UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

NICHOLAS DE AQUINO LAU

Análise Comparativa de Performance e Modelagem entre um Banco de Dados Relacional e um de Documentos para a base de dados do BDQueimadas (INPE)

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia da Computação

Orientador: Prof[a]. Dr. Renata de Matos Galante

Porto Alegre
2021

“Computers are useless.

They can only give you answers.”

— PABLO PICASSO

AGRADECIMENTOS

Primeiramente gostaria de agradecer a toda minha família, em especial meu pai, Richard Lau, minha mãe, Claudia Simone de Aquino Lau, meu irmão, Walter Lau Neto, minha tia, Nelza Maria de Aquino e minha avó, Astrid Seibert Lau, que estiveram presentes e me apoiaram incondicionalmente durante todas etapas de minha vida e não mediram esforços para que eu realizasse mais este sonho. Também agradeço a minha namorada Ana Luísa Fabris de Moura que esteve presente durante todos esses anos, me aconselhando em cada decisão e me oferecendo apoio sempre que necessário. Gostaria de deixar um agradecimento especial ao meus amigos (que são quase família) Matheus Tavares Frigo, Luís Augusto Weber Mercado e Vilmar Antonio Fonseca, pela grande amizade desde o início da faculdade. Amizade esta que não se limitou apenas ao âmbito acadêmico, mas passou também pelas mais diversas ocasiões da vida e que com certeza sem eles eu não teria chegado até aqui. Por fim, agradeço também a todos os professores que fizeram parte de minha trajetória acadêmica, me proporcionando um ensino de muita qualidade e excelência. Em especial, agradeço a minha orientadora Renata Galante por toda a ajuda e suporte fornecidos, não só nesta etapa final da faculdade, como também em todo o decorrer dela.

RESUMO

Com o passar dos anos e o avanço da tecnologia, o panorama dos dados mudou. Cada vez mais ouve-se falar de *Big Data*, *Internet of Things*, de um grande volume e uma vasta variedade de dados, sejam eles gerados em nuvem, em dispositivos móveis ou nas redes sociais. Para acompanhar essas mudanças, os bancos de dados *Not Only SQL* (NoSQL), os quais são não relacionais, evoluíram a fim de fornecer um melhor suporte aos desenvolvedores a criar sistemas de banco de dados que suportem o armazenamento dessas novas informações. Logo, é de extrema importância o estudo entre os diferentes tipos de bancos de dados, tanto dos novos bancos NoSQL, como os antigos bancos SQL, a fim de compreender os diferentes cenários os quais cada um se encaixa melhor. Este trabalho visa comparar o desempenho de dois bancos de dados, um NoSQL e outro SQL, considerando aspectos como: modelagem e projeto, criação do banco de dados, inserção de dados, consultas, espaço de armazenamento, dentre outros. No geral, o banco de dados SQL obteve uma melhor performance nas consultas, no entanto, para os outros aspectos apresentou algumas desvantagens e estas devem ser consideradas ao decidir-se por um sistema ou outro para uma determinada aplicação.

Palavras-chave: Data modeling. NoSQL. SQL. MySQL. MongoDB. performance.

A Comparative Analysis of Performance and Modeling between a Relational Database and a Document Database for the database BDQueimadas(INPE)

ABSTRACT

As the years have passed and technology has advanced, the data landscape has changed. Nowadays we hear more and more about Big Data, Internet of Things, a large volume and vast variety of data, whether generated in the cloud, on mobile devices, or in social networks. Not Only SQL (NoSQL) databases, which are non-relational, have kept up with these changes and evolved to better support developers in creating database systems that support the storage of this new type of information. In this context, it is crucial to assess these different databases, both the new NoSQL databases and the old SQL databases, in order to understand the different scenarios that each one best fits into. This works aims to compare the performance of two databases, one NoSQL database and the other a SQL database, considering aspects such as (i) modeling and design, (ii) database creation, (iii) data insertion, (iv) queries and (v) the required storage space. In general, the SQL database performed better in queries, however, for the other aspects it had some drawbacks and these should be considered when deciding between one system and another for a particular application.

Keywords: Data modeling. NoSQL. SQL. MySQL. MongoDB. performance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IoT	Internet of Things
NoSQL	Not Only SQL
SQL	Structured Query Language
INPE	Instituto Nacional de Pesquisas Espaciais
ISAM	Index Sequential Access Method
DBMS	Database Management System
SGBD	Sistema de Gerenciamento de Banco de Dados
IMS	Information Management System
CODASYL	Conference/Committee on Data Systems Languages
RDBMS	Relational Database Management System
GFS	Google File System
RAC	Real Application Clusters
AWS	Amazon Web Services
MMS	MongoDB Management Service
DDL	Data Definition Language
DML	Data Manipulation Language
FRP	Fire Radiative Power

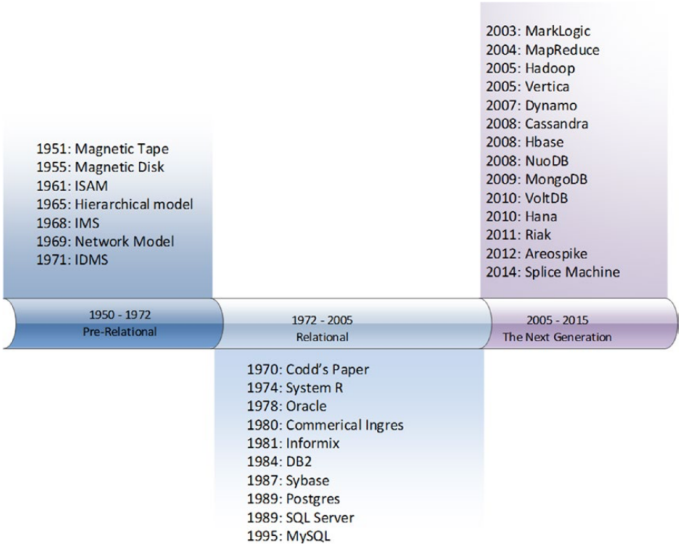
SUMÁRIO

1 INTRODUÇÃO

Atualmente, ouvimos cada vez mais falar em *Big Data*, *Internet of Things* (IoT), *Web 2.0*, computação em nuvem, de um grande volume e uma grande variedade de dados. Esses dados, sejam eles gerados em nuvem, dispositivos móveis ou até mesmo pelas redes sociais, necessitam de uma nova forma de modelagem e armazenamento. As aplicações modernas também exigem uma alta escabilidade e concorrência devido ao grande volume de dados que estão sendo processados atualmente, o que, os já conhecidos bancos relacionais tendem a não fornecer de uma forma eficiente. Como podemos ver em (HAN et al., 2011), com o avanço da tecnologia durante os últimos anos, principalmente nas áreas relacionadas a dados, houve a necessidade de aprimorar os banco de dados a fim de que eles acompanhassem todas essas mudanças.

Grandes empresas, como Google[1] e Amazon[2], tiveram que lidar logo com essa necessidade devido ao seu pioneirismo no processamento de um grande volume de dados. Mesmo os bancos de dados relacionais comerciais mais caros da época não forneciam a escalabilidade necessária para suprir as necessidades atuais. Para isso, essas empressas passaram a desenvolver seus próprios modelos de bancos de dados, como o Google BigTable[3] e o Amazon Dynamo[4], os quais serviram de inspiração para o surgimento de um novo modelo mais tarde (STRAUCH; SITES; KRIHA, 2011).

Figura 1.1: Linha do tempo dos principais lançamentos e inovações de banco de dados.



Fonte: Harrison (2015)

[1] https://www.google.com
[2] https://www.amazon.com
[3] https://cloud.google.com/bigtable
[4] https://aws.amazon.com/dynamodb

Para acompanhar essa necessidade, os bancos de dados *Not Only SQL* (NoSQL) surgiram, a fim de fornecer um melhor suporte aos desenvolvedores a criarem banco de dados que suportam o armazenamento dessas novas informações. Diferentemente dos bancos de dados relacionais, esse novo modelo é não relacional e possui diferentes características como: altamente portável, alta escalabilidade, flexível, não suporta a *Structured Query Language* (SQL), entre outras.

Como podemos ver na Figura 1.1, houve uma explosão dessa nova geração de banco de dados, onde muitos sistemas de banco de dados novos surgiram no mercado em um curto período de tempo. Cada sistema possui características distintas, com diferentes maneiras de modelar os dados a serem armazenados. Entre esses novos modelos para representação dos dados, podemos listar quatro tipos:

- Base de dados chave-valor;
- Base de dados de documentos;
- Base de dados orientada a colunas;
- Base de dados de grafos.

Com uma enorme variedade de sistemas e modelos de bancos de dados no mercado, incluindo tanto a nova gerção, NoSQL, como os antigos bancos de dados relacionais, muitas vezes os desenvolvedores ficam em dúvida sobre qual sistema de banco de dados utilizar. Por isso, é tão importante compararmos e estudarmos os diferentes cenários os quais cada um destes modelos se encaixa.

Este trabalho visa comparar dois diferentes sistemas e modelos de bancos de dados, comparando uma abordagem mais moderna, NoSQL, com uma mais tradicional, a relacional. Para este fim, será comparado um banco de dados de documentos, através do MongoDB, com um banco de dados relacional, através do MySQL. Para obtermos diferentes perspectivas de uso para cada caso é necessário compreender e comparar os diferentes aspectos de cada modelo, como: modelagem e projeto, criação do banco de dados, consultas, espaço de armazenamento, uso de memória, dentre outros.

Para este estudo comparativo entre ambos os modelos foi escolhido um conjunto de dados fornecido pelo INPE (Instituto Nacional de Pesquisas Espaciais) (INPE, 2019), o qual monitora focos de incêncidos e queimadas através de imagens via satélite em diferentes regiões do Brasil, assim como de alguns países da América do Sul, como Argentina, Bolívia, Chile, Colômbia, Equador, entre outros. Estes dados são atualizados diariamente e possui registros desde, aproximadamente, o ano de 1999, logo possui uma

grande quantidade de dados.

Após os experimentos realizados para o presente trabalho pode-se observar que, no geral, o banco de dados SQL obteve uma melhor performance nas consultas. No entanto, para os outros aspectos analisados ele apresentou algumas desvantagens e estas devem ser consideradas ao decidir-se por um sistema ou outro para uma determinada aplicação. Todos esses pontos serão discutidos e abordados em mais detalhes nos próximos Capítulos.

O texto está organizado da seguinte maneira, o Capítulo 2 descreve a fundamentação teórica, dando uma breve introdução ao leitor dos principais fatos históricos assim como uma base conceitual sobre banco de dados. Além disso, também descreve os trabalhos relacionados, mostrando suas similaridades e diferenças. No Capítulo 3, é apresentada a proposta de modelagem para os dados escolhidos considerando as duas abordagens de banco de dados: relacional e de documentos. No Capítulo 4 temos os testes comparativos de desempenho entre os dois sistemas de bancos de dados escolhidos. O Capítulo 5 finaliza o trabalho e apresenta sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os fundamentos sobre banco de dados, assim como as principais características e diferenças entre os modelos abordados neste trabalho. Na Seção 2.1, descrevemos um breve histórico sobre banco de dados no geral, passando por diversos modelos e abordagens propostas que sugiram com o passar dos anos e os avanços da tencologia em seus mais diferentes aspectos. Na Seção 2.2, discutiremos um pouco mais a fundo sobre a abordagem NoSQL, citando um pouco sobre suas principais características e motivações, assim como os diferentes modelos propostos dentro desta nova abordagem. Na Seção 2.3, discutiremos sobre o SGBD NoSQL (MongoDB) escolhido, apresentando um pouco sobre sua história e suas principais características. Na Seção 2.4, apresentaremos alguns trabalhos relacionados, abordando seus principais pontos e relacionando-os com o presente trabalho.

2.1 Perspectiva histórica dos modelos e sistemas de bancos de dados

Um Sistema de Gerenciamento de Bancos de Dados (SGBD), conhecido também em inglês como *Database-Management System* (DBMS), tem como principal objetivo armazenar um conjunto de dados e prover maneiras eficientes de consultar esses dados quando necessário. Como podemos ver em (SILBERSCHATZ et al., 1997), com o passar dos anos, informações dos mais diversos tipos e formatos começaram a fazer parte da vida das grandes empresas. Por causa dessa necessidade de armazenar e consultar estes dados de forma eficiente, diversas técnicas e conceitos de modelagem foram propostos pelos cientistas de computação do mundo todo, propondo assim, diferentes sistemas para gerenciar esses dados. A seguir, analisaremos, de uma perspectiva mais histórica, os modelos de banco de dados utilizados nos útlimos anos pelos sistemas de gerenciamento de bancos de dados.

Como podemos ver em (HARRISON, 2015), coletar e organizar informações foi um dos fatores que fizeram a civilização humana e a tecnologia se desenvolverem. Podemos observar isso desde muito cedo, quando por exemplo, foram criados os primeiros dicionários e enciclopédias, os quais possuíam as suas informações bem estruturadas e organizadas. Outro exemplo dos primórdios da organização de dados que podemos citar são as bibliotecas e também a organização de arquivos indexados.

A partir do surgimento dos primeiros computadores eletrônicos, que tiveram como

precedente a Segunda Guerra Mundial, tivemos uma pequena evolução nos banco de dados, os quais passaram a utilizar fitas magnéticas que possibilitavam avançar e voltar através do conjunto de dados nela presente. Em torno de 1950, surgiu o disco magnético giratório, o que possibilitou também o acesso rápido e direto a dados individuais gravados nestes discos. Os primeiros métodos de indexação também surgiram nessa época, como o ISAM, de *Index Sequential Access Method*.

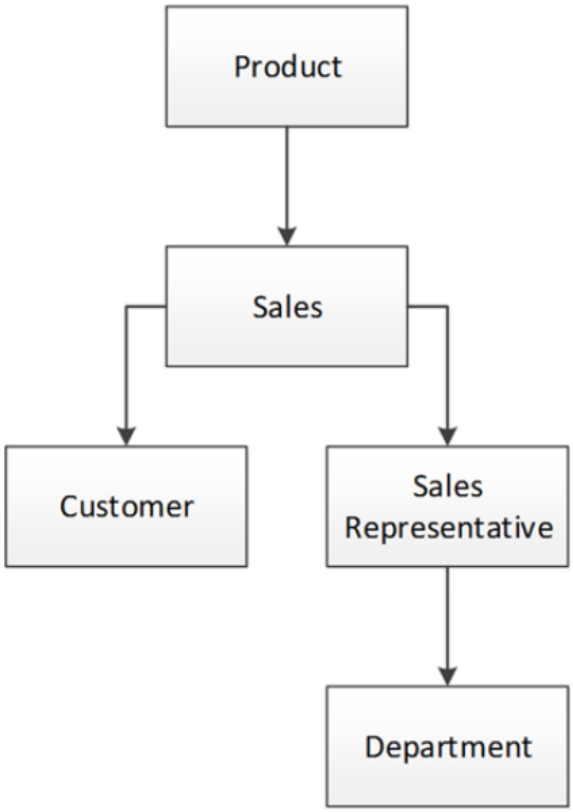
A primeira grande evolução nos bancos de dados surgiu a partir de um problema na produtividade das aplicações: cada nova aplicação havia a necessidade de reescrever seu próprio código para o tratamento de dados, isso acabava gerando um retrabalho desnecessário para todas as empresas. Além disso, alguns problemas de concorrência e de corrupção de dados também começaram a surgir, uma vez que o tratamento desses casos exigia o uso de técnicas mais complexas e sofisticadas, e que eram difíceis de serem replicadas para cada aplicação. Para tentar resolver este problema idealizou-se a modularização dos componentes de uma aplicação, externalizando assim a responsabilidade do tratamento de dados da aplicação. Esta nova camada responsável pelo tratamento de dados ficou conhecida como o SGBD e teria como principal objetivo reduzir o retrabalho por parte dos programadores das aplicações, assim como garantir a integridade dos dados. Os primeiros modelos de SGBDs que surgiram na época foram o modelo em rede (TAYLOR; FRANK, 1976) e o modelo hierárquico (TSICHRITZIS; LOCHOVSKY, 1976). Abaixo algumas considerações sobre cada modelo:

- **Modelo hierárquico**: Esse modelo foi criado pensando em uma das estruturas mais simples e mais conhecidas na sociedade, que é a estrutura hierárquica. Essa estrutura pode ser encontrada tanto naturalmente no mundo animal, como dentro de empresas e organizações governamentais. Este modelo foi introduzido pela IBM e pôde ser encontrado em seu próprio SGBD, o IBM IMS (*Information Management System*). Ele é organizado em uma estrutura de árvore, assim cada registro é conectado através de ligações a outros registros, de modo que cada registro filho tenha apenas um parente, enquanto cada parente pode ter um ou mais filhos. Esta estrutura pode ser observada na Figura 2.1. Para consultar dados neste banco de dados toda a árvore deve ser percorrida começando pelo seu nodo raíz. Assim, acaba-se tendo uma estrutura simples, porém nada flexível, uma vez que cada relacionamento está limitado a uma relação de um-para-muitos. Algumas desvantagens desse modelo são:
    - Devido a estrutura restrita do modelo, operações de inserção e remoção se

tornam complexas;

- Para representar ligações de muitos-para-muitos é preciso realizar a duplicação de dados, tornando ela uma representação ineficiente e inapropriada;
- Dependendo dos tipos de dados o modelo hierárquico pode não ser tão simples de ser modelado, uma vez que este modelo pode não ser natural para algumas estruturas.

Figura 2.1: Modelo hierárquico.

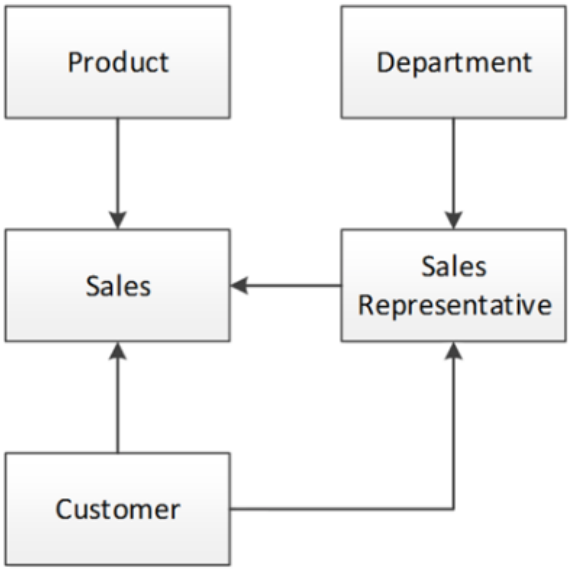


Fonte: Harrison (2015)

- **Modelo em rede**: Esse modelo foi adotado pelo CODASYL (*Conference/Committee on Data Systems Languages*), por isso muitas vezes esse modelo é conhecido como o modelo CODASYL. Como podemos ver em (BACHMAN, 1969), o modelo em rede apresentou um modelo mais flexível na maneira de representar os registros e suas relações. Este modelo pode ser visto com um grafo, em que cada registro é um nodo e suas relações são expressas pelos arcos, permitindo assim que cada registro tenha múltiplas relações. Sua estrutura pode ser observada na Figura 2.2.

Esses modelos ficaram conhecidos como modelos navegacionais. Ambos esses modelos prevaleceram durante o período dos computadores *mainframe* (computadores de grande porte da época, os quais possuíam um grande desempenho, uma alta escabilidade

Figura 2.2: Modelo em rede.



Fonte: Harrison (2015)

e segurança, e eram dedicados ao processamento de um grande volume de dados). No entanto, eles ainda possuíam alguns pontos negativos como, serem pouco flexíveis na sua estrutura e também na capacidade de realizar buscas. Foram modelos que acabaram se concentrando mais no processamento das quatro operações básicas dos banco de dados, conhecidas como CRUD, de *Create*, *Read*, *Update* e *Delete*. Logo, quando eram necessárias operações de buscas, principalmente aquelas mais complexas, era necessário a criação de códigos complexos e repetitivos. Como as exigências do mercado começaram a mudar e essas consultas mais elaboradas começaram a se tornar cada vez mais frequentes, novamente o mundo da tecnologia, no contexto de banco de dados, se encontrava em uma situação crítica e então uma segunda revolução nesse meio foi necessária.

Diante de todas dificuldades listadas acima, uma nova proposta de modelo foi feita por Edgar Codd, ela pode ser vista em (CODD, 2002). Este modelo definiu as principais ideias do modelo de banco de dados relacionais, que se tornou um dos modelos mais importantes e mais usados para sistemas de banco de dados durante décadas. Os bancos de dados relacionais ofereceram uma base sólida para tratar derivabilidade, redundância e consistência das relações. Além disso, criaram uma estrutura que possibilita o uso por pessoas que não são especiliazadas no assunto, uma vez que busca separar a implementação lógica da física. *Os futuros usuários de grandes bancos de dados devem ser protegidos de terem que saber como os dados estão organizados na máquina (na representação interna)*, (CODD, 2002). Ele buscou definir como um conjunto de dados deve ser apresentado ao usuário, ao invés de como ele deve ser armazenado no disco.

Nos modelos antigos a representação dos dados nas bases de dados era correspondente ao armazenamento físico na base de dados, ao invés de uma representação lógica que facilitasse a compreensão. Alguns conceitos chaves do modelo relacional são: Tuplas, que correspondem as linhas; Relações, que são coleções de tuplas distintas e correspondem a uma tabela; Restrições, que garantem a consistência do banco de dados e Operações, como *joins*, *projections*, *unions* e etc. Através deste modelo uma linha de uma tabela deve ser facilmente identificada e acessada eficientemente por uma única chave-valor, e todas colunas desta linha devem ser dependentes desta chave.

No entanto, o modelo relacional não havia definido como um banco de dados deveria lidar com requisições concorrentes para alteração dos dados. Essas alterações simultâneas levariam a problemas de consistência e integridade dos dados. Foi então que Jim Gray definiu um modelo de transação que foi amplamente aceito e passou a ser utilizado pela maioria das implementações de sistemas de bancos de dados. Como podemos ver em (GRAY; REUTER, 1992), este modelo de transação ficou conhecido como *ACID transactions*, onde uma transação deveria possuir as seguintes características:

- **Atomicidade**: A transação deve ser indivisível, isto é, ou todas transações são aplicadas a base de dados ou nenhuma é;
- **Consistência**: O banco de dados permanece consistente tanto antes quanto depois da execução de uma transação;
- **Isolamento**: Mesmo que muitas transações possam ser feitas ao mesmo tempo, uma transação não deve perceber os efeitos de outras transações que estão ocorrendo simultaneamente;
- **Durabilidade**: Quando uma transação é finalizada com sucesso, é esperado que essas mudanças persistam mesmo em caso de falha no sistema ou no hardware.

A partir de então teve-se uma explosão de sistemas de bancos de dados que passaram a implementar o modelo relacional. A IBM desenvolveu o System R, que foi pioneiro no uso da linguagem SQL. Também tivemos o INGRES (HELD; STONEBRAKER; WONG, 1975), que foi desenvolvido por Gerald Held, Mike Stonebreaker e companhia, o qual utilizava a linguagem QUEL, ou seja, não utilizava a linguagem SQL. Na mesma época, Larry Ellison fundou uma das empresas mais conhecidas, inclusive nos dias atuais, a *Oracle Corporation*, a qual desenvolveu os primeiros sistemas de bancos de dados relacionais disponíveis comercialmente. Alguns anos mais tarde, a linguagem SQL passou a ser a preferida de todos clientes que utilizavam os bancos de dados relacionais,

passando assim a ser a linguagem dominante no mercado. Após essa estabilização no mercado, diversos outros sistemas surgiram nas décadas seguintes como: Microsoft SQL Server (MICROSOFT, 1994), MySQL (ORACLE, 1995), Sybase (SYBASE, 1988), entre outros. Todos eles seguiam os princípios básicos definidos nos anos anteriores, ou seja, utilizavam o modelo relacional, a linguagem SQL e seguiam o modelo de transações ACID.

Os bancos de dados relacionais permanecem em alta até os dias de hoje, no entanto, novamente o passar dos anos levou a mudanças em diversos aspectos da tecnologia e consequentemente das tecnologias envolvidas, como linguagens de programação, banco de dados, entre outros.

Como vimos anteriormente, a Google e a Amazon foram as empresas pioneiras em lidar com uma grande quantidade de dados. Mesmo os RDBMS, do inglês *Relational Database Management System*, mais conhecidos e expansivos fornecidos pela Oracle não forneciam a escalabilidade e a agilidade necessária que a grande quantidade de dados exigia dessas empresas. O Google acabou criando o *Google File System* (GFS), que formou uma base para seu sistema de armazenamento. Além disso, também criou o algoritmo *MapReduce*, usado para o processamento distribuído e paralelo de dados. Todos esses projetos serviram como base para o projeto *Hadoop*, criado nos anos seguintes e que serviu como uma tecnologia que possibilitou o gerenciamento desse grande volume de dados. Nesse período a Oracle tentou criar um sistema de banco de dados relacionais que se adaptasse as novas necessidades de escalabilidade exigidas, o *Oracle Real Applications Clusters* (RAC), porém este sistema era economicamente inviável para as empresas.

Outra alternativa que surgiu na época para os problemas de escalabilidade dos bancos relacionais, foi uma técnica conhecida como *Sharding*. Esta técnica, que pode ser observada na Figura 2.3, consiste em dividir cada registro de uma tabela em diversos bancos de dados e cada banco de dados pode ser guardado em diferentes nodos computáveis. Para cada um desses bancos de dados é aplicada uma técnica conhecida como *shared-nothing*, ou seja, cada fragmento do banco de dados (composto por alguns dos registros), é independente dos outros fragmentos do bancos de dados e funciona de forma autônoma. Desta forma, os sistemas de bancos de dados que aplicam essa técnica podem lidar com um maior número de *requests* (pedidos), dos usuários. Esta técnica é uma forma de escalonamento conhecido como escalonamento horizontal ou *scale-out*. O escalonamento horizontal permite uma escalabilidade quase que ilimitada, permitindo assim que os bancos de dados lidem com um grande número de dados e uma grande carga de trabalho. No

sentido oposto, temos o escalonamento vertical, também conhecido como *scaling up*, que envolve uma atualização no hardware utilizado nas máquinas, adicionando mais memória RAM ou mais CPUs.

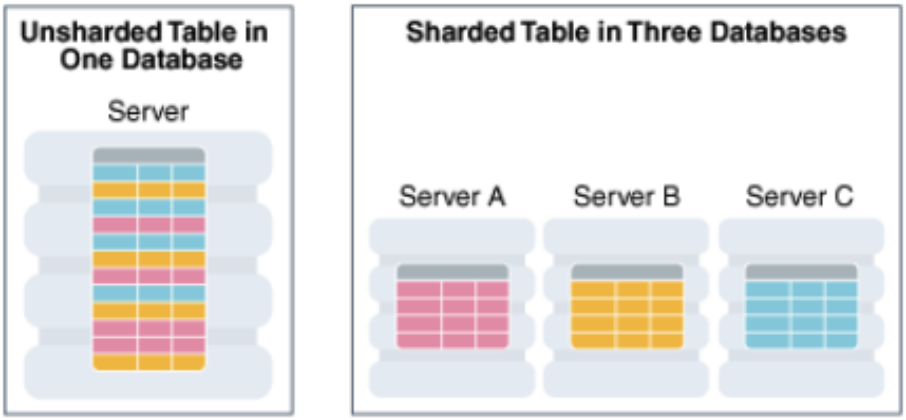
As principais vantangens que podemos observar no *Sharding* são:

- Redução da carga sobre cada servidor. Como cada servidor vai possuir uma pequena parte dos dados, cada consulta vai ser distribuída entre os diferentes servidores.
- Essa redistribuição entre servidores também reduz o espaço de armazenamento, tanto de indexes como de registros, utilizado em cada máquina pelo banco de dados.
- Consultas mais rápidas, uma vez que cada consulta terá que percorrer apenas a porção de dados correspondente ao servidor que esta está sendo realizada.
- Desacoplamento de um ponto único de falha. Se um dos nodos falha, o restante dos dados distribuídos entre os outros nodos permanecem funcionando.

E as principais desvantagens são:

- Complexidade de implementação;
- Muitas vezes a fragmentação do banco de dados acaba se tornando desbalanceada com o passar do tempo, levando aos mesmos problemas apresentados por bancos de dados que não utilizam a técnica;
- Retornar o banco de dados ao momento antes de aplicar a técnica de *sharding* é uma tarefa muito complexa;
- Aumento do tempo das consultas, quando essas necessitam passar por mais de um dos nodos/servidores.

Figura 2.3: Arquitetura de um banco de dados que aplica a técnica de *sharding*.



Fonte: Sundarappa (2018)

No entanto, como vimos ateriormente, os termos *Big Data*, *IoT*, *Cloud Computing*, passaram a fazer parte do dia a dia da maioria das empresas e *startups* ligadas a tecnologia. As aplicações deixaram de ser no padrão conhecido de cliente-servidor e passaram a ser *web-based*, ou seja, cujos os servidores da aplicação e o armazenamento de dados passaram a ser acessíveis via internet. O conceito de Web 2.0 começou a se espalhar rapidamente, houve uma revolução na forma como a internet era usada e vista. Esse novo conceito passou a fazer parte da vida dos usuários e da maioria das aplicações. Assim, as soluções propostas anteriormente já não davam mais conta da necessidade das empresas e dos usuários. Como podemos ver em (MONIRUZZAMAN; HOSSAIN, 2013), organizações passaram a coletar cada vez mais um grande volume de dados, sejam eles gerados pelos usuários, sistemas ou sensores. As principais dificuldades em lidar com esses dados se davam basicamente pelas seguintes características do *Big Data*, (VENKATRAMAN et al., 2016):

- Alta velocidade dos dados, ou seja, um fluxo muito intenso e contínuo, vindo de diferentes fontes;
- Variedade de dados, através de dados estruturados, semi-estruturados ou sem qualquer estrutura;
- Grande volume de dados. Bancos de dados com um grande tamanho, atigindo terabytes ou até petabytes de dados;
- Complexidade dos dados, com dados distribuídos em diferentes locais ou *data-centers*.

As consultas de dados baseadas em SQL acabaram perderdendo a eficiência nesse novo cenário, e possuiam uma variedade de limitações quanto as necessidades das empresas em lidar com a análise do *Big Data*, como vimos acima. Além disso, soluções baseadas em *clusters* surgiram como uma solução para grandes bancos de dados, porém o SQL não suportava tal arquitetura. Devido a todos os fatores comentados anteriormente, surgiu a necessidade do uso de uma nova tecnologia para lidar com tal situação.

Os banco de dados NoSQL surgiram para contornar tais problemas. Esses bancos foram projetados para possuir características que facilitavam o gerenciamento do *Big Data*. Os bancos NoSQL tiveram como base as ideias propostas por Michael Stonebreaker em (STONEBRAKER et al., 2018). Nele Michael sugere que a arquitetura relacional já não mais se encaixa nas necessidades atuais, como o grande volume de dados, e que talvez uma única arquitetura possa não ter a solução ótima para todos os casos. Assim

foram propostas duas arquiteturas, a *H-Store* e a *C-Store*, as quais foram as bases para os novos bancos não relacionais no futuro, que possuíam algumas características dos bancos relacionais mas divergiam na sua arquitetura. Nos anos seguintes teve-se uma explosão no crescimento de bancos de dados que seguiam essa arquitetura não relacional. Esses bancos possuiam diferentes modelos e possuíam como principais características uma maior flexibilidade e escalabilidade horizontal. Além disso, buscavam uma forma fácil de armazenar e recuperar os dados independente da estrutura que eles possuíssem. Nos bancos relacionais isso já não era possível devido a sua rígida arquitetura. Apesar de todas essas características, inicialmente esses bancos ainda possuíam algumas falhas, como a falta de consistência nos dados. No entanto, com o aumento exponencial do uso de bancos de dados NoSQL diversas aspectos estão sendo aprimorados a fim de superar e corrigir estas falhas.

2.2 NoSQL

Como vimos anteriormente os bancos NoSQL, de *Not Only SQL*, são aqueles que não utilizam as regras tradicionais do modelo relacional. Também vimos que foram diversos os fatores que influenciaram o surgimento de um novo paradigma para a modelagem de dados, onde o principal motivo se deu pelo atual cenário, em que as tecnologias de *Big Data*, *Cloud Computing* e *IoT* estão em alta e o modelo NoSQL surgiu para atender de uma melhor forma tais abordagens. Em (PAGÁN; CUADRADO; MOLINA, 2015), podemos observar diferentes pontos que este novo modelo proporcionou:

- Evitar a complexidade desnecessária imposta pelo modelo relacional. Essa estrutura rígida do modelo relacional proporciona algumas vantagens, como diferentes funcionalidades e a consistência de dados. No entanto, muitas vezes isso é mais do que o necessário para alguns casos de uso, o que acaba levando a uma sobrecarga desnecessária de trabalho;
- Os bancos de dados NoSQL possuem uma escalabilidade horizontal e não dependem de um único hardware que esteja sempre disponível. Pode-se adicionar e remover *clusters* com um esforço muito menor comparado ao aplicar as técnicas de *sharding* em bancos de dados relacionais;
- Evitar o mapeamento necessário e custoso que havia entre a estrutura dos dados apresentada nas linguagens de programação orientada a objetos e a estrutura exi-

gida pelo modelo relacional. A orientação a objetos surigu como um novo paradigma de programação, em que se mistura os atributos e seus comportamentos em uma só entidade, com a finalidade de aumentar o encapsulamento. Por se diferenciar da abordagem tradicional, em que os dados e a lógica eram separados entre si, os bancos de dados relacionais dificultavam a modelagem dessas novas entidades. A maioira dos desenvolvedores que utilizavam a orientação a objetos não se sentiam satisfeitos com a abordagem relacional devido a disparidade entre as duas abordagens. Assim, a maioria dos bancos de dados NoSQL foram criados, também, com o objetivo de facilitar o armazenamento dessas novas estruturas.

- Complexidade e custo para criação de *clusters* de base de dados. Característica a qual reforça novamente a ideia de que o custo para adicionar novos *clusters* é muito menor comparado ao custo de aplicar as técnicas de *sharding*.
- Comprometer a confiabilidade em troca de performance. De acordo com as características e finalidades de algumas aplicações se opta por uma troca entre a confiabilidade e a performance.

No entanto, como os bancos NoSQL tinham como principal característica fornecer uma maior flexibilidade e performance, eles não forneciam as garantias do modelo de transação ACID, vistas na seção anterior.

Em (BREWER, 2000), Eric Brewer propôs o teorema CAP, o qual foi amplamente adotado por sistemas de bancos de dados distribuídos nos anos seguintes. CAP é um acrônimo para:

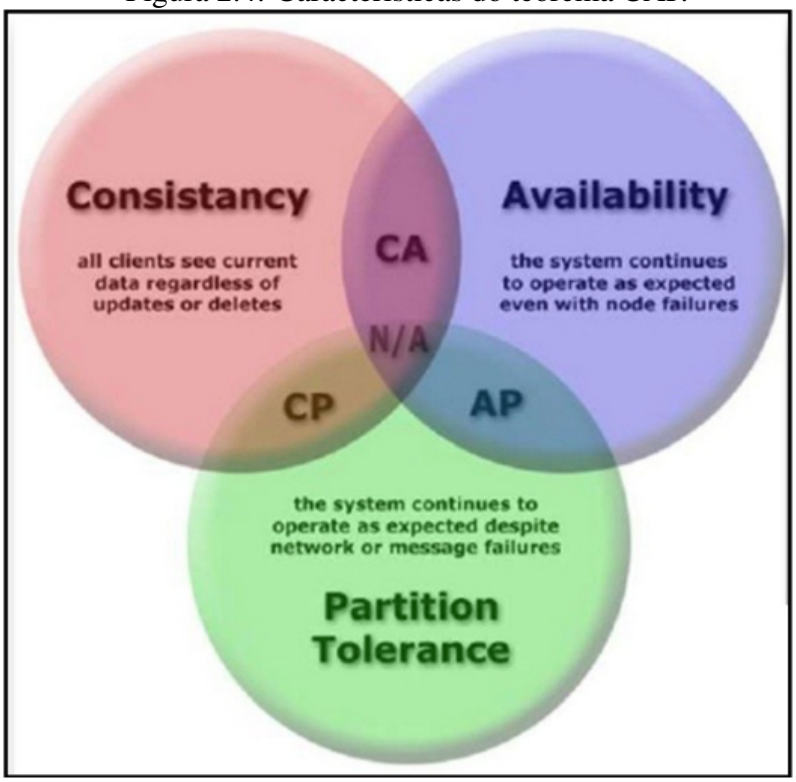
- *Consistency*: Todos usuários do banco de dados tem uma visão idêntica do dado (mesma versão) em um dado instante;
- *Availability*: Mesmo em um evento de falha o banco de dados continua operacional;
- *Partition tolerance*: O banco de dados pode se manter operacional mesmo se houver uma falha de rede entre dois segmentos do sistema distribuído.

As quais eram as principais características deste teorema. Além disso, estava proposto que um sistema de dados distribuído poderia possuir apenas duas das três características listadas acima, como podemos ver na Figura 2.4

Assim, os bancos NoSQL poderiam ser classificados preliminarmente da seguinte forma (HAN et al., 2011):

- *Consistency and Availability (CA)*: O banco de dados está interessado mais na disponibilidade e consistência dos dados, utilizando assim de técnicas de replicação

Figura 2.4: Características do teorema CAP.



Fonte: Moniruzzaman e Hossain (2013)

para garantir tais características;

- *Consistency and Partition tolerance (CP)*: O banco guarda os dados em nodos distribuídos e garante a consistência desses dados, porém não oferece um bom suporte a disponibilidade dos dados;
- *Availability and Partition tolerance (AP)*: O banco garante a disponibilidade e a tolerância de partições ao manter o banco disponível mesmo com falhas na comunicação entre os nodos. Quando o problema da partição for resolvido, os nodos serão sincronizados entre si, porém não é garantido que todos nodos terão os mesmos dados.

Em (MONIRUZZAMAN; HOSSAIN, 2013), podemos ver que a maioria dos bancos de dados NoSQL flexibilizaram a consistência dos dados a fim de obter uma melhor disponibilidade e particionamento. Surgiu assim o teorema BASE, que é o acrônimo de ***B****asically* ***a****vailable*, ***S****oft-state* e ***E****ventual consistency*. Ele é o oposto do ACID e possui características derivadas do teorema CAP.

No Capítulo 1, vimos que existem quatro modelos para o tratamento de dados NoSQL, abaixo descreveremos cada um deles com mais de detalhes:

- Chave-valor: É uma das arquiteturas mais simples, onde os registros (valores) são

salvos associados a uma chave. Como sua estrutura é simples, as consultas tendem a serem mais rápidas comparadas as bases de dados relacionais, além de suportar uma grande quantidade de dados e concorrência.

- Orientada a colunas: Usa uma tabela como modelo de dados, mas não permite associação entre as tabelas. Os dados são armazenados por colunas, cada coluna é um índice do banco de dados, processa consultas concorrentemente (i.e. cada coluna é tratada como um processo). Em geral, a vantagem desse modelo de dados é uma aplicação mais adequada na agregação e no armazenamento de dados.
- Documentos: A arquitetura em documentos é muito parecida com a de chave-valor, porém o "valor"em um documento é armazenado geralmente em formato JSON, BSON ou XML, os quais são formatos semi-estruturados. Esses formatos facilitam a modelagem para os desenvolvedores, já que eles, no geral, se mapeiam diretamente para os objetos das aplicações modernas, tornando assim a modelagem natural. Além disso, os documentos suportam indíces em valores secundários, diferentemente do modelo chave-valor. Outro ponto em que se diferem é que no modelo de documentos tanto as chaves como os valores são totalmente pesquisáveis.
- Grafos: A arquitetura em grafos é totalmente baseada na teoria dos grafos. Por ser focada em grafos, suas estruturas principais são os nodos, as arestas e as propriedades. É focada totalmente nas relações entre os dados e por isso acaba tendo aplicações mais específicas como, por exemplo, redes sociais. A busca é otimizada ao usar técnicas livres de índices, como a baseada na adjacência dos nodos. Essa estrutura permite armazenamentos rápidos, em conforme com as diretrizes do teorema ACID e com suporte ao *rollback*.

Na Tabela 2.1 abaixo podemos ver um comparativo entre as diferentes arquiteturas explicadas acima de acordo com alguns aspectos em específico.

Tabela 2.1: Comparativo entre os modelos NoSQL.

	Performance	**Escalabilidade**	**Flexibilidade**	**Complexidade**	**Funcionalidade**
Chave-valor	alta	alta	alta	nenhum	variável
Colunas	alta	alta	moderado	baixo	mínimo
Documentos	alto	variável(alto)	alto	baixo	variável (baixo)
Grafos	variável	variável	baixo	moderado	teoria dos grafos

Fonte: Pagán, Cuadrado e Molina (2015)

Para o presente trabalho a arquitetura que irá representar os bancos NoSQL será a orientada a documentos, através do MongoDB. A escolha desta arquitetura se deu por algumas de suas características gerais, como a boa performance, a escalabilidade e uma

alta flexibilidade, uma vez que, com o avanço da tecnologia os satélites podem apresentar uma maior precisão, aumentando assim o número de focos registrados. Além disso, novas tecnologias podem passar a coletar uma maior diversidade de dados, tornando-se necessário a possibildiade de adicionar registros com propriedades diferentes das já existentes. Ao analisar os outros modelos, logo descarta-se a opção da base de dados baseada em grafos, tendo em vista que ela possui uma baixa flexibilidade e uma estrutura pouco vantajosa para a base de dados escolhida. Os bancos de dados do tipo chave-valor possuem características semelhantes aos bancos de documentos, porém se diferem no momento da realização das consultas. Em bancos do tipo chave-valor as consultas são realizadas apenas pelas chaves, enquanto para o banco de documentos elas podem ser realizadas também pelos valores. Para o presente trabalho a possibilidade de buscas por valores é interessante, logo elimina-se também os bancos do tipo chave-valor das opções. Já os bancos de dados baseados em colunas possuem uma flexibilidade moderada, no entanto devido a suas outras características, como uma boa performance e também boa escalabilidade, seriam um interessante caso de estudo para o futuro.

2.3 MongoDB

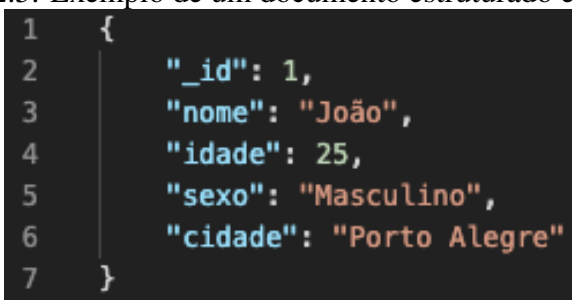
O MongoDB (MONGODB, 2007) é um dos sistemas de gerenciamento de banco de dados orientado a documentos mais usados hoje em dia. Por ser bastante popular, possui uma boa documentação que pode ser facilmente encontrada na internet. Ele foi criado em 2007 por Dwight Merriman, Eliot Horowitz e Kevin Ryan. Ambos trabalharam na empresa *DoubleClick* e durante este período passaram por dificuldades quanto a escalabilidade e agilidade dos bancos de dados, quando estes lidavam com um grande número de dados. Frustrados com esses problemas eles resolveram criar um banco de dados que os solucionava. Assim surgiu o MongoDB, ele foi desenvolvido visando um fácil desenvolvimento e escalonamento para os banco de dados. Além disso, possui três versões:

- *MongoDB Community Server*, o qual é de graça e disponível para Windows, Linux e OS X;
- *MongoDB Enterprise Server*, com fins comerciais e está disponível como parte do pacote para empresas.
- *MongoDB Atlas*, o qual está disponível como um serviço totalmente *on-demand*.

Ele funciona com o Amazon Web Services (AWS), Microsoft Azure e Google Cloud.

Ao invés dos dados serem salvos em tabelas com linhas e colunas, o MongoDB é, como dito anteriormente, um banco de dados orientado a documentos, ou seja, os dados são estruturados em formatos como o JSON. Esses dados são então salvos no banco de dados no formato BSON, o qual é uma representação binária do JSON. Na Figura 2.5 podemos ver um exemplo de um registro estruturado em JSON.

Figura 2.5: Exemplo de um documento estruturado em JSON.



Fonte: Autor.

Desde o início, o MongoDB foi construído em uma arquitetura que facilitasse a escala, ou seja, uma estrutura que permite que diversas máquinas trabalhem em conjunto a fim de criar sistemas rápidos e eficientes ao lidarem com uma grande quantidade dados.

Algumas das principais caraterísticas do MongoDB são:

- Consultas *ad hoc*: Suporta consultas por campos, por *range* e expressões regulares. O MongoDB suporta consultas *ad hoc* indexando documentos BSON e utilizando uma linguagem de consulta própria.
- *Schema-less*: O MongoDB é uma base de dados sem esquema, assim acaba se tornando mais flexível do que o tradicional modelo relacional. Isso acaba reduzindo o trabalho de mapeamento entre um dado e a estrutura do banco de dados.
- Indexação: Os campos do documento podem ser indexados com índices primários ou secundários.
- Replicação: Fornece alta disponibilidade através do uso de conjunto de réplicas de dados. Um conjunto de réplicas consiste em duas ou mais cópias dos dados. Cada membro do conjunto pode agir como réplica primária ou secundária a qualquer momento. Todas leituras e escritas são feitas na réplica primária e na réplica secundária se mantém apenas a cópia da primária. Quando a réplica primária falha, o conjunto de réplicas utiliza do método da eleição para determinar qual das réplicas secundárias deve tornar-se a primária.

- Balanço de carga: O MongoDB escala horizontalmente utilizando da técnica de *auto-sharding*. Ele pode rodar em diversos servidores, balanceando os dados automaticamente ou duplicando eles para manter o sistema rodando mesmo em caso de alguma falha no hardware.
- *MongoDB Management Service* (MMS): É uma ferramenta que permite monitorar as bases de dados, assim como as máquinas onde são guardadas. Permite também a criação de *backups* dos dados. Além disso, disponibiliza métricas de performance, ajudando a otimizar as implementações feitas. Consegue também criar alertas personalizados os quais ajudam a identificar problemas antes que eles atinjam de fato a aplicação.

2.4 Trabalhos Relacionados

Nesta seção discutiremos sobre alguns dos trabalhos relacionados, abordando alguns dos principais pontos levantados em cada trabalho e relacionando-os com o presente trabalho.

Leavitt (2010) cita diferentes limitações que os bancos relacionais apresentam frente aos problemas atuais, como escalabilidade, flexibilidade e a complexidade de implementação. Em contrapartida, os bancos NoSQL proporcionam diferentes características que tendem a ser mais vantajosas para o cenário atual, o qual apresenta um grande volume de dados sendo estes, em sua maioria, não estruturados. Em geral, os bancos de dados NoSQL são mais eficientes nas consultas, realizando elas de forma mais rápida que os bancos relacionais. No entanto esses bancos ainda apresentam alguns contrapontos, como apresentar problemas de confiabilidade, já que não seguem as características do ACID e pouca familiaridade por parte das empresas. Os bancos NoSQL eram uma aposta para a época, eles seriam largamente utilizados por aplicações que lidassem com dados desestruturados e que necessitavam de escalabildiade. Além disso, apostava-se que os bancos NoSQL seriam utilizados apenas em casos especiais, não substituindo assim os tradicionais bancos relacionais, mas sim se tornando uma melhor alternativa para certos casos.

Em (LI; MANOHARAN, 2013) os autores fizeram uma comparação entre os diferentes bancos de dados, tanto NoSQL quanto SQL, avaliando a performance deles para um banco de dados composto por elementos chave-valor. Nele podemos observar diversos resultados, como os tempos para leitura, escrita, busca, remoção e também o número

de operações realizada em cada banco de dados testado. Ao final, os autores mostram que para cada um dos pontos testados há diferentes resultados na performance, e mesmo que os bancos de dados NoSQL sejam, geralmente, mais otimizados para armazenarzenamento de chave-valor, nem sempre os bancos NoSQL performaram melhor que o banco SQL testado, levando assim a conclusão de que os bancos de dados devem ser testados, não somente na frase de projeto da aplicação, mas também em intervalos regulares após essa fase, buscando sempre se adaptar para o modelo mais adequado.

Hecht e Jablonski (2011) comparam diversos modelos NoSQL através de uma pesquisa orientada a casos de usos. Neste artigo são apresentadas as dificuldades em escolher um banco de dados NoSQL para certos casos de uso. Como base para a comparação são usados o modelo dos dados, o suporte fornecido as consultas, o particionamento, a replicação e os controles de concorrência disponível para cada banco testado. Cada banco de dados NoSQL é específico para certas ocasiões, portanto é necessário que seja feito um estudo de caso para cada aplicação visando identificar qual dos diferentes modelos melhor se encaixa. Hecht e Jasblonksi também citam alguns pontos importantes que o desenvolvedor deve considerar para identificar o modelo mais adequado para uma certa aplicação. São eles:

- Analisar qual modelo evita transformações ou mapeamentos dos dados para as arquiteturas disponíveis, evitando assim uma complexidade de implementação desnecessária;
- Quais tipos de consultas devem ser suportadas pelo banco de dados;
- Além disso, deve-se analisar quais características serão priorizadas: alta performance, alta disponibilidade ou consistência.

Ao final do artigo são apresentadas algumas conclusões em respeito aos casos de uso de cada um dos modelos de bancos de dados NoSQL. No geral, os bancos de dados baseados em chave-valor são indicados para operações simples e muito rápidas. Os bancos baseados em documentos oferecem uma boa flexibilidade e boas opções de consulta. Os bancos baseados em colunas são interessantes para bases de dados muito grandes e que têm de ser escalonadas para grandes dimensões. Já os bancos de dados baseados em grafos devem ser usados em domínios nos quais as entidades são tão importantes quanto as relações entre elas, como por exemplo, nas redes sociais.

Podemos observar assim que a área de banco de dados está em constante evolução e atualmente os bancos de dados NoSQL ganharam visibilidade devido a suas caraterís-

ticas que armonizam com o cenário atual. Além disso, percebemos que não existe uma fórmula padrão para se definir o modelo ideal para cada aplicação, fazendo-se necessário um estudo de caso particular para cada cenário.

3 MAPEAMENTO DOS DADOS SOBRE FOCOS DE QUEIMADAS NO BRASIL PARA OS MODELO LÓGICOS DE BANCOS DE DADOS RELACIONAIS E DE DOCUMENTOS

Neste capítulo são apresentadas e descritas as duas abordagens propostas para o mapeamento lógico e físico para o BDQueimadas (INPE, 2019), o qual apresenta um conjunto de dados fornecido pelo Instituto Nacional de Pesquisas Espaciais (INPE) que monitora focos de incêncidos e queimadas através de imagens via satélite de diferentes países, sendo a primeira delas o banco de dados relacional e a segunda em banco de dados baseados em documentos. A escolha pelo modelo relacional e do modelo de documentos, em detrimento de outros, se deu para que fosse possível a comparação entre a abordagem mais tradicional e popular entre as aplicações (modelo relacional) e o modelo de documentos, o qual é um dos modelos que vem crescendo no mercado nos últimos anos e possui diversas características, as quais foram vistas anteriormente, interessantes para o cenário atual. Além disso, o modelo de documentos permite armazenar os dados como um único documento, ou seja, de uma forma mais próxima ao arquivo que é disponibilizado para download pelo site do INPE.

Inicialmente este capítulo descreve o domínio do problema, que são os dados sobre os focos de incêndio e queimadas de determinadas regiões. Em seguida, são especificados os modelos lógicos para cada uma das abordagens propostas, a relacional (Seção 3.2) e a de documentos (Seção 3.3).

3.1 Descrição do Domínio

O domínio escolhido, como dito anteriormente, foi o de monitoramento de queimadas e incêndios fornecido pelo Instituto Nacional de Pesquisas Espaciais (INPE).

Cada foco de incêncido é registrado, armazenado e disponibilizado para visualização no site BDQueimadas (INPE, 2019). Para cada foco, são guardadas informações pontuais de cada queimada/incêndio, onde são armazenadas informações como a data e a hora em que determinado foco foi registrado, o satélite que o registrou, suas coordenadas geográficas (latitude e longitude), o país, o estado e o município em que o foco se encontra, assim como algumas informações mais específicas como, a quantos dias sem chuva essa região se encontrava até a detecção do foco, o risco de fogo previsto para o dia da

detecção do foco, o valor da precipitação no dia até o momento da detecção do foco e o *Fire Radiative Power* (FRP), o qual é a taxa média de calor emitida por um incêndio.

3.2 Modelagem para Banco de Dados Relacional

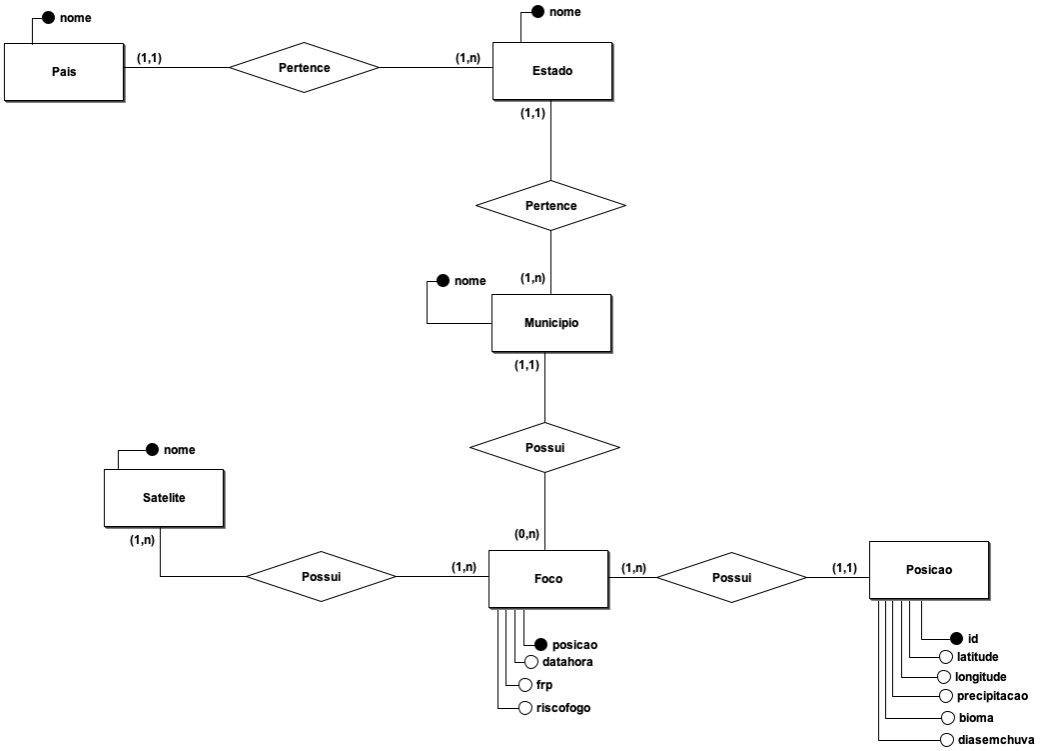
Para representar os dados em uma abordagem relacional é necessário a construção de um modelo que possua as principais entidades capazes de representar todo conjunto de dados fornecido pelos INPE, o qual é composto por diversas informações providas pelos satélites, assim como demonstrar de forma mais clara e objetiva como estas entidades se relacionam. O objetivo desta seção é realizar a modelagem dos dados para a abordagem relacional, a qual será realizada e abordada através de um diagrama entidade-relacionamento. Além disso, também serão abordadas as restrições de integridade aplicadas para o modelo proposto.

Primeiramente é construído o modelo conceitual, no qual é possível identificar as principais entidades e seus relacionamentos para que posteriormente seja possível criar um banco de dados relacional, armazenando informações e realizando consultas. Tal modelo pode ser visto na Figura 3.1. Através deste modelo conceitual desenvolveu-se o modelo lógico, representado na Figura 3.2. Neste modelo podemos observar as seguintes entidades:

- **Pais**: Representa o país em que cada foco está localizado.
- **Estado**: Representa os estados de cada país em que cada foco está localizado.
- **Municipio**: Representa os municípios de cada estado em que cada foco está localizado.
- **Foco**: Representa os focos e suas características.
- **Posicao**: Representa a posição em que o foco está localizado, assim como características relacionadas a esta posição.
- **Satelite_has_Foco**: Representa os focos encontrados por um determinado satélite. Um foco pode ser encontrado por diferentes satélites, assim como um satélite encontra vários focos diferentes.
- **Satelite**: Representa os satélites utilizados para o monitoramento e identificação dos focos.

O modelo proposto também visou garantir algumas restrições de integridade ne-

Figura 3.1: Modelo conceitual para o domínio dos dados de focos de incêncido fornecido pelo INPE.

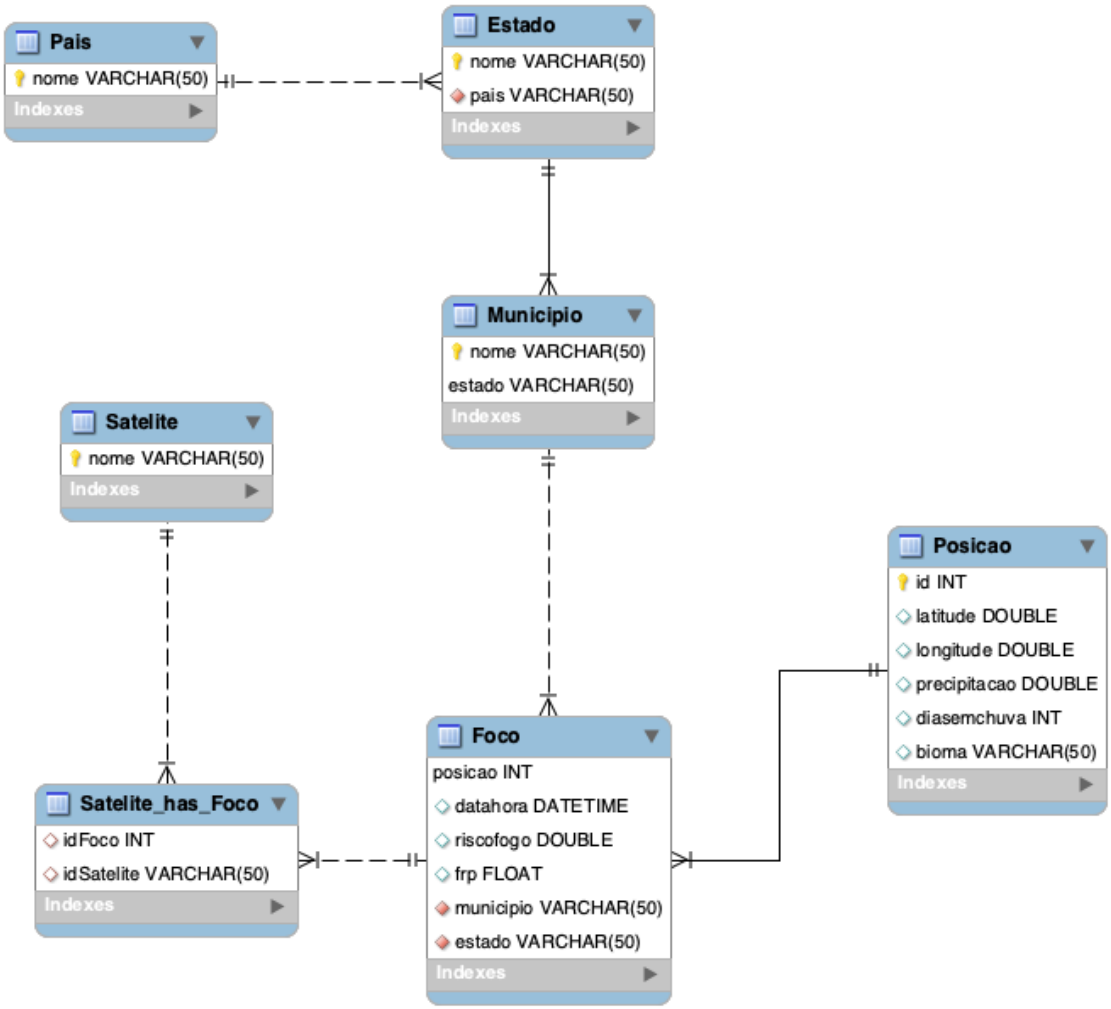


Fonte: Autor.

cessárias para o caso em questão, como:

- **Integridade referencial**: Cada chave estrangeira do modelo corresponde a um valor de uma chave primária existente na tabela de origem. Assim, garante-se a consistência entre as tuplas que relacionam as entidades. Como exemplo, podemos analisar o atributo "estado"existente na tabela Municipio, o qual referencia a chave primária da tabela Estado, isto é, o atributo "nome".
- **Integridade de entidade (chave)**: Todas tabelas possuem chaves primárias que identificam de maneira única cada elemento, mantendo assim todos os registros distintos uns dos outros.
- **Integridade de domínio**: Todos atributos das tabelas possuem um conjunto de valores permitidos, uma vez que são definidos os seus tipos e seus tamanhos explicitamente. Como alguns atributos podem conter o valor nulo, foi necessário permitir a estes campos específicos a atribuição de valores nulos.

Figura 3.2: Diagrama entidade-relacionamento para o domínio dos dados de focos de incêncido fornecido pelo INPE.



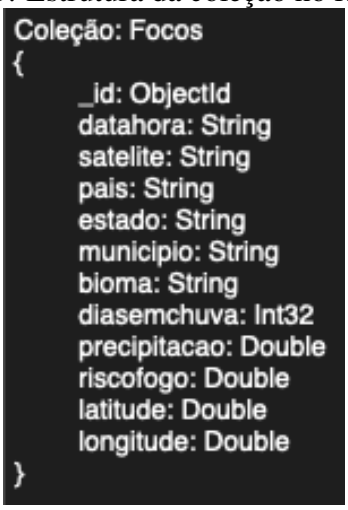
Fonte: Autor.

3.3 Modelagem para Banco de Dados baseado em Documentos

Visando focar em todas características e facilidades que MongoDB e o NoSQL proporcionam para a criação de base de dados e suas coleções, o arquivo CSV disponbilizado para download no BDQueimadas, presente no site do INPE, foi simplesmente inserido no MongoDB através do comando *mongoimport*, o qual serve para importar os dados para a base de dados do MongoDB, criando assim de forma automática a estrutura necessária para suportar os dados disponbilizados. Na Figura 3.3, podemos observar a estrutura criada pelo MongoDB para armazenar os dados fornecidos através do arquivo CSV. A coleção é composta por todos atributos encontrados no CSV, além de um ID único para cada documento a ser inserido na base de dados.

Vale ressaltar que, devido a flexibilidade proposta pelo MongoDB, esta estrutura

Figura 3.3: Estrutura da coleção no MongoDB.



Fonte: Autor.

da Figura 3.3 não está presente necessariamente em todos documentos da base de dados, já que para alguns registros certos atributos possuem valores nulos e estes atributos acabam sendo removidos da estrutura destes registros durante a importação automática dos dados.

3.4 Considerações sobre os modelos propostos

Neste capítulo, foram descritos os dois modelos propostos para o armazenamento de um conjunto de dados disponibilizado pelo INPE, no BDQueimadas. Um modelo é baseado em bancos de dados relacionais, implementado através do MySQL, e o outro modelo é o baseado em documentos, onde utilizou-se o MongoDB.

Para ambas modelagens procurou-se criar modelos simples e extensíveis. No entanto, para a implementação de um modelo relacional diversas etapas adicionais são necessárias, como a criação do modelo conceitual, lógico para então criar-se o físico. Todas essas etapas são extremamente importantes para a criação de um banco relacional com um bom desempenho e que atenda as necessidades dos usuários a longo prazo. Por estes motivos o desenvolvimento do banco de dados relacional se tornou mais complexo, demandou um esforço maior e uma maior quantidade de tempo em comparação ao desenvolvimento do banco de dados de documentos.

Para o banco não relacional a modelagem ocorre de forma muito mais simples e direta, tendo em vista que basta inserir os dados exportados em arquivos CSV para o MongoDB e então o próprio sistema de gerenciamento de banco de dados cria as coleções e suas estruturas de forma automatizada. Esta agilidade e flexibilidade na criação da base

de dados condiz com algumas das principais características propostas por este sistema de gerenciamento de banco de dados e que foram vistas nos capítulos anteriores.

4 AVALIAÇÃO EXPERIMENTAL

Neste Capítulo, é descrita a avaliação experimental entre os dois bancos de dados escolhidos para o presente trabalho, o MySQL (utilizando da *engine* InnoDB) e o MongoDB. Inicialmente, na Seção 4.1, é descrito o ambiente computacional utilizado para a execução da avaliação experimental. Em seguida, na Seção 4.2, são detalhados como os dados foram obtidos e mapeados para os modelos físicos apresentados no Capítulo 3. Na Seção 4.3, são apresentados os resultados de desempenho relativo ao armazenamento de tais dados. Na Seção 4.4, é apresentado o plano de escolha das consultas realizadas neste trabalho, assim como o resultado do tempo de execução das mesmas. Já na Seção 4.5, são apresentados os resultados gerais para os experimentos realizados.

4.1 Ambiente computacional

Os experimentos foram executados em um processador Intel Core i7, 2.7 GHz, quad-core, com uma memória RAM de 16GB de armazenamento e 1600MHz de frequência, e com um SSD de 512GB, com desempenho de 1830MB/s para leitura e 1300MB/s para escrita. O sistema operacional utilizado é macOS Catalina, versão 10.15.7.

Os SGBDs escolhidos para o banco de dados baseado em documentos e o banco de dados relacional foram respectivamente, o MongoDB, com o auxílio do MongoDB Compass Community Edition, para facilitar e melhorar a visualização dos documentos inseridos no banco de dados, e o MySQL, utilizando a *engine* InnoDB e com auxílio do MySQL Workbench para a criação tanto do diagrama entidade-relacionamento, como também das tabelas de cada entidade. Além disso, o MySQL Workbench facilitou tanto na obtenção de alguns dados de armazenamento, como, por exemplo, a quantidade de espaço ocupada pelo banco de dados completo, assim como na visualização dos dados nas tabelas.

Os *scripts* utilizados para o MongoDB foram desenvolvidos em Python, versão 3.8.5, com o auxílio do gerenciador de pacotes Anaconda. Para executar as funções necessária no MongoDB, utilizou-se a biblioteca PyMongo.

4.2 Base de Dados - BDQueimadas

A base de dados utilizada foi a disponibilizada pelo INPE através da aplicação BDQueimadas (INPE, 2019), responsável por monitorar e diponibilizar de forma visual as informações sobre os focos de queimadas da América do Sul. Para o presente trabalho, optou-se por aplicar alguns filtros para a geração dos dados, visando obter dados mais completos e interessantes para o comparativo. Assim sendo, a base de dados sobre os focos ficou restrita ao Brasil, tendo em vista que para os outros países da América do Sul poucas informações eram disponibilizadas para cada foco registrado, logo muitas colunas do arquivo acabavam ficando nulas.

Outro ponto a se considerar para a filtragem dos dados foram os anos a serem selecionados. Para datas anteriores a 2014, poucas informações sobre os focos estavam disponíveis e assim, pelo mesmo motivo anterior, optou-se por utilizar registros somente a partir do ano de 2014 até os dias atuais, ou seja, até o ano de 2021.

Além disso, é necessário considerar qual será o satélite escolhido como fonte dos registros dos focos, e optou-se pelo satélite de referência. Na própria página do BDQueimadas encontramos algumas informações referentes aos satélites e seus funcionamentos. Nela encontramos recomendações do próprio INPE para o uso do satélite de referência como fonte dos dados, pois é ele que é usado para compor oficialmente a série temporal ao longo dos anos e assim permitir a análise de tendências nos números de focos entre regiões para os períodos de interesse. Além disso, os focos fornecidos pelo satélite de referência coincidem com o conjunto de focos disponibilizado pela NASA e também pela Universidade de Maryland, uma vez que utilizam do mesmo algoritmo para detecção de queimadas, gerando assim produtos finais mais confiáveis. É recomendado o uso de todos os satélites disponíveis apenas em ocasiões específicas, tendo em vista que focos podem ser registrados de diferentes fontes, fazendo-se necessário uma análise própria da evolução do evento para identificar se o foco é repetido ou é um novo foco em um lugar já registrado anteriormente. Outro ponto a se considerar é que alguns satélites, como, por exemplo, o satélite europeu MSG-03, se localizam em uma longitude desfavorável para o registro dos focos no Brasil e acabam não cobrindo algumas regiões do país. No exemplo anterior, do MSG-03, a região oeste do país acaba ficando de fora da área de cobertura do satélite. A posição dos satélites também influencia na captação das imagens, uma vez que pelo ângulo em que está localizado as imagens acabam apresentando píxeis distorcidos, o que dificulta a identificação dos focos pelos algoritmos.

Após este estudo sobre quais dados seriam escolhidos para os testes a coletânea de dados foi obtida a partir da aplicação disponível no site do BDQueimadas (INPE, 2019). A partir dos filtros aplicados gerou-se uma base de dados com aproximadamente 1,2 milhões de registros de focos.

4.2.1 Mapeamento físico para o banco de dados MySQL

Para realizar a inserção dos dados no modelo desenvolvido para o banco de dados MySQL foi realizada uma etapa de pré-processamento dos dados. Nesta etapa, foi feita a separação das colunas do arquivo CSV original em diversos arquivos CSV para que fosse possível, e também facilitasse, a carga dos dados para as tabelas criadas no modelo proposto, visto no Capítulo 3. Este algoritmo de pré-processamento pode ser visto no Algoritmo 1 a seguir e foi desenvolvido em Python 3 utilizando, principalmente, a biblioteca Pandas, a qual é frequentemente utilizada no mundo da programação para a manipulação e análise de dados.

Algoritmo 1: Pré-processamento dos dados

```
   Entrada: Focos_2014_2021.csv
   Saída: Um arquivo CSV para cada tabela modelada.
1  início
2      Ler arquivo CSV de entrada;
3      Inserir uma coluna de id no arquivo CSV;
       /* A troca de valores NaN para None das colunas
          facilita a importação no MySQL Workbench    */
4      Trocar os valores NaN para None;
5      para cada Tabela modelada em SQL faça
6          Selecionar as colunas de interesse;
7          Exportar novo CSV apenas com as colunas selecionadas;
8      fim
9  fim
```

Assim, para cada tabela foi criado um arquivo CSV específico e de acordo com seus atributos. Para carregar os dados corretamente, basta executar o *script* disponibilizado e que pode ser visto no Apêndice A. Este *script* executa os comandos de *LOAD DATA* existentes no MySQL, carregando cada arquivo criado de acordo com sua tabela. Para este fim, cada tabela deve ser carregada em uma ordem específica, já estabelecida corretamente no algoritmo disponibilizado.

4.2.2 Mapeamento físico para o banco de dados MongoDB

Para o banco de dados MongoDB o mapeamento dos dados para o modelo físico se torna muito mais simples, tendo em vista que o MongoDB facilita a criação dos modelos, criando-os de forma automática de acordo com o arquivo CSV ou JSON inserido, como vimos na Seção 3.3. Então, para este SGBD bastou fazer o *download* dos dados de acordo com a Seção 4.2 e inserí-lo diretamente no MongoDB, que criou o modelo também já visto na Seção 3.3.

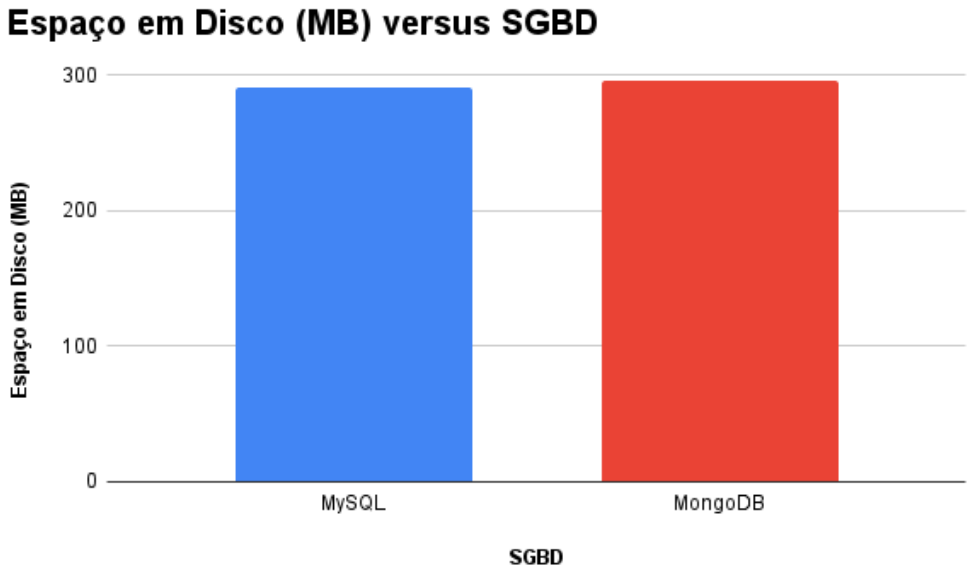
4.3 Experimento 1: Avaliação do Armazenamento Físico

Este experimento visa analisar e comparar o espaço de armazenamento físico necessário para armazenar a base de dados, obtida na Seção 4.2, para cada um dos SGBDs escolhidos. Serão analisados para cada um dos SGBDs, o espaço em disco utilizado, o número total de registros e o tempo de carga.

Para carregar os dados no MySQL foram usados os comandos de LOAD DATA, vistos no *script* encontrado no Apêndice A. Já para o MongoDB foi utilizado o comando *mongoimport*, realizado a partir do terminal. A inserção do dados no MySQL ocupou 291.10 MB de armazenamento e foram inseridos no total aproximadamente 3.560.860 registros. A carga dos dados levou aproximadamente 1 minuto e 19 segundos. Para o MongoDB a inserção de dados ocupou aproximadamente 296.2 MB e foram inseridos 1.185.163 registros. A carga de dados levou aproximadamente 39,5 segundos. Vale ressaltar que para o armazenamento ocupado pelo MongoDB foi considerado o tamanho da coleção criada e não da base de dados no sistema, uma vez que a base de dados acaba sendo comprimida e assim possui um tamanho muito menor, de aproximadamente 75 MB. Como o MySQL foi testado com as configurações padrões, e por este motivo não está configurado com as configurações de compactação disponíveis pela *engine* InnoDB, optou-se por comparar com o tamanho da coleção não comprimida do MongoDB. Os resultados obtidos para cada SGBD podem ser analisados graficamente na Figura 4.1, que apresenta o espaço em disco utilizado, na Figura 4.2, que apresenta o número de registros, e na Figura 4.3, que apresenta o tempo de carga.

A partir destes resultados podemos observar que o banco de dados MySQL e o banco de dados do MongoDB acabam ocupando praticamente o mesmo espaço, mesmo o MongoDB possuindo um número muito menor de registros. Isso se deve ao fato de que

Figura 4.1: Comparativo entre o espaço em disco ocupado pelo MySQL e pelo MongoDB.



Fonte: Autor.

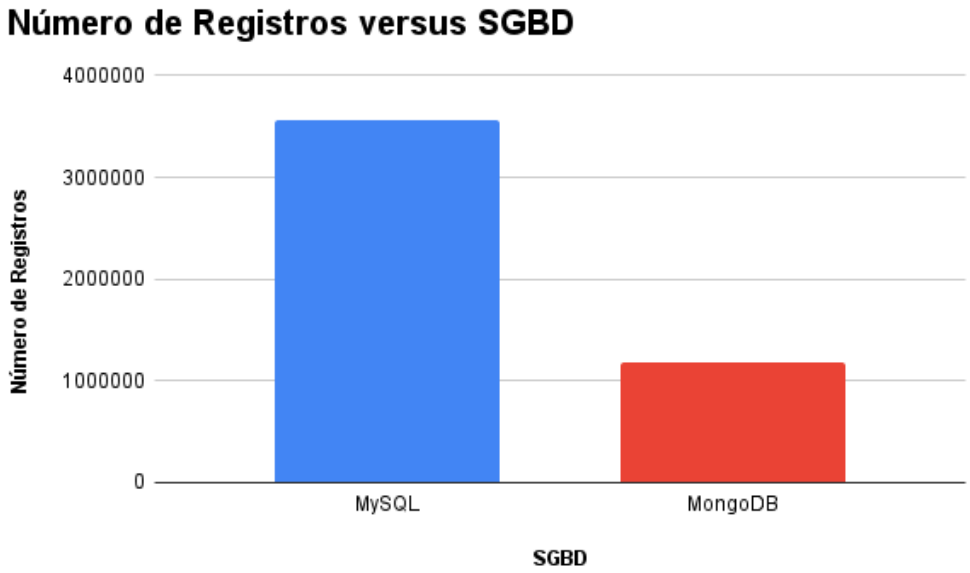
o MongoDB realiza uma pré alocação para futuras inserções de dados, buscando assim previnir (ou pelo menos minimizar) a fragmentações do disco. Como o MongoDB busca a flexibilidade, ele permite que cada registro tenha uma estrutura completamente diferente, assim ele acaba salvando junto a cada registro a sua estrutura, o que o leva a ocupar um espaço maior. Diferentemente, o MySQL possui uma estrutura rígida para todos registros de uma tabela, permitindo assim com que ele salve a estrutura desses registros uma única vez para cada tabela.

Além disso, quanto ao tempo de carga podemos observar que o MongoDB foi mais rápido, possivelmente devido ao número de registros total que acabou sendo menor para o MongoDB do que para o MySQL.

4.4 Experimento 2: Avaliação do Desempenho e Complexidade das Consultas

Esta seção apresenta os cenários escolhidos e as respectivas consultas que foram realizadas para avaliar o tempo de execução para cada um dos SGBDs escolhidos. Cada uma das consultas foi executada 10 vezes para obter o valor médio de cada consulta, já que a duração de cada uma delas varia de acordo com cada execução. Além disso, em um primeiro momento optou-se por utilizar o comando *SQL_NO_CACHE* durantes as consultas em MySQL, com o objetivo de evitar que as consultas ficassem em cache, tendo em vista que o MongoDB também não realiza o cache das consultas. No entanto, notou-se

Figura 4.2: Comparativo entre o número de registros em cada um dos bancos de dados, MySQL e MongoDB.



Fonte: Autor.

que o comando não estava alterando o resultado das consultas e descobriu-se que a partir do MySQL 8, o qual é a versão utilizada neste trabalho, por padrão o cache das consultas está desabilitado, não sendo necessário a utilização do comando *SQL_NO_CACHE*.

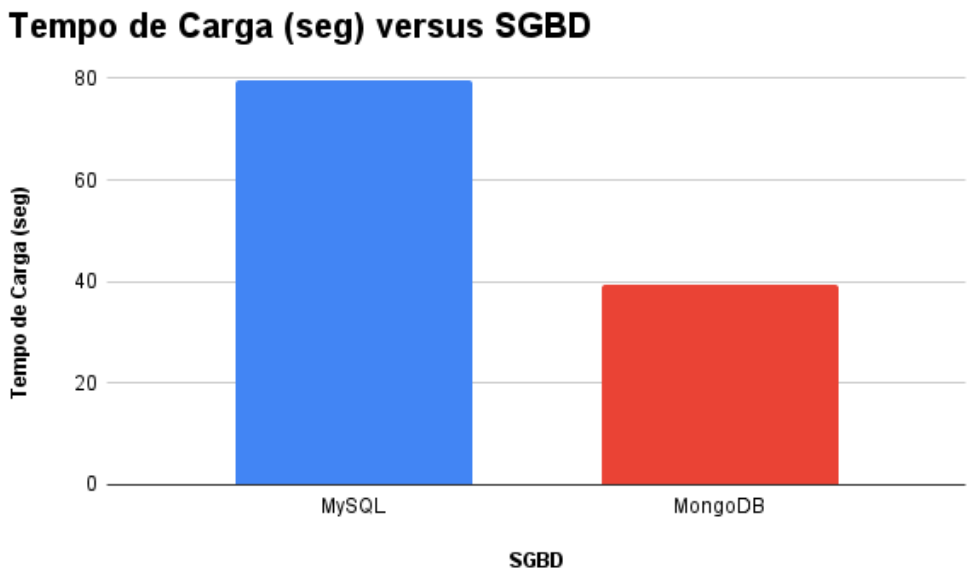
4.4.1 Cenários para Avaliação

Para avaliar o tempo de execução das consultas, tanto no MongoDB quanto no MySQL, foram propostos alguns cenários, visando possíveis consultas de interesse a serem realizadas na base de dados. Assim, foram propostas as seguintes consultas:

- Listagem de todos municípios e seus respectivos estados;
- Número de focos por estado;
- Número de focos por país;
- Focos com o maior *Fire Radiative Power* (FRP) e seu respectivo bioma (onde estão as queimadas mais intensas);
- Biomas os quais apresentam focos com maior risco de fogo para região;
- Número de focos por bioma no ano de 2019.

A seguir serão avaliados a complexidade da criação das consultas e sua performance para cada um dos cenários propostos.

Figura 4.3: Comparativo entre o tempo de carga no MySQL e no MongoDB.



Fonte: Autor.

4.4.2 Consulta 1: Listagem de todos municípios e seus respectivos estados

Esta consulta busca obter apenas uma lista de todos os munícipios e seu respectivo estado na base de dados disponibilizada. Para as consultas foram usados os comandos a seguir:

Consulta MySQL:

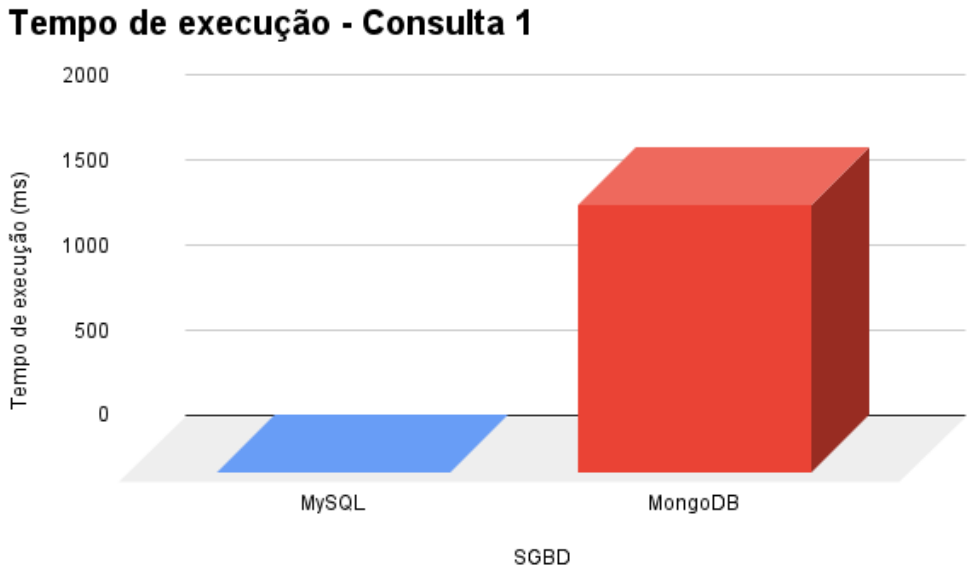
```
SELECT * FROM Municipio;
```

Consulta MongoDB:

```
db.focos.find({}, {"municipio": 1, "estado": 1}).explain()['executionStats']
```

Como podemos observar, para ambos bancos de dados as consultas são simples e não demandaram muito esforço para o desenvolvimento. Na Figura 4.4, podemos observar o tempo de execução de cada uma das consultas para cada banco de dados. Como é uma consulta simples, o MySQL acaba utilizando apenas uma tabela para a obtenção dos dados requisitados e como anteriormente havia sido feito o pré-processamento dos dados, ele acaba percorrendo um número muito menor de registros para obter o resultado necessário. Desta forma, o MySQL obtém uma performance melhor que o modelo proposto no MongoDB, o qual percorre, obrigatoriamente, todos os registros inseridos.

Figura 4.4: Comparativo entre os tempos de execução para a Consulta 1 em cada SGBD.



Fonte: Autor.

4.4.3 Consulta 2: Número de focos por estado

Esta consulta busca obter o número de focos encontrado em cada um dos estados listados no banco de dados. A seguir, podem ser visualizados os comandos utilizados para a consulta em cada um dos SGBDs.

Consulta MySQL:

```
SELECT COUNT(posicao) AS NumeroDeFocos, M.estado FROM Foco F
INNER JOIN Municipio M ON F.municipio = M.nome AND F.estado = M.estado
INNER JOIN Estado E ON M.estado = E.nome
GROUP BY (estado);
```

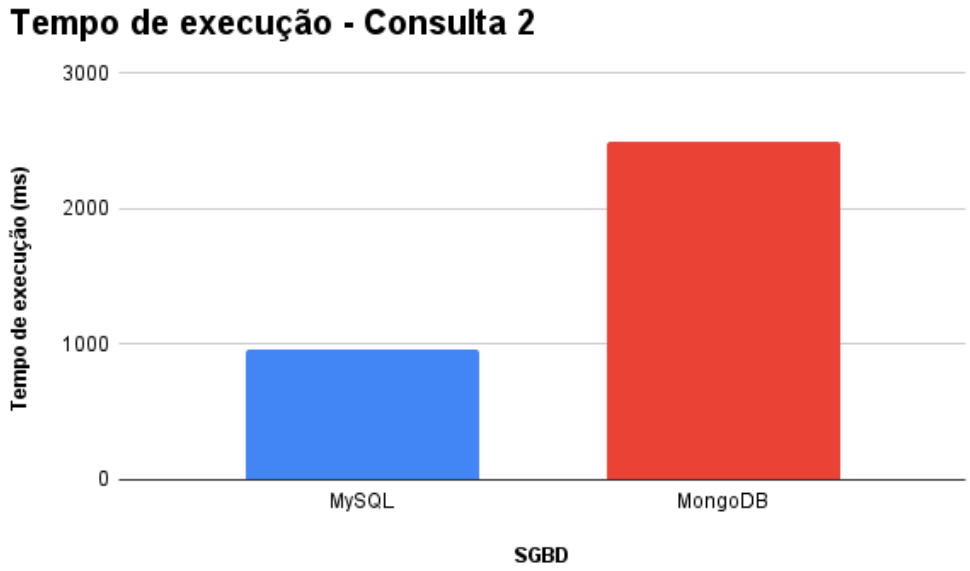
Consulta MongoDB:

```
db.focos.aggregate([ { "$group": {"_id": "$estado","count": {"$sum": 1}}}])
```

Nesta consulta foi necessário um esforço de desenvolvimento maior por parte do MySQL, tendo em vista a necessidade da junção de diversas tabelas a fim de obter-se o resultado necessário. Já para o MongoDB, a consulta foi um pouco mais complexa do que a anterior, porém ainda sim foi realizada de uma forma simples, tendo em vista a estrutura adotada e os comandos que são realizados de uma forma mais intuitiva. Na Figura 4.5, podemos observar o tempo de execução para cada uma das consultas acima.

Desta vez, podemos observar que o MySQL demorou mais tempo para a execução da consulta se comparado com a consulta anterior. No entanto, ainda assim demorou menos tempo para execução que o MongoDB. Isso possivelmente se deve ao fato de que o *aggregation framework*, utilizado através do comando *aggregate* no MongoDB, não possui uma boa performance devido a algumas conversões internas. O comando *aggregate* possui *pipelines* que consistem em um ou mais estágios que processam os documentos, sendo eles responsáveis por agrupar, filtrar ou até aplicar cálculos, como o valor máximo, médio, o total, entre outros, sobre um conjunto de documentos. Cada um desses estágios transforma o documento enquanto ele percorre o *pipeline* e a saída de um desses estágios serve como entrada para o próximo. Para cada estágio, o *framework* necessita buscar o arquivo BSON do documento em questão e convertê-lo para objetos internos ao *pipeline* para o processamento e no final da execução ele é convertido de volta ao formato BSON.

Figura 4.5: Comparativo entre os tempos de execução para a Consulta 2 em cada SGBD.



Fonte: Autor.

4.4.4 Consulta 3: Número de focos por país

Esta consulta visa obter o número de focos encontrado em cada país encontrado no banco de dados. A seguir podem ser visualizados os comandos utilizados para cada a consulta em cada um dos SGBDs.

Consulta MySQL:

```
SELECT COUNT(posicao) AS NumeroDeFocos, pais FROM Foco F
INNER JOIN Municipio M ON F.municipio = M.nome AND F.estado = M.estado
INNER JOIN Estado E ON M.estado = E.nome
INNER JOIN Pais P ON E.pais = P.nome
GROUP BY (pais);
```

Consulta MongoDB:

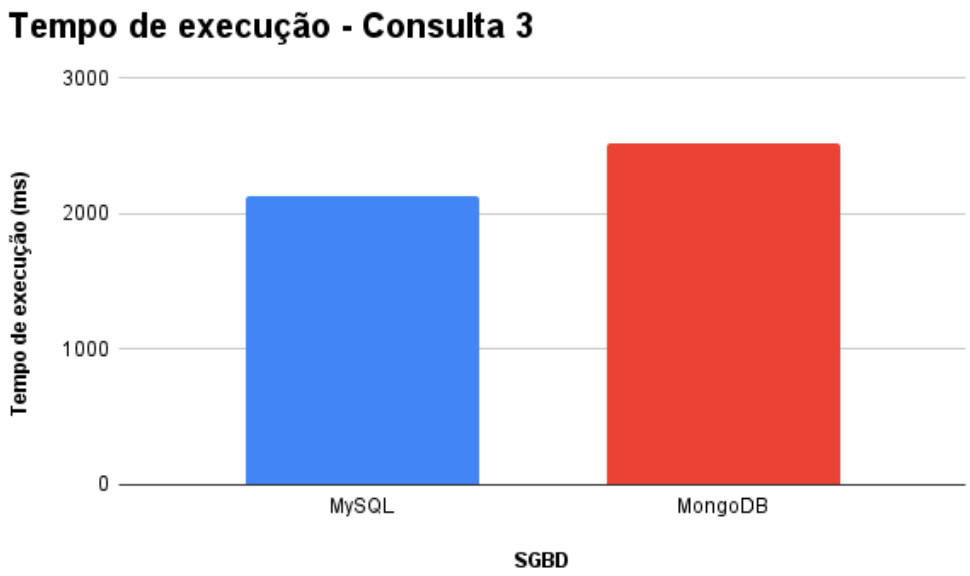
```
db.focos.aggregate([ { "$group": {"_id": "$pais","count": {"$sum": 1}}}])
```

Novamente, para esta consulta, a complexidade de desenvolvimento para o MySQL foi maior, tendo em vista as diversas junções necessárias nas tabelas a fim de obter o resultado esperado. A complexidade para a consulta no MongoDB se manteve a mesma da consulta anterior, já que ambas consultas acabaram sendo muito parecidas. Na Figura 4.6, podemos observar o tempo de execução para cada consulta. O MySQL teve um tempo maior de execução se comparado a consulta anterior, e isso pode ser justificado pela maior quantidade de junções necessárias para obter o resultado esperado. No entanto, ainda assim o MySQL foi mais rápido que o MongoDB, e novamente isso pode ser justificado pelo uso do *aggregation framework* por parte da consulta no MongoDB. Também podemos perceber que as consultas 2 e 3 possuem uma estrutura muito parecida no MongoDB, logo elas tendem a manter um tempo de execução muito parecido, já que realizaram praticamente a mesma tarefa. Por outro lado, no MySQL, as consultas 2 e 3 tiveram uma maior variação nos tempos de execução, pois de acordo com os dados requeridos foi necessário realizar um maior número de junções entre as tabelas para a consulta 3 em comparação a consulta 2.

4.4.5 Consulta 4: Focos com o maior *Fire Radiative Power* (FRP) e seu respectivo bioma

Esta consulta tem como o objetivo identificar onde estão localizados os biomas que possuem os maiores focos de incêncido. O FRP é uma técnica que utiliza de dados detectados remotamente para conseguir quantificar a quantidade média de biomassa queimada para cada foco. Ele mede a energia radiada por unidade de tempo através da queima da vegetação. Para realizar esta consulta foram utilizados os seguintes comandos em cada SGBD:

Figura 4.6: Comparativo entre os tempos de execução para a Consulta 3 em cada SGBD.



Fonte: Autor.

Consulta MySQL:

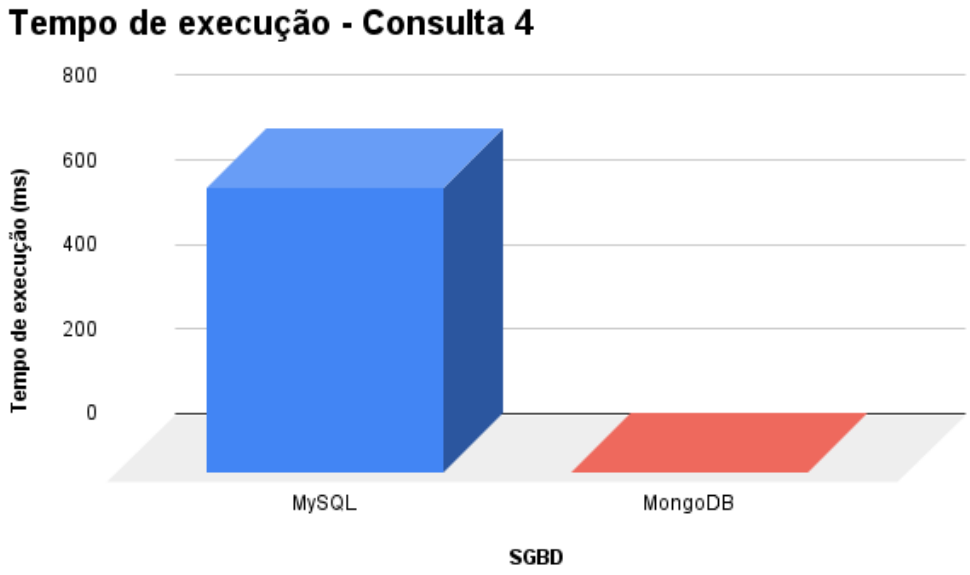
```
SELECT frp, bioma FROM Foco F
INNER JOIN Posicao P
ON F.posicao = P.id
ORDER BY frp DESC LIMIT 5;
```

Consulta MongoDB:

```
db.focos.find({}, {"frp": 1, "bioma": 1 }).sort([("frp", -1)]).limit(5)
```

Em termos de complexidade, novamente a consulta em MySQL necessitou um pouco mais de cuidado no desenvolvimento em comparação a consulta realizada no MongoDB. Pode-se observar na Figura 4.7 que desta vez o MongoDB foi consideravelmente mais rápido que o MySQL. Isto se deve ao fato de que para o modelo proposto para o MongoDB necessitou-se apenas de um comando *find*, especificando os campos necessários e a realização do *sort*, para ordená-los de acordo. Já para o MySQL foi necessária a realização de uma junção entre duas tabelas e também a realização de um ordenamento através do comando *ORDER BY*. Outro ponto a ser considerado é sobre os algoritmos de *sort* e suas varáveis internas, como, por exemplo, o tamanho dos *buffers* que pode ser alterado no MySQL através da variável de sistema *sort_buffer_size*, aplicados em cada um dos comandos utilizados por cada SGBD. Possivelmente, o *sort* utilizado pelo MongoDB tende a ser mais eficiente do que o MySQL para o caso em específico.

Figura 4.7: Comparativo entre os tempos de execução para a Consulta 4 em cada SGBD.



Fonte: Autor.

4.4.6 Consulta 5: Biomas os quais apresentam focos com maior risco de fogo

A partir desta consulta obtem-se os biomas do país que apresentaram focos com maior risco de fogo. O risco de fogo é o valor previsto para o mesmo dia da ocorrência do foco e é calculado essencialmente a partir do histórico de precipitação nos últimos 120 dias. Além disso, também leva em consideração outros fatores como, temperatura máxima do ar e da umidade relativa do ar mínima, bem como o tipo de vegetação e a ocorrência de focos anteriormente. O risco de fogo varia entre os valores de 0 a 1, sendo 1 risco de fogo considerado crítico e 0 é o risco de fogo mínimo. Para esta consulta optou-se por filtrar valores acima de 0.7, os quais indicariam regiões com um alto risco de fogo. Para realizar esta consulta foram utilizados os seguintes comandos em cada um dos SGBDs:

Consulta MySQL:

```
SELECT COUNT(riscofogo), bioma FROM Foco F
INNER JOIN Posicao P ON F.posicao = P.id
WHERE riscofogo >= 0.7
GROUP BY(bioma);
```

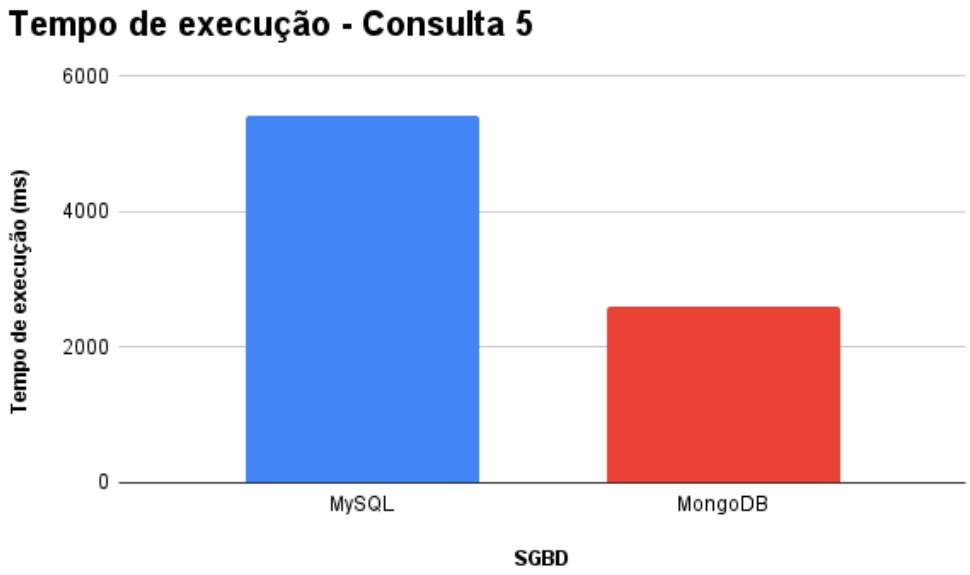
Consulta MongoDB:

```
db.focos.aggregate([ { "$match": { "riscofogo": {"$gte": 0.7}, } }, { "$group": {
```

"_id": "$bioma", "count": { "$sum": 1 } }])

Na Figura 4.8, podemos observar que o MongoDB foi ligeiramente mais rápido na execução da consulta especificada. Analisando os resultados anteriores, uma justificativa para tal resultado seria possivelmente por uma melhor otimização do comando *match* utilizado no MongoDB em relação a cláusula *WHERE* utilizada no MySQL, assim como as estruturas em que ambos comandos são aplicados (tabelas, para o MySQL, e documentos em formato JSON, para o MongoDB).

Figura 4.8: Comparativo entre os tempos de execução para a Consulta 5 em cada SGBD.



Fonte: Autor.

4.4.7 Consulta 6: Número de focos por bioma no ano de 2019

Esta consulta visa obter o número de focos em cada bioma para o ano de 2019, possibilitando consultar qual o bioma mais afetado durante o período. Para realizar esta consulta foram utilizados os seguintes comandos em cada um dos SGBDs:

Consulta MySQL:

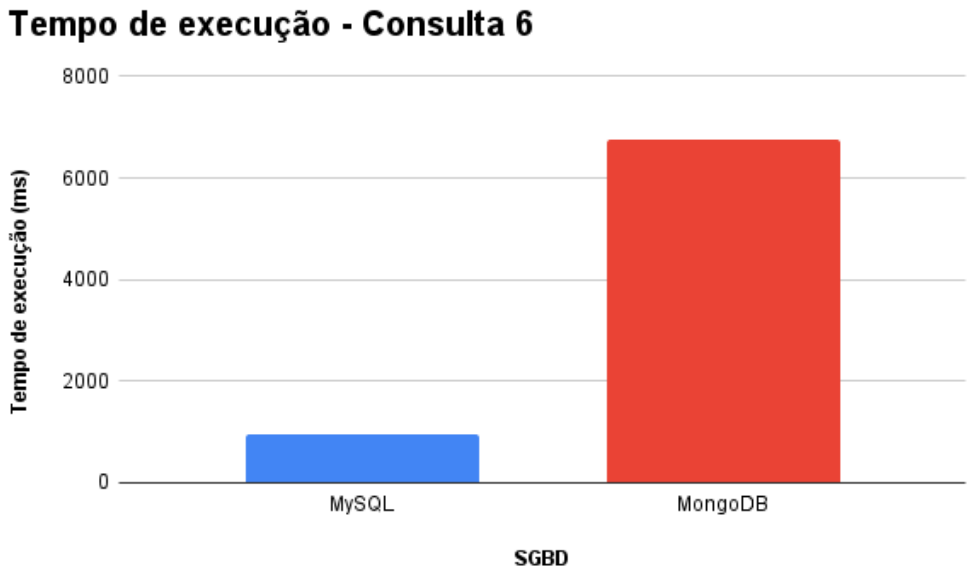
```
SELECT COUNT(posicao) AS NumeroDeFocos, bioma FROM Foco F
INNER JOIN Posicao P ON F.posicao = P.id
WHERE (datahora >= '2019-01-01' AND datahora <= '2019-12-31')
GROUP BY(bioma);
```

Consulta MongoDB:

db.focos.aggregate([{ "$addFields": { "datahora": { "$dateFromString": { "dateString": "$datahora"} } }}, { "$match": { "datahora": { "$gte": datetime.datetime(2019,1,1), "$lt": datetime.datetime(2019,12,31) }}}, { "$group": { "_id": "$bioma", "count": { "$sum": 1 } } }])

O desenvolvimento de ambas as consultas se tornou mais complexa. Pode-se considerar a consulta realizada no MongoDB ainda mais complexa de criar do que a do MySQL, pois foi necessário realizar a conversão do atributo *datahora* de *string* para o tipo *ISODate*, o qual é um formato que se encaixa no padrão ISO (*International Organization for Standardization*) para as datas expressas no calendário numérico. Para realizar esta conversão foram necessários alguns comandos extras, como *$addFields* e *$dateFromString*, além dos comandos de *$match* e *$group*. Na Figura 4.9, podemos ver que a consulta em MongoDB acabou executando em um tempo maior a consulta realizada em MySQL, possivelmente devido as conversões e comandos extras que foram necessárias para a realização da consulta. Um possível ponto a se considerar para o modelo proposto, seria a troca do tipo do atributo *datahora* de todos os registros salvos no banco de dados do MongoDB para o tipo *Date*, facilitando na hora da criação da consulta e, possivelmente, diminuindo seu tempo de execução. Esta troca do tipo do atributo também possivelmente diminuiria o espaço de armazenamento utilizado pelo banco de dados, tendo em vista que o tipo *Date* ocupa menos espaço que o tipo *string*.

Figura 4.9: Comparativo entre os tempos de execução para a Consulta 6 em cada SGBD.



Fonte: Autor.

4.5 Avaliação Geral dos Resultados

Através dos experimentos realizados para o presente trabalho, pode-se analisar diferentes fatores na avaliação de desempenho para cada um dos sistemas de gerenciamento de banco de dados escolhidos. Entre estes fatores podemos listar, o tempo de carga dos dados, o espaço em disco utilizado pelos bancos de dados, a duração média das consultas, complexidade de criação dos comandos de consulta e também complexidade de modelagem. Sendo que a complexidade de criação das consultas e a complexidade de modelagem serão analisadas de forma indireta.

Analisando o tempo de carga, percebemos que o MongoDB inseriu os dados de forma mais rápida se comparado ao tempo obtido para a inserção dos dados pelo MySQL, sendo assim o MongoDB escolhido como melhor opção para este caso. Vale ressaltar que o tempo de carga deve variar de acordo com a modelagem proposta para cada um dos modelos, como por exemplo, para o modelo MySQL proposto no presente trabalho obteve-se um número muito maior de registros em comparação ao MongoDB e isso pode ter influenciado de forma negativa a importação dos dados para este banco.

Já ao analisar o espaço necessário em disco para armazenar os dados utilizados, ambos os modelos obtiveram um espaço parecido de armazenamento. Vale ressaltar que para a presente comparação foi utiliazado o espaço ocupado pela coleção criada no MongoDB. Ao analisar o espaço ocupado pelo *database* obteve-se um resultado muito menor, algo em torno dos 75 MB de espaço utilizado, já que é realizada a compressão dos dados para armazenamento. Como para o MySQL não foi habilitado a compactação na *engine* InnoDB, assim ele acaba ocupando mais espaço no total se comparado ao espaço utilizado pelos dados compactados no MongoDB.

No que se refere ao tempo de execução das consultas criadas, no MySQL elas foram realizadas, no geral, de forma mais eficiente e rápidas do que no MongoDB. Esse resultado pode ser observado na Figura 4.10, a qual apresenta o resultado geral do tempo de execução para todas as consultas. O MySQL possui uma tecnologia mais madura se comparada ao MongoDB, o que o torna mais otimizado ao realizar algumas consultas. Porém, deve-se considerar alguns pontos, como o modelo proposto para ambos casos, o uso de índices, o qual não foi adotado para o MongoDB e que poderia melhorar o resultado de algumas consultas, além de detalhes internos de implementação de cada sistemas. O MySQL, através da *engine* InnoDB possui diversas *caches*, entre elas o *InnoDB Buffer Pool*, o qual é um espaço de memória que contém muitas estruturas de dados do InnoDB,

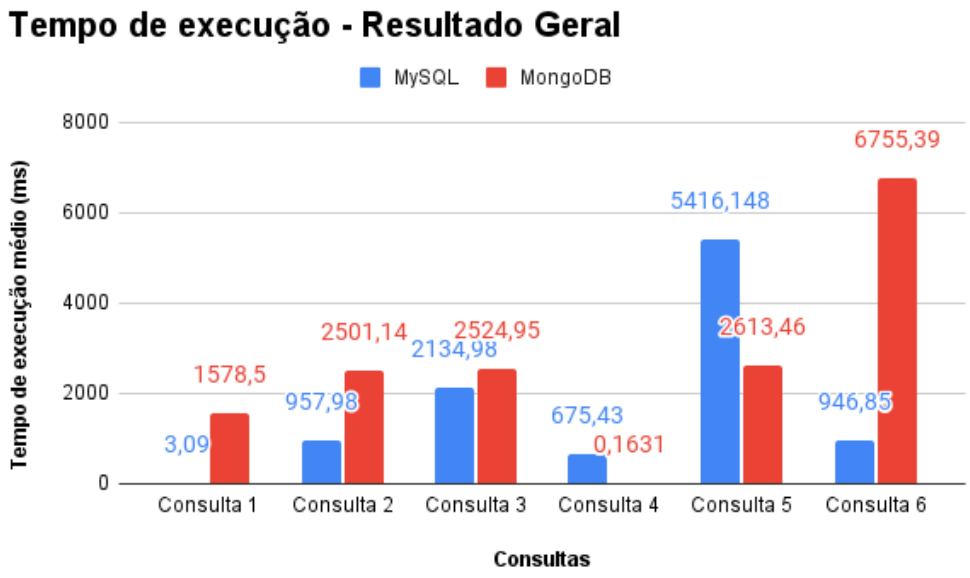
como *buffers*, *caches*, índices e até registros, e que melhora a performance das consultas subsequentes a primeira consulta.

Quanto a complexidade de modelagem e da realização das consultas, pode-se observar que o MySQL possui uma carga de trabalho muito maior para a criação das consultas assim como para a modelagem em comparação ao MongoDB. Para a modelagem em MySQL, foram necessárias diversas etapas até que se chegou no modelo proposto, o que levou uma quantidade de tempo muito maior em comparação ao MongoDB, que bastou inserir os dados no banco de dados que a estrutura para cada registro foi criada de forma automática. Já para as consultas, podemos ver que, em geral, elas são muito mais claras e objetivas de serem realizadas através do MongoDB, uma vez que utilizam uma linguagem mais simples e de fácil compreensão por parte do programador. Para o MySQL é necessário um estudo prévio muito maior, não somente sobre a linguagem SQL, mas também sobre todo o conceito proposto pelo modelo relacional.

Considerando os pontos discutidos anteriormente, podemos perceber que ambos os sistemas de banco de dados possuem suas vantagens e desvantagens, e ambos se saíram parelhos de uma forma geral. Para o presente trabalho, pode-se dizer que o MySQL obteve um desempenho melhor nas consultas, que tem um maior peso na maioria dos casos, porém deve-se considerar que a adição de índices aos registros no MongoDB poderia levar a uma melhora no desempenho do modelo em alguns casos. Assim sendo, o mais indicado a se fazer antes de optar entre um dos modelos, é estudar como cada um dos modelos se comporta no seu caso em específico, assim como buscar otimizá-lo da melhor forma possível. Devem ser considerados por cada desenvolvedor na hora de escolher entre um dos modelos pontos como:

- O armazenamento necessário em disco é importante para minha solução?
- O tempo de execução das consultas pesa mais que os demais fatores?
- O tempo da carga de dados influencia?
- O trabalho/tempo consumido para modelar a solução em função da performance obtida é vantajoso?
- Necessito de flexbilidade dos dados?
- A minha solução trabalhará com um grande volume de dados?

Figura 4.10: Comparativo entre os tempos de execução para todas as consultas em cada SGBD. Consulta 1: Listagem de todos municípios e seus respectivos estados; Consulta 2: Número de focos por estado; Consulta 3: Número de focos por país; Consulta 4: ocos com o maior *Fire Radiative Power* (FRP) e seu respectivo bioma; Consulta 5: Biomas os quais apresentam focos com maior risco de fogo; Consulta 6: Número de focos por bioma para o ano de 2019.



Fonte: Autor.

5 CONCLUSÃO

Este trabalho apresentou a análise de desempenho entre os sistemas de gerenciamento de bancos de dados MySQL (utilzando a *engine* InnoDB) e o MongoDB, analisando diversas características como: tempo de carga para os dados, o número de registros armazenados, o espaço ocupado pelos dados no disco, o tempo de execução de algumas consultas, além da complexidade de implementação de cada modelo. Para que fosse possível realizar tal comparativo, os bancos de dados foram alimentados com uma base de dados real, a de registro focos de incêndio, fornecida pelo Instituto Nacional de Pesquisas Espaciais (INPE) através da aplicação web do BDQueimadas (INPE, 2019). Através deste trabalho pode-se observar as seguintes diferenças entre os bancos:

- **Modelagem**: Para o MySQL foram necessárias mais etapas para o desenvolvimento do modelo final proposto presente neste trabalho. Primeiro, criou-se o modelo conceitual, posteriormente o modelo lógico e somente a partir do modelo lógico que desenvolveu-se o modelo físico. Todas essas etapas são de suma importância para o desenvolvimento de qualquer banco relacional. Elas visam obter um modelo para o banco de dados que atenda as necessidades dos usuários a longo prazo (evitando problemas futuros) e também apresente um bom desempenho; Para o MongoDB não foi necessário criar nenhum modelo prévio, visto que para este sistema basta carregar os arquivos CSVs diretamente na plataforma e eles são modelados automaticamente pela ferramenta. Esta modelagem automática pode levar a algumas estruturas não otimizadas. Durante o desenvolvimento do presente trabalho, observou-se que o atributo *datahora* se modelado para o tipo *Date* ao invés de ser uma *string*, poderia resultar em uma melhor performance para o banco estabelecido.
- **Carga de dados**: Para o MySQL, foi necessário criar uma etapa de pré-processamento dos dados visando facilitar a carga dos dados para o banco de dados. Esta etapa de pré-processamento pode ser observada no Algoritmo 1. Ainda assim, o MongoDB obteve um tempo de carga menor em comparação ao MySQL, já que, devido ao modelo proposto, este teve um menor número total de registros salvos no banco de dados.
- **Espaço em Disco**: Ambos os bancos tiveram um espaço de disco parecido ao comparar os dois modelos não compactados.
- **Consultas**: No geral as consultas no MySQL foram mais rápidas do que no MongoDB, porém, como foi discutido anteriormente, alguns fatores podem ter influen-

ciado neste desempenho, como a falta do uso de índices no MongoDB. No entanto, as consultas realizadas no MongoDB, em sua maioria, foram mais simples de serem desenvolvidas se comparadas com as consultas do MySQL. Nele não foi necessário realizar junções de documentos visto que todo conteúdo foi armazenado em um único documento. Isto é um ponto positivo, já que o MongoDB não foi otimizado pra estas operações de junção e isso poderia levar a um desempenho pior.

Através destes resultados podemos ver que ambos os sistemas de banco de dados possuem suas vantagens e desvantagens, e diversos pontos devem ser levados em questão ao decidir-se por um sistema ou outro. O MongoDB tende a ser um SGBD que aceita uma maior flexibilidade dos dados a serem armazenados, estes podendo variar em seu formato e tipo de informação, e tudo isso é armazenado de uma forma simples e fácil. Enquanto isso, no MySQL, os dados são fixos e bem estruturados, o que torna difícil o armazenamento de dados não estruturados. Sendo assim, o MongoDB tende a ser vantajoso neste momento da era dos dados em que, em função da *Internet of Things* e do *Big Data*, eles tendem a ser não estruturados e em grande volume. Já o MySQL parece mais vantajoso em performance para os casos de dados bem estruturados e com um pequeno volume de dados. Além disso, há diversos outros pontos que devem ser explorados, como o uso de técnicas de *sharding*, escalabilidade, a adição de índices no MongoDB, a compactação por parte do MySQL, o uso dos diferentes tipos e tamanhos de *cache* para cada sistema, entre outros.

Como possíveis melhorias deste trabalho e também como trabalho futuro, poderia-se considerar a aplicação de alguns dos pontos citados acima durante a comparação entre os dois bancos, como técnica de *sharding*, aplicação de índices no MongoDB, variação de algumas váriaveis internas do MySQL, além de outros pontos como: o uso de uma diferente *engine* para o MySQL, uma análise sobre a segurança de cada banco, sobre a consistência dos dados em diferentes situações, a disponibilidade de cada modelo e até mesmo um comparativo com outros modelos de bancos NoSQL, como o modelo baseado em colunas.

REFERÊNCIAS

BACHMAN, C. W. Data structure diagrams. **SIGMIS Database**, Association for Computing Machinery, New York, NY, USA, v. 1, n. 2, p. 4–10, jul. 1969. ISSN 0095-0033. Disponível em: <https://doi.org/10.1145/1017466.1017467>.

BREWER, E. A. Towards robust distributed systems. In: PORTLAND, OR. **PODC**. [S.l.], 2000. v. 7, n. 10.1145, p. 343477–343502.

CODD, E. F. A relational model of data for large shared data banks. In: **Software pioneers**. [S.l.]: Springer, 2002. p. 263–294.

GRAY, J.; REUTER, A. **Transaction processing: concepts and techniques**. [S.l.]: Elsevier, 1992.

HAN, J. et al. Survey on nosql database. In: IEEE. **2011 6th international conference on pervasive computing and applications**. [S.l.], 2011. p. 363–366.

HARRISON, G. **Next Generation Databases: NoSQLand Big Data**. 1st. ed. USA: Apress, 2015. ISBN 1484213300.

HECHT, R.; JABLONSKI, S. Nosql evaluation: A use case oriented survey. In: **2011 International Conference on Cloud and Service Computing**. [S.l.: s.n.], 2011. p. 336–341.

HELD, G.; STONEBRAKER, M.; WONG, E. Ingres: A relational data base system. In: **Proceedings of the May 19-22, 1975, national computer conference and exposition**. [S.l.: s.n.], 1975. p. 409–416.

INPE. **Banco de Dados de queimadas**. 2019. <https://queimadas.dgi.inpe.br/queimadas/bdqueimadas#mapa>.

LEAVITT, N. Will nosql databases live up to their promise? **Computer**, IEEE, v. 43, n. 2, p. 12–14, 2010.

LI, Y.; MANOHARAN, S. A performance comparison of sql and nosql databases. In: IEEE. **2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)**. [S.l.], 2013. p. 15–19.

MICROSOFT. **Microsoft MySQL Server**. 1994. <https://www.microsoft.com/pt-br/sql-server>.

MONGODB. 2007. <https://www.mongodb.com>.

MONIRUZZAMAN, A.; HOSSAIN, S. A. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. **arXiv preprint arXiv:1307.0191**, 2013.

ORACLE. **MySQL**. 1995. <https://www.mysql.com>.

PAGÁN, J. E.; CUADRADO, J. S.; MOLINA, J. G. A repository for scalable model management. **Software & Systems Modeling**, Springer, v. 14, n. 1, p. 219–239, 2015.

SILBERSCHATZ, A. et al. **Database system concepts**. [S.l.]: McGraw-Hill New York, 1997. v. 4.

STONEBRAKER, M. et al. The end of an architectural era: It's time for a complete rewrite. In: **Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker**. [S.l.: s.n.], 2018. p. 463–489.

STRAUCH, C.; SITES, U.-L. S.; KRIHA, W. Nosql databases. **Lecture Notes, Stuttgart Media University**, v. 20, p. 24, 2011.

SUNDARAPPA, N. **Oracle Java Scalability with Sharded Database**. 2018. <https://blogs.oracle.com/dev2dev/java-scalability-with-sharded-database>.

SYBASE. **Sybase**. 1988. <https://getmanta.com/technologies/databases/sap-ase-sybase>.

TAYLOR, R. W.; FRANK, R. L. Codasyl data-base management systems. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 8, n. 1, p. 67–103, mar. 1976. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/356662.356666>.

TSICHRITZIS, D. C.; LOCHOVSKY, F. H. Hierarchical data-base management: A survey. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 8, n. 1, p. 105–123, mar. 1976. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/356662.356667>.

VENKATRAMAN, S. et al. Sql versus nosql movement with big data analytics. **International Journal of Information Technology and Computer Science**, v. 8, n. 12, p. 59–66, 2016.

APÊNDICE A — SCRIPT SQL PARA A CARGA DE DADOS NO BANCO DE DADOS MYSQL

```
SET profiling = 1;

-- ############################{Paises}############################
LOAD DATA INFILE '/Users/nicholaslau/DataModeling/paises.csv'
INTO TABLE Pais
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS (@dummy, nome);

-- ############################{Estados}############################
LOAD DATA INFILE '/Users/nicholaslau/DataModeling/estados_paises.csv'
INTO TABLE Estado
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS (pais, nome) set nome=nome, pais=pais;

-- ############################{Municipios}############################
LOAD DATA INFILE '/Users/nicholaslau/DataModeling/
                  municipios_estados.csv'
INTO TABLE Municipio
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS (estado, nome) set estado=estado, nome=nome;

-- ############################{Satelites}############################
LOAD DATA INFILE '/Users/nicholaslau/DataModeling/satelites.csv'
INTO TABLE Satelite
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS (@dummy, nome);
```

```
-- ############################{Posição}############################
LOAD DATA INFILE '/Users/nicholaslau/DataModeling/posicao.csv'
INTO TABLE Posicao
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
        (id, latitude, longitude, @precipitacao, @diasemchuva, bioma)
          set id=id, latitude=latitude, longitude=longitude,
          precipitacao=NULLIF(@precipitacao,''),
          diasemchuva=NULLIF(@diasemchuva,''), bioma=bioma;

-- ############################{Focos}############################
LOAD DATA INFILE '/Users/nicholaslau/DataModeling/focos.csv'
INTO TABLE Foco
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS (@id, datahora, @riscofogo, @frp, municipio, estado)
              set posicao=@id, datahora=datahora,
              riscofogo=NULLIF(@riscofogo,''),
              frp=NULLIF(@frp,''), municipio=municipio, estado=estado;

-- ############################{Satelite_has_focos}#####################
LOAD DATA INFILE '/Users/nicholaslau/DataModeling/sateliteHasFoco.csv'
INTO TABLE Satelite_has_Foco
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS (@id, @satelite) set idFoco=@id, idSatelite=@satelite;

SHOW PROFILES;
```

APÊNDICE B — SCRIPT SQL PARA CRIAÇÃO DO BANCO DE DADOS NO MYSQL

```
-- MySQL Script generated by MySQL Workbench
-- Mon Nov  8 19:43:04 2021
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,
    STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,
    ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
-- -----------------------------------------------------
-- Schema focos
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema focos
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `focos` DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci ;
USE `focos` ;

-- -----------------------------------------------------
-- Table `focos`.`Pais`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `focos`.`Pais` (
  `nome` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`nome`))
```

```
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `focos`.`Estado`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `focos`.`Estado` (
  `nome` VARCHAR(50) NOT NULL,
  `pais` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`nome`),
  INDEX `FK_Pais` (`pais` ASC) VISIBLE,
  CONSTRAINT `FK_Pais`
    FOREIGN KEY (`pais`)
    REFERENCES `focos`.`Pais` (`nome`)
    ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `focos`.`Municipio`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `focos`.`Municipio` (
  `nome` VARCHAR(50) NOT NULL,
  `estado` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`nome`, `estado`),
  INDEX `FK_Estado` (`estado` ASC) VISIBLE,
  CONSTRAINT `FK_Estado`
    FOREIGN KEY (`estado`)
    REFERENCES `focos`.`Estado` (`nome`))
ENGINE = InnoDB
```

```
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `focos`.`Posicao`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `focos`.`Posicao` (
  `id` INT NOT NULL,
  `latitude` DOUBLE NULL DEFAULT NULL,
  `longitude` DOUBLE NULL DEFAULT NULL,
  `precipitacao` DOUBLE NULL DEFAULT NULL,
  `diasemchuva` INT NULL DEFAULT NULL,
  `bioma` VARCHAR(50) NULL DEFAULT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `focos`.`Foco`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `focos`.`Foco` (
  `posicao` INT NOT NULL,
  `datahora` DATETIME NULL DEFAULT NULL,
  `riscofogo` DOUBLE NULL DEFAULT NULL,
  `frp` FLOAT NULL DEFAULT NULL,
  `municipio` VARCHAR(50) NOT NULL,
  `estado` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`posicao`),
  INDEX `FK_Municipio` (`municipio` ASC, `estado` ASC) VISIBLE,
  CONSTRAINT `FK_Municipio`
    FOREIGN KEY (`municipio` , `estado`)
```

```
    REFERENCES `focos`.`Municipio` (`nome` , `estado`),
  CONSTRAINT `FK_Posicao`
    FOREIGN KEY (`posicao`)
    REFERENCES `focos`.`Posicao` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `focos`.`Satelite`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `focos`.`Satelite` (
  `nome` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`nome`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `focos`.`Satelite_has_Foco`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `focos`.`Satelite_has_Foco` (
  `idFoco` INT NULL DEFAULT NULL,
  `idSatelite` VARCHAR(50) NULL DEFAULT NULL,
  UNIQUE INDEX `idFoco` (`idFoco` ASC) VISIBLE,
  INDEX `FK_Satelite` (`idSatelite` ASC) VISIBLE,
  CONSTRAINT `FK_Foco`
    FOREIGN KEY (`idFoco`)
    REFERENCES `focos`.`Foco` (`posicao`),
  CONSTRAINT `FK_Satelite`
    FOREIGN KEY (`idSatelite`)
    REFERENCES `focos`.`Satelite` (`nome`))
```

```
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```