UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

WILLIAM WEBER BERRUTTI

Um Estudo Comparativo da Modelagem e Avaliação de Desempenho entre Bancos de Dados Relacional e de Documentos para os Dados Abertos da Câmara dos Deputados do Brasil

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

Orientadora: Profª. Drª. Renata Galante
Coorientadora: Profª. Drª. Carla M.D.S. Freitas

Porto Alegre
2019

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me capacitado nesta longa jornada na faculdade, também meus professores, técnicos administrativos, meus pais, familiares, amigos e colegas por me darem apoio e instrução em todos os momentos.

RESUMO

Este trabalho apresenta um estudo comparativo entre as abordagens de bancos de dados relacional e de documentos, usando, respectivamente MySQL (com a *engine* InnoDB) e MongoDB, para armazenar e consultar dados referentes à Câmara dos Deputados do Brasil. Foram realizados testes que visam analisar o desempenho de ambas as abordagens, considerando espaço de armazenamento, complexidade na escrita das consultas e tempo de processamento. Resultados mostram que, considerando os três fatores em conjunto, o MySQL é melhor, pois ocupa menos espaço e tem melhor desempenho no tempo de processamento. Por outro lado, a representação dos dados no formato de documento pelo MongoDB representa a estrutura original dos dados. A análise resultante deste trabalho serve de *backend* uma aplicação de visualização interativa de dados de votações da Câmara que permite aos usuários acompanhar, de forma visual, como os deputados estão votando.

Palavras-chave: MySQL. MongoDB. Banco de Dados Relacional. InnoDB. Banco de Dados orientado a Documentos. Avaliação de Desempenho. Câmara dos Deputados do Brasil. Dados Políticos. Dados Abertos.

A Comparative Study of Modeling and Performance Evaluation between the Relational and Document Database for the Open Data of the Brazilian Chamber of Deputies

ABSTRACT

This work presents a comparative study between relational and document database approaches, using respectively MySQL (with the InnoDB engine) and MongoDB, to store and consult data referring to the Chamber of Deputies of Brazil. Tests were performed to analyze the performance of both approaches, considering storage space, query writing complexity and processing time. Results show that considering the three factors together, MySQL is better because it takes up less space and performs better in processing time. On the other hand, a representation of the data in the MongoDB document format represents an original data structure. The analysis resulting from this work serves as a backend to an interactive visualization application of House voting data that allows users to visually monitor how deputies are voting.

Keywords: MySQL. MongoDB. Relational Database. InnoDB. Document-based Database. Performance Evaluation. Brazil's Chamber of Deputies. Political Data. Open Data.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BD	Banco de Dados
CSV	Comma Separated Values
DB	Data Base
DBMS	Data Base Management System
DDD	Domain Driven Design
DML	Document Modeling Language
EER	Extended Entity Relationship
ER	Entity Relationship
GAF	General Access Frequency
HTML	Hyper-Text Markup Language
JSON	JavaScript Object Notation
MAF	Minimal Access Frequency
MySQL	My Structured Query Language
NoSQL	No Structured Query Language ou Not Only Structured Query Language
OO	Object Oriented / Orientado a Objetos
POSIX	Portable Operating System Interface for Unix
QUEL	Query Language
SGBD	Sistema Gerenciador de Banco de Dados
XML	eXtensive Markup Language

SUMÁRIO

1 INTRODUÇÃO

As iniciativas de dados abertos pelos governos em geral visam dar ao cidadão comum a possibilidade de conhecer e acompanhar as diversas atividades desenvolvidas no âmbito das administrações públicas, desde o nível municipal até o federal. No Brasil, diversas entidades e setores governamentais disponibilizam dados para acesso, variando desde arquivos em formatos específicos que podem ser buscados isoladamente até formulários online através dos quais consultas específicas podem ser feitas.

Este trabalho trata da modelagem de dados disponibilizados pelo portal de dados abertos da Câmara dos Deputados do Brasil, a fim de servir como *backend* para outro sistema, o CivisAnalysis (de Borja; Freitas, 2015), que é uma aplicação de visualização interativa de dados de votações da Câmara. O objetivo principal desse sistema é permitir aos usuários acompanhar, de forma visual, como os deputados estão votando. A primeira versão do CivisAnalysis foi desenvolvida por (de Borja; Freitas, 2015), utilizando dados armazenados diretamente em memória, tendo em vista que testes iniciais com armazenamento em banco de dados mostraram desempenho insatisfatório. Uma nova versão do CivisAnalysis (SILVA; SPRITZER; FREITAS, 2018) implementada com uma abordagem de múltiplas visões coordenadas, incorporou novas visualizações desses mesmos dados, mas ainda trabalhando com dados em memória.

As informações da Câmara estão disponibilizadas através de uma API (*Application Programming Interface*) para consultas aos dados, podendo ser usada em navegadores para obter dados ou arquivos referentes à Câmara, dos mais variados contextos, como, por exemplo, gastos parlamentares, lista de deputados, lista de partidos, etc. As informações retornadas podem ser de diversos formatos, dentre eles, CSV (*Comma Separated Values*) e JSON (*JavaScript Object Notation*).

Antes de decidir qual abordagem de banco de dados seria o ideal (relacional ou orientada a documentos), é necessário fazer a modelagem dos dados, criar os bancos, alimentá-los com dados reais da Câmara, fazer testes de desempenho comparativos e avaliar em quais situações cada solução seria melhor, possivelmente adotando uma estratégia híbrida.

Existem diversos trabalhos que fazem comparação entre bancos de dados relacionais e não-relacionais. (HOMRICH; MERGEN, 2018) comparam dois bancos de dados populares, Neo4J e MySQL. Ao fazer consultas em dados complexos, usando apenas recursos nativos de suas linguagens, o Neo4J não mostrou eficiência tão boa

quanto o MySQL. Já (SEUFITELLI; MORO; DAVIS JR., 2015) apresentam uma forma híbrida de armazenar dados de esquemas físicos geográficos, usando bancos de dados. (COSTA; VILAIN; MELLO, 2016) usaram uma ferramenta que automatiza a migração de bancos de dados relacionais para bancos de dados de chave-valor, fazendo também testes de desempenho. Outro trabalho (KOLONKO, 2018) testa e compara desempenho de bancos de dados relacional (OracleDB) e não-relacional (MongoDB), usando *YCSB Benchmark*, enquanto (Puangsaijai; Puntheeranurak, 2017) fazem uma comparação entre um banco de dados relacional (MariaDB) e chave-valor (Redis) para *Big Data*. (VICKNAIR et al., 2010) fazem testes de desempenho comparativos entre bancos de dados Relacional (MySQL) e orientado a Grafo (Neo4J).

Este trabalho tem, por similaridade, a avaliação de desempenho entre diferentes abordagens de bancos de dados, e tem como diferencial os bancos de dados analisados, que, no caso, são MySQL e MongoDB. São relatados testes de desempenho entre um banco de dados relacional, o MySQL, e um banco de dados de documentos, o MongoDB, para verificar quais situações cada um seria o mais adequado, ou se uma solução híbrida seria melhor. Antes de executar as consultas dos cenários de teste, são inseridos todos os dados nos bancos, e obtidos o tempo necessário para inserção e o espaço em disco usado. Depois, os cenários de teste são executados, fazendo consultas, e analisados seus tempos para avaliar o desempenho de cada banco, individualmente. Como contribuição deste trabalho, é feita avaliação de desempenho entre dois bancos de dados distintos, MySQL e MongoDB, para determinar qual seria o mais adequado para a aplicação CivisAnalysis usar.

O texto está organizado da seguinte maneira. O Capítulo 2 descreve a fundamentação teórica e compara os principais trabalhos relacionados. No Capítulo 3, é apresentada, em detalhes, a proposta de modelagem dos dados abertos da Câmara dos Deputados. No Capítulo 4, estão documentados testes de desempenho para avaliação dos bancos de dados MySQL e Mongo, usando as informações da Câmara dos Deputados. Finalmente, o Capítulo 5 finaliza o trabalho e apresenta sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Este capítulo aborda tanto os fundamentos sobre bancos de dados como os trabalhos relacionados. Está organizado da seguinte maneira. Na seção 2.1 introduzimos os aspectos principais de modelagem relacional e não-relacional necessários ao trabalho. A seção 2.2 descreve os principais trabalhos relacionados, dentre eles, os que falam sobre bancos de dados de documentos, comparam bancos de dados relacionais e não-relacionais e apresentam análise de desempenho.

2.1 Abordagens Relacional e Não-Relacional

Baseado em (SILBERSCHATZ; KORTH; SUDARSHAN, 2011), é possível definir que Banco de Dados é uma área da computação que visa armazenar grandes volumes de dados relacionados entre si em meios permanentes, para que a recuperação desses dados possa ser feita de forma rápida para consultas, inserções ou atualizações de informações. Existem vários tipos de bancos de dados, que se dividem em dois grandes grupos: relacionais (dados estruturados armazenados em tabelas) e não-relacionais (dados não-estruturados armazenados em arquivos, grafos, chave-valor ou colunas). Cada contexto vai exigir um determinado tipo de banco de dados que seja o mais adequado para a situação, ou potencialmente uma solução híbrida, devido a restrições de negócio, ou pela forma como os dados estão sendo gerados ou disponibilizados.

Um banco de dados relacional é estruturado em tabelas que se relacionam entre si por chaves, evitando duplicidade de dados, e podem fazer consultas por junções. Por exemplo, num sistema, há a tabela Aluno, que tem os campos: idAluno, nome, idade; a tabela Disciplinas, que tem os campos: idDisciplina, nome. Como diversas disciplinas podem ter diversos alunos, então é necessário ter uma tabela de histórico, esta contendo as referências para os IDs de cada tabela, dataInicial, dataFinal, statusAprovacao, por exemplo. Se quisermos obter todas as informações de todos os alunos de graduação da disciplina Y, então é necessário juntar as informações dessas 3 tabelas, e devolver o resultado da consulta, possivelmente filtrando os campos de interesse para não haver campos desnecessários mostrados.

Para um banco de dados relacional entrar em funcionamento, é necessário definir previamente um esquema (como as tabelas estão estruturadas, seus relacionamentos, gatilhos e procedimentos), já para os bancos de dados de documentos, isso não é

necessário, o que agiliza o desenvolvimento de aplicações onde os dados são volumosos e não estão bem estruturados, voltadas a *Big Data*, por exemplo. O armazenamento de informações se dá por documentos, e a recuperação de dados em consultas é bastante agilizado, devido a inexistência de junções, que afetam o desempenho de bancos de dados MySQL. Mas, as garantias de integridade que um banco MySQL têm, praticamente não existem em bancos NoSQL. Em suma, os cenários ideais de um banco de dados MySQL são quando os dados têm relação entre si e é necessário manter integridade de dados; já os cenários ideais para banco de dados NoSQL são quando as relações não existem, são desconhecidas ou evoluem constantemente, quando o projeto do sistema é mais simples (agilizando desenvolvimento de aplicações) e quando precisa de velocidade e escalabilidade.

Já um banco de dados de documentos é estruturado em documentos, ou seja, insere-se documentos no formato CSV ou JSON, por exemplo, e fazem consultas neles, devolvendo resultados a partir destes documentos. Um banco de dados de documentos, pela forma que é estruturado, não é convencional fazer operações de junções, mas é possível de se fazer. Se tivermos um documento contendo itens de compras, com os campos: lista de itens e lista de quantidades, por exemplo, uma consulta possível seria obter a lista dos itens mais vendidos para cada compra, fazendo uma consulta neste documento.

O capítulo 4 do livro "Next Generation Databases - NoSQL, NewSQL, and Big Data"(HARRISON, 2015) apresenta um panorama geral do que é um banco de dados de documentos, afirmando que tal abordagem surgiu por causa de 3 razões: primeiro, um conflito entre programação orientada a objetos e o modelo relacional de banco de dados que frustrava os desenvolvedores de software e que motivou o movimento de banco de dados em meados de 1990; segundo, porque os formatos de documentos são autodescritivos, não precisando perguntar ao programa que os criou como estão formados e suportando acesso a consultas ad hoc no banco de dados, que estava ausente em armazenamentos puros de chave-valor; e terceiro, porque eles se alinharam bem aos paradigmas dominantes de programação web, em particular o AJAX (*Asynchronous JavaScript and XML*). Um banco de dados de documentos está entre a rigidez de um esquema de banco de dados relacional e um modelo sem esquema de chave-valor, não precisando de esquema, apesar de seus documentos terem alguma estruturação. Também fala que os primeiros bancos de dados de documentos foram construídos em torno do XML (*eXtensive Markup Language*), pois ele dava muitas facilidades aos programadores

Web, fazendo com que DBMSs relacionais adotassem também o formato XML, permitindo operações em suas linguagens declarativas nesses arquivos. Mas os maiores problemas do XML são o desperdício de espaço e o custo computacional de manipulação desses arquivos, sendo necessário usar JSON. Os bancos de dados XML são voltados a sistemas de gerência de conteúdo, já os bancos de dados JSON são voltados ao suporte de aplicações web operacionais. JSON pode ser usado em AJAX. Também fala sobre como dados são modelados em bancos de dados relacionais e de documentos, sendo que a abordagem de documentos poderia usar a 3ª forma normal dos relacionais, mas não seria adequado pois geralmente há menor quantidade de coleções, além de não ser natural para o programador e dos bancos de dados de documentos não suportarem operações de junções. Também fala dos bancos de dados JSON primitivos, como o CrouchDB, MemBase, CrouchBase e como eles evoluíram ao longo do tempo até chegar ao MongoDB. O foco da modelagem em bancos de dados relacionais está em como representar a natureza dos dados a serem armazenados, enquanto que em bancos de dados de documentos está na natureza das consultas. Muitos DBMSs estão adotando JSON também, devido a sua popularidade, à produtividade dos programadores e acessibilidade dos dados. No futuro, todos os DBMSs adotarão o JSON.

2.2 Trabalhos Relacionados

(CLIFTON; GARCIE-MOLINA, 1988) propõem a abordagem de banco de dados fundamentada em documentos, para permitir integridade, compartilhamento, segurança e busca eficiente, sendo genérico o suficiente para suportar os mais variados tipos de documentos, e apresentam uma nova linguagem declarativa, chamada DML (CLIFTON; GARCIE-MOLINA, 1988). Os autores consideram que documentos parcialmente estruturados seriam o ideal para uma base de dados de documentos. A ideia do trabalho é ter, na maioria dos casos, um servidor de documentos e diversos usuários em suas estações de trabalho, que fazem requisições ao servidor, no estilo similar a um banco de dados relacional. O artigo define documentos como se fossem um conjunto de triplas, onde cada tripla contém informações do seu tipo, a chave, e seu valor. O tipo da tripla define o propósito da mesma, além de definir os tipos de dados da chave e dos dados. A chave serve para indexação, e o valor é um dado do documento propriamente dito, que também pode servir para indexação ou armazenamento de arquivos como textos ou imagens. O artigo também define DML (CLIFTON; GARCIE-MOLINA, 1988), que não

é uma linguagem de programação completa, mas sim algo do estilo do SQL e QUEL, em que pode ser embutida na linguagem de programação hospedeira. Nela, as categorias de consulta são relativas à recuperação de documentos e suas cadeias de ponteiros, e buscas por documentos a partir de um critério. Além disso, também é necessário que somente certas triplas de um documento possam ser retornadas pela busca. O artigo também define aspectos sobre esta linguagem (mais especificamente sobre operações em conjuntos, filtros básicos, conjuntos de documentos, conjuntos de filtros, correspondência de variáveis, operações sobre ponteiros, iterações, e operações básicas) e sobre o sistema no geral (tipos de dados, tipos das triplas, índices, tipos das palavras de busca, e gerência de transações).

(LIMA; MELLO, 2015) descrevem um processo que converte um modelo conceitual em representações lógicas eficientes em um banco de dados de documentos NoSQL, baseadas na carga de trabalho do sistema. O trabalho propõe um modelo lógico a partir de dois parâmetros de entrada: o esquema conceitual e a informação de carga de trabalho do sistema (quantidade de dados instanciados na aplicação e operações principais sobre esses dados). Os autores usaram o EER (*Extended Entity Relationship*), pois existem construtos essenciais para a modelagem conceitual para mapear o esquema em arquivos JSON. O contexto do trabalho é sobre *e-commerce*, e os resultados mostram que obtiveram menos acessos ao banco de dados pelas consultas das aplicações. Para avaliar a proposta (comparando a quantidade de acessos), experimentos mostraram o projeto de *datasets* de *e-commerce* e compararam o esquema reprojetado com o método proposto. Para tanto, considerou-se o volume de dados anotado em esquemas conceituais EER, e as operações para o esquema. Eles fazem uso do GAF (*General Access Frequency*), que calcula a frequência de acesso para cada entidade, e do MAF (*Minimal Access Frequency*), que calcula a frequência mínima de acesso, informada pelo projetista do software. Usam a ideia de agregados, provenientes do DDD (*Domain Driven Design*), que é uma abordagem OO (*Object Oriented*). Resumidamente, os agregados são coleções de dados relacionados, que contêm hierarquia e são considerados como uma unidade.

(HOMRICH; MERGEN, 2018) comparam dois bancos de dados populares, Neo4J e MySQL (este usando as *engines* InnoDB e MyISAM), ao fazerem consultas em dados complexos de uma rede rodoviária da Califórnia, usando apenas recursos nativos de suas linguagens. O resultado mostrou que o Neo4J não conseguiu a boa eficiência mostrada pelo MySQL. No artigo, os autores apresentam uma noção geral das características de

cada banco de dados, incluindo a forma como as entidades e os relacionamentos são modelados e as respectivas linguagens declarativas.

(SEUFITELLI; MORO; DAVIS JR., 2015) apresentam uma forma híbrida de armazenar dados de esquemas físicos geográficos, devido à observação dos autores de que é necessário mapear restrições espaciais geográficas. O que os motivou a fazer uma abordagem híbrida foi perceber que há um crescente número de DBMSs não-relacionais, sendo possível, em tese, usar recursos de alguns deles para mapear restrições espaciais, combinando uma abordagem usando DBMSs relacionais e não-relacionais. Com isso, a ideia é fazer uso do melhor desempenho de DBMSs não-relacionais com a possibilidade de definir restrições espaciais geográficas dos DBMSs relacionais. Eles também perceberam que não é possível mapear um esquema conceitual de geografia física em DBMSs não relacionais, o que é possível em DBMSs relacionais. Os autores concluem que há dificuldades em mapear dados geográficos para DBMSs NoSQL, sendo necessário melhorias na abordagem feita pelo artigo.

Santos e co-autores (SANTOS; MORO; DAVIS, 2015) descrevem como armazenar dados móveis e espaciais usando bancos de dados relacionais e NoSQL, mostrando que o projeto físico dos bancos de dados deve evoluir para obter o desempenho de bancos de dados NoSQL e manter a consistência de bancos de dados MySQL. A tendência está sendo adotar bancos de dados NoSQL para dados espaciais, pois requerem poucas atualizações e muitas consultas. Mas, não é algo trivial garantir a consistência em dados espaciais, sendo necessário criar funções complexas no DBMS ou na aplicação. O objetivo do artigo é identificar pontos fortes e fracos de ferramentas de gerência de dados espaciais, fazendo testes de desempenho para diferentes consultas comuns para aplicações espaciais (apesar do artigo não focar em benchmarking). Os DBMSs avaliados são PostgreSQL com PostGIS e *pgrouting*, MongoDB e Neo4J com Neo4J-Spatial. Os resultados dos testes variaram muito, sendo necessário definir quais seriam as funcionalidades mais comuns e qual seria o dado alvo para selecionar o DBMS espacial mais adequado. O artigo apresenta uma metodologia, que consiste em obter dados espaciais reais (para que os índices e consultas sejam feitos da maneira mais realista possível), definir um conjunto de consultas, inserções e atualizações, criação de uma interface genérica (para que novos bancos de dados possam ser facilmente adicionados), e demonstrar que as avaliações de desempenho devem ser orientadas a dados e às características da aplicação. O artigo ainda propõe uma nova métrica de avaliação calculando vértices por segundos, além de medir o tempo de execução.

2.2.1 Análise Comparativa

Esta seção faz uma análise comparativa entre este trabalho e os trabalhos relacionados citados na seção anterior. Primeiramente, são definidos os quesitos de comparação e, na sequência, é apresentada a comparação propriamente dita.

A seguir, são litados os itens de comparação definidos para comparar o presente trabalho e os trabalhos relacionados::

- Uso de Bancos de Dados Relacionais: indica se o trabalho analisado avalia banco de dados relacional;
- Uso de Bancos de Dados de Documentos: indica se o trabalho analisado avalia banco de dados de documentos;
- Comparação entre Bancos de Dados Relacionais e Não-Relacionais: se o trabalho faz uma análise comparativa entre bancos de dados relacionais e não-relacionais;
- Medição do número de acessos ao BD: se o trabalho mede a quantidade de requisições feitas ao BD;
- Medição do tempo para realizar consultas: se o trabalho utiliza métricas para avaliar consultas de dados;
- Medição do tempo de carga de dados: se o trabalho faz medições de tempo ao carregar dados ao BD;
- Medição do espaço ocupado pelos dados do BD: se o trabalho avalia métricas para avaliar o espaço ocupado pelos dados armazenados.

A Tabela 2.1 apresenta uma visão geral dos trabalhos relacionados de acordo com os quesitos mencionados acima.

Tabela 2.1: Tabela comparativa a partir de diversos itens entre os trabalhos relacionados

Artigo	Usa Banco de Dados Relacional?	Usa Banco de Dados Não-Relacional?	Faz comparação entre BDs Relacionais e BDs Não-Relacionais?	Mede o número de acessos ao BD?	Mede o tempo para realizar uma consulta?	Mede o tempo de carga dos dados?	Mede o espaço ocupado pelos dados do BD?
(CLIFTON; GARCIE-MOLINA, 1988)	Não	Sim	Não	Não	Não	Não	Não
(LIMA; MELLO, 2015),	Não	Sim	Não	Sim	Sim	Não	Não
(HOMRICH; MERGEN, 2018),	Sim	Sim	Sim	Não	Sim	Sim	Não
(SEUFITELLI; MORO; DAVIS JR., 2015)	Sim	Sim	Sim	Não	Não	Não	Não
(SANTOS; MORO; DAVIS, 2015)	Sim	Sim	Sim	Sim	Sim	Sim	Sim

Fonte: autor

Dependendo do contexto, é mais adequado ou implementar o sistema usando somente banco de dados relacional, ou somente usando não-relacional, ou uma solução híbrida. Por isso, é necessário fazer um estudo sobre as características de cada um, procurando, assim, determinar qual solução implementar num sistema real. O banco de dados adotado pode ser centralizado ou distribuído.

Os trabalhos mais completos são os de Santos e co-autores e Homrich e co-autores (HOMRICH; MERGEN, 2018) (SANTOS; MORO; DAVIS, 2015). Aprimoramentos possíveis nesses trabalhos seriam: avaliar o desempenho entre todas as abordagens existentes de bancos de dados (relacional, orientada a objetos, documentos, grafos, chave-valor, colunar, etc.) para que as análises de avaliação de desempenho pudessem ser mais ricas, e criar *benchmarks*, para que as avaliações sejam feitas de forma mais automática, podendo ser usadas como padrões de indústria.

O trabalho apresentado no presente texto poderia também fazer comparações de desempenho entre todas as abordagens de bancos de dados, identificando com melhor precisão qual seria a mais adequada, possivelmente adotando uma solução híbrida, e também fazer uso dos *benchmarks* citados no parágrafo anterior.

3 MAPEANDO DADOS ABERTOS DA CÂMARA DE DEPUTADOS PARA MODELOS LÓGICOS DE BANCOS DE DADOS RELACIONAIS E DE DOCUMENTOS

Este capítulo descreve as duas abordagens propostas para o mapeamento lógico e físico dos dados das votações na Câmara de Deputados para bancos de dados relacionais e de documentos. A escolha dos modelos relacionais e de documentos, em detrimento de outros, se deu para que fosse possível a comparação entre o modelo mais popular utilizado pelas aplicações comerciais (relacional) e o modelo de documentos, que permite armazenar as proposições votadas como um único documento, de forma mais próxima ao que ocorre no armazenamento dos dados disponíveis para download pela Câmara de Deputados. Além disso, os modelos baseados em documentos têm sido usados extensivamente em aplicações atuais que utilizam o formato JSON para processamento de dados.

Este capítulo, inicialmente, descreve o domínio de problema, que é o processo de votação na Câmara de Deputados. Em seguida, para cada uma das abordagens de mapeamento propostas, Relacional (Seção 3.2) e Documentos (Seção 3.3), são especificados o modelo lógico e o algoritmo de mapeamento para o modelo físico de armazenamento dos dados.

3.1 Descrição do Domínio

O domínio escolhido para o trabalho foi o de Dados Abertos da Câmara de Deputados com o objetivo de acompanhar como os deputados votam e, assim, se distribuem no espaço conhecido como *espectro político*.

Cada proposição (Projeto de Lei, Medida Provisória, Projeto de Emenda Constitucional) pode ser aberta em vários tópicos a serem votados, sendo que as votações podem ocorrer em diferentes momentos (por ex, dias da semana, meses ou até anos). Por exemplo, a proposição identificada por PL 8703/2007 foi votada em 2 sessões, cujos objetivos eram diferentes (a 1ª sessão tinha por objetivo o requerimento de votação nominal, e a 2ª, a votação de um artigo desta proposição), e que ocorreram em datas e horas diferentes (a 1ª sessão foi em 04/10/2017, às 20:47 horas, e a 2ª, em 04/10/2017, às 22:18 horas). Para cada sessão, a bancada partidária orienta os votos para cada

partido, e os deputados podem ou não acatar as orientações de voto de seus partidos.

Cada deputado pode fazer parte de uma frente parlamentar, e cada partido pode fazer parte de um bloco partidário. As frentes são uniões de deputados e de outros parlamentares que trabalham por uma causa (por ex, Frente Parlamentar em Defesa do Trânsito Seguro, que, em tese, trabalha para que tenhamos um trânsito mais seguro), e os blocos partidários são uniões de partidos, que podem sugerir votos aos deputados de cada partido em cada sessão de votação.

3.2 Modelagem Relacional

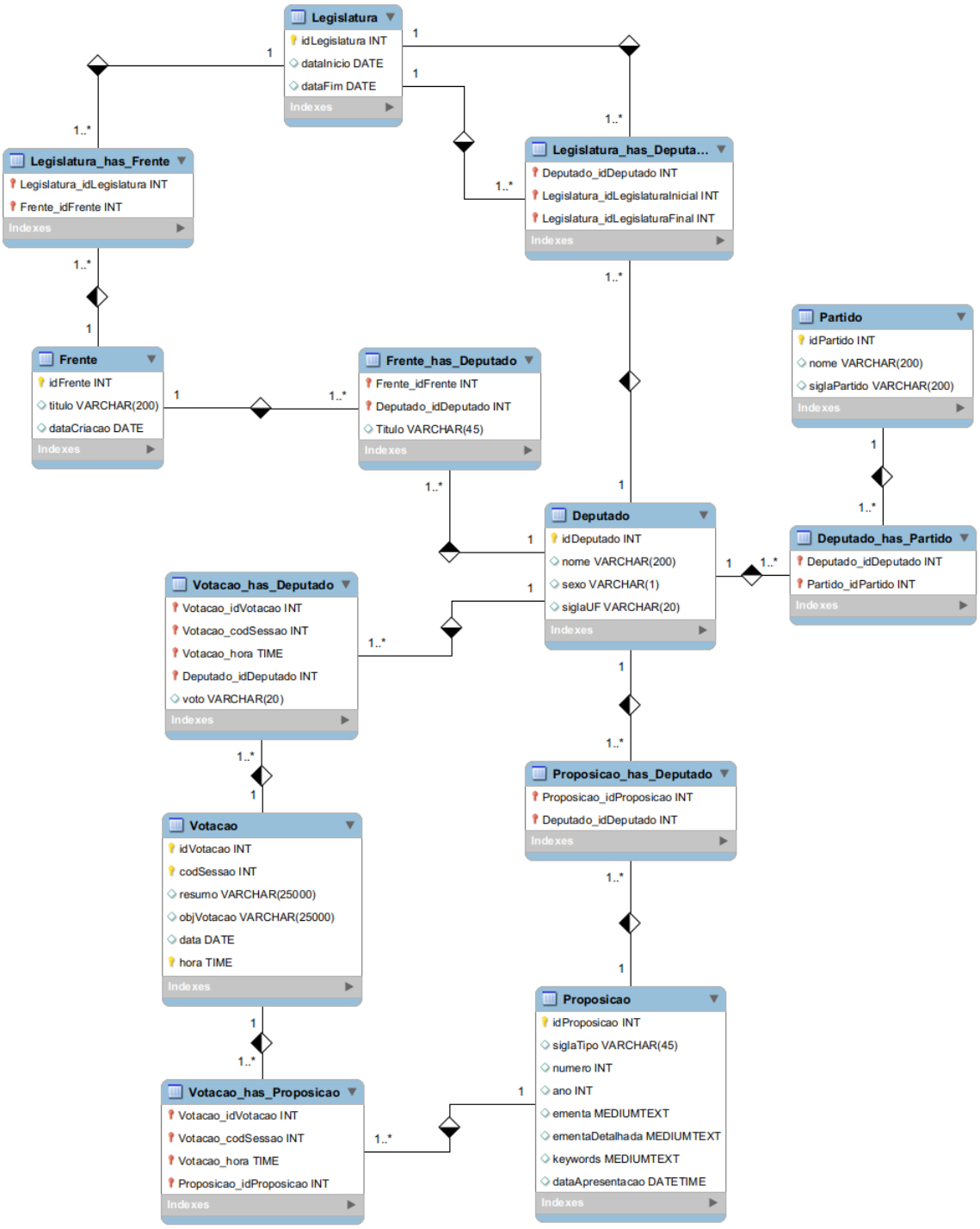
O objetivo desta seção é fazer a modelagem de dados para criar um banco de dados relacional utilizando um Diagrama Entidade Relacionamento. Com este diagrama, o entendimento das informações da Câmara se torna mais fácil antes de começar a implementação do banco de dados. Em seguida, é especificado o algoritmo de mapeamento para o armazenamento físico do banco de dados.

A Figura 3.1 ilustra a modelagem ER (Entidade-Relacionamento) dos Dados Abertos da Câmara dos Deputados. A ideia é representar Deputados, Frentes, Legislaturas, Partidos, Proposições e Votações em forma de diagrama para que seja possível identificar as principais entidades e seus relacionamentos, e posteriormente criar um banco de dados relacional e armazenar suas informações e realizar consultas de dados.

As entidades são as seguintes:

- Deputado: representa um deputado na Câmara.
- Frente: é uma união de deputados que trabalham por uma causa.
- Legislatura: representa o período de tempo de para o qual os deputados foram eleitos.
- Partido: armazena informações de um partido da Câmara.
- Proposicao: é uma proposta de lei autorada por um deputado ou por alguém da população.
- Votacao: representa um conjunto de sessões de votações de proposições.

Figura 3.1: Diagrama ER do domínio dos Dados Abertos da Câmara dos Deputados



Fonte: autor

Os relacionamentos são os seguintes:

- Deputado_has_Partido: representa um histórico dos partidos dos quais um deputado fez parte.
- Frente_has_Deputado: contém um histórico de deputados e as frentes das quais

participou em seu(s) mandato(s).

- Legislatura_has_Deputado: armazena um histórico de legislaturas de todos os deputados.
- Legislatura_has_Frente: representa um histórico das legislaturas que uma frente existiu.
- Proposicao_has_Deputado: representa a autoria das proposições a serem votadas em plenário.
- Votacao_has_Deputado: representa as votações de um deputado.
- Votacao_has_Proposicao: representa as votações de uma proposição. Uma proposição pode ter vários tópicos a serem votados, sendo necessário criar uma entidade especialmente para este fim.

3.3 Modelagem de Documentos

O objetivo desta seção é fazer a modelagem de dados para criar um banco de dados de documentos. Com esta modelagem, o entendimento das informações da Câmara se torna mais fácil antes de começar a implementação do banco de dados. Em seguida, é especificado o algoritmo de mapeamento para o armazenamento físico do banco de dados.

As coleções foram modeladas simplesmente inserindo os arquivos CSVs de dados no Mongo. Para as votações, foram inseridos os JSONs, obtidos da Câmara dos Deputados, através de requisições HTML GET. A Figura 3.2 ilustra a estruturação das coleções.

As coleções são as seguintes:

- Deputados: representam os deputados na câmara.
- DeputadosPartidos: são informações dos partidos e seus deputados.
- Frentes: são uniões de deputados que trabalham por uma causa.
- FrentesDeputados: armazena um conjunto de frentes e os deputados que as compõem.
- Legislaturas: representa os períodos de tempo para os quais há eleições de Deputados Federais.
- Partidos: armazena informações de partidos da câmara.
- Proposicoes: representa as propostas a serem votadas e que foram criadas por um

deputado ou por alguém da população.

- Votacoes: representam os dados dos conjuntos de sessões de votações de proposições.

Figura 3.2: Imagem da estruturação das coleções no Mongo

```
Coleção: deputados
{
    _id: Int32
    nome: String
    ufNascimento: String
    idLegislaturaInicial: Int32
    idLegislaturaFinal: Int32
}

Coleção: deputadosPartidos
{
    _id: Int32
    dados_id_legislatura: Int32
    dados_nome: String
    dados_sigla_partido: String
    dados_sigla_uf: String
    dados_id_partido: String
}

Coleção: frentes
{
    _id: Int32
    titulo: String
    dataCriacao: String
}

Coleção: frentesDeputados
{
    _id: ObjectId
    titulo: String
    frente_id: Int32
}

Coleção: legislaturas
{
    _id: Int32
    dataInicio: String
    dataFim: String
}

Coleção: partidos
{
    _id: Int32
    dados_nome: String
    dados_sigla: String
}
```

```
Coleção: proposicoes
{
    _id: Int32
    siglaTipo: String
    numero: Int32
    ano: Int32
    ementa: String
    ementaDetalhada: String
    keywords: String
    dataApresentacao: String
}

Coleção: votacoes
{
    _id: Int32
    proposicao: {
        Sigla: String
        Numero: String
        Ano: String
        Votacoes: (Mixed) {
            Votacao: [
                @Resumo: String
                @Data: String
                @Hora: String
                @ObjVotacao: String
                @codSessao: String
                votos: {
                    Deputado: [
                        @Nome: String
                        @IdeCadastro: String
                        @Partido: String
                        @UF: String
                        @Voto: String
                    ]
                }
                orientacaoBanca: {
                    bancada: [
                        @Sigla: String
                        @Orientacao: String
                    ]
                }
            ]
        }
    }
}
```

Fonte: autor

3.4 Considerações Finais

Neste capítulo foram apresentados a explicação do domínio de aplicação das modelagens, que são no contexto da Câmara dos Deputados do Brasil; seus modelos de dados, voltados para MySQL e Mongo; e os algoritmos de mapeamento, que ilustram de forma geral o como deve ser feito o mapeamento dos dados para o armazenamento físico no banco de dados.

Em termos de facilidades, a modelagem visou ser a mais simples e extensível possível, não precisando fazer tanto esforço para implementar funções auxiliares, ou funcionalidades novas ao sistema (incluindo o mapeamento de campos e entidades que não foram contemplados neste trabalho).

Dentre as limitações desta modelagem estão: nem todos os atributos das entidades foram mapeados, nem outras informações do site da Câmara, como, por exemplo, gastos parlamentares, órgãos, situações dos deputados, dos partidos e das proposições, eventos, entre outros. A implementação foi feita em Python 3 para sistemas Linux. Dessa forma, qualquer sistema operacional que seja UNIX-like e siga os padrões POSIX permite a execução dos *scripts*.

4 AVALIAÇÃO EXPERIMENTAL

Este capítulo descreve a avaliação de desempenho entre os dois bancos de dados contemplados neste trabalho: MySQL (com a *engine* InnoDB) e Mongo. São dados detalhes da infraestrutura computacional utilizada neste trabalho, os SGBDs usados e a base de dados. Na sequência, na Seção 4.3, é detalhado como foi feito o mapeamento físico do modelo do domínio para os SGBDs e, na Seção 4.4, é detalhado como foram planejadas e executadas as consultas. Na Seção 4.5, é descrita uma avaliação que cobre todas as consultas feitas a partir dos cenários de teste.

4.1 Ambiente Computacional e Obtenção dos Dados

O trabalho foi desenvolvido num computador configurado com 4 processadores Intel Core i3, velocidade de 2.93 GHz e 12 GB de RAM. O sistema operacional utilizado é o Kubuntu 18.04, com KDE Plasma versão 5.12.7, KDE Frameworks versão 5.44.0, Qt versão 5.9.5, versão do kernel 4.15.0-54 *generic* e sistema de arquivos ext4.

Os SGBDs usados foram MySQL (com a *engine* InnoDB), com auxílio do MySQL Workbench para criar o Diagrama ER e as tabelas, e Mongo, com auxílio do MongoDB *Compass Community Edition*, para melhor visualização dos documentos inseridos no banco de dados. Não foi usada indexação no MongoDB. Os scripts foram desenvolvidos em Python 3 (MySQL *Connector* e PyMongo).

A base de dados usada, conforme já mencionado, foi a disponibilizada pela Câmara dos Deputados do Brasil, de acordo com a iniciativa de Dados Abertos.

Foi utilizada a nova API dos Dados Abertos da Câmara dos Deputados, aba Arquivos. Para os partidos, os dados foram obtidos fazendo uma consulta na aba API RESTful. Para as votações, foi necessário fazer consultas em HTML na API antiga, para obter as proposições de fato votadas em plenário e suas votações.

Ao todo, foram obtidos 7 arquivos CSV (sendo que um deles, o de proposições, é uma fusão de 40 arquivos, com proposições dos anos de 1980 até 2019) e 1330 arquivos JSON de votações de proposições, que foram obtidos através de requisições HTML GET, totalizando 2632815 registros no banco de dados.

Para armazenar estes dados no banco de dados relacional, foi necessária a constante exploração de dados, no sentido de, além de ver o que a API RESTful retornava de resultados, ver que informações continham os arquivos na aba Arquivos, ver

como as votações seriam obtidas do site, obter os dados relevantes referentes as votações e saber como os dados seriam inseridos com primitivas SQL (para o banco de dados relacional) e Mongo (para o banco de dados de documentos). Esse processo será detalhado na seção 4.2.

4.2 Mapeamento Físico para Sistemas Gerenciadores de Bancos de Dados

Inicialmente, antes de realizar a carga dos dados nos dois bancos que buscamos avaliar, é necessária uma etapa de pré-processamento para otimizar o tempo de inserção dos dados.

Para o banco de dados MySQL:

1) Conectar ao BD;

2) Obter os nomes das tabelas principais e das tabelas de histórico do BD criado, mantendo as principais antes das de histórico, e armazenar em uma lista;

3) Abrir e executar cada arquivo de script SQL, baseado na lista de entidades do passo 2;

4) Após a carga de dados de todas as entidades no BD, fechar a conexão do BD.

Para o banco de dados Mongo:

1) Conectar ao Mongo;

2) Obter a listagem dos arquivos CSV, removendo CSVs desnecessários (os de votações);

3) Para cada arquivo CSV, remover a extensão CSV, criar a coleção de dados, obter as colunas de interesse e inserir os dados no Mongo.

Para obter as votações:

1) Abrir o arquivo allMotionsPerYear.json, que contém todas as proposições votadas em Plenário;

2) Gravar em arquivos JSON as respostas do HTML GET para cada proposição votada, salvando no nome do arquivo o idVotacao e idProposicao, para facilitar /

simplificar processamento

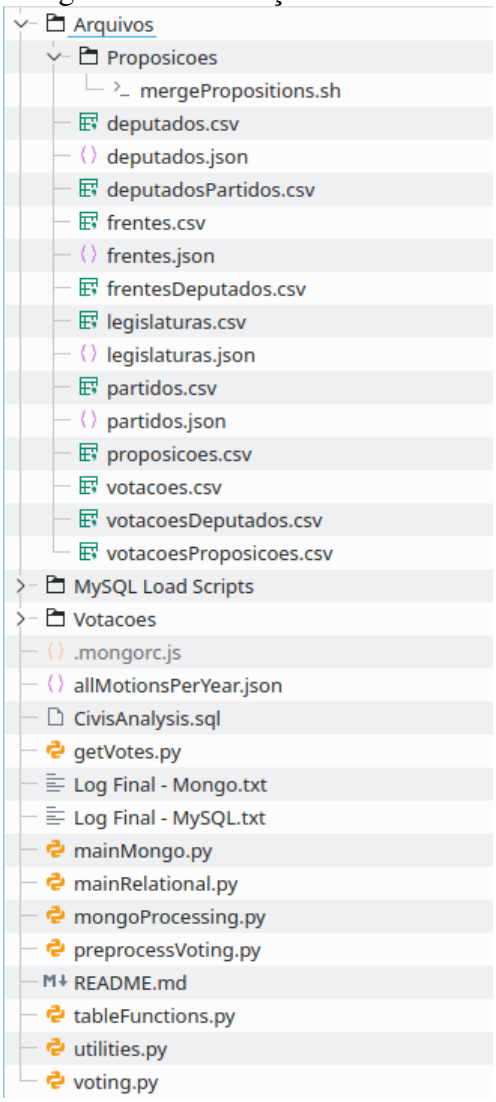
Para pré-processar as votações:

1) Abrir cada arquivo de votação;

2) Criar 3 CSVs, contendo as entradas correspondentes para as tabelas que contém dados sobre as votações (Votacao, Votacao_has_Deputado e Votacao_has_Proposicao).

Ao criar CSVs extras referentes às votações, votações de deputados e proposições votadas (que são baseados nos arquivos JSON de votações), precisamos eliminar entradas inválidas das votações (pois, aparentemente, não são apenas deputados que podem votar nas sessões de votações), e eliminar arquivos de votações cujos conteúdos são duplicados (para não haver erro de chaves primárias no SGBD MySQL); após isto, é necessário filtrar campos que não seriam interessantes neste momento, e inserir informações das entidades nos bancos de dados. O arquivo “allMotionsPerYear.json” foi cedido por Rodrigo Nunes Moni da Silva. A estruturação dos diretórios está na figura 4.1.

Figura 4.1: Imagem da estruturação dos diretórios do projeto



Fonte: autor

4.2.1 Mapeamento Físico para o Banco de Dados MySQL

Para inserir os dados no banco de dados MySQL, é necessário executar, um por um, os scripts de LOAD DATA (que estão na subpasta “MySQL Load Scripts”). Para que cada um seja executado, é necessário abrir o arquivo SQL, e executar as instruções deste arquivo no MySQL. Este algoritmo não tem entradas, nem saídas. Devemos tomar cuidado em executar os *scripts* referentes às entidades principais antes dos *scripts* referentes às entidades de histórico, para que não dê erro de restrições de integridade referencial. Note que, antes de executar este programa, é necessário baixar os arquivos do site da Câmara dos Deputados, e salvá-los na subpasta “Arquivos”, no formato CSV.

No Apêndice A - Scripts SQL para cargas de dados no banco de dados MySQL tem os *scripts* usados para carregar os dados ao banco de dados relacional, e no Apêndice D - Script SQL para criação do banco de dados MySQL contém o *script* para criação das tabelas do banco de dados usado neste trabalho.

O algoritmo para mapear os dados para o banco é o seguinte:

Algoritmo 1: CARGA DE ARQUIVOS DE DADOS NO MYSQL

Entrada: n/a

Saída: n/a

```
1 início
2     conectarBD_MySQL()
3     ObterListaEntidades_MySQL()
      /* É necessário reordenar as entidades para que o
         processamento nelas seja feita nas tabelas
         principais, depois nas de histórico          */
4     ReordenarListaEntidades_MySQL()
5     para cada entidade da Lista Ordenada de Entidades faça
          /* Abre o arquivo que contém o script SQL de
             LOAD DATA                                  */
6         abrirArquivo_SQLLoad()
          /* Executa o script de LOAD DATA              */
7         executarArquivo_SQLLoad()
8     fim
9 fim
```

4.2.2 Mapeamento Físico para o Banco de Dados MongoDB

Para inserir os dados no banco de dados Mongo, é necessário, antes de mais nada, listar os arquivos CSV da pasta “Arquivos”, e remover os CSVs desnecessários para este processamento, que são referentes às votações (pois essas informações serão inseridas de outra forma, direto dos arquivos de votações cujo formato é JSON). Depois disso, para cada arquivo CSV da subpasta “Arquivos”, remover a extensão “.csv”, criar a coleção de dados, pré-processar para obter os campos de interesse e inserir os dados obtidos no pré-processamento no Mongo. Note que, antes de executar este programa, é necessário

baixar os arquivos do site da Câmara dos Deputados, e salvá-los na subpasta “Arquivos”, no formato CSV.

O algoritmo para mapear os dados é o seguinte:

Algoritmo 2: CARGA DE ARQUIVOS DE DADOS NO MONGO

```
Entrada: n/a
Saída: n/a
1  início
2      CONECTARBD_MONGO()
3      OBTERLISTACSVS()
       /* É necessário remover os CSVs dos arquivos de
          votação                                        */
4      REMOVERELEMENTOSDESNECESSARIOSLISTACSVS()
5      para cada arquivo de dados CSV faça
           /* Remove a extensão '.csv' e cria a coleção
              de dados com o nome do arquivo CSV         */
6          CRIARCOLECAO()
           /* Obtém os campos de interesse do CSV        */
7          PREPROCESSARCSV()
           /* Insere o conteúdo na coleção               */
8          INSERIRCONTEUDO()
9      fim
10 fim
```

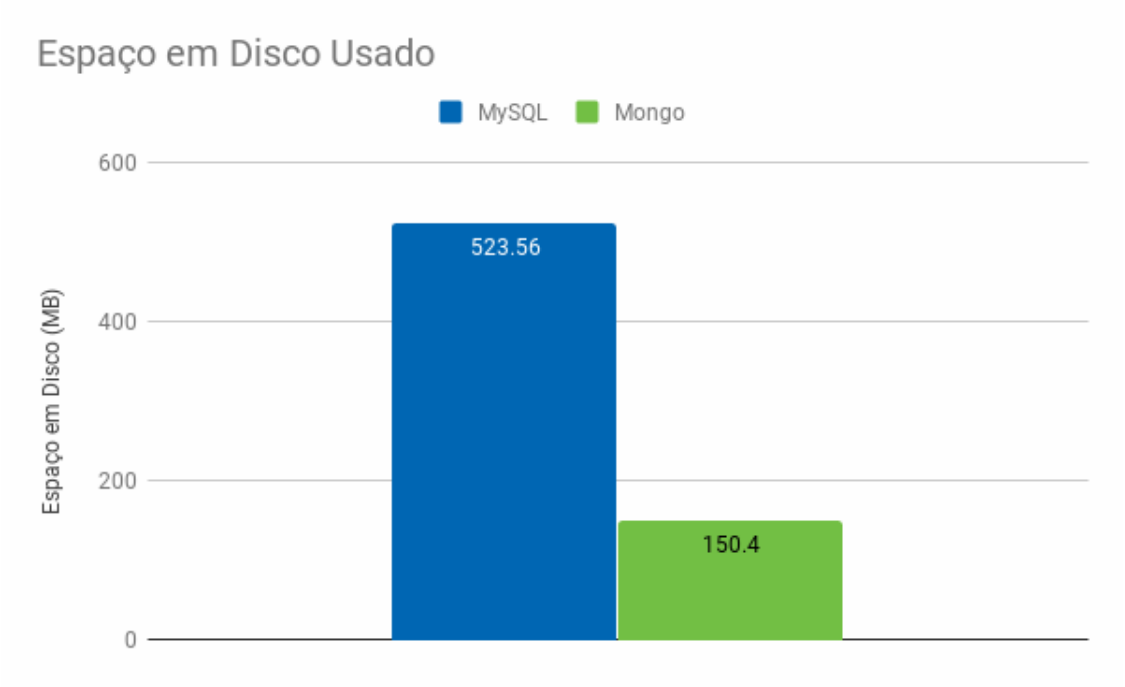
4.3 Experimento I: Avaliação do Armazenamento Físico

No MySQL, foram inseridos 2.622.389 registros em aproximadamente 5.58 minutos, totalizando 523.5625 MB de dados; no Mongo, foram 699.940 registros em aproximadamente 5.88 minutos, totalizando 150.4MB de dados.

Assim, como mostrado nas figuras 4.2, 4.3 e 4.4, respectivamente, o MySQL consumiu muito mais espaço em disco, teve mais registros inseridos, mas carregou os dados ligeiramente mais rápido em relação ao Mongo. Isso se deve ao fato que foi feita uma otimização na parte de carga de dados do MySQL, ao pré-processar os arquivos de votação, para que criasse 3 arquivos CSV a mais, sendo, respectivamente, Votacao,

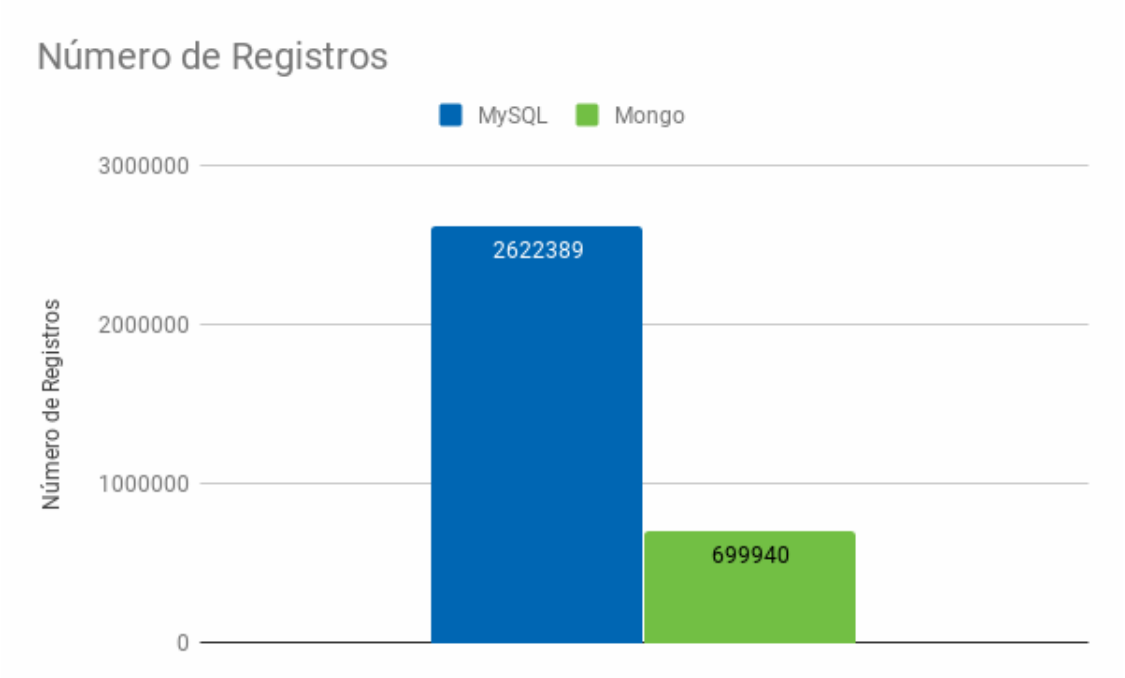
Votacao_has_Deputado e Votacao_has_Proposicao, sendo possível usar somente scripts de LOAD DATA para carregar todas as informações necessárias ao banco, deixando os códigos em Python 3 mais simples de entender.

Figura 4.2: Espaço em Disco Ocupado no MySQL e Mongo



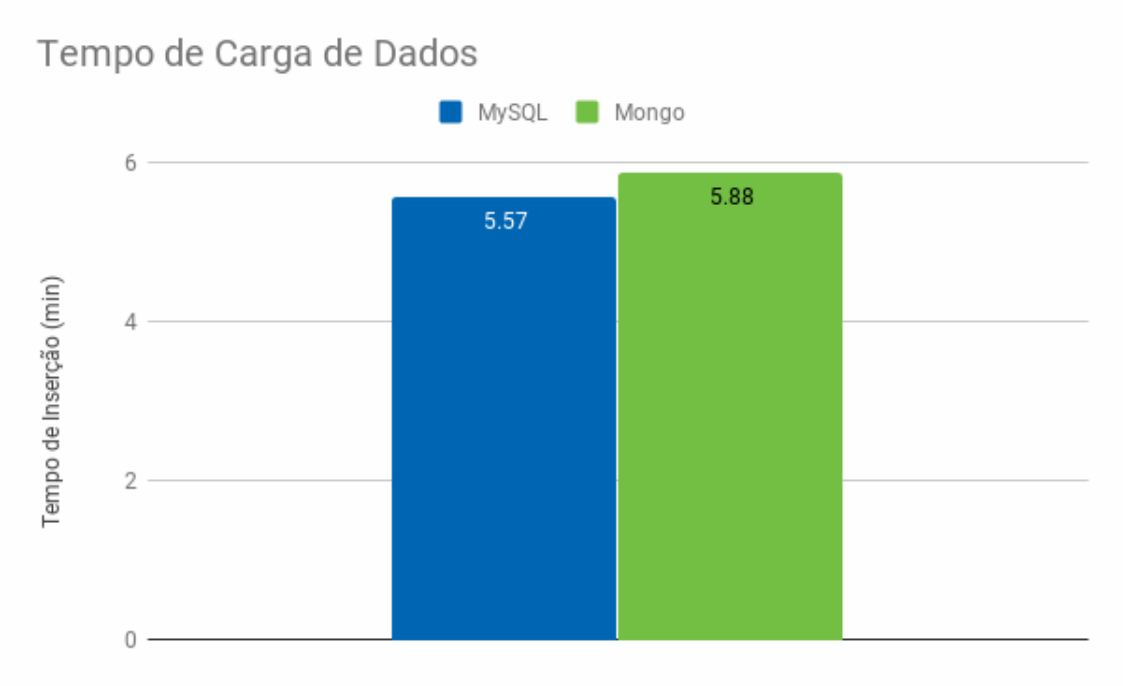
Fonte: autor

Figura 4.3: Número de Registros inseridos no MySQL e Mongo



Fonte: autor

Figura 4.4: Tempo de Carga no MySQL e Mongo



Fonte: autor

4.4 Experimento II: Avaliação da Complexidade e do Tempo de Processamento de Consultas

Esta seção descreve os experimentos que avaliaram o tempo de processamento de consultas tanto em MySQL quanto no Mongo. Inicialmente foram especificados quatro cenários, detalhados em 7 consultas. Estes cenários foram elaborados pelo Rodrigo Nunes Moni da Silva, e aproveitados no presente trabalho. Estes cenários representam os principais requisitos do sistema CivisAnalysis. Uma captura de tela está ilustrada na Figura 4.5.

Figura 4.5: Captura de tela da execução do CivisAnalysis 2.0

Fonte: autor

As consultas para MySQL e Mongo estão listadas, respectivamente, nos Apêndice B - Scripts para consultas no banco de dados MySQL e Apêndice C - Scripts para consultas no banco de dados Mongo. Cada consulta foi executada 11 vezes, sendo que a 1ª vez foi usada para obter o tempo inicial para carga de dados em memória, e mais 10 vezes, para que as consultas sejam feitas com dados em memória, fazendo a média dessas 11 execuções, totalizando 144 execuções (77 para MySQL e 77 para Mongo). Foi necessário executar 10 vezes pois a duração das consultas poderia mudar um pouco, e assim obteve-se o valor médio de duração das consultas.

4.4.1 Cenário de Avaliação

Para testar a avaliação do processamento de consultas, foram elaborados alguns cenários de teste e suas consultas, tanto em MySQL quanto em Mongo. Os cenários de teste são os seguintes:

- Cenário 1: Obter todas proposições de fato votadas em Plenário.
- Cenário 2: Obter dados sobre as proposições (i.e., Autores, Ementa, Explicação Ementa, Indexação, etc).
- Cenário 3: Obter todas as votações para cada proposição.
- Cenário 4: Obter deputados e seus dados (i.e., Partido, Estado pelo qual foram eleitos).

A partir dos cenários, foram elaboradas as seguintes consultas nos bancos de dados:

- Consulta 1: Buscar o ID de todas as Proposições de fato votadas em Plenário.
- Consulta 2: Buscar os dados sobre todas as Proposições que foram votadas em Plenário.
- Consulta 3: Buscar todas as informações sobre as votações de cada proposição votada em Plenário.
- Consulta 4: Buscar os dados básicos de todos os Deputados.
- Consulta 5: Buscar os dados dos partidos ao qual pertencem os Deputados.
- Consulta 6: Buscar os dados das Frentes Parlamentares.
- Consulta 7: Buscar os dados das legislaturas em que os Deputados tem mandato.

4.4.2 Consulta 1- Buscar o ID de todas as Proposições de fato votadas em Plenário

Esta consulta visa obter uma informação básica (que é o identificador único das proposições) do histórico de proposições votadas em plenário. Como pode ser visto na Tabela 4.1, ambas as consultas nos dois bancos de dados são simples, tendo demandado pouco esforço para sua elaboração. A consulta requer acesso a uma tabela apenas, no caso do MySQL, e a uma coleção no caso Mongo.

Tabela 4.1: Consulta 1 - Buscar o ID de todas as Proposições de fato votadas em Plenário

Nome da Consulta	Descrição	Apêndices para Referência
ID das Proposições	Este script contém o retorno do identificador único de todas as proposições existentes na Câmara.	B, C

Fonte: autor

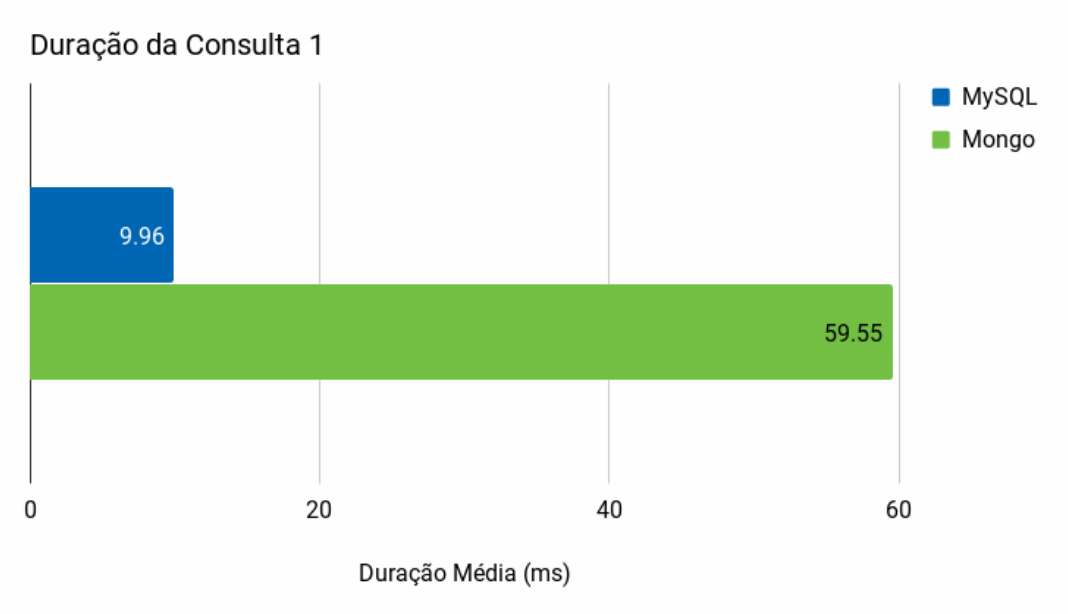
Como podemos ver na Figura 4.6, o MySQL demandou menos tempo para executar a consulta, quando comparado ao Mongo, pois uma seleção simples em MySQL é mais rápida que uma busca em Mongo.

Tabela 4.2: Consulta 2 - Buscar os dados sobre todas as Proposições que foram votadas em Plenário

Nome da Consulta	Descrição	Apêndices para Referência
Informações Proposições	Este script contém o retorno das informações relacionadas às proposições votadas em Plenário.	B, C

Fonte: autor

Figura 4.6: Desempenho da consulta para “Buscar o ID de todas as Proposições de fato votadas em Plenário” nos bancos de dados MySQL e Mongo

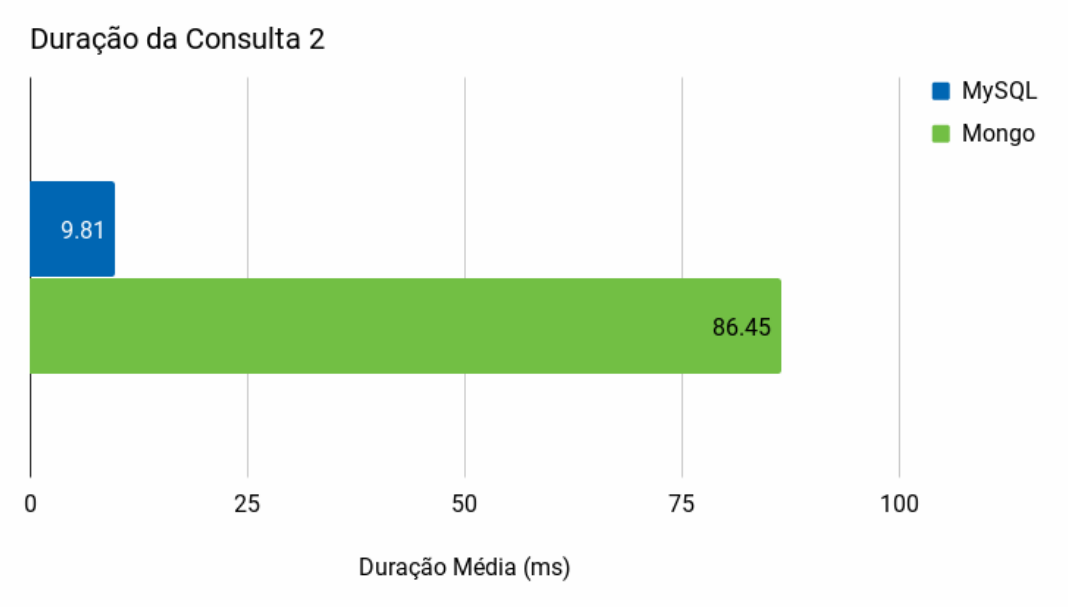


Fonte: autor

4.4.3 Consulta 2 - Buscar os dados sobre todas as Proposições que foram votadas em Plenário

Esta consulta visa obter várias informações das proposições a partir do histórico de proposições votadas em plenário. As consultas podem ser visualizadas na Tabela 4.2. Ambas as consultas nos dois bancos de dados são complexas, sendo necessário fazer junções (em MySQL) ou agregações (no Mongo), e demandam algum tempo para elaborá-las. Para a consulta do Mongo, foi necessário implementar uma função de contagem de tempo em JavaScript, para que o tempo fosse mais preciso.

Figura 4.7: Desempenho da consulta “Buscar os dados sobre todas as Proposições que foram votadas em Plenário” nos bancos de dados MySQL e Mongo



Fonte: autor

Como podemos ver na Figura 4.7, o MySQL demandou menos tempo para executar a consulta, se comparado com o Mongo. Isso se deve ao fato de que, no Mongo, é feita uma agregação, e em seguida uma filtragem de resultados. Como o Mongo não foi projetado para fazer agregações (junções), demorou mais que o MySQL.

4.4.4 Consulta 3 - Buscar todas as informações sobre as votações de cada proposição votada em Plenário

Esta consulta visa obter várias informações a respeito das votações a partir do histórico de proposições votadas em plenário. A descrição da consulta em ambos os SGBDs pode ser visualizada na Tabela 4.3. A consulta no MySQL é complexa, sendo necessário fazer muitas junções. Já no Mongo, é uma consulta simples. A elaboração da consulta do MySQL demanda algum tempo, enquanto que a do Mongo, é relativamente rápida.

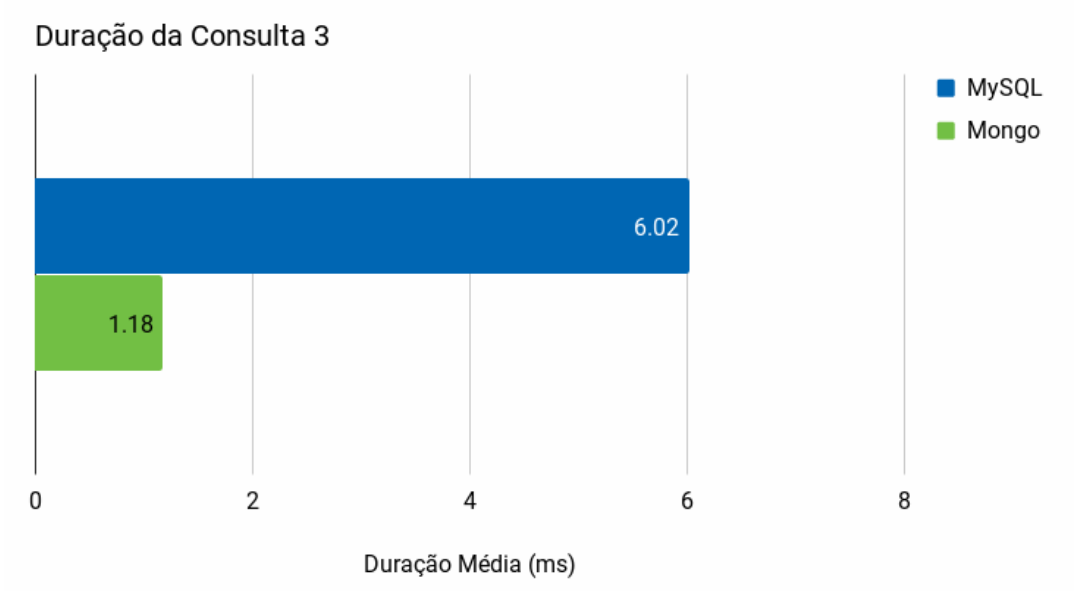
Tabela 4.3: Consulta 3 - Buscar todas as informações sobre as votações de cada proposição votada em Plenário

Nome da Consulta	Descrição	Apêndices para Referência
Proposições Votadas	Este script contém o retorno das votações para cada proposição que foi votada em sessões do Plenário.	B, C

Fonte: autor

Em termos de tempo de execução, como podemos ver na Figura 4.8, o MySQL precisou de mais tempo para executar a consulta, se comparado ao Mongo, pois, em MySQL, foi necessário usar junções, que são operações demoradas, enquanto que no Mongo, foi usada uma busca simples.

Figura 4.8: Desempenho da consulta “Buscar todas as informações sobre as votações de cada proposição votada em Plenário” nos bancos de dados MySQL e Mongo



Fonte: autor

4.4.5 Consulta 4 - Buscar os dados básicos de todos os Deputados

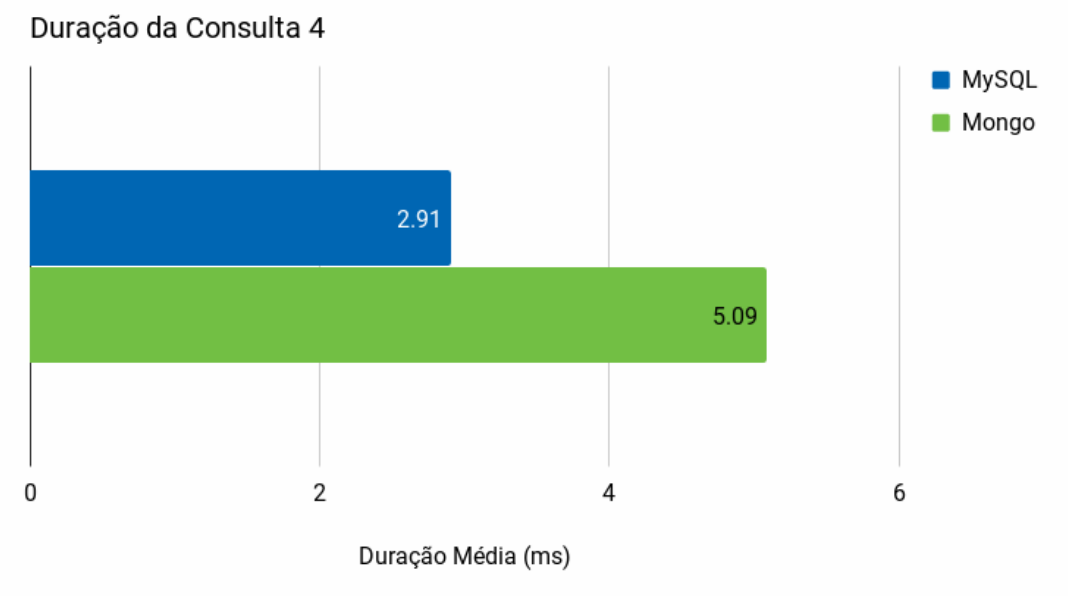
Esta consulta visa obter as informações básicas dos deputados da câmara e pode ser visualizada na Tabela 4.4. Ambas as consultas nos dois bancos de dados são simples, tendo demandado pouco tempo para elaborá-las.

Tabela 4.4: Consulta 4 - Buscar os dados básicos de todos os Deputados

Nome da Consulta	Descrição	Apêndices para Referência
Informações Básicas dos Deputados	Este script contém o retorno das informações básicas para cada deputado da Câmara.	B, C

Fonte: autor

Figura 4.9: Desempenho da consulta “Buscar os dados básicos de todos os Deputados” nos bancos de dados MySQL e Mongo



Fonte: autor

Como podemos ver no gráfico 4.9, o MySQL precisou de pouco tempo para executar a consulta, se comparado com o Mongo. Isso se deve ao fato de que uma seleção em MySQL é mais rápida que uma busca em Mongo.

4.4.6 Consulta 5 - Buscar os dados dos partidos ao qual pertencem os Deputados

Esta consulta visa obter informações referentes aos deputados e partidos ao qual pertencem. As consultas podem ser vistas na Tabela 4.5. A consulta no MySQL é complexa, sendo necessário fazer muitas junções, já no Mongo, é simples. A elaboração da consulta do MySQL demanda algum tempo, enquanto que a do Mongo, é relativamente rápida.

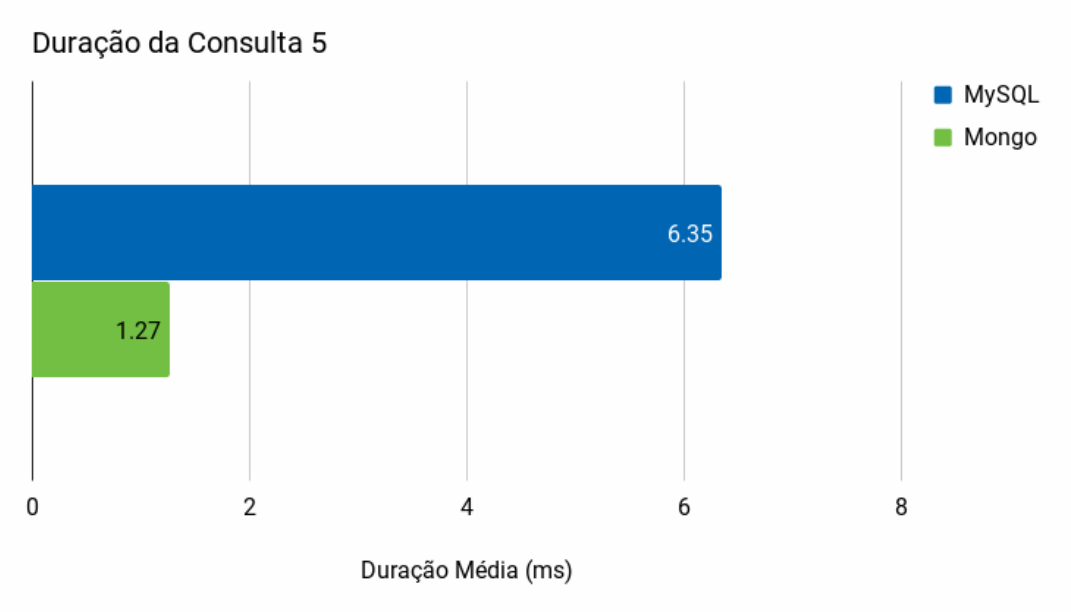
Tabela 4.5: Consulta 5 - Buscar os dados dos partidos ao qual pertencem os Deputados

Nome da Consulta	Descrição	Apêndices para Referência
Informações dos Partidos dos Deputados	Este script contém o retorno das informações dos partidos de cada deputado da Câmara.	B, C

Fonte: autor

Como podemos ver na Figura 4.10, o MySQL precisou de mais tempo para executar a consulta, se comparado com o Mongo, pois, em MySQL, foi necessário usar junções, que são operações demoradas, enquanto que no Mongo, foi usada uma busca simples.

Figura 4.10: Desempenho da consulta “Buscar os dados dos partidos ao qual pertencem os Deputados” nos bancos de dados MySQL e Mongo



Fonte: autor

4.4.7 Consulta 6 - Buscar os dados das Frentes Parlamentares

Esta consulta visa obter informações referentes aos deputados e frentes a qual pertencem. Ela pode ser vista na Tabela 4.6. A consulta no MySQL é complexa, sendo necessário fazer muitas junções; já no Mongo, é uma consulta simples. A elaboração da consulta do MySQL demanda algum tempo, enquanto que a do Mongo, é relativamente

rápida.

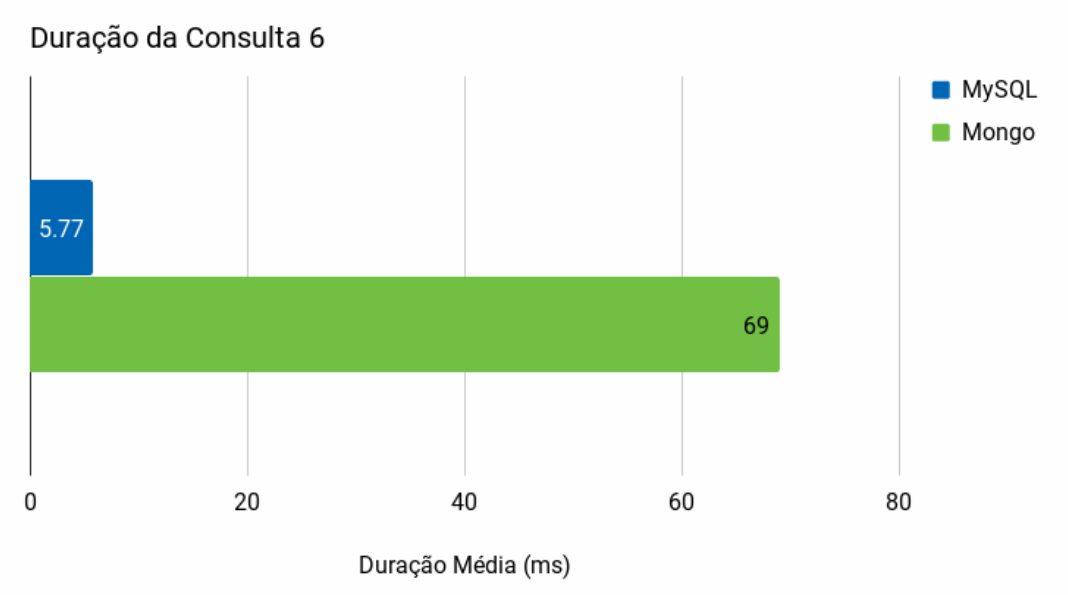
Tabela 4.6: Consulta 6 - Buscar os dados das Frentes Parlamentares

Nome da Consulta	Descrição	Apêndices para Referência
Informações das Frentes dos Deputados	Este script contém o retorno das informações das frentes de cada deputado da Câmara.	B, C

Fonte: autor

As medidas de tempo do processamento das consultas podem ser vistas na Figura 4.11. Observa-se que o MySQL demandou pouco tempo para executar a consulta, se comparado com o Mongo. Isso se deve ao fato que, nesta coleção, o ID do Mongo é sobre um campo ObjectId, ao invés de ser sobre um inteiro. Possivelmente o índice criado sobre esses objetos faça com que as consultas demorem mais, se comparado com índices criados sobre inteiros.

Figura 4.11: Desempenho da consulta “Buscar os dados das Frentes Parlamentares” nos bancos de dados MySQL e Mongo



Fonte: autor

4.4.8 Consulta 7 - Buscar os dados das legislaturas em que os Deputados têm mandato

Esta consulta visa obter informações referentes aos deputados e suas legislaturas, e pode ser vista na Tabela 4.7. A consulta no MySQL é complexa, sendo necessário fazer muitas junções; já no Mongo, é uma consulta simples. A elaboração da consulta do MySQL demanda algum tempo, enquanto que a do Mongo, é relativamente rápida.

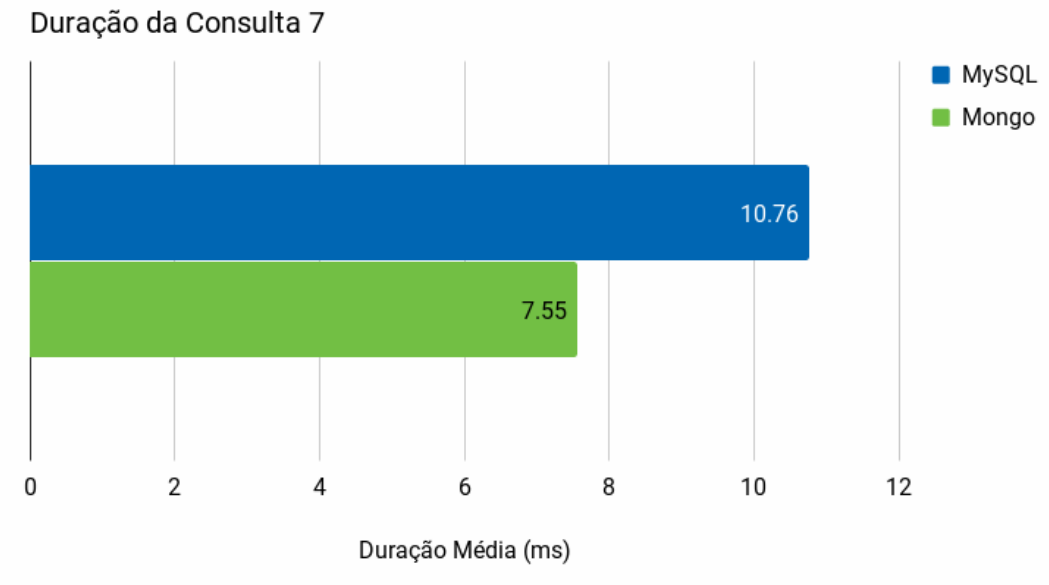
Tabela 4.7: Consulta 7 - Buscar os dados das legislaturas em que os Deputados tem mandato

Nome da Consulta	Descrição	Apêndices para Referência
Informações das Legislaturas dos Deputados	Este script contém o retorno das informações das legislaturas de cada deputado da Câmara.	B, C

Fonte: autor

Quanto ao tempo de execução, como podemos ver na Figura 4.12, o MySQL demandou mais tempo para executar a consulta, se comparado com o Mongo, pois, em MySQL, foi necessário usar junções, que são operações demoradas, enquanto que no Mongo, foi usada uma busca simples.

Figura 4.12: Desempenho da consulta “Buscar os dados das legislaturas em que os Deputados têm mandato” nos bancos de dados MySQL e Mongo



Fonte: autor

4.5 Avaliação Geral dos Resultados

Há 3 fatores a considerar na avaliação de desempenho realizada neste trabalho: tempo de carga de dados, espaço em disco consumido e duração média das consultas. A decisão sobre qual banco de dados usar será determinada pelas limitações do contexto de implementação do sistema.

Como o tempo de carga foi similar entre os dois bancos, não faz muita diferença qual deles selecionar. Mas, como o MySQL teve tempo de carga ligeiramente menor, então ele deve ser avaliado como melhor opção, neste caso.

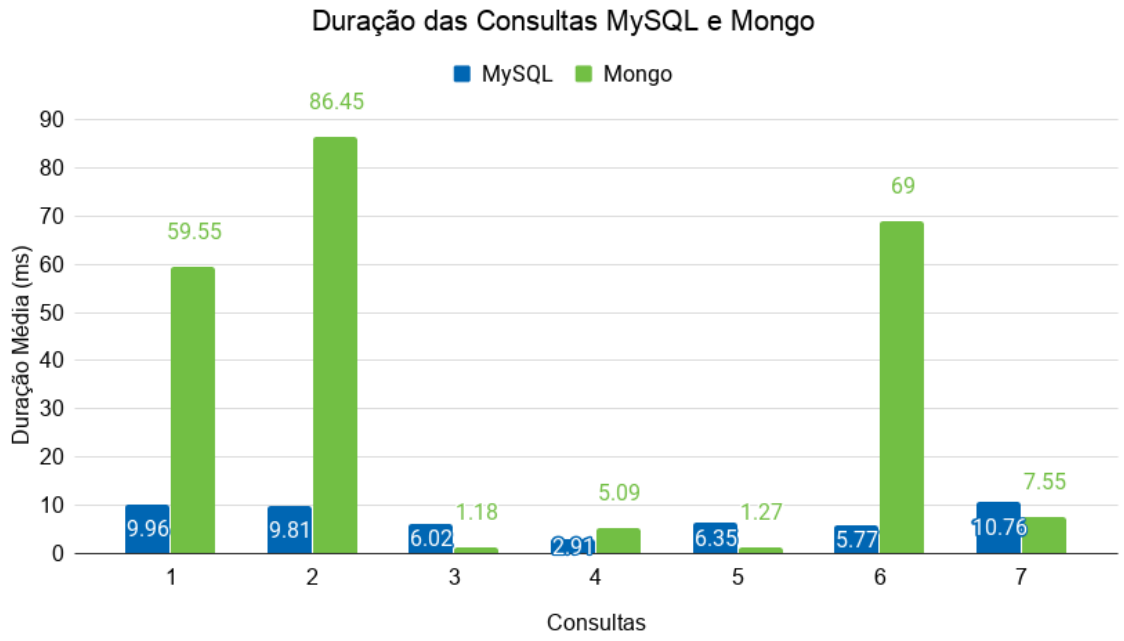
Avaliando o espaço necessário em disco para armazenar os dados utilizados neste trabalho, o Mongo demandou menor espaço porque os dados ficam agrupados dentro dos documentos. O MySQL teve muito mais espaço em disco consumido porque não foi habilitado compactação na *engine* InnoDB. Por padrão, ela usa 16KB de tamanho de página, possivelmente havendo fragmentação interna, além dos dados armazenados conterem dados do usuário e do sistema. Se o contexto do sistema precisar de economia de espaço em disco, talvez o Mongo seja a alternativa mais viável.

No que se refere a tempo de execução das consultas, no MySQL elas demoraram pouco tempo, enquanto que no Mongo, seus tempos oscilaram, pois é uma tecnologia mais madura que Mongo, sendo mais otimizada ao realizar consultas, com auxílio de seus índices. Se este fator for o principal no contexto do sistema, então MySQL deve ser selecionado como melhor.

Considerando os 3 fatores de desempenho em conjunto, o MySQL se sai melhor, apesar de consumir mais espaço em disco, se comparado ao Mongo. Podemos argumentar que, a longo prazo, o que vai exigir mais do sistema são as consultas dos usuários, então, o tempo de execução deve ter um peso maior na avaliação.

O gráfico de desempenho está apresentado a seguir.

Figura 4.13: Desempenho das consultas nos banco de dados MySQL e Mongo. Consulta 1: Buscar o ID de todas as Proposições de fato votadas em Plenário; Consulta 2: Buscar os dados sobre todas as Proposições que foram votadas em Plenário; Consulta 3: Buscar todas as informações sobre as votações de cada proposição votada em Plenário; Consulta 4: Buscar os dados básicos de todos os Deputados; Consulta 5: Buscar os dados dos partidos ao qual pertencem os Deputados; Consulta 6: Buscar os dados das Frentes Parlamentares; Consulta 7: Buscar os dados das legislaturas em que os Deputados têm mandato.



Fonte: autor

Para o MySQL:

- A consulta 4 foi a que menos demorou, pois é uma seleção simples numa tabela com poucos milhares de registros.
- A consulta 7 foi a que mais demorou, possivelmente pela forma que os dados do usuário estão combinados com os dados do sistema na *engine* usada (InnoDB).

Para o Mongo:

- A consulta 3 foi a que menos demorou, possivelmente pela forma que os dados estão organizados em disco.
- A consulta 2 demorou muito porque é a única consulta que faz agregações, sendo um processo lento no Mongo.

No geral, podemos tirar as seguintes conclusões:

- As consultas em MySQL, em geral, demoraram pouco tempo, comparado com o Mongo, que houve oscilações.
- MySQL demandou mais espaço em disco que o Mongo.
- O mapeamento para MySQL resultou em mais registros que o para o Mongo.
- MySQL demandou menos tempo para inserir dados do que o Mongo.

Em suma, o banco de dados MySQL obteve melhor desempenho, sendo uma opção viável para a utilização no CivisAnalysis.

5 CONCLUSÕES

Este trabalho fez análise de desempenho entre os bancos de dados MySQL (com a *engine* InnoDB) e Mongo, analisando os tempos para carga de dados, tempos para consultas, e espaço em disco consumido. Para tanto, foi necessário alimentar ambos os bancos com dados reais, no caso, da Câmara dos Deputados do Brasil, provenientes da Lei dos Dados Abertos, para que os testes fossem feitos o mais próximo possível da realidade.

As principais diferenças entre bancos de dados são as seguintes:

- Modelagem: no MySQL, foi necessário criar um diagrama ER, pois é uma etapa de suma importância para qualquer banco de dados relacional, que determina se o banco de dados vai ter um desempenho bom e atender às necessidades dos usuários a longo prazo. Também, desenvolver um banco de dados relacional se assemelha ao desenvolvimento em cascata, da engenharia de *software*, sendo necessário demandar bastante tempo na modelagem conceitual, para que dê menos problemas no futuro do sistema; para o Mongo, não foi necessário criar uma modelagem, visto que bastava carregar os dados dos documentos, diretamente.
- Consultas: as consultas em Mongo são geralmente mais simples de serem escritas, se comparadas com MySQL, raramente precisando fazer junções de arquivos (ou agregações, usando o termo do Mongo), visto que o conteúdo é armazenado agregado em um documento.
- Carga de Dados: em MySQL foi mais direto de ser feito a carga dos dados, enquanto que no Mongo, foi necessário pré-processar cada coluna de interesse dos arquivos CSV, criando também um campo "_id", para depois carregar os dados.
- Espaço em Disco: o MySQL consumiu muito espaço em disco comparado com o Mongo, pois não foi feita a compactação dos dados. Talvez se fosse usada outra *engine*, o consumo de espaço fosse diferente.

Com os resultados deste trabalho, o CivisAnalysis poderia usar um banco de dados relacional para armazenar os dados da Câmara dos Deputados do Brasil, tornando a experiência do usuário mais agradável. Além disso, os *scripts* feitos para este trabalho podem ser estendidos de forma facilitada, agilizando a adição de novas funcionalidades ao CivisAnalysis.

Dentre as possíveis melhorias deste trabalho estão: fazer testes de desempenho com MySQL usando outras *engines*, e também, para todas (inclusive a usada neste trabalho), habilitar a compactação de dados. Para isso, possivelmente, não será necessário alterar muito os scripts, além de habilitar a *flag* de compressão, basta fazer *forward engineering* no MySQL Workbench, especificando a *engine* que será utilizada; outra melhoria possível seria adaptar os scripts em Python 3 para rodar em sistemas que não seguem os padrões POSIX; outra melhoria seria usar mais dados da Câmara (além do que foi feito), como gastos parlamentares, blocos partidários, etc, fazer mais testes de desempenho, para ficar mais completa a análise, e fazer uma modelagem mais adequada ao CivisAnalysis, na parte do MongoDB.

REFERÊNCIAS

CLIFTON, C.; GARCIE-MOLINA, H. The design of a document database. In: **Proceedings of the ACM Conference on Document Processing Systems**. New York, NY, USA: ACM, 1988. (DOCPROCS '88), p. 125–134. ISBN 0-89791-291-8. Available from Internet: <http://doi.acm.org/10.1145/62506.62528>.

COSTA, A. da; VILAIN, P.; MELLO, R. Uma camada para o mapeamento de instruções sql dml para o banco de dados nosql chave-valor voldemort. In: **Anais do XII Simpósio Brasileiro de Sistemas de Informação**. Porto Alegre, RS, Brasil: SBC, 2016. p. 224–231. ISSN 0000-0000. Available from Internet: <https://sol.sbc.org.br/index.php/sbsi/article/view/5966>.

de Borja, F. G.; Freitas, C. M. D. S. Civisanalysis: Interactive visualization for exploring roll call data and representatives' voting behaviour. In: **2015 28th SIBGRAPI Conference on Graphics, Patterns and Images**. [S.l.: s.n.], 2015. p. 257–264. ISSN 2377-5416.

HARRISON, G. **Next Generation Databases - NoSQL, NewSQL, and Big Data**. [S.l.]: Springer International Publishing, 2015. 57–63 p. ISBN 978-1-4842-1330-8.

HOMRICH, É. P.; MERGEN, S. L. S. Comparação entre mysql e neo4j para o acesso a dados complexos usando linguagens declarativas. In: **Anais da XIV Escola Regional de Banco de Dados (ERBD 2018)**. Porto Alegre, RS, Brasil: SBC, 2018. ISSN 2595-413X. Available from Internet: <https://sol.sbc.org.br/index.php/erbd/article/view/2827>.

KOLONKO, K. **Performance comparison of the most popular relational and non-relational database management systems**. 66 p. Dissertation (Master) — , Department of Software Engineering, 2018.

LIMA, C. de; MELLO, R. dos S. A workload-driven logical design approach for nosql document databases. In: **Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services**. New York, NY, USA: ACM, 2015. (iiWAS '15), p. 73:1–73:10. ISBN 978-1-4503-3491-4. Available from Internet: <http://doi.acm.org/10.1145/2837185.2837218>.

Puangsaijai, W.; Puntheeranurak, S. A comparative study of relational database and key-value database for big data applications. In: **2017 International Electrical Engineering Congress (iEECON)**. [S.l.: s.n.], 2017. p. 1–4.

SANTOS, P. O.; MORO, M. M.; DAVIS, C. A. Comparative performance evaluation of relational and nosql databases for spatial and mobile applications. In: CHEN, Q. et al. (Ed.). **Database and Expert Systems Applications**. Cham: Springer International Publishing, 2015. p. 186–200. ISBN 978-3-319-22849-5.

SEUFITELLI, D. B.; MORO, M. M.; DAVIS JR., C. A. Desafios no mapeamento de esquemas conceituais geográficos para esquemas físicos híbridos sql/nosql. In: KÖRTING, T. S.; FILETO, R. (Ed.). **XVI Brazilian Symposium on GeoInformatics, Campos do Jordão, São Paulo, Brazil, November 29 - December 2, 2015.** MCTI/INPE, 2015. p. 119–124. Available from Internet: <http://urlib.net/8JMKD3MGPDW34P/3KP33M8>.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. **Database System Concepts**. 6. ed. [S.l.]: Mc Graw Hill, 2011. ISBN 978-0-07-352332-3.

SILVA, R. N. M. da; SPRITZER, A.; FREITAS, C. D. S. Visualization of roll call data for supporting analyses of political profiles. In: IEEE. **2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)**. [S.l.], 2018. p. 150–157.

VICKNAIR, C. et al. A comparison of a graph database and a relational database: A data provenance perspective. In: **Proceedings of the 48th Annual Southeast Regional Conference**. New York, NY, USA: ACM, 2010. (ACM SE '10), p. 42:1–42:6. ISBN 978-1-4503-0064-3. Available from Internet: <http://doi.acm.org/10.1145/1900008.1900067>.

APÊNDICE A - SCRIPTS SQL PARA CARGAS DE DADOS NO BANCO DE DADOS MYSQL

Figura 5.1: Script MySQL para carga de dados a partir de arquivos CSV dos Deputados

```
LOAD DATA LOCAL INFILE 'Arquivos/deputados.csv'
INTO TABLE Deputado
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5,@col6,@col7,@col8,
 @col9,@col10,@col11,@col12,@col13)

SET

idDeputado = (SELECT SUBSTRING_INDEX(@col1, '/', -1)),
nome = @col5,
siglaUF = @col12;
```

Fonte: autor

Figura 5.2: Script MySQL para carga de dados a partir de arquivos CSV das Frentes

```
LOAD DATA LOCAL INFILE 'Arquivos/frentes.csv'
INTO TABLE Frente
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5,@col6,@col7,@col8,
 @col9,@col10,@col11,@col12,@col13,@col14,@col15,@col16,
 @col17,@col18,@col19,@col20)

SET

idFrente = @col1,
titulo = @col3,
dataCriacao = @col4;
```

Fonte: autor

Figura 5.3: Script MySQL para carga de dados a partir de arquivos CSV das Legislaturas

```
LOAD DATA LOCAL INFILE 'Arquivos/legislaturas.csv'
INTO TABLE Legislatura
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5)

SET

idLegislatura = @col1,
dataInicio = @col3,
dataFim = @col4;
```

Fonte: autor

Figura 5.4: Script MySQL para carga de dados a partir de arquivos CSV dos Partidos

```
LOAD DATA LOCAL INFILE 'Arquivos/partidos.csv'
INTO TABLE Partido
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4)

SET

idPartido = @col1,
nome = @col2,
siglaPartido = @col3;
```

Fonte: autor

Figura 5.5: Script MySQL para carga de dados a partir de arquivos CSV das Proposições

```
LOAD DATA LOCAL INFILE 'Arquivos/proposicoes.csv'
INTO TABLE Proposicao
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5,@col6,@col7,@col8,
@col9,@col10,@col11,@col12,@col13,@col14,@col15,@col16,
@col17,@col18,@col19,@col20,@col21,@col22,@col23,@col24,
@col25,@col26,@col27,@col28,@col29,@col30)

SET

idProposicao = @col1,
siglaTipo = @col3,
numero = @col4,
ano = @col5,
ementa = @col8,
ementaDetalhada = @col9,
keywords = @col10,
dataApresentacao = @col11;
```

Fonte: autor

Figura 5.6: Script MySQL para carga de dados a partir de arquivos CSV das Votações

```
LOAD DATA LOCAL INFILE 'Arquivos/votacoes.csv'
INTO TABLE Votacao
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5,@col6)

SET

idVotacao = @col1,
codSessao = @col2,
resumo = @col3,
objVotacao = @col4,
data = @col5,
hora = @col6;
```

Fonte: autor

Figura 5.7: Script MySQL para carga de dados a partir de arquivos CSV do histórico de Partidos e seus Deputados

```
LOAD DATA LOCAL INFILE 'Arquivos/deputadosPartidos.csv'
INTO TABLE Deputado_has_Partido
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5,@col6,@col7)

SET

Deputado_idDeputado = @col1,
Partido_idPartido = (SELECT SUBSTRING_INDEX(@col7, '/', -1));
```

Fonte: autor

Figura 5.8: Script MySQL para carga de dados a partir de arquivos CSV do histórico de Frentes e seus Deputados

```
LOAD DATA LOCAL INFILE 'Arquivos/frentesDeputados.csv'
INTO TABLE Frente_has_Deputado
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5,@col6,@col7,@col8,
 @col9,@col10,@col11,@col12,@col13,@col14)

SET

Frente_idFrente = @col1,
Deputado_idDeputado = @col4;
```

Fonte: autor

Figura 5.9: Script MySQL para carga de dados a partir de arquivos CSV do histórico de Legislaturas e seus Deputados

```
LOAD DATA LOCAL INFILE 'Arquivos/deputados.csv'
INTO TABLE Legislatura_has_Deputado
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5,@col6,@col7,@col8,
 @col9,@col10,@col11,@col12,@col13)

SET

Deputado_idDeputado = (SELECT SUBSTRING_INDEX(@col1, '/', -1)),
Legislatura_idLegislaturaInicial = @col3,
Legislatura_idLegislaturaFinal = @col4;
```

Fonte: autor

Figura 5.10: Script MySQL para carga de dados a partir de arquivos CSV do histórico de Legislaturas e suas Frentes

```
LOAD DATA LOCAL INFILE 'Arquivos/frentes.csv'
INTO TABLE Legislatura_has_Frente
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5,@col6,@col7,@col8,
@col9,@col10,@col11,@col12,@col13,@col14,@col15,@col16,
@col17,@col18,@col19,@col20)

SET

Legislatura_idLegislatura = @col5,
Frente_idFrente = @col1;
```

Fonte: autor

Figura 5.11: Script MySQL para carga de dados a partir de arquivos CSV do histórico de Proposições e seus Deputados

```
LOAD DATA LOCAL INFILE 'Arquivos/proposicoes.csv'
INTO TABLE Proposicao_has_Deputado
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ';'
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5,@col6,@col7,@col8,
@col9,@col10,@col11,@col12,@col13,@col14,@col15,@col16,
@col17,@col18,@col19,@col20,@col21,@col22,@col23,@col24,
@col25,@col26,@col27,@col28,@col29,@col30)

SET

Proposicao_idProposicao = @col1,
Deputado_idDeputado = (SELECT SUBSTRING_INDEX(@col20, '/', -1));
```

Fonte: autor

Figura 5.12: Script MySQL para carga de dados a partir de arquivos CSV do histórico de Votações e seus Deputados

```
LOAD DATA LOCAL INFILE 'Arquivos/votacoesDeputados.csv'
INTO TABLE Votacao_has_Deputado
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4,@col5)

SET

Votacao_idVotacao = @col1,
Votacao_codSessao = @col2,
Votacao_hora = @col3,
Deputado_idDeputado = @col4,
voto = @col5;
```

Fonte: autor

Figura 5.13: Script MySQL para carga de dados a partir de arquivos CSV do histórico de Proposições votadas em Plenário

```
LOAD DATA LOCAL INFILE 'Arquivos/votacoesProposicoes.csv'
INTO TABLE Votacao_has_Proposicao
CHARACTER SET 'utf8'
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(@col1,@col2,@col3,@col4)

SET

Votacao_idVotacao = @col1,
Votacao_codSessao = @col2,
Votacao_hora = @col3,
Proposicao_idProposicao = @col4;
```

Fonte: autor

APÊNDICE B - SCRIPTS PARA CONSULTAS NO BANCO DE DADOS MYSQL

Consulta 1:

```
SELECT Proposicao_idProposicao
FROM Votacao_has_Proposicao;
```

Consulta 2:

```
SELECT idProposicao, siglaTipo, numero, ano, ementa,
       ementaDetalhada, keywords, dataApresentacao,
       Votacao_idVotacao, Votacao_codSessao, Votacao_hora
FROM Proposicao as P JOIN Votacao_has_Proposicao as VP
     ON P.idProposicao = VP.Proposicao_idProposicao;
```

Consulta 3:

```
SELECT idVotacao, Proposicao_idProposicao,
       Deputado_idDeputado, voto, codSessao, resumo,
       objVotacao, data, hora
FROM Votacao_has_Deputado as VD JOIN Votacao as V
     ON VD.Votacao_idVotacao = V.idVotacao and
     VD.Votacao_codSessao = V.codsessao and
     VD.Votacao_hora = V.hora
     JOIN Votacao_has_Proposicao as VP
     ON VP.Votacao_idVotacao = V.idVotacao and
     VP.Votacao_codSessao = V.codsessao and
     VP.Votacao_hora = V.hora;
```

Consulta 4:

```
SELECT *
FROM Deputado;
```

Consulta 5:

```
SELECT idDeputado, D.nome, sexo, siglaUF, idPartido,
       P.nome, siglaPartido
```

```
FROM Deputado as D JOIN Deputado_has_Partido as DP
     ON D.idDeputado = DP.Deputado_idDeputado
     JOIN Partido as P
     ON P.idPartido = DP.Partido_idPartido;
```

Consulta 6:

```
SELECT idDeputado, D.nome, sexo, siglaUF, idFrente,
       FD.titulo, F.titulo, dataCriacao
FROM Deputado as D JOIN Frente_has_Deputado as FD
     ON D.idDeputado = FD.Deputado_idDeputado
     JOIN Frente as F
     ON F.idFrente = FD.Frente_idFrente;
```

Consulta 7:

```
SELECT idDeputado, D.nome, sexo, siglaUF,
       Legislatura_idLegislaturaInicial,
       Legislatura_idLegislaturaFinal, dataInicio, dataFim
FROM Deputado as D JOIN Legislatura_has_Deputado as LD
     ON D.idDeputado = LD.Deputado_idDeputado
     JOIN Legislatura as L
     ON L.idLegislatura = LD.Legislatura_idLegislaturaFinal;
```

APÊNDICE C - SCRIPTS PARA CONSULTAS NO BANCO DE DADOS MONGO

Consulta 1:

```
db.votacoes.find({}, {'_id' : 1}).explain("executionStats")
```

Consulta 2:

```
time(() => db.votacoes.aggregate([
{
    $lookup:
    {
        from: 'proposicoes',
        localField: '_id',
        foreignField: '_id',
        as: 'DetalhesProposicao'
    }
},
{
    $project:
    {
        _id: 0, DetalhesProposicao: 1
    }
}
]) )
```

Consulta 3:

```
db.votacoes.find().explain("executionStats")
```

Consulta 4:

```
db.deputados.explain("executionStats").find()
```

Consulta 5:

```
db.deputadosPartidos.find().explain("executionStats")
```

Consulta 6:

```
db.frentesDeputados.find().explain("executionStats")
```

Consulta 7:

```
db.deputados.find({'idLegislaturaInicial' : 1, 'idLegislaturaFinal' : 1
```

APÊNDICE D - SCRIPT SQL PARA CRIAÇÃO DO BANCO DE DADOS MYSQL

```
-- 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 CivisAnalysis
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema CivisAnalysis
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `CivisAnalysis` ;
USE `CivisAnalysis` ;

-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Deputado`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Deputado` (
  `idDeputado` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(200) NULL,
  `sexo` VARCHAR(1) NULL,
  `siglaUF` VARCHAR(20) NULL,
  PRIMARY KEY (`idDeputado`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Legislatura`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Legislatura` (
  `idLegislatura` INT NOT NULL,
  `dataInicio` DATE NULL,
  `dataFim` DATE NULL,
  PRIMARY KEY (`idLegislatura`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Proposicao`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Proposicao` (
  `idProposicao` INT NOT NULL,
  `siglaTipo` VARCHAR(45) NULL,
  `numero` INT NULL,
  `ano` INT NULL,
  `ementa` MEDIUMTEXT NULL,
  `ementaDetalhada` MEDIUMTEXT NULL,
  `keywords` MEDIUMTEXT NULL,
  `dataApresentacao` DATETIME NULL,
  PRIMARY KEY (`idProposicao`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Votacao`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Votacao` (
  `idVotacao` INT NOT NULL AUTO_INCREMENT,
  `codSessao` INT NOT NULL,
```

```
  `resumo` VARCHAR(25000) NULL,
  `objVotacao` VARCHAR(25000) NULL,
  `data` DATE NULL,
  `hora` TIME NOT NULL,
  PRIMARY KEY (`idVotacao`, `codSessao`, `hora`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Partido`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Partido` (
  `idPartido` INT NOT NULL,
  `nome` VARCHAR(200) NULL,
  `siglaPartido` VARCHAR(200) NULL,
  PRIMARY KEY (`idPartido`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Legislatura_has_Deputado`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Legislatura_has_Deputado`
(
  `Deputado_idDeputado` INT NOT NULL,
  `Legislatura_idLegislaturaInicial` INT NOT NULL,
  `Legislatura_idLegislaturaFinal` INT NOT NULL,
  PRIMARY KEY (`Deputado_idDeputado`,
  `Legislatura_idLegislaturaInicial`,
  `Legislatura_idLegislaturaFinal`),
  INDEX `fk_Legislatura_has_Deputado_Deputado1_idx`
  (`Deputado_idDeputado` ASC),
  INDEX `fk_Legislatura_has_Deputado_Legislatura2_idx`
  (`Legislatura_idLegislaturaFinal` ASC),
```

```
  CONSTRAINT `fk_Legislatura_has_Deputado_Deputado1`
    FOREIGN KEY (`Deputado_idDeputado`)
    REFERENCES `CivisAnalysis`.`Deputado` (`idDeputado`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Legislatura_has_Deputado_Legislatura1`
    FOREIGN KEY (`Legislatura_idLegislaturaInicial`)
    REFERENCES `CivisAnalysis`.`Legislatura` (`idLegislatura`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Legislatura_has_Deputado_Legislatura2`
    FOREIGN KEY (`Legislatura_idLegislaturaFinal`)
    REFERENCES `CivisAnalysis`.`Legislatura` (`idLegislatura`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Frente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Frente` (
  `idFrente` INT NOT NULL,
  `titulo` VARCHAR(200) NULL,
  `dataCriacao` DATE NULL,
  PRIMARY KEY (`idFrente`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Frente_has_Deputado`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Frente_has_Deputado` (
  `Frente_idFrente` INT NOT NULL,
```

```
  `Deputado_idDeputado` INT NOT NULL,
  `Titulo` VARCHAR(45) NULL,
  PRIMARY KEY (`Frente_idFrente`, `Deputado_idDeputado`),
  INDEX `fk_Frente_has_Deputado_Deputado1_idx`
  (`Deputado_idDeputado` ASC),
  INDEX `fk_Frente_has_Deputado_Frente1_idx`
  (`Frente_idFrente` ASC),
  CONSTRAINT `fk_Frente_has_Deputado_Frente1`
    FOREIGN KEY (`Frente_idFrente`)
    REFERENCES `CivisAnalysis`.`Frente` (`idFrente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Frente_has_Deputado_Deputado1`
    FOREIGN KEY (`Deputado_idDeputado`)
    REFERENCES `CivisAnalysis`.`Deputado` (`idDeputado`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Legislatura_has_Frente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Legislatura_has_Frente` (
  `Legislatura_idLegislatura` INT NOT NULL,
  `Frente_idFrente` INT NOT NULL,
  PRIMARY KEY (`Legislatura_idLegislatura`, `Frente_idFrente`),
  INDEX `fk_Legislatura_has_Frente_Frente1_idx`
  (`Frente_idFrente` ASC),
  INDEX `fk_Legislatura_has_Frente_Legislatura1_idx`
  (`Legislatura_idLegislatura` ASC),
  CONSTRAINT `fk_Legislatura_has_Frente_Legislatura1`
    FOREIGN KEY (`Legislatura_idLegislatura`)
    REFERENCES `CivisAnalysis`.`Legislatura` (`idLegislatura`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Legislatura_has_Frente_Frente1`
    FOREIGN KEY (`Frente_idFrente`)
    REFERENCES `CivisAnalysis`.`Frente` (`idFrente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Proposicao_has_Deputado`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Proposicao_has_Deputado` (
  `Proposicao_idProposicao` INT NOT NULL,
  `Deputado_idDeputado` INT NOT NULL,
  PRIMARY KEY (`Proposicao_idProposicao`, `Deputado_idDeputado`),
  INDEX `fk_Proposicao_has_Deputado_Deputado1_idx`
  (`Deputado_idDeputado` ASC),
  INDEX `fk_Proposicao_has_Deputado_Proposicao1_idx`
  (`Proposicao_idProposicao` ASC),
  CONSTRAINT `fk_Proposicao_has_Deputado_Proposicao1`
    FOREIGN KEY (`Proposicao_idProposicao`)
    REFERENCES `CivisAnalysis`.`Proposicao` (`idProposicao`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Proposicao_has_Deputado_Deputado1`
    FOREIGN KEY (`Deputado_idDeputado`)
    REFERENCES `CivisAnalysis`.`Deputado` (`idDeputado`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Deputado_has_Partido`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Deputado_has_Partido` (
  `Deputado_idDeputado` INT NOT NULL,
  `Partido_idPartido` INT NOT NULL,
  PRIMARY KEY (`Deputado_idDeputado`, `Partido_idPartido`),
  INDEX `fk_Deputado_has_Partido_Partido1_idx`
  (`Partido_idPartido` ASC),
  INDEX `fk_Deputado_has_Partido_Deputado1_idx`
  (`Deputado_idDeputado` ASC),
  CONSTRAINT `fk_Deputado_has_Partido_Deputado1`
    FOREIGN KEY (`Deputado_idDeputado`)
    REFERENCES `CivisAnalysis`.`Deputado` (`idDeputado`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Deputado_has_Partido_Partido1`
    FOREIGN KEY (`Partido_idPartido`)
    REFERENCES `CivisAnalysis`.`Partido` (`idPartido`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Votacao_has_Deputado`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Votacao_has_Deputado` (
  `Votacao_idVotacao` INT NOT NULL,
  `Votacao_codSessao` INT NOT NULL,
  `Votacao_hora` TIME NOT NULL,
  `Deputado_idDeputado` INT NOT NULL,
  `voto` VARCHAR(20) NULL,
  PRIMARY KEY (`Votacao_idVotacao`, `Votacao_codSessao`,
```

```
  `Votacao_hora`, `Deputado_idDeputado`),
  INDEX `fk_Votacao_has_Deputado_Deputado1_idx`
  (`Deputado_idDeputado` ASC),
  INDEX `fk_Votacao_has_Deputado_Votacao1_idx`
  (`Votacao_idVotacao` ASC, `Votacao_codSessao` ASC,
  `Votacao_hora` ASC),
  CONSTRAINT `fk_Votacao_has_Deputado_Votacao1`
    FOREIGN KEY (`Votacao_idVotacao` , `Votacao_codSessao`,
    `Votacao_hora`)
    REFERENCES `CivisAnalysis`.`Votacao` (`idVotacao` ,
    `codSessao` , `hora`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Votacao_has_Deputado_Deputado1`
    FOREIGN KEY (`Deputado_idDeputado`)
    REFERENCES `CivisAnalysis`.`Deputado` (`idDeputado`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `CivisAnalysis`.`Votacao_has_Proposicao`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `CivisAnalysis`.`Votacao_has_Proposicao` (
  `Votacao_idVotacao` INT NOT NULL,
  `Votacao_codSessao` INT NOT NULL,
  `Votacao_hora` TIME NOT NULL,
  `Proposicao_idProposicao` INT NOT NULL,
  PRIMARY KEY (`Votacao_idVotacao`, `Votacao_codSessao`,
  `Votacao_hora`, `Proposicao_idProposicao`),
  INDEX `fk_Votacao_has_Proposicao_Proposicao1_idx`
  (`Proposicao_idProposicao` ASC),
  INDEX `fk_Votacao_has_Proposicao_Votacao1_idx`
```

```
  (`Votacao_idVotacao` ASC, `Votacao_codSessao` ASC,
  `Votacao_hora` ASC),
  CONSTRAINT `fk_Votacao_has_Proposicao_Votacao1`
    FOREIGN KEY (`Votacao_idVotacao` , `Votacao_codSessao` ,
    `Votacao_hora`)
    REFERENCES `CivisAnalysis`.`Votacao` (`idVotacao` , `codSessao` ,
    `hora`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Votacao_has_Proposicao_Proposicao1`
    FOREIGN KEY (`Proposicao_idProposicao`)
    REFERENCES `CivisAnalysis`.`Proposicao` (`idProposicao`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


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