extensão foi utilizada para criar um campo para armazenar as *tags*.

Como já explicado no Capítulo 2, a extensão PostGIS adiciona suporte a objetos geográficos ao PostgreSQL, o que permite a execução de uma vasta gama de consultas utilizando funções geográficas, como uniões, distâncias e áreas. Sua primeira versão foi desenvolvida pela companhia *Refractions Research* em 2001, sob licença aberta [4].

Após adicionadas as extensões, fez-se necessário a criação ou utilização de um esquema (*schema*), que consiste na estruturação do banco de dados, isto é, a organização das tabelas e relações dentro do SGBD. Para este projeto, foi utilizado um *script* de criação de esquema pré configurado, fornecido juntamente com a ferramenta de inserção Osmosis [19]. A Figura 3.1 mostra a execução do *script* de criação do esquema denominado *pgsnapshot_schema_0.6.sql*, definido pela ferramenta Osmosis.

```
psql -h localhost -U postgres -f "pgsnapshot_schema_0.6.sql" OSM
```

Figura 3.1: Utilização via linha de comando do esquema pré definido pelo Osmosis.

No comando acima, o argumento -h consiste no nome do *host* aonde o PostgreSQL está rodando, o argumento -U consiste no nome do usuário realizando a operação, o argumento -f consiste no nome do arquivo no qual os comandos a serem executados se encontram e, por fim, o argumento OSM nada mais é do que o nome da base de dados sendo utilizada.

A ferramenta Osmosis consiste em uma aplicação desenvolvida em *Java* e utilizável via linha de comando para realizar processamento de dados provenientes do OpenStreetMap, como por exemplo a inserção de arquivos osm em uma base de dados armazenada no PostgreSQL, como é o caso desse estudo de caso. A Figura 3.2 mostra a utilização dessa ferramenta.

```
osmosis --read-xml file="/Users/liechtenstein-latest.osm" --write-pgsql host="localhost" database="OSM" user="postgres"
```

Figura 3.2: Utilização via linha de comando da ferramenta Osmosis para inserção da amostra de Liechtenstein na base de nome OSM.

Como é possível perceber, a utilização via linha de comando demanda o uso de alguns argumentos específicos. No caso dessa aplicação, os argumentos e uma breve explicação de sua utilidade são listados a seguir:

- –read-xml file=: Esse argumento é responsável por realizar a leitura do conteúdo de um arquivo xml de extensão osm, localizado no caminho fornecido após o sinal de igualdade próximo ao identificador *file*;
- –write-pgsql host= database= user=: Argumento responsável por popular uma base de dados PostGIS vazia. O identificador *host* diz respeito ao endereço da máquina aonde o PostgreSQL está sendo executado, enquanto os identificadores *database* e *user* se referem ao nome da base de dados e usuário executando a operação, respectivamente.

Os procedimentos acima foram realizados 10 vezes para cada uma das quatro amostras de dados, totalizando 40 inserções. Em bancos de dados, este número de repetições é o suficiente para atingir estimativas confiáveis. As execuções foram do tipo quente, ou seja, quando os SGBD já estavam sendo utilizados, sem realizar a reinicialização do sistema entre cada inserção. Para automatizar o processo de inserção e medição de velocidade, uma rotina foi desenvolvida utilizando as linguagens *bash* e *Python*. Tais *scripts* podem ser encontrados no *GitHub*[2].

Inserção no MongoDB

A inserção das amostras no MongoDB necessita menos passos práticos. Para a população da base de dados, foi utilizado o MongOSM, um conjunto de utilitários desenvolvidos em *Python* por Ian Dees e disponibilizados no *GitHub*[3] que possibilitam, dentre outras coisas, a inserção de arquivos no formato osm no MongoDB.

Para a utilização desses utilitários, é necessário a linguagem de programação *Python* na versão 2.5 e da distribuição denominada Pymongo[4], um conjunto de ferramentas para utilizar o MongoDB em conjunto com a linguagem de programação *Python*.

[2]https://github.com/vUriarte/TCC
[3]`https://github.com/iandees/mongosm`
[4]`https://api.mongodb.com/python/current`

No escopo deste trabalho, o utilitário presente no MongOSM necessário para a inserção consiste na rotina *insert_osm_data.py*. Sua utilização é simples, conforme demonstra a Figura 3.3.

```
python insert_osm_data.py liechtenstein-latest.osm
```

Figura 3.3: Utilização via linha de comando do MongOSM para inserção da amostra de Liechtenstein.

A linha de comando acima basicamente executa a rotina *insert_osm_data.py* utilizando, neste exemplo, o arquivo *liechtenstein-latest.osm* como fonte dos dados a serem adicionados. A rotina cria uma base chamada OSM contendo três coleções de documentos: *nodes*, *ways* e *relations*, populando-as.

Os procedimentos acima foram realizados de maneira quente para cada uma das quatro amostras de dados, totalizando 40 inserções. O processo também fez uso da simples rotina *bash* para automatizar as inserções, presente no *GitHub*[5].

Inserção no Neo4j

A inserção dos dados no Neo4j foi realizada através da biblioteca *Neo4j Spatial*, que conta com uma classe java chamada *OSMImporter* dedicada à inserção de arquivos xml com a extensão osm. A Figura 3.4 ilustra como a classe em questão foi utilizada.

```
java -cp target/classes;target/dependency/* org.neo4j.gis.spatial.osm.OSMImporter osm-db-col colombia-latest.osm
```

Figura 3.4: Linha de comando responsável pela inserção da amostra referente a Colômbia.

Como é evidente na Figura 3.4, foi utilizada a linguagem de programação *Java* para executar a classe fornecida pelo *Neo4j Spatial*. Os seguintes argumentos foram passados via linha de comando:

- -cp target/classes;target/dependency/*: O comando -cp é responsável por apontar o caminho de classes ou bibliotecas necessárias para a execução ou compilação do programa escrito na linguagem *Java*;
- org.neo4j.gis.spatial.osm.OSMImporter: Este é o nome da classe propriamente dita, a classe a ser executada;
- osm-db-col : O nome da base de dados a ser criada e populada;
- colombia-latest.osm: O arquivo a ser inserido na base de dados.

[5]https://github.com/vUriarte/TCC

De modo análogo às inserções nos outros SGBD, foram realizadas 10 execuções quentes para quatro amostras de dados, totalizando 40 inserções. Também foi utilizado a rotina *bash* para automatização das inserções.

3.2.3 Consultas Espaciais

Para aferir o desempenho dos sistemas gerenciadores de banco de dados, foram desenvolvidas consultas, ou *queries*, equivalentes nos 3 SGBD, todos utilizando as mesmas amostras de dados. Como o MongoDB é o SGBD que possui menos funções espaciais implementadas, estas foram utilizadas como modelo. São elas:

- *Within*: Seleciona os resultados que estão dentro de uma certa geometria;
- *Intersects*: Seleciona os resultados que cruzam, intersectam uma dada geometria;
- *Near*: Dado um ponto, retorna todos os resultados que se encontram a uma certa distância de forma ordenada, do mais próximo ao mais distante.

Como nem todos os SGBD possuem os mesmos procedimentos espaciais, foram elaboradas consultas equivalentes utilizando as funções disponíveis. A Tabela 3.3 resume as funções utilizadas em cada um dos bancos.

Tabela 3.3: Tabela mostrando as funções espaciais de cada banco utilizadas em cada consulta.

	MongoDB	**PostgreSQL**	**Neo4j**
Within	*$geoWithin* *$centerSphere*	*ST_DWithin* *ST_GeomFromText*	*spatial.withinDistance*
Intersects	*$geoIntersects* *$geometry*	*ST_Intersects,* *ST_GeomFromText*	*spatial.intersects*
Near	*$near* *$geometry*	*ST_DWithin* *ST_Distance* *ST_MakePoint*	*spatial.withinDistance*

As consultas elaboradas para cada SGBD estão presentes no *GitHub*, embora algumas delas apareçam nas subseções que se seguem. Com vistas a melhorar o entendimento das consultas, estas serão explicadas de forma mais detalhada a seguir.

Consultas *Within*

Dado um certo ponto ou polígono de referência, a consulta *Within* realiza a busca de objetos espaciais contidos em um intervalo, como por exemplo um raio de 12km de distância. Nesse trabalho, foi definido um ponto geográfico, equivalente a um par ordenado

de coordenadas geográficas, para cada amostra de dados, que foi utilizado como ponto de partida em todas as *queries* desse tipo, independente do SGBD, conforme a Tabela 3.4.

Tabela 3.4: Locais e coordenadas geográficas dos pontos utilizados em cada amostra de dados.

	Local	**Coordenadas Geográficas**
Liechtenstein	Castelo de Vaduz	longitude: 9.5242002 latitude: 47.1405701
Colômbia	Aeroporto Internacional El Dorado, Bogotá	longitude: -74.14606074859216 latitude: 4.700912450000001
Chile	Aeroporto Internacional Comodoro Arturo Merino Benítez, Santiago	longitude: -70.79417391985464 latitude: -33.388679249999996
América Central	Aeroporto Internacional Augusto C. Sandino, Manágua, Nicarágua	longitude: -86.168755740802 latitude: 12.14039905

As consultas *Within* elaboradas estão expressas nas linguagens de consulta dos sistemas gerenciadores de banco de dados na Figura 3.5

Foi, então, realizada uma consulta por amostra utilizando os pontos definidos como centro de um círculo de raio igual a 12km. Foram realizadas 10 execuções quentes para cada *query*, por SGBD.

MongoDB

```
db.nodes.find(
        {loc:
            {$geoWithin:
                { $centerSphere:
                    [[  9.5242002,
                        47.1405701 ],
                            12/ 6371 ]
                }
            }
        }
    ).explain("executionStats")
```

PostgreSQL

```
SELECT geom
FROM nodes
WHERE ST_DWithin(geom::geography,
            ST_GeomFromText('POINT(9.5242002
                                47.1405701)', 4326)::geography,12000);
```

Neo4j

```
CALL
spatial.withinDistance('/dir/do/osm/sample.osm',
                        {longitude:9.5242002,
                        latitude:47.1405701},12);
```

Figura 3.5: Consultas *Within* com raio de 12km nas linguagens de consulta dos respectivos SGBD.

Graficamente, as consultas Within com raio de 12km representam o que está exposto nas Figuras 3.6, 3.7, 3.8, 3.6 e 3.9 referente, respectivamente, a Liechtenstein, Colômbia, Chile e América Central.

Figura 3.6: Representação no mapa da consulta *Within* com raio de 12km para Liechtenstein.

Figura 3.7: Representação no mapa da consulta *Within* com raio de 12km para a Colômbia.

Consultas *Intersects*

Como na anterior, estas *queries* necessitam um ponto ou polígono de referência. Nesse caso, porém, foi definido um polígono limitador para cada uma das amostras de dados. Estes polígonos são denominados *bounding boxes*, ou caixas delimitadoras. Textualmente, a *bounding box* é um conjunto de coordenadas geográficas que formam um retângulo. A Tabela 3.5 mostra os polígonos utilizados nesse trabalho.

Figura 3.8: Representação no mapa da consulta *Within* com raio de 12km para o Chile.

Figura 3.9: Representação no mapa da consulta *Within* com raio de 12km para a América Central.

Com a ajuda do site *geojson.io* [11], é possível representar visualmente as *Bounding Boxes* presentes na Tabela 3.5. Esta representação pode ser observada na Figura 3.10.

Tabela 3.5: Representação textual das *Bounding Boxes* utilizadas nas consultas.

	Bounding Box
Liechtenstein	[[[9.5097211464, 47.1558570829], [9.5586009272, 47.1558570829], [9.5586009272, 47.2109385806], [9.5097211464, 47.2109385806], [9.5097211464, 47.1558570829]]]
Colômbia	[[[-75.7982040942, 5.065987494], [-74.0512200777, 5.065987494], [-74.0512200777, 5.9344189261], [-75.7982040942, 5.9344189261], [-75.7982040942, 5.065987494]]]
Chile	[[[-71.8031632681, -34.3363776491], [-70.9475924911, -34.3363776491], [-70.9475924911, -33.4103920585], [-71.8031632681, -33.4103920585], [-71.8031632681, -34.3363776491]]]
América Central	[[[-89.0083365836, 13.2433273728], [-83.6675302742, 13.2433273728], [-83.6675302742, 15.6250254777], [-89.0083365836, 15.6250254777], [-89.0083365836, 13.2433273728]]]

Por fim, foram realizadas 10 execuções quentes por consulta. A Figura 3.11 ilustra uma consulta já nas linguagens dos respectivos SGBD.

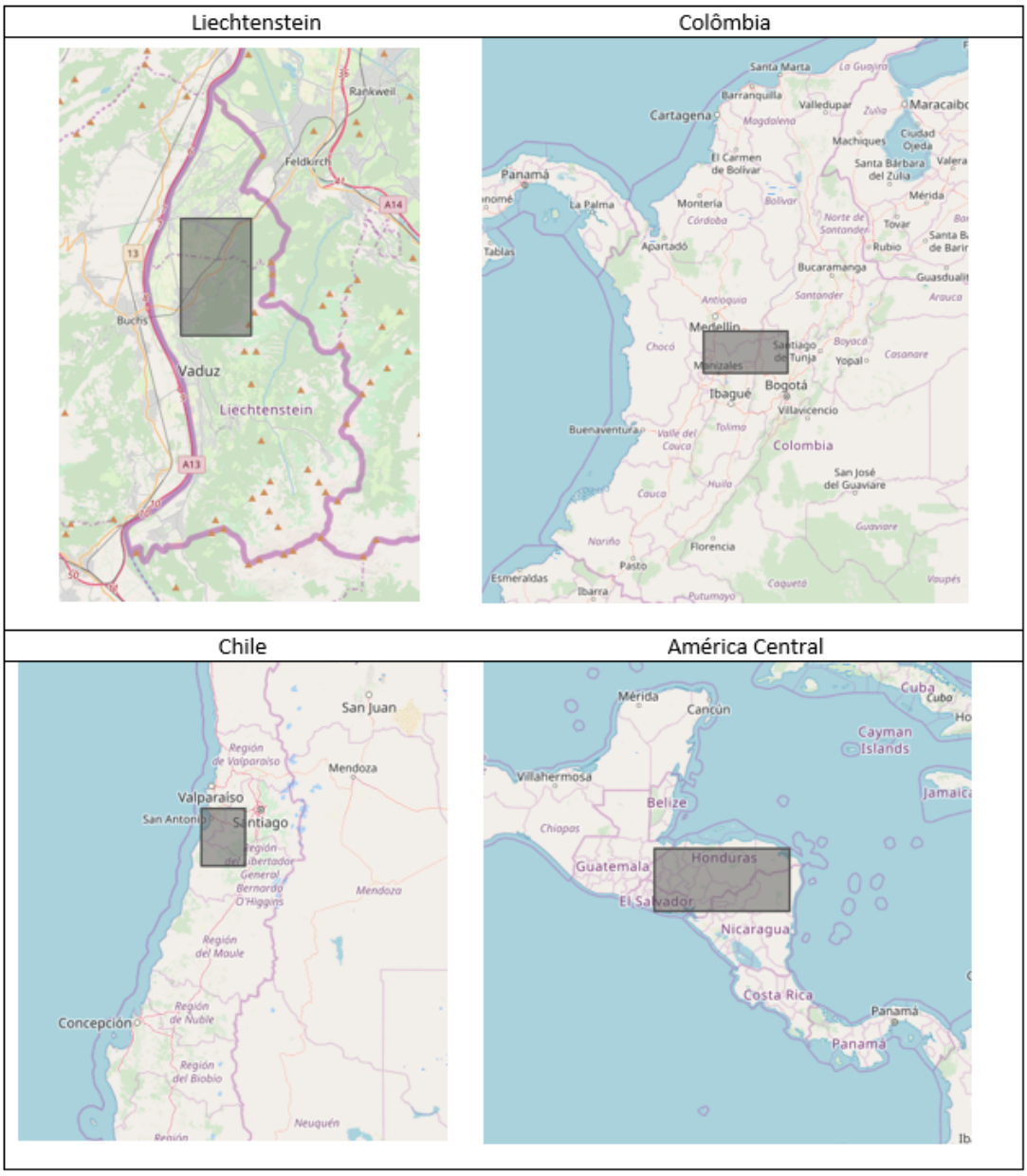

Figura 3.10: Representação visual das *Bounding Boxes*.

Consultas *Near*

As consultas *Near* são similares às consultas *Within*, ambas retornam resultados em um dado raio de distância a partir de um dado ponto. A consulta *Near*, porém, retorna tais dados por ordem de distância, do resultado mais perto do ponto de referência até o mais distante dentro do raio imposto pela consulta.

Desse modo, faz-se necessário a definição de um ponto de referência que será a origem do raio de busca. Tais pontos estão definidos na Tabela 3.6.

É importante ressaltar que nem todos os SGBD utilizados nesse trabalho possuem a

MongoDB

```
db.nodes.find(
        {loc:
            {$geoIntersects:
                {$geometry:
                    {type: "Polygon",
                    coordinates: [[[9.5097211464,47.1558570829],
                                   [9.5586009272,47.1558570829],
                                   [9.5586009272,47.2109385806],
                                   [9.5097211464,47.2109385806],
                                   [9.5097211464,47.1558570829]]]
                    }
                }
            }
        }
    ).explain("executionStats")
```

PostgreSQL

```
SELECT geom
FROM nodes
WHERE  ST_Intersects(ST_GeomFromText('POLYGON(( 9.5097211464 47.1558570829,
                                                9.5586009272 47.1558570829,
                                                9.5586009272 47.2109385806,
                                                9.5097211464 47.2109385806,
                                                9.5097211464 47.1558570829))', 4326), geom);
```

Neo4j

```
WITH
'POLYGON((  9.5097211464 47.1558570829,
            9.5586009272 47.1558570829,
            9.5586009272 47.2109385806,
            9.5097211464 47.2109385806,
            9.5097211464 47.1558570829))'
            as polygon
            CALL spatial.intersects('/dir/do/osm/sample.osm',polygon) YIELD node
            RETURN node;
```

Figura 3.11: Estrutura das consultas *Intersects* para cada SGBD.

Tabela 3.6: Locais e coordenadas geográficas dos pontos utilizados em cada amostra de dados.

	Local	**Coordenadas Geográficas**
Liechtenstein	Castelo de Vaduz	longitude: 9.5242002 latitude: 47.1405701
Colômbia	Aeroporto Internacional El Dorado, Bogotá	longitude: -74.14606074859216 latitude: 4.700912450000001
Chile	Aeroporto Internacional Comodoro Arturo Merino Benítez, Santiago	longitude: -70.79417391985464 latitude: -33.388679249999996
América Central	Aeroporto Internacional Augusto C. Sandino, Manágua, Nicarágua	longitude: -86.168755740802 latitude: 12.14039905

função *Near* implementada, sendo necessária a utilização de uma ou mais funções que se comportem de maneira análoga a ela, como pode ser observado na Figura 3.12. O raio definido foi de 12km e cada consulta foi executada de maneira quente um total de 10 vezes.

MongoDB

```
db.nodes.find(
        {loc:
            {$near:
                {$geometry:
                    {type: "Point",
                    coordinates:[9.5242002,
                                 47.1405701]},
                    $maxDistance: 12000}
            }
        }).explain("executionStats")
```

PostgreSQL

```
select
geom
from nodes
where ST_DWithin(geom::geography,
        ST_MakePoint(9.5242002,
                47.1405701)::geography, 12000)
                order by ST_Distance(geom::geography,
                        ST_MakePoint(9.5242002,
                                47.1405701)::geography);
```

Neo4j

```
CALL
spatial.withinDistance('/dir/do/osm/sample.osm',
                    {longitude:9.5242002,
                    latitude:47.1405701},12);
```

Figura 3.12: Consultas *Near* com raio 12km nos respectivos SGBD.

3.3 Resultados Experimentais

Esta seção é dedicada a apresentar os resultados obtidos nos testes de inserção e consultas espaciais.

3.3.1 Resultados das Inserções

Inserções no PostgreSQL

Primeiramente, foi feita a inserção da amostra referente a Liechtenstein, com tamanho aproximado de 50.9 MB, ou seja, um arquivo de tamanho bastante reduzido se comparado aos três outros utilizados nesta monografia. O menor tempo mensurado foi de 9 segundos, enquanto o maior tempo foi de 10 segundos, com uma média de 9 segundos para as inserções.

A segunda bateria de inserções foi realizada com os arquivos da Colômbia, com tamanho aproximado de 2.54 GB, 50 vezes maior que o arquivo anterior. Dessa vez, as inserções foram realizadas em minutos, sendo 7 minutos e 2 segundos o menor tempo mensurado e 7 minutos e 11 segundos o maior tempo mensurado. A média do tempo das inserções foi de 7 minutos e 6 segundos.

Na terceira bateria, foram utilizados os arquivos referente ao Chile, com tamanho aproximado de 3.18GB, 62 vezes maior que a amostra de Liechtenstein e 1.25 vezes maior que a Colômbia. A inserção mais demorada foi de 15 minutos e 30 segundos e a mais rápida foi de 14 minutos e 37 segundos. Em média, as inserções foram realizadas em 15 minutos e 1 segundo.

A quarta e última bateria de inserções no PostgreSQL foi realizada com os arquivos referente à América Central, com tamanho aproximado de 8.07 GB, 158.54 vezes maior que Liechtenstein, 3.17 vezes maior que a Colômbia e 2.54 vezes maior que o Chile. Mesmo com essa diferença de tamanho considerável entre as últimas amostras, as inserções ainda foram realizada em minutos, tendo a mais demorada levado 55 minutos e 24 segundos e a mais rápida 54 minutos e 6 segundos. A média do tempo das inserções foi de 54 minutos e 39 segundos.

A Figura 3.13 ilustra as 10 inserções para cada amostra de dados no PostgreSQL.

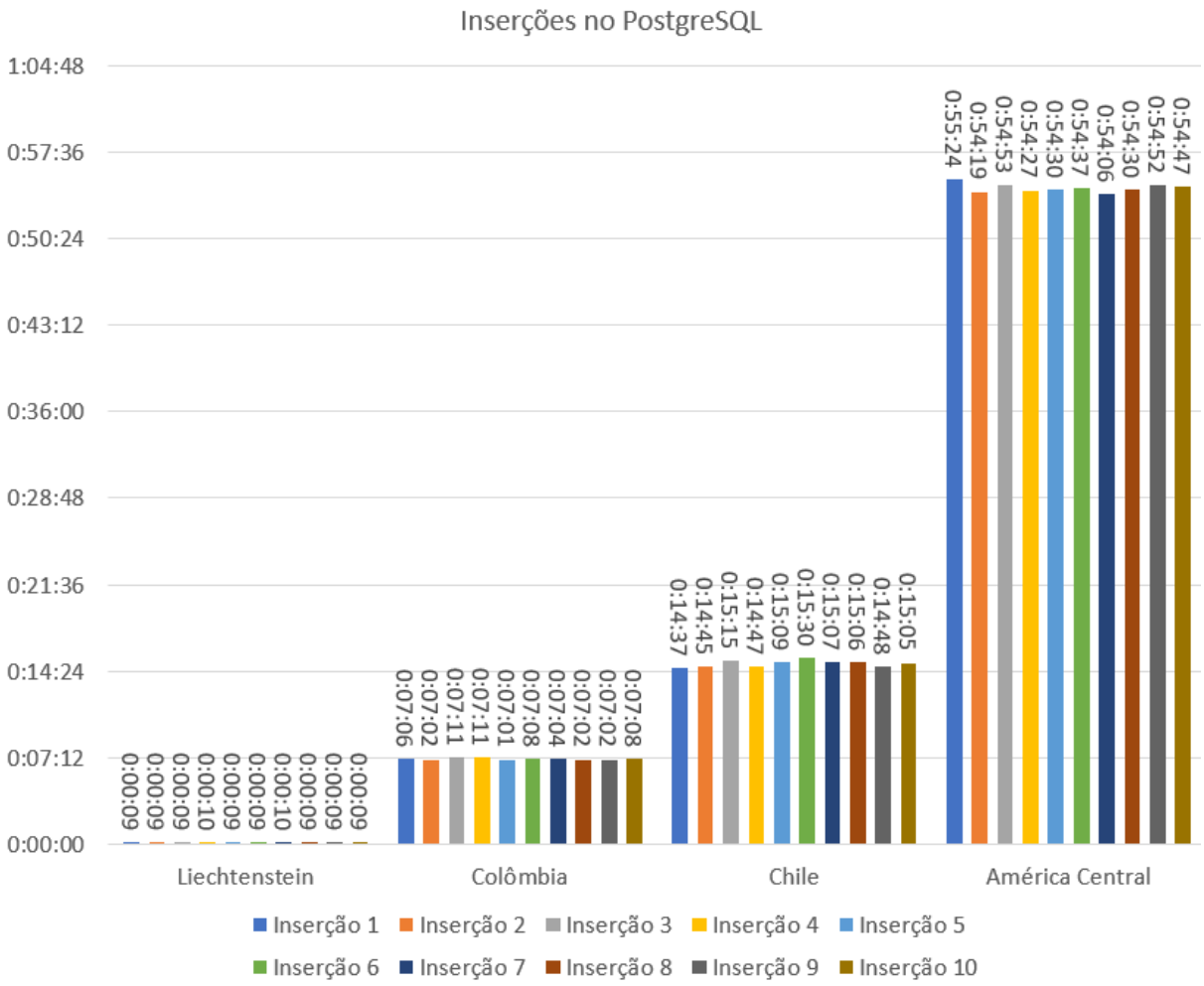

Figura 3.13: Gráfico das 10 inserções de cada amostra de dados no PostgreSQL.

Com base nesses dados, é possível perceber que não houve um aumento anormal de tempo entre as amostras: a segunda amostra é 50 vezes maior que a primeira e levou 47 vezes mais tempo para ser inserida; a terceira amostra é 1.25 vezes maior que a segunda e levou 2 vezes mais tempo na inserção; a última amostra é 2.54 vezes maior que a terceira e levou 3.6 vezes mais tempo para ser inserida.

Inserções no MongoDB

De modo análogo às inserções no PostgreSQL, o primeiro conjunto de dados a ser inserido foi Liechtenstein. A inserção mais demorada se deu em 30 segundos, enquanto a inserção mais rápida se deu em 29 segundos. Em média, as inserções levaram 30 segundos.

Colômbia foi a segunda amostra a ser inserida. No pior caso, foi necessário 17 minutos e 50 segundos para inserção, enquanto o melhor caso se deu em 17 minutos e 41 segundos. A média de tempo de inserção foi de 17 minutos e 45 segundos.

A terceira bateria de inserções foi referente ao Chile. A inserção mais lenta ocorreu em 25 minutos e 18 segundos, enquanto a melhor inserção levou 25 minutos e 37 segundos. O tempo médio de inserção foi de 25 minutos e 26 segundos.

A América Central foi a última amostra a ser inserida no MongoDB. Dessa vez, foi necessário mais do que horas para a inserção: o pior caso se deu em 2 horas, 32 minutos e 1 segundo, enquanto o melhor caso se deu em 2 horas, 31 minutos e 8 segundos. Em média, as inserções levaram 2 horas, 31 minutos e 20 segundos para acontecer.

A Figura 3.14 mostra as 10 inseções de cada amostra no MongoDB.

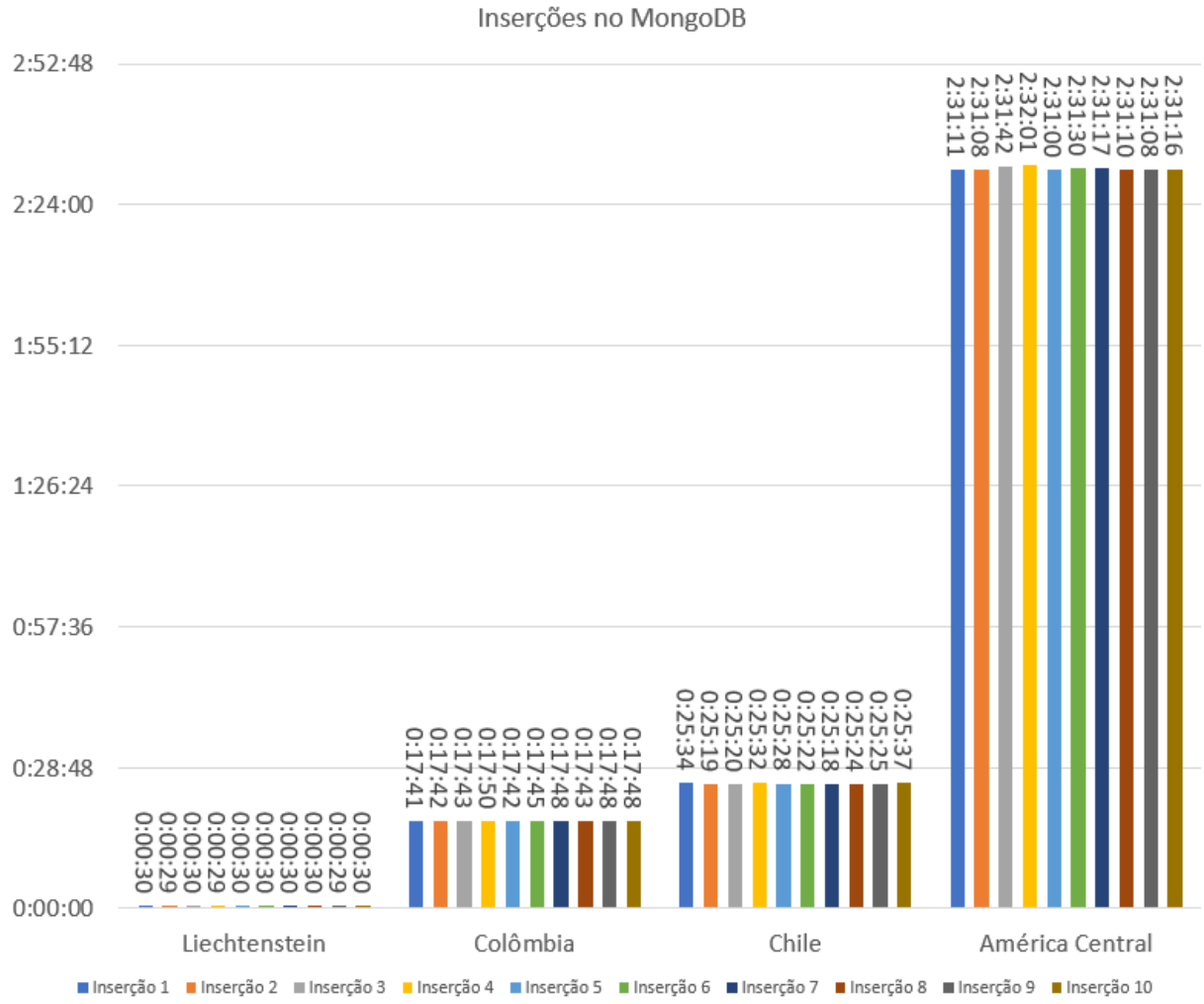

Figura 3.14: Gráfico das 10 inserções de cada amostra de dados no MongoDB.

Pelos dados obtidos acima, é possível ver uma acentuada degradação de performance entre a amostra referente ao Chile e a referente à América Central, passando de uma média de 25 minutos e 26 segundos na primeira para 2 horas, 31 minutos e 20 segundos na segunda. Isso pode ser explicado pelo grande crescimento da complexidade da amostra: enquanto o Chile possui 19780888 nós, 1202708 caminhos e 13133 relações, a

América Central conta com 46472572 nós, 6621393 caminhos e 74313 relações. Esse fator, juntamente com a utilização dos *scripts* de inserção na linguagem *Python*, resultam no aumento acentuado nos tempos de inserção.

Inserções no Neo4j

Por fim, foram realizadas as inserções no Neo4j, começando por Liechtenstein. Essa amostra, mesmo que pequena, já foi suficiente para que a população do banco de dados fosse mais demorada que alguns segundos, atingindo a marca de 1 minuto. No pior caso, a inserção demorou 1 minuto e 25 segundos, enquanto o melhor caso se deu em 1 minutos e 18 segundos. Em média, foi necessário 1 minuto e 21 segundos para a inserção.

A segunda amostra, referente à Colômbia, necessitou, no pior caso, de 1 hora, 46 minutos e 8 segundos. No melhor caso, foi necessário 1 hora, 41 minutos e 58 segundos. A média de tempo necessário foi de 1 hora, 43 minutos e 21 segundos.

A Figura 3.15 mostra os tempos das 10 inserções das amostras de Liechtenstein e da Colômbia no Neo4j.

O Chile e a América Central resultaram, nas 10 inserções, em estouro de memória *heap* da máquina virtual *Java* (JVM). A área *heap* da JVM consiste numa porção de memória limitada criada na inicialização da máquina virtual que serve para alocação de *arrays* e de todas as instâncias das classes *Java* utilizadas. Com o tempo, o *heap* vai se enchendo e fica sujeito a estouro caso novos objetos necessitem de memória enquanto objetos antigos ainda a ocupam.

A quantidade de memória *heap* pode ser ajustada através de alterações no documento de configuração do Neo4j. No sistema operacional utilizado nesse projeto, macOS, o arquivo é denominado *neo4j.conf*. A Figura 3.16 mostra a configuração em questão.

Para tentar resolver o problema de pouca memória para a JVM, as seguintes medidas foram tomadas:

- dbms.memory.heap.max_size=2048m: Alocar 2.048GB para a *heap*;
- dbms.memory.heap.max_size=3048m: Alocar 3.048GB para a *heap*;
- dbms.memory.heap.max_size=4500m: Alocar 4.5GB para a *heap*;
- dbms.memory.heap.max_size=6048m: Alocar 6.048GB para a *heap*.

Mesmo com todos esses ajustes, não foi possível inserir os dados de forma correta. Com a *heap* em 2.048GB, 3.048GB e 4.5GB, o estouro de memória *heap* persistiu, enquanto 6.048GB causou o estouro da memória total da máquina virtual. Mais detalhes sobre esse problema são apresentados na subseção seguinte, destinada às conclusões referentes às inserções.

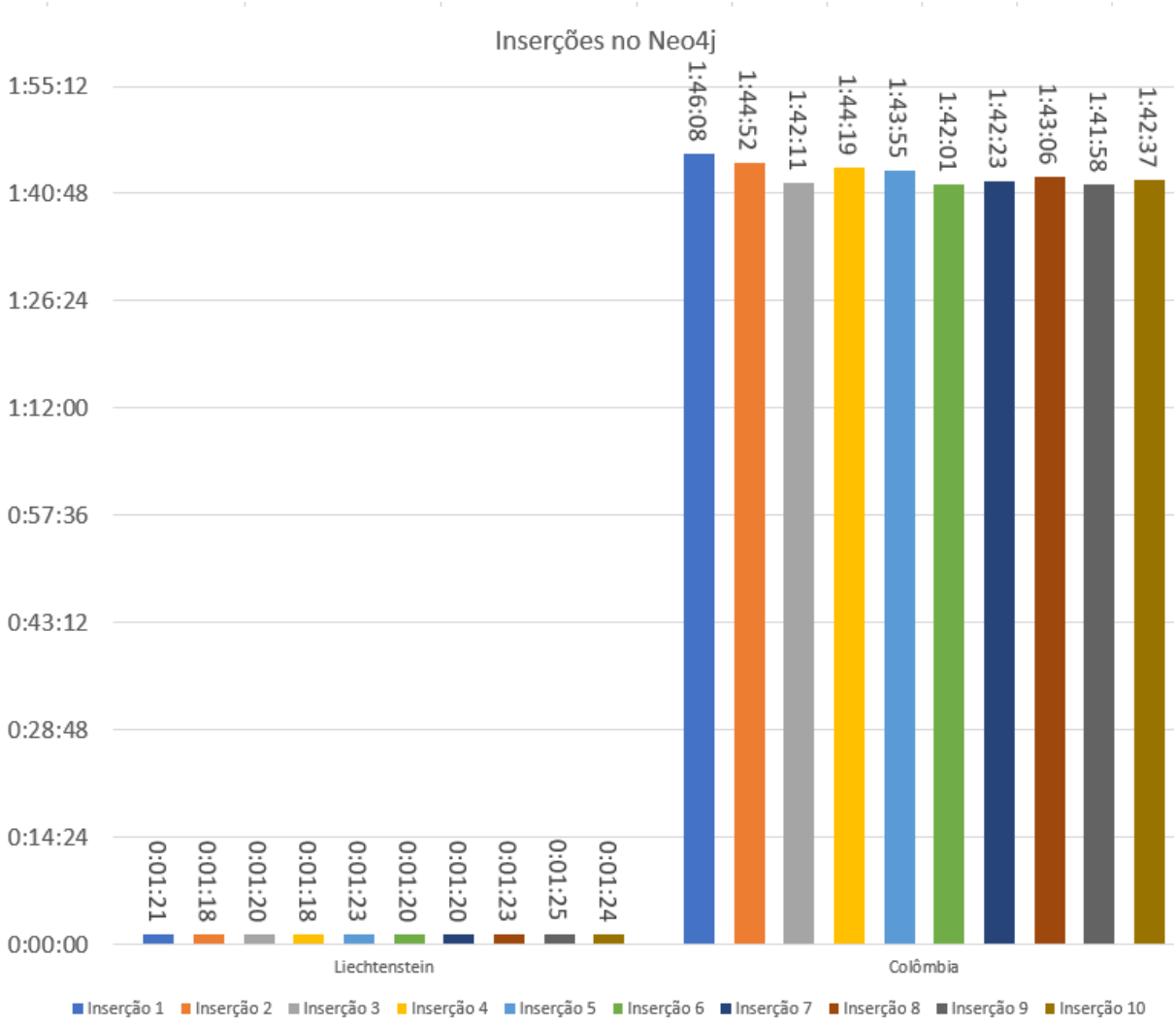

Figura 3.15: Gráfico das 10 inserções de Liechtenstein e da Colômbia no Neo4j.

De acordo com Craig Taverner, funcionário da empresa desenvolvedora do Neo4j e criador do repositório oficial do *Neo4j Spatial* no *GitHub*, esse é um problema recorrente em arquivos de tamanho elevado [20]. O que ocorre é que a inserção de dados através da utilização da classe *OSMImporter* é realizada em duas fases: a fase de *batch insert*, onde os dados são propriamente inseridos, e a fase de indexação, onde os dados inseridos são indexados para representar da melhor maneira possível a organização original do arquivo osm. Basicamente, durante a construção do grafo, é necessário a correlação de nós previamente importados com novas realações, o que torna necessário uma solução que permita a consulta aos identificadores dos nós importados [36].

A solução implementada no *Neo4j Spatial* utiliza *Apache Lucene* para contornar esse problema. Trata-se de uma biblioteca de código aberto, desenvolvida em *Java*, que implementa um motor de busca textual para aplicações, principalmente multiplataformas, que necessitam realizar pesquisas em textos, através da criação de índices com base nesses

```
#*****************************************************************
# Neo4j configuration
#
# For more details and a complete list of settings, please see
# https://neo4j.com/docs/operations-manual/current/reference/configuration-settings/
#*****************************************************************

# The name of the database to mount
#dbms.active_database=graph.db

# Paths of directories in the installation.
#dbms.directories.data=data
#dbms.directories.plugins=plugins
#dbms.directories.certificates=certificates
#dbms.directories.logs=logs
#dbms.directories.lib=lib
#dbms.directories.run=run

# This setting constrains all `LOAD CSV` import files to be under the `import` directory. Remove or comment it out to
# allow files to be loaded from anywhere in the filesystem; this introduces possible security problems. See the
# `LOAD CSV` section of the manual for details.
dbms.directories.import=import

# Whether requests to Neo4j are authenticated.
# To disable authentication, uncomment this line
dbms.security.auth_enabled=false

# Enable this to be able to upgrade a store from an older version.
#dbms.allow_format_migration=true

# Java Heap Size: by default the Java heap size is dynamically
# calculated based on available system resources.
# Uncomment these lines to set specific initial and maximum
# heap size.
#dbms.memory.heap.initial_size=512m
#dbms.memory.heap.max_size=3048m
#dbms.memory.heap.max_size=6048m
#dbms.memory.heap.max_size=4500m
dbms.memory.heap.max_size=2048m
```

Figura 3.16: Arquivo de configuração do Neo4j em ambiente *macOS*.

textos. O *Neo4j Spatial* utiliza os índices criados pelo *Lucene* em conjunto com tabelas *hash* para realizar a busca pelos nós no processo de criação do *index*, durante a inserção dos dados. Esse processo, porém, se torna muito custoso e demorado conforme o aumento do tamanho da base de dados utilizada [3, 14].

Durante os últimos anos, houve discussões acerca da implementação de uma nova solução de inserção de documentos com a extensão osm, porém até o momento nada foi desenvolvido de fato. Desse modo, o único modo atual de contornar o problema de estouro de memória é com a adição de mais memória *RAM* na máquina, o que não convém para esse trabalho.

A segunda amostra é 50 vezes maior que a primeira amostra e levou 76.5 vezes mais para ser inserida. A degradação de performance já foi notável neste caso, onde houve uma diferença, em média, de 1 hora e 42 minutos entre as amostras. Para os demais arquivos, não foi possível mensurar os tempos de inserção devido a limitação de hardware, mais precisamente de memória RAM.

Comparações

A Tabela 3.7 contém o tempo médio de inserção das amostras de dados pro SGBD. A Figura 3.17 ilustra o gráfico de comparação entre os SGBD.

Tabela 3.7: Tabela contendo os tempos médios de inserção de cada amostra por SGBD.

	PostgreSQL	**MongoDB**	**Neo4j**
Liechtenstein	0:00:09	0:00:30	0:01:21
Colômbia	0:07:06	0:17:45	1:43:21
Chile	0:15:01	0:25:26	-
América Central	0:54:39	2:31:20	-

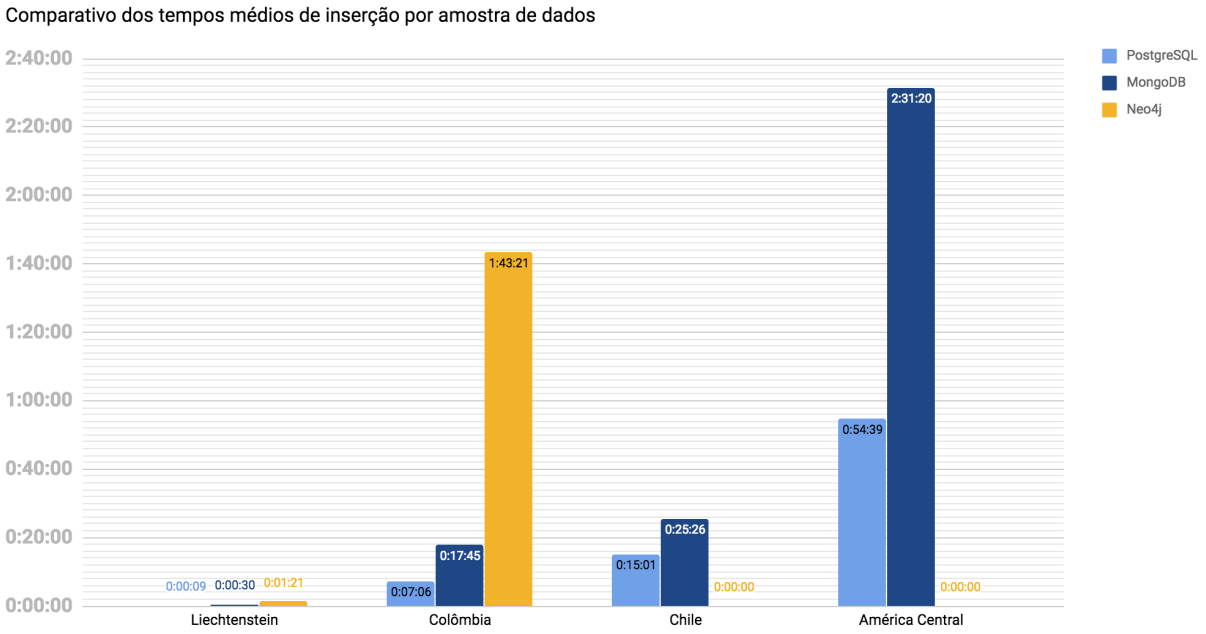

Figura 3.17: Gráfico comparativo dos tempos de inserção agregados por amostra de dados.

Com base no gráfico e nos resultados obtidos nos experimentos, é possível observar que o PostgreSQL se apresenta como o SGBD mais ágil na hora de inserir os dados provenientes do OpenStreetMap, seguido pelo MongoDB, com o Neo4j sendo o mais demorado e ineficiente entre eles.

Para Liechtenstein, o PostgreSQL se mostrou 3.3 vezes mais rápido que o MongoDB e 9 vezes mais rápido que o Neo4j; Para a Colômbia, a diferença foi de 2.5 vezes em relação ao MongoDB e 14.6 vezes em relação ao Neo4j.

Comparando o MongoDB ao Neo4j, as inserções de Liechtenstein foram, em média, 2.7 vezes mais rápidas no MongoDB, enquanto para a Colômbia a diferença foi de 5.8 vezes.

Para as amostras do Chile e da América Central, houve o estouro de memória por parte do Neo4j, logo a comparação se dará apenas entre o PostgreSQL e o MongoDB. O PostgreSQL se mostrou em média 1.7 vezes mais rápido que o MongoDB nas inserções referentes ao Chile, e 2.77 vezes em relação à América Latina.

A Tabela 3.8 resume o que foi dito nos parágrafos acima, apresentando a diferença no tempo de inserção das amostras por cada SGBD em horas, minutos, segundos e quantas vezes isso representa.

Tabela 3.8: Tabela contendo as diferenças entre os tempos médios de inserção de cada amostra nos SGBD

	Diferença entre PostgreSQL e MongoDB	**Diferença entre PostgreSQL e Neo4j**	**Diferença entre MongoDB e Neo4j**
Liechtenstein	00:00:21 =>3.3x	00:01:12 =>9x	00:00:51 =>2.7x
Colômbia	00:10:39 =>2.5x	01:36:15 =>14.6x	01:25:36 =>5.8x
Chile	00:10:25 =>1.7x	-	-
América Central	01:36:41 =>2.77x	-	-

Em geral, houve uma diferença notável de performance entre o PostgreSQL e o MongoDB, porém essa diferença se manteve sem alterações absurdas durante todos os testes, apresentando resultados similares tanto para a menor quanto para a maior mostra de dados. Parte dessa diferença de desempenho poderia ser resolvida com a confecção de novos *scripts* de inserção em linguagens de programação que apresentam execução mais rápida que o *Python*, como *C++* e até mesmo *Java*. Mesmo assim, o MongoDB se apresentou como uma alternativa viável ao PostgreSQL nos testes de inserção.

No caso do Neo4j, é perceptível que, conforme ocorria o aumento do tamanho do arquivo de testes, os resultados pioravam consideravelmente, tanto em relação ao PostgreSQL, onde esse efeito foi ainda mais acentuado, quanto em relação ao MongoDB, onde a diferença foi menor. Fica evidente que, enquanto as classes de inserção do *Neo4j Spatial* continuarem a utilizar *Lucene* em conjunto com tabelas *hashes*, fica inviável a utilização desse para lidar com grandes volumes de dados. Para amostras menores, ao que tange a inserção dos dados, há a possibilidade de adoção do Neo4j, porém com um desempenho notavelmente inferior aos outros SGBD analisados.

3.3.2 Resultados das consultas

O objetivo dessa seção é ilustrar os resultados experimentais acerca das consultas espaciais aos SGBD analisados nesse trabalho. Com vistas a melhorar o entendimento geral, essa

seção será dividida por função espacial: *Within*, *Intersects* e *Near*.

Consultas *Within*

Primeiramente, foram realizadas as consultas com a amostra de dados referente a Liechtenstein. Os resultados, em milisegundos, estão expressos na Tabela 3.9.

Tabela 3.9: Tabela contendo os resultados das execuções quentes das consultas *within* para Liechtenstein.

	Raio 12km		
	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*1107 ms*	*430 ms*	*6240 ms*
Consulta 2	559 ms	239 ms	2798 ms
Consulta 3	553 ms	239 ms	2727 ms
Consulta 4	564 ms	236 ms	2743 ms
Consulta 5	542 ms	238 ms	2765 ms
Consulta 6	551 ms	239 ms	2771 ms
Consulta 7	550 ms	241 ms	2712 ms
Consulta 8	549 ms	240 ms	2768 ms
Consulta 9	546 ms	239 ms	2780 ms
Consulta 10	553 ms	238 ms	2800 ms
Média sem primeira consulta	*552 ms*	*239 ms*	*2763 ms*

Como explicita a tabela, a primeira consulta foi consideravelmente mais demorada que as outras. Isso ocorre pois é realizado acesso ao disco rígido para buscar os dados armazenados nos SGBD. Após o primeiro acesso, os dados ficam em *cache*, o que torna os resultados substancialmente mais rápidos. As médias foram calculadas desconsiderando a primeira consulta.

Com a menor amostra de dados, houve o retorno de aproximadamente 241487 resultados. Nessa situação, o PostgreSQL se mostrou o mais ágil, executando 2.3 e 11.5 vezes mais rapidamente que o MongoDB e o Neo4j, respectivamente. O MongoDB, por sua vez, apresentou um resultado mediano, conseguindo ser 5 vezes mais veloz que o Neo4j.

Utilizando a Colômbia, um *dataset* substancialmente maior que Liechtenstein, o cenário muda. Os resultados estão apresentados na Tabela 3.10.

Nessas consultas, o MongoDB obteve os menores tempos de execução, atingindo uma *performance* 3.4 vezes superior ao PostgreSQL e 11 vezes melhor que o Neo4j. É notável que o tamanho da amostra, em conjunto com o aumento do número de objetos retornados de aproximadamente 241487 para aproximadamente 274854, foi o suficiente para causar

Tabela 3.10: Tabela contendo os resultados das execuções quentes das consultas *within* para Colômbia.

	Raio 12km		
	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*3272 ms*	*7463 ms*	*25559 ms*
Consulta 2	770 ms	2664 ms	12370 ms
Consulta 3	809 ms	2667 ms	8589 ms
Consulta 4	807 ms	2680 ms	8495 ms
Consulta 5	813 ms	2679 ms	8412 ms
Consulta 6	790 ms	2693 ms	8345 ms
Consulta 7	802 ms	2689 ms	8255 ms
Consulta 8	795 ms	2672 ms	8224 ms
Consulta 9	781 ms	2700 ms	8210 ms
Consulta 10	797 ms	2693 ms	8264 ms
Média sem primeira consulta	*796 ms*	*2682 ms*	*8796 ms*

uma degradação de desempenho perceptível no PostgreSQL, enquanto o MongoDB se comportou de forma parecida com o teste anterior. O Neo4j novamente ficou muito atrás dos demais, atingindo praticamente 9 segundos por consulta, 6 a mais que o PostgreSQL.

Para a amostra do Chile, algo interessante ocorreu. O MongoDB, novamente o SGBD mais veloz, apresentou uma performance média *superior* ao caso de teste anterior, com um aquivo de tamanho menor. Isso é explicado pelo número de registros retornados, de aproximadamente 196391, inferior ao número retornado no *dataset* da Colômbia. Isso significa que o tamanho do conjunto de dados utilizados, por si só, não é parâmetro para definir a velocidade de execução das consultas *within*, a quantidade de resultados retornados apresenta uma forte influência nesse caso.

O PostgreSQL, por sua vez, teve um desempenho 1.68 vezes pior que o caso anterior, demonstrando que o tamanho do *dataset* utilizado é fator predominante na degradação de desempenho. A título de comparação, o MongoDB se mostrou 6.5 vezes mais rápido que o PostgreSQL. Não é possível traçar uma comparação em relação ao Neo4j, visto que este apresentou estouro de memória no momento da inserção dessa amostra de dados.

Por fim, a Tabela 3.12 apresenta os resultados utilizando a maior das amostras, a da Améria Central. A situação é análoga a anterior, com o MongoDB apresentando um desempenho ainda melhor com esse *dataset*, novamente explicado pelo fato de ter 135488 retornados contra 196391 do teste anterior. O PostgreSQL apresentou resultados ainda piores, executando as consultas 2 vezes mais lentamente que no teste com a Colômbia.

Tabela 3.11: Tabela contendo os resultados das execuções quentes das consultas *within* para o Chile.

	Raio 12km		
	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*2807 ms*	*11487 ms*	-
Consulta 2	699 ms	4459 ms	-
Consulta 3	666 ms	4467 ms	-
Consulta 4	672 ms	4498 ms	-
Consulta 5	696 ms	4627 ms	-
Consulta 6	673 ms	4531 ms	-
Consulta 7	680 ms	4520 ms	-
Consulta 8	691 ms	4492 ms	-
Consulta 9	690 ms	4567 ms	-
Consulta 10	695 ms	4441 ms	-
Média sem primeira consulta	*685 ms*	*4511 ms*	-

Tabela 3.12: Tabela contendo os resultados das execuções quentes das consultas *within* para a América Central.

	Raio 12km		
	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*1816 ms*	*26188 ms*	-
Consulta 2	396 ms	9081 ms	-
Consulta 3	398 ms	8874 ms	-
Consulta 4	401 ms	8898 ms	-
Consulta 5	395 ms	9049 ms	-
Consulta 6	394 ms	9094 ms	-
Consulta 7	408 ms	9120 ms	-
Consulta 8	391 ms	9079 ms	-
Consulta 9	396 ms	8919 ms	-
Consulta 10	388 ms	9023 ms	-
Média sem primeira consulta	*396 ms*	*9015 ms*	-

Com base nos dados obtidos nessas consultas, fica evidente que o MongoDB apresenta um desempenho superior ao PostgreSQL quando submetido a grandes volumes de dados, ao passo que esse último se comporta melhor com pequenas quantidades de dados.

O Neo4j, por sua vez, não obteve bons resultados em nenhum dos testes, levando sempre alguns segundos para retornar os resultados enquanto os outros SGBD retornavam

em questão de milésimos. Esse mal desempenho pode ser explicado, em parte, pois a função espacial utilizada, *spatial.withinDistance*, se comporta de maneira um pouco diferente das implementações análogas no MongoDB e no PostgreSQL: enquanto esses dois últimos não realizam a ordenação dos resultados por distância, o Neo4j o faz. Na verdade, a função *spatial.withinDistance* se assemelha às consultas *Near*.

O gráfico da Figura 3.18 agrega todos os resultados obtidos nessa subseção, facilitando a percepção de padrões e tendências. Nesse gráfico, é notável a presença de uma degradação de desempenho no PostgreSQL, conforme cresce o tamanho da amostra de pesquisa. Já o MongoDB não apresenta grandes saltos de performance durante as consultas,

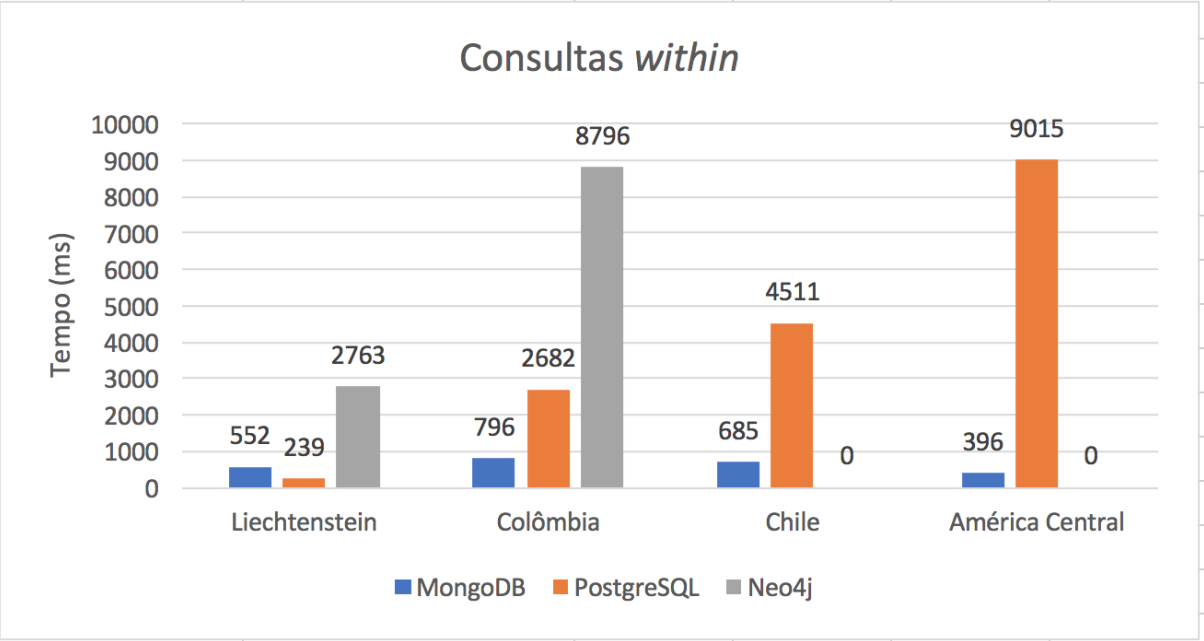

Figura 3.18: Gráfico agregando os resultados das consultas por amostra de dados.

Consultas *Intersects*

A Tabela 3.13 apresenta os resultados das consultas utilizando Liechtenstein. O PostgreSQL novamente se mostrou superior aos demais, 6.5 vezes em relação ao MongoDB e 17.3 vezes em relação ao Neo4j. Este último, por sua vez, conseguiu um resultado um pouco melhor nessas consultas, se mostrando apenas 2.6 vezes mais lento que o MongoDB, enquanto havia se comportado 5 vezes mais devagar nas consultas *Within* com a mesma amostra de dados.

Para a Colômbia, como é possível observar na Tabela 3.14, o MongoDB e o Neo4j apresentaram resultados muito similares, diferindo por apenas 58 ms. O PostgreSQL, novamente, mostrou-se superior aos dois para consultas *Intersect*, com uma execução 11.4 vezes mais rápida.

Tabela 3.13: Tabela contendo os resultados das execuções quentes das consultas *Intersects* para Liechtenstein.

	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*453 ms*	*204 ms*	*2397 ms*
Consulta 2	205 ms	35 ms	587 ms
Consulta 3	201 ms	32 ms	534 ms
Consulta 4	206 ms	31 ms	555 ms
Consulta 5	208 ms	32 ms	508 ms
Consulta 6	202 ms	30 ms	534 ms
Consulta 7	203 ms	30 ms	529 ms
Consulta 8	207 ms	32 ms	534 ms
Consulta 9	199 ms	30 ms	519 ms
Consulta 10	205 ms	30 ms	528 ms
Média sem primeira consulta	*204 ms*	*31 ms*	*536 ms*

Tabela 3.14: Tabela contendo os resultados das execuções quentes das consultas *Intersects* para a Colômbia.

	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*5005 ms*	*1409 ms*	*14456 ms*
Consulta 2	3495 ms	308 ms	3655 ms
Consulta 3	3520 ms	312 ms	3548 ms
Consulta 4	3548 ms	308 ms	3623 ms
Consulta 5	3517 ms	308 ms	3591 ms
Consulta 6	3586 ms	309 ms	3634 ms
Consulta 7	3543 ms	312 ms	3602 ms
Consulta 8	3527 ms	332 ms	3559 ms
Consulta 9	3570 ms	305 ms	3599 ms
Consulta 10	3534 ms	314 ms	3556 ms
Média sem primeira consulta	*3538 ms*	*312 ms*	*3596 ms*

Das consultas utilizando a Colômbia para as consultas utilizando o Chile houve pouca mudança no número de resultados retornado, de 435640 para 505400, o que fez com que os tempos não tivessem grandes oscilações entre as amostras, como pode-se observar na Tabela 3.15. A exceção é o Neo4j, que não é possível mensurar devido ao estouro de memória.

Por fim, os tempos decorridos nas consultas no *dataset* referente à America Central estão expressas na tabela 3.16 e reafirmam a superioridade do PostgreSQL frente ao MongoDB e ao Neo4j para as consultas *Intersect*. Nem mesmo o crescimento da base

Tabela 3.15: Tabela contendo os resultados das execuções quentes das consultas *Intersects* para o Chile.

	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*7648 ms*	*1512 ms*	-
Consulta 2	3712 ms	328 ms	-
Consulta 3	3789 ms	318 ms	-
Consulta 4	3799 ms	311 ms	-
Consulta 5	3755 ms	312 ms	-
Consulta 6	3867 ms	329 ms	-
Consulta 7	3798 ms	320 ms	-
Consulta 8	3690 ms	324 ms	-
Consulta 9	3767 ms	313 ms	-
Consulta 10	3730 ms	327 ms	-
Média sem primeira consulta	*3767 ms*	*320 ms*	-

de dados ou o grande aumento de resultados para 2528195 foi o suficiente para que o MongoDB executasse as *queries* em menos tempo que o PostgreSQL. A Figura 3.19 ilustra os resultados obtidos com as consultas *Intersects*.

Tabela 3.16: Tabela contendo os resultados das execuções quentes das consultas *Intersects* para a América Central.

	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*33946 ms*	*6679 ms*	-
Consulta 2	17388 ms	1631 ms	-
Consulta 3	16904 ms	1590 ms	-
Consulta 4	16844 ms	1586 ms	-
Consulta 5	16885 ms	1579 ms	-
Consulta 6	16870 ms	1570 ms	-
Consulta 7	16978 ms	1587 ms	-
Consulta 8	16788 ms	1599 ms	-
Consulta 9	16860 ms	1569 ms	-
Consulta 10	16963 ms	1589 ms	-
Média sem primeira consulta	*16942 ms*	*1589 ms*	-

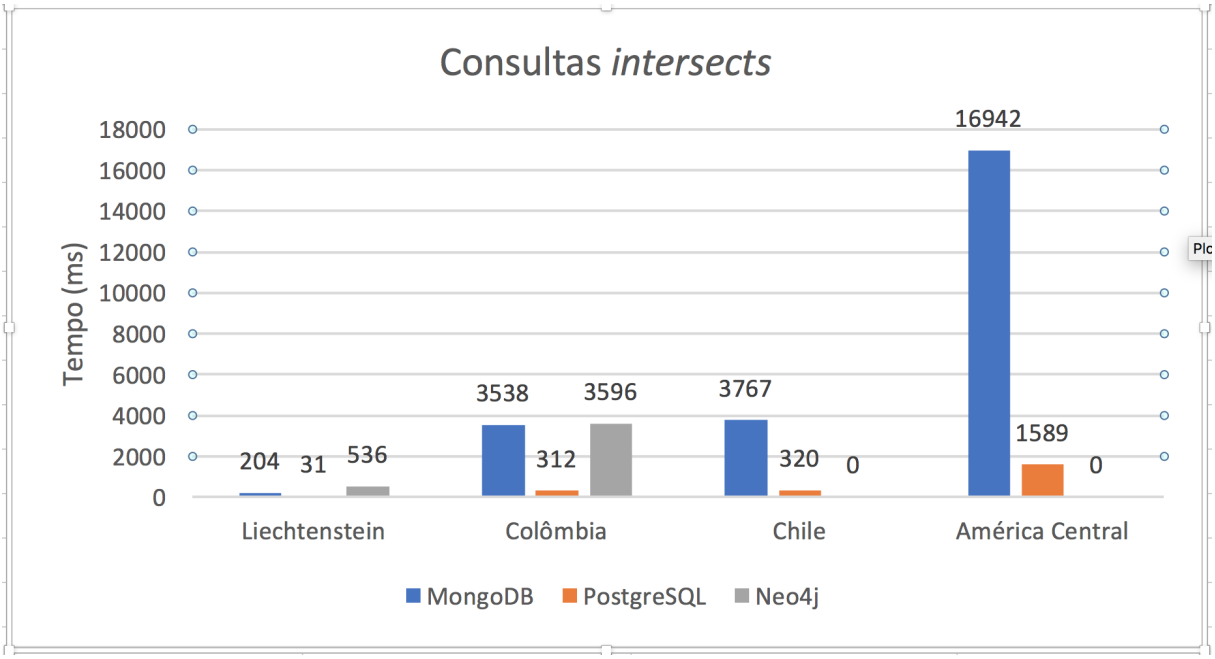

Figura 3.19: Gráfico agregando os resultados das *queries* por amostra de dados.

Consultas *Near*

Como já dito anteriormente, as consultas *Near* são similares às *Within*, porém retornam os resultados de forma ordenada da menor distância para a maior. Nestes testes, a consulta elaborada no Neo4j foi a mesma utilizada para as consultas *Within*, visto que a função *spatial.withinDistance()* presente no *Neo4j Spatial* já retorna, por padrão, os resultados de forma ordenada. A Tabela 3.17 contém os tempos de execução para Liechtenstein.

De forma semelhante às consultas *Within*, o PostgreSQL se comportou melhor para pequenos volumes de dados, executando 1.86 vezes mais rapidamente que o MongoDB e 4.71 vezes mais rapidamente que o Neo4j.

A partir da amostra de dados da Colômbia, porém, o MongoDB passa a frente do PostgreSQL, apresentando desempenho 2.15, 3.98 e 13.75 vezes superior com os *datasets* da Colômbia, Chile e América Central respectivamente. Os tempos de consulta estão presentes nas Tabelas 3.18, 3.19 e 3.20 respectivamente.

A Tabela 3.20, em especial, mostra novamente a degradação do desempenho no PostgreSQL conforme o volume de dados aumenta. Com 135340 resultados na América Central contra 195817 no Chile, houve aumento de, em média, 5045 ms, o que equivale a uma *performance* 2.11 vezes pior para o *dataset* de maior tamanho com menos resultados. Nessa mesma situação, o MongoDB foi 1.63 vezes mais rápido na maior amostra de dados com menor resultados.

Tabela 3.17: Tabela contendo os resultados das execuções quentes das consultas *Near* para Liechtenstein.

		Raio 12km	
	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*1901 ms*	*817 ms*	*6240 ms*
Consulta 2	1063 ms	581 ms	2798 ms
Consulta 3	1072 ms	584 ms	2727 ms
Consulta 4	1199 ms	585 ms	2743 ms
Consulta 5	1061 ms	591 ms	2765 ms
Consulta 6	1057 ms	582 ms	2771 ms
Consulta 7	1163 ms	582 ms	2712 ms
Consulta 8	1103 ms	597 ms	2768 ms
Consulta 9	1068 ms	584 ms	2780 ms
Consulta 10	1056 ms	588 ms	2800 ms
Média sem primeira consulta	*1094 ms*	*586 ms*	*2763 ms*

Tabela 3.18: Tabela contendo os resultados das execuções quentes das consultas *Near* para a Colômbia.

		Raio 12km	
	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*4667 ms*	*7910 ms*	*25559 ms*
Consulta 2	1472 ms	3101 ms	12370 ms
Consulta 3	1447 ms	3149 ms	8589 ms
Consulta 4	1449 ms	3098 ms	8495 ms
Consulta 5	1420 ms	3112 ms	8412 ms
Consulta 6	1414 ms	3126 ms	8345 ms
Consulta 7	1414 ms	3118 ms	8255 ms
Consulta 8	1454 ms	3102 ms	8224 ms
Consulta 9	1427 ms	3097 ms	8210 ms
Consulta 10	1458 ms	3113 ms	8264 ms
Média sem primeira consulta	*1443 ms*	*3113 ms*	*8796 ms*

A Figura 3.20 ilustra resumidamente os testes realizados nos parágrafos acima, o que facilita a visualização principalmente da tendência do PostgreSQL de degradar a *performance* conforme cresce o tamanho do banco de dados.

Tabela 3.19: Tabela contendo os resultados das execuções quentes das consultas *Near* para o Chile.

		Raio 12km	
	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*3582 ms*	*12057 ms*	-
Consulta 2	1188 ms	12057 ms	-
Consulta 3	1124 ms	4504 ms	-
Consulta 4	1104 ms	4534 ms	-
Consulta 5	1154 ms	4567 ms	-
Consulta 6	1108 ms	4487 ms	-
Consulta 7	1118 ms	4508 ms	-
Consulta 8	1138 ms	4515 ms	-
Consulta 9	1138 ms	4515 ms	-
Consulta 10	1135 ms	4520 ms	-
Média sem primeira consulta	*1133 ms*	*4514 ms*	-

Tabela 3.20: Tabela contendo os resultados das execuções quentes das consultas *Near* para a América Central.

		Raio 12km	
	MongoDB	**PostgreSQL**	**Neo4j**
Consulta 1	*2311 ms*	*26673 ms*	-
Consulta 2	660 ms	9584 ms	-
Consulta 3	652 ms	9508 ms	-
Consulta 4	700 ms	9625 ms	-
Consulta 5	725 ms	9770 ms	-
Consulta 6	714 ms	9572 ms	-
Consulta 7	711 ms	9504 ms	-
Consulta 8	694 ms	9485 ms	-
Consulta 9	723 ms	9299 ms	-
Consulta 10	678 ms	9684 ms	-
Média sem primeira consulta	*695 ms*	*9559 ms*	-

3.4 Observações

Durante a realização desse trabalho, alguns fatores não mensuráveis foram observados. Primeiramente, é notável a larga utilização do PostgreSQL em conjunto com o PostGIS para aplicações espaciais, principalmente aquelas que envolvem dados provenientes do

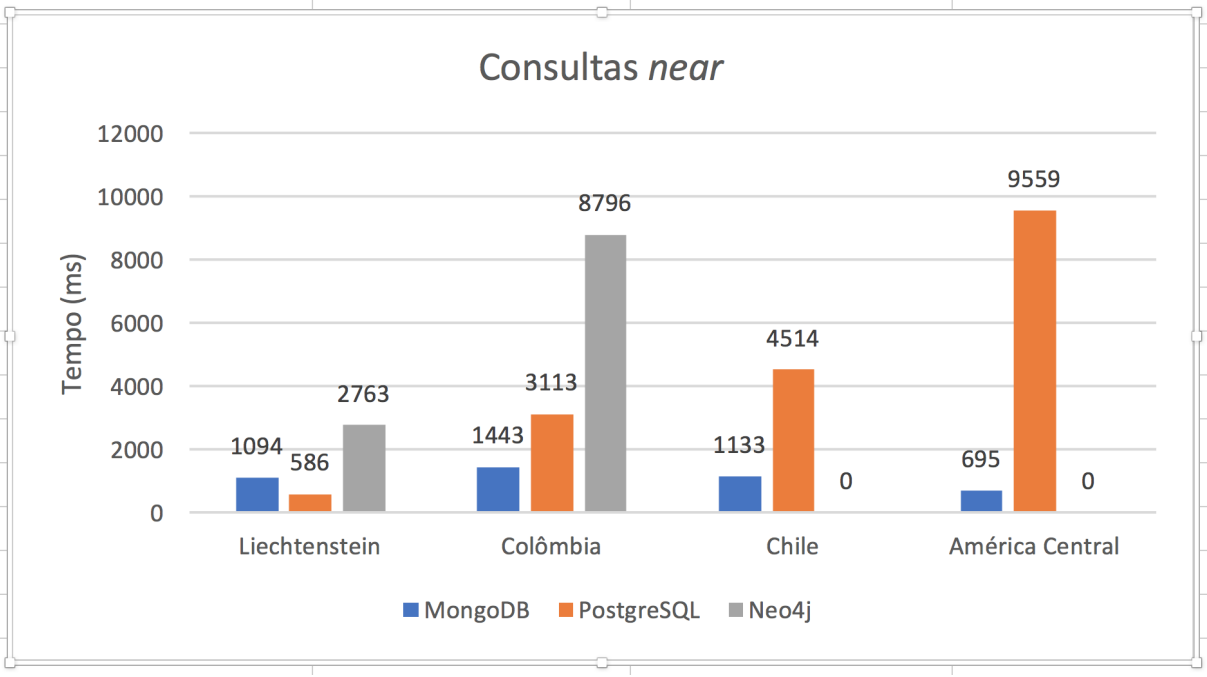

Figura 3.20: Gráfico agregando os resultados das *queries* por amostra de dados.

OpenStreetMap, enquanto o mesmo não pode ser dito para o MongoDB e o Neo4j. Dessa forma, há uma falta de suporte e artigos especializados acerca da utilização do OSM em conjunto com esses SGBD. Basicamente, além das documentações oficiais, todo o suporte vem de sites de questões como o *Stackoverflow* [24].

Nas últimas versões, o Neo4j trouxe algumas funcionalidades geoespaciais simples, longe da complexidade e das ferramentas presentes no PostGIS. A extensão *Neo4j Spatial* recebe poucas atualizações e melhoras, tendo tido seus ultimos *commits* realizados apenas para garantir o funcionamento com as novas versões do Neo4j. Na área destinada a problemas e dúvidas no *GitHub* oficial, fala-se há anos da implementação de melhorias e refatoração de métodos de indexação para corrigir os problemas de inserção e lentidão das consultas, porém nada foi de fato feito, e a ferramenta já demonstra sérios problemas para lidar com os casos de usos da atualidade.

O MongoDB trouxe melhorias relativas às funcionalidades espaciais, já presentes, nas últimas versões, o que mostra uma atenção dos desenvolvedores para com casos de uso envolvendo dados geoespaciais. Não há, porém, de maneira oficial, um suporte específico aos dados do OpenStreetMap, cabendo ao usuário utilizar alguma ferramenta de tratamento e inserção de terceiros ou desenvolver a sua própria.

Capítulo 4

Conclusão e Trabalhos Futuros

Antes apenas consumidores, os usuários da *web* passam a se tornar produtores de conteúdo, gerando dados dos mais diversos tipos como, por exemplo, informações geográficas e de localização. O OpenStreetMap é, provavelmente, um dos melhores exemplos disso, uma plataforma colaborativa onde os participantes cooperam para construir uma representação fiel do globo terrestre.

Nesse contexto, torna-se necessário a criação de ferramentas capazes de lidar com grandes volumes de dados flexíveis. Nesse sentido, surgem os Bancos de dados não relacionais, ou NoSQL. Dentre os diversos tipos de bancos não relacionais, destacam-se os orientados a documentos, sendo o MongoDB seu representante maior, e os orientados a grafos, com o avanço da popularidade do Neo4j.

Desse modo, esse trabalho visou investigar a viabilidade do emprego desses dois bancos não relacionais em conjunto com dados provenientes do OpenStreetMap, analisando o desempenho desses na inserção e na realização de consultas espaciais. Além disso, visou-se a comparação com um banco relacional amplamente utilizado para esses fins, o PostgreSQL.

Durante a realização dessa pesquisa, foi possível a aquisição de amostras de dados geográficos de diversos tamanhos originárias do OpenStreetMap, mais especificamente as informações referentes a Liechtenstein, Colômbia, Chile e América Central. Essas amostras foram, então, submetidas aos três SGBD, obtendo-se êxito em todas as inserções de Liechtenstein e Colômbia, enquanto Chile e América Central não puderam ser importadas para o Neo4j devido a restrições de memória física e comportamento indesejado das rotinas de inserção, mais precisamente a *API* de indexação *Lucene*. Nesse estágio do trabalho, o PostgreSQL se mostrou amplamente superior aos demais, levando muito menos tempo para importar a mesma quantidade de dados que os outros SGBD aqui estudados.

Com os bancos populados, foram realizadas consultas espaciais de tipos diferentes, com vistas a observar o desempenho de cada SGBD e como estes se comportavam com o aumento gradativo do volume de dados. Aqui, o MongoDB se mostrou uma poderosa

ferramenta frente ao tradicional PostgreSQL, apresentando um desempenho superior a este último em 2 dos 3 tipos de consultas realizadas. O PostgreSQL levou vantagem apenas nas consultas de interseção de dados, enquanto o MongoDB foi superior nas consultas *within* e *near*. O Neo4j, por sua vez, não se comportou bem, ficando de fora dos testes com o Chile e a América Central, e apresentando um desempenho inferior aos demais nos testes com Liechtenstein e Colômbia.

O MongoDB, no geral, é a melhor opção para a utilização em conjunto com amostras de dados do OpenStreetMap de tamanhos elevados. Embora este demore mais tempo para ser populado que o PostgreSQL, este atraso é recuperado durante a realização das consultas ao banco, visto que a inserção é realizada apenas uma vez enquanto as consultas são realizadas diversas vezes ao longo do ciclo de vida da aplicação.

No que tange ao suporte a dados geoespaciais de forma geral, o PostgreSQL, com sua extensão PostGIS, se consolida como o SGBD mais rico em funções e procedimentos, seguido pelo Neo4j em conjunto com o *Neo4j Spatial*, que apresenta uma boa gama de funcionalidades, embora mais enxuta que o PostGIS. O MongoDB, apesar de ser o ganhador no quesito desempenho das consultas, é o mais pobre em relação a funcionalidades espaciais implementadas, embora estas sejam feitas de forma nativa, e não através de extensões.

4.1 Trabalhos futuros

Algumas ideias de possíveis trabalhos futuros são:

- Desenvolvimento de novos *scripts* para a inserção de dados no MongoDB em uma linguagem mais rápida que *Python*, como *C++* ou até mesmo *Java*;
- Construção de uma interface web que permita a visualização dos dados geográficos armazenados nos bancos NoSQL em um mapa, possivelmente em *Javascript*
- Execução de consultas mais complexas, com caminhos e relações envolvidas;
- Teste de *performance* utilizando *sharding* no MongoDB e no PostgreSQL.

Referências

[1] *3.4.10. spatial functions.* https://neo4j.com/docs/developer-manual/current/cypher/functions/spatial. Accessed: 2018-11-10. 31

[2] *Amazon dynamodb.* https://aws.amazon.com/dynamodb/. Accessed: 2017-11-20. 17

[3] *Apache lucene 7.5.0 documentation.* http://lucene.apache.org/core/7_5_0/index.html. Accessed: 2018-11-10. 54

[4] *Chapter 1. introduction.* https://postgis.net/docs/manual-2.5/postgis_introduction.html. Accessed: 2018-01-10. 37

[5] *Cloud bigtable.* https://cloud.google.com/bigtable/. Accessed: 2017-11-20. 17

[6] *Databases and data access apis.* https://wiki.openstreetmap.org/wiki/Databases_and_data_access_APIs. Accessed: 2017-09-29. 28

[7] *Db-engines - knowledge base of relational and nosql database management systems.* https://db-engines.com/. Accessed: 2017-11-06. 16, 22

[8] *Db-engines ranking - trend popularity.* https://db-engines.com/en/ranking_trend. Accessed: 2017-11-06. ix, 23, 24

[9] *Dbms popularity broken down by database model.* https://db-engines.com/en/ranking_categories. Accessed: 2017-11-06. ix, 20, 22, 23

[10] *Geofabrik.* https://www.geofabrik.de/. Accessed: 2018-01-06. 35

[11] *Geojson.io.* http://geojson.io/. Accessed: 2018-20-11. 44

[12] *Geospatial queries.* https://docs.mongodb.com/manual/geospatial-queries/. Accessed: 2017-09-29. 27

[13] *Introduction to mongodb.* https://docs.mongodb.com/master/introduction/. Accessed: 2017-09-29. 24

[14] *Lucene tutorial.* http://www.lucenetutorial.com/basic-concepts.html. Accessed: 2018-11-10. 54

[15] *Mongosm.* https://github.com/iandees/mongosm. Accessed: 2017-09-29. 28

[16] *Museu de arte da pampulha*. https://pt.wikipedia.org/wiki/Museu_de_Arte_da_Pampulha. Accessed: 2017-09-30. ix, 10

[17] *Neo4j spatial*. https://github.com/neo4j-contrib/spatial. Accessed: 2018-01-06. 31

[18] *Neo4j spatial*. https://github.com/neo4j-contrib/spatial. Accessed: 2017-11-25. 32

[19] *Osmosis*. https://wiki.openstreetmap.org/wiki/Osmosis#Downloading. Accessed: 2018-02-10. 37

[20] *Out of memory for importing osm data*. https://github.com/neo4j-contrib/spatial/issues/273. Accessed: 2018-02-20. 53

[21] *Postgis*. https://postgis.net/. Accessed: 2017-11-05. 14

[22] *Project voldemort: A distributed database*. http://www.project-voldemort.com/voldemort/. Accessed: 2017-11-20. 17

[23] *Refraction research*. http://www.refractions.net/. Accessed: 2017-11-05. 14

[24] *Stack overflow*. https://stackoverflow.com/. Accessed: 2018-12-10. 66

[25] *Stats*. https://wiki.openstreetmap.org/wiki/Stats. Accessed: 2017-10-02. 10

[26] *Vector based gis*. https://geogra.uah.es/patxi/gisweb/GISModule/GIST_Vector.htm. Accessed: 2017-09-30. ix, 5

[27] *What is cassandra?* http://cassandra.apache.org/. Accessed: 2017-11-20. 17

[28] *What is raster data?* http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=What_is_raster_data? Accessed: 2017-09-30. ix, 6

[29] *What is the difference between the 'spatial' built into neo4j and the community spatial plugin?* https://community.neo4j.com/t/what-is-the-difference-between-the-spatial-built-into-neo4j-and-the-community-spa 159. Accessed: 2018-11-10. 32

[30] *Why is columnstore important?* https://mariadb.com/resources/blog/why-is-columnstore-important/. Accessed: 2018-12-10. ix, 17

[31] Abadi, Daniel J, Peter A Boncz e Stavros Harizopoulos: *Column-oriented database systems*. Proceedings of the VLDB Endowment, 2(2):1664–1665, 2009. 17

[32] Abramova, Veronika e Jorge Bernardino: *Nosql databases: Mongodb vs cassandra*. Em *Proceedings of the International C* Conference on Computer Science and Software Engineering*, páginas 14–22. ACM, 2013. 12, 15, 16, 17, 21

[33] Amat, Guillermo, Javier Fernandez, Alvaro Arranz e Angel Ramos: *Using open street maps data and tools for indoor mapping in a smart city scenario*. 2014. 32

[34] Amirian, Pouria, Anahid Basiri, Guillaume Gales, Adam Winstanley e John McDonald: *The next generation of navigational services using openstreetmap data: The integration of augmented reality and graph databases.* Em *OpenStreetMap in GIScience*, páginas 211–228. Springer, 2015. 32

[35] Anderson, Jennings, Robert Soden, Kenneth M Anderson, Marina Kogan e Leysia Palen: *Epic-osm: A software framework for openstreetmap data analytics.* Em *2016 49th Hawaii International Conference on System Sciences (HICSS)*, páginas 5468–5477. IEEE, 2016. 32

[36] Baas, BLP: *Nosql spatial–neo4j versus postgis.* Tese de Mestrado, 2012. 33, 53

[37] Bogdan George Tudorica, Cristian Bucur: *A comparison between several nosql databases with comments and notes.* Computational Intelligence and AI in Games, IEEE Transactions on, páginas 1–5, 2011. 14

[38] Câmara, Gilberto, Antônio Miguel Monteiro, Suzana Druck Fucks e Marília S Carvalho: *Spatial analysis and gis: a primer.* National Institute for Space Research. Brazil, 2004. 5

[39] Casanova, Marco Antonio, Gilberto Câmara, Clodoveu Davis, Lúbia Vinhas e GR de Queiroz: *Banco de dados geográficos.* MundoGEO Curitiba, 2005. 4

[40] Cattell, Rick: *Scalable sql and nosql data stores.* Acm Sigmod Record, 39(4):12–27, 2011. 16, 26, 27

[41] Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes e Robert E Gruber: *Bigtable: A distributed storage system for structured data.* ACM Transactions on Computer Systems (TOCS), 26(2):4, 2008. 14

[42] Codd, Edgar F: *A relational model of data for large shared data banks.* Communications of the ACM, 13(6):377–387, 1970. 11

[43] DeCandia, Giuseppe, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall e Werner Vogels: *Dynamo: amazon's highly available key-value store.* ACM SIGOPS Operating Systems Review, 41(6):205–220, 2007. 14

[44] Douglas, Korry e Susan Douglas: *PostgreSQL: a comprehensive guide to building, programming, and administering PostgresSQL databases.* SAMS publishing, 2003. 13

[45] Drake, Joshua D e John C Worsley: *Practical PostgreSQL.* " O'Reilly Media, Inc.", 2002. 13

[46] Elmasri, Ramez: *Fundamentals of database systems.* Pearson Education India, 2008. 11

[47] Elmasri, Ramez, Shamkant B Navathe e Rinaldo de Oliveira Morais: *Sistemas de banco de dados.* 2005. 4, 11, 12

[48] Francia, Steve: *MongoDB and PHP*. " O'Reilly Media, Inc.", 2012. ix, 26

[49] Goodchild, Michael F: *Citizens as sensors: the world of volunteered geography*. GeoJournal, 69(4):211–221, 2007. 1

[50] Güting, Ralf Hartmut: *An introduction to spatial database systems*. The VLDB Journal—The International Journal on Very Large Data Bases, 3(4):357–399, 1994. 4

[51] Huang, Hongcheng e Ziyu Dong: *Research on architecture and query performance based on distributed graph database neo4j*. Em *Consumer Electronics, Communications and Networks (CECNet), 2013 3rd International Conference on*, páginas 533–536. IEEE, 2013. 28, 29

[52] Kuznetsov, Sergey D e Andrey V Poskonin: *Nosql data management systems*. Programming and Computer Software, 40(6):323–332, 2014. 14, 17, 21, 25, 26, 27

[53] Leavitt, Neal: *Will nosql databases live up to their promise?* Computer, 43(2):12–14, 2010. 14, 16, 21

[54] Ma, Ding, Mats Sandberg e Bin Jiang: *Characterizing the heterogeneity of the openstreetmap data and community*. ISPRS International Journal of Geo-Information, 4(2):535–550, 2015. 8, 11

[55] Macedo, José Alexandre e Claudia Boeres: *Graph database*. 30

[56] Maia, Daniel Cosme Mendonça: *Arquitetura de armazenamento de dados para sistemas de informação geográfica voluntária utilizando banco de dados nosql baseado em documento*. http://ppgi.unb.br/normatives, 2016. 18, 21

[57] Martinez, Federico e Ariel Aizemberg: *Bases de datos de grafos con manejo de datos espaciales*. Em *Simposio Argentino de GRANdes DAtos (AGRANDA 2015)-JAIIO 44 (Rosario, 2015)*, 2015. 33

[58] McCreary, Dan e Ann Kelly: *Making sense of nosql*. Shelter Island: Manning, páginas 19–20, 2014. 20

[59] Milani, André: *PostgreSQL-Guia do Programador*. Novatec Editora, 2008. 13

[60] Miller, Justin J: *Graph database applications and concepts with neo4j*. Em *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, volume 2324, 2013. 19, 31

[61] Momjian, Bruce: *PostgreSQL: introduction and concepts*, volume 192. Addison-Wesley New York, 2001. 13

[62] MongoDB: *Mongodb architecture guide*. White Paper. 24, 25, 26, 27, 29

[63] MongoDB: *Top 5 considerations when evaluating nosql databases*. White Paper. 24

[64] Moniruzzaman, A B M e Syed Akhter Hossain: *Nosql database: New era of databases for big data analytics - classification, characteristics and comparison*. International Journal of Database Theory and Application, 06(04), 2013. ix, 2, 14, 16, 17, 18, 21, 22

[65] Neis, Pascal e Dennis Zielstra: *Recent developments and future trends in volunteered geographic information research: The case of openstreetmap*. Future Internet, 6(1):76–106, 2014. 11

[66] Obe, Regina e Leo Hsu: *Postgis in action*. GEOInformatics, 14(8):30, 2011. 14

[67] O'reilly, Tim: *What is web 2.0*, 2005. 1

[68] Padhy, Rabi Prasad, Manas Ranjan Patra e Suresh Chandra Satapathy: *Rdbms to nosql: reviewing some next-generation non-relational database's*. International Journal of Advanced Engineering Science and Technologies, 11(1):15–30, 2011. 14, 24, 25

[69] Pokorny, Jaroslav: *Nosql databases: a step to database scalability in web environment*. International Journal of Web Information Systems, 9(1):69–82, 2013. 15, 16

[70] Pourabbas, Elaheh: *Geographical Information Systems: Trends and Technologies*. CRC Press, 2014. 5, 6, 7

[71] Raj, Pethuru: *Handbook of research on cloud infrastructures for big data analytics*. IGI Global, 2014. 14

[72] Ramm, Frederik, Jochen Topf e Steve Chilton: *OpenStreetMap: using and enhancing the free map of the world*. UIT Cambridge Cambridge, 2011. 1

[73] Roberto, Evandro, Maristela Terto de Holanda, Aletéia PF de Araújo e Marcio Victorino: *Geographic data in a graph oriented database: A study with neo4j and postgresql*. Em *Information Systems and Technologies (CISTI), 2017 12th Iberian Conference on*, páginas 1–6. IEEE, 2017. 33

[74] Robinson, Ian, Jim Webber e Emil Eifrem: *Graph Databases: New Opportunities for Connected Data*. " O'Reilly Media, Inc.", 2015. ix, 29, 30

[75] Rodriguez, Marko A e Peter Neubauer: *Constructions from dots and lines*. Bulletin of the American Society for Information Science and Technology, 36(6):35–41, 2010. ix, 18, 19

[76] Santos, Erik Urbanski e Marcos Alberto Lopes da Silva: *Abordagem ao banco de dados orientado a grafos neo4j em um nível empresarial*. 28

[77] Schmid, Stephan, Eszter Galicz e Wolfgang Reinhardt: *Performance investigation of selected sql and nosql databases*. AGILE 2015–Lisbon, páginas 9–12, 2015. 27

[78] Schmid, Stephan, Eszter Galicz e Wolfgang Reinhardt: *Wms performance of selected sql and nosql databases*. Em *Military Technologies (ICMT), 2015 International Conference on*, páginas 1–6. IEEE, 2015. 33

[79] Sehra, Sukhjit Singh, Jaiteg Singh e Hardeep Singh Rai: *A systematic study of openstreetmap data quality assessment*. Em *Information Technology: New Generations (ITNG), 2014 11th International Conference on*, páginas 377–381. IEEE, 2014. 10

[80] Sharma, Neeraj, Liviu Perniu, Raul F Chong, Abhishek Iyer, Chaitali Nandan, Adi Cristina Mitea, Mallarswami Nonvinkere e Mirela Danubianu: *database fundamentals*. IBM Canada, páginas 96–101, 2010. 12

[81] Stančić, Baldo, Vlado Cetl e Mario Mađer: *Testing the potential of volunteered geographic information in the case of openstreetmap in croatia*. Kartografija i geoinformacije: časopis Hrvatskoga kartografskog društva, 13(22):48–69, 2014. 8

[82] Technology, Neo: *The neo4j developer manual*. https://neo4j.com/docs/developer-manual, 2016. 28, 29, 31

[83] Uchoa, Helton Nogueira e Paulo Roberto Ferreira: *Geoprocessamento com software livre*. Publicação eletrônica. Rio de Janeiro–RJ, 2004. 14

[84] Van Bruggen, Rik: *Learning Neo4j*. Packt Publishing Ltd, 2014. 18, 29

[85] Wang, Guoxi e Jianfeng Tang: *The nosql principles and basic application of cassandra model*. Em *Computer Science & Service System (CSSS), 2012 International Conference on*, páginas 1332–1335. IEEE, 2012. 14

[86] Wang, Guoxi e Jianfeng Tang: *The nosql principles and basic application of cassandra model*. Em *Computer Science & Service System (CSSS), 2012 International Conference on*, páginas 1332–1335. IEEE, 2012. 15, 16