Mapeamento Objeto-Relacional no Contexto de Desenvolvimento Web

Igor Donatti G. da Silva[1], **Matheus Anderson L. Gomes**[1], **Silvana M. Melo**[1]

[1]Faculdade de Ciências Exatas e Técnológicas (FACET)
Universidade Federal da Grande Dourados (UFGD)
Caixa Postal 364 – 79.804-97015.064 – Dourados – MS – Brasil

{igor.silva069,matheus.gomes498}@academico.ufgd.edu.br

silvanamelo@ufgd.edu.br

Resumo. *Com o avanço da tecnologia nos ambientes de desenvolvimento de software, nota-se um aumento da utilização de frameworks ORMs buscando realizar a abstração do banco de dados e facilitar o processo de desenvolvimento. Devido ao custo associado ao mapeamento objeto relacional, neste trabalho buscamos avaliar dois dos principais ORMs do mercado, considerando a sua popularidade de uso, o Hibernate e o Prisma. A análise é realizada por meio de uma avaliação de desempenho entre os dois ORMs, no contexto de um ambiente web cliente servidor, sob operações de CRUD no banco de dados. Os resultados mostram que o Prisma obteve uma performance superior em termos de análise de tempo de execução de consultas em relação ao Hibernate em um ambiente similar de desenvolvimento. Embora promissores, os resultados devem ser validados em novos estudos empíricos e sob diferentes contextos a fim de garantir a generalização dos resultados.*

Abstract. *With the advancement of technology in software development environments, there has been an increasing utilization of ORM frameworks to abstract the database and facilitate the development process. Due to the associated cost of object-relational mapping, this study aims to evaluate two of the leading market ORMs, considering their popularity of use: Hibernate and Prisma. The analysis is performed by comparing the performance of the two ORMs in the context of a client-server web environment, focusing on CRUD operations in the database. The results show that Prisma outperformed Hibernate in terms of query execution time analysis within a similar development environment. However, while promising, these findings should be validated in further empirical studies and across different contexts to ensure the generalization of the results.*

1. Introdução

No atual contexto globalizado, os sistemas web estão presentes no dia a dia da população. Portanto, surgem tecnologias cada dia mais avançadas. Neste contexto, a persistência de dados é uma tarefa essencial para grande parte das aplicações. Em geral, a persistência dos dados é garantida por meio de banco de dados relacionais, que armazenam os dados em tabelas e colunas. Porém, a manipulação dos dados não é uma tarefa simples, especialmente em aplicações orientadas a objetos, por possuírem estruturas mais complexas. Os ORMs (acrônimo para *Object-Relational Mapping*), surgem como proposta de

solução para lidar com este problema, fornecendo uma camada de abstração entre o banco de dados e código da aplicação.

Os ORMs têm se tornado muito populares nos últimos anos, com uma grande variedade de ferramentas existentes. Devido a grande variedade de opções de ferramentas torna-se importante uma maneira de avaliá-las a fim de selecionar a mais adequada para um determinado contexto.

Buscando identificar avaliações empíricas entre ORMs, nesse estudo foi conduzido um mapeamento sistemático da literatura. Como resultado foi possível observar que os estudos avaliam diversos ORMs, como: Hibernate, DJango ORM, Apache, Entity Framework e SQL queries. Com foco em sua maioria, no framework Hibernate. Entretanto, nenhum estudo apresentava avaliação do framework Prisma, mesmo ele tendo se mostrado uma ferramenta importante para o Typescript e com crescimento de uso na comunidade de desenvolvedores [1]. O mapeamento sistemático e seus resultados serviram como base para a definição desse trabalho, que tem como foco a avaliação experimental dos ORMs Hibernate e Prisma, a fim de comparar seu desempenho sob condições específicas de desenvolvimento no contexto web.

O restante deste trabalho está estruturado da seguinte forma: a Seção 2 apresenta uma fundamentação teórica dos conceitos envolvidos na pesquisa; a Seção 3 apresenta os principais trabalhos relacionados; a Seção 4 descreve a metodologia e as principais fases de condução da pesquisa; a Seção 5 trata do planejamento e condução da avaliação de desempenho proposta para os ORMs selecionados; e por fim, a Seção 6 traz algumas conclusões e propostas para trabalhos futuros.

2. Fundamentação teórica

Por muito tempo, desenvolvedores deveriam mapear e criar manualmente as tabelas relacionais de um banco de dados, confeccionando scripts SQL de maneira direta e escrita, essa criação acaba por muitas vezes ser confusa e complexa, visto que um sistema back-end é construído em meio orientação objeto. Com tal, o surgimento da impedância dos dados foi inevitável, e como forma de resolução foi criado as técnicas de mapeamento conhecidas como ORMs.

Object-Relational Mapping (ORM) é uma técnica confeccionada para lidar com a impedância de dados, onde trata-se do mapeamento de objetos de um sistema orientado a objetos para tabelas em um banco de dados relacional. Além de uma solução, tal técnica acaba por fornecer também um maior desempenho quando relacionada à construção de um back-end, visto que tal técnica leva a abstração das complexidades das operações de banco de dados.

Dentre funcionalidades atribuídas aos ORMs podemos citar, a sua interface de programação de alto nível, oferecendo operações de *CRUD (Create, Read, Update, Delete)* no banco de dados de forma abstrata e muitas vezes com uma maior simplicidade em operações complexas. Outra funcionalidade é seu auto-mapeamento das classes e objetos da aplicação para tabelas e colunas correspondentes no banco de dados, assim excluindo a necessidade de criação de scripts SQL para tais funcionalidades e do mapeamento de

[1] `https://www.prisma.io/dataguide/database-tools/top-nodejs-orms-query-builders-and-database-libraries`

objetos para tabelas, auxiliando o acesso e a manipulação de dados.

Logo, suas utilizações se tornam inerentes ao desenvolvimento de aplicações web, trazendo inúmeras vantagens, como por exemplo, o maior desempenho, manipulação de dados facilitada e a diminuição de erros. Contudo, não somente vantagens são trazidas à tona por sua utilização, existindo algumas desvantagens sucintas, uma de suas desvantagens se vem por conta da possibilidade de menor eficiência perante scripts SQL manuais, sendo ocasionados especialmente em projetos grandes e complexos.

A Tabela 1 apresenta alguns dos principais ORMs para Node.js e Java. A partir de uma análise preliminar da literatura [Torres et al. 2017, Prisma 2022] é possível listar alguns dos principais ORMs para Node.js e Java, considerando características como: (i) suporte a bancos de dados, sendo esta, a capacidade do ORM de se conectar e interagir com diferentes bancos de dados; (ii) suporte a cache de segundo nível, sendo esta, a capacidade de armazenar objetos em cache para melhorar o desempenho; (iii) facilidade de uso, sendo esta, a facilidade de configuração e utilização do ORM; (iv) documentação e comunidade ativa, sendo esta, a disponibilidade de documentação e uma comunidade ativa; (v) performance, sendo esta, uma característica importante para garantir a qualidade do sistema. Esses ORMs figuram como os mais utilizados no mercado, incluindo desde os mais antigos aos mais novos com características promissoras. É importante destacar o Prisma como um importante ORM para essas linguagens, com notável vantagem em todas as categorias apresentadas, sendo portanto o objeto de estudo desse trabalho.

Tabela 1. Principais ORMs para desenvolvimento web

ORMS (Object-relational mapping)	Linguagem de programação	Versão	Suporte avários bancos de dados	Suporte a cache de segundo nível	Facilidade de uso	Documentação e comunidade ativa
JPA/Hibernate	Java	6.2	Amplo	Sim	Difícil	Excelente
MyBatis	Java	3.5.11	Amplo	Sim	Difícil	Razoável
Sequelize	Javascript	6.28	Amplo	Não	Fácil	Boa
Prisma	Typescript	4.9.4	MySQL, PostgreSQL, SQLite, SQLServer	Não	Fácil	Excelente
TypeORM	Javascript	0.3.11	Amplo	Sim	Fácil	Boa

3. Trabalhos relacionados

A fim de identificar estudos que abordassem avaliações empíricas de ORMs, foi realizado um mapeamento sistemático da literatura. Nesse estudo foram considerados três bases de dados diferentes, sendo elas: IEEExplore, ACM e Scopus. Para construção da string de busca foram utilizadas as seguintes palavras-chaves: experimento, e suas variantes: estudo empírico, estudo controlado, estudo de caso, avaliação, avalação empírica e comparação, além das palavras ORM e mapeamento objeto relacional. Com as strings de busca formadas, as pesquisas nas bases de dados retornaram 221 estudos, dos quais, após a aplicação dos critérios de inclusão e exclusão, resultaram em 12 artigos incluídos,

por possuírem como objetivo uma avaliação empírica de um ou mais ORMs. Os estudos incluídos ao final do mapeamento, e que portanto compreendem os trabalhos relacionados a essa pesquisa são apresentados na Tabela 2.

Tabela 2. Artigos incluidos

Id	Titulo	Ano
1	Energy Efficiency of ORM Approaches: an Empirical Evaluation	2016
2	The influence of optimisations on the performance of an object relational mapping tool	2009
3	An Evaluation of the Hibernate Object-Relational Mapping for Processing Interactive Social Networking Actions	2014
4	Comparing the performance of object databases and ORM tools	2006
5	Enhanced segment trees in object-relational mapping	2013
6	Performance Evaluation of Transparent Persistence Layer in Java Applications	2010
7	Performance Analysis of .NET Based Object–Relational Mapping Frameworks	2014
8	Object Oriented Application Cooperation Methods with Relational Database (ORM) based on J2EE Technology	2007
9	Understanding the Energy, Performance, and Programming Effort Trade-offs of Android Persistence Frameworks	2016
10	Investigating the Effects of Object-Relational Impedance Mismatch on the Efficiency of Object-Relational Mapping Frameworks	2020
11	Performance Comparison of CRUD Methods usingNET Object Relational Mappers: A Case Study	2020
12	A Comparative Study of the Features and Performance of ORM Tools in a .NET Environment	2010

Os trabalhos incluídos no mapeamento sistemático compreendem estudos que avaliam desepenho de ORMs sob diferentes perspectivas, como tempo, impacto energético, uso de mémoria, entre outros. [Giuseppe et al. 2016] avaliam o impacto energético dos frameworks de Mapeamento Objeto-Relacional.

Foram encontrados diversos estudos que avaliam o Hibernate como framework principal. [Pieter et al. 2009] investigam a influência das técnicas de otimização e das técnicas recomendadas pelo fornecedor no Hibernate. Já [Pieter et al. 2014] investigam o tempo de *overhead* ao utilizar o Hibernate. Ainda no contexto do Hibernate, [Michał et al. 2013] propõem soluções para atender as necessidades de aplicativos mais complexos e desafiadoras, utilizando ORM para materializações adequadas no banco de dados e avaliando seu desempenho em tais tarefas.

Também foram encontrados estudos que exerciam comparações entre ORMs distintos. Em [Pieter et al. 2006] é relatado um estudo sobre os aspectos de desempenho do Hibernate e DB40, utilizando um benchmark para comparar Sistemas de Gerenciamento de Banco de Dados Orientados a Objetos. No estudo de [Zhiyu and Zhiang 2010] o OJB e o Hibernate são comparados juntamente com a integração de ambos com o Spring nos testes de desempenho. No estudo de [Aleksandra and Przemyslaw 2014] o objetivo da pesquisa é analisar o desemepnho e comparar o Hibernate, JPOX e JPA em relação ao seu desempenho e velocidade de aprendizado. Em [Doina et al. 2020] é realizada uma

comparação de desempenho entre Entity Framework Core, Hibernate e Dapper, o objetivo principal do artigo é realizar uma análise comparativa do impacto que um ORM tem no desempenho da aplicação. Por fim [Stevica and Dragan 2010], apresentam uma análise comparativa do Hibernate e do Entity Framework, vários mecanismos de consulta foram descritos e testados em comparação com a abordagem convencional de consulta SQL como referência.

Também foram encontrados estudos com outros ORMs. [Jing et al. 2016] relatam um estudo de seis frameworks de persistênca para Android, são eles: ActiveAndroid, greenDAO, OrmLite, Sugar ORM, Android SQLite e Java Realm, avaliando o desempenho utilizando benchmarks populares. Já [Derek et al. 2020], apresentam uma pesquisa com profissionais de banco de dados sobre a eficácia das ferramentas ORM.

Apesar da variedade de estudos encontrados na literatura, a maioria dos estudos visa uma pesquisa ou comparação do framework Hibernate, porém não foi encontrado nenhum estudo que tivesse como objetivo avaliação do ORM Prisma, que se mostra muito popular no mercado, sendo utilizado principalmente no contexto de desenvolvimento web. O que motivou a condução dessa pesquisa a fim de oferecer a comunidade uma base que possa ser utilizada como comparação e fundamentação para estudos futuros, sobre esse ORM.

4. Materiais e Métodos

Nessa seção descrevemos as principais etapas da pesquisa, caracterizando a metodologia utilizada para sua condução. Em seguida são apresentadas as etapas de planejamento e condução da avaliação de desempenho proposta.

4.1. Metodologia

A pesquisa em questão seguiu uma metodologia de condução de estudos empíricos [Wohlin et al. 2012] como um estudo experimental para avaliação de desempenho, considerando os ORMs Prisma e Hibernate, focando na comparação entre esses dois frameworks amplamente utilizados no contexto de interação com bancos de dados. A metodologia adotada reflete os princípios fundamentais de avaliação de desempenho, incorporando elementos do campo de Planejamento de Experimentos e outras práticas recomendadas. A Figura 1 ilustra as etapas de condução da pesquisa, conforme descrito abaixo.

Figura 1. Metodologia de Condução da Pesquisa

O primeiro passo foi estabelecer os objetivos da pesquisa, que envolviam a avaliação do desempenho do Prisma em relação ao Hibernate. Isso incluiu determinar as principais métricas de desempenho a serem analisadas, como tempo de resposta das consultas e taxa de transferência.

Para criar cenários de teste comparáveis e realistas, optamos por utilizar um benchmark disponibilizado pelo MySQL. Esse benchmark simula cargas de trabalho comuns

em bancos de dados, permitindo uma avaliação de ambos os frameworks sob as mesmas condições. O benchmark pode ser acessado através do link: `https://dev.mysql.com/doc/employee/en/employees-introduction.html`.

Com base nas informações fornecidas na discussão anterior, foram definidas métricas específicas para avaliar o desempenho dos ORMs. Essas métricas foram escolhidas para capturar aspectos cruciais do desempenho, como tempo de execução de consultas e eficiência no uso de recursos. As métricas definidas são apresentadas na Tabela 3 juntamente com as suas descrições.

A partir do benchmark do MySQL e das métricas definidas, foram criado cenários de teste que refletiam situações reais de uso dos ORMs. Isso incluiu a definição de consultas e operações típicas que os frameworks enfrentariam em uma aplicação. Também foram feitas as funções responsáveis pelas requisições HTTP que posteriormete foram utilizadas para a coleta de dados.

Os cenários de teste foram executados tanto com o Prisma quanto com o Hibernate, e os dados relevantes foram coletados. Isso incluiu a medição das métricas definidas e a captura de informações sobre o consumo de recursos do sistema durante os testes.

A coleta de dados permitiu a análise estatística dos resultados. Com base nas técnicas de Planejamento de Experimentos, foram comparadas as métricas de desempenho dos dois frameworks. A análise envolveu uma interpretação estatística dos resultados obtidos durante a coleta de dados.

Os principais resultados são discutidos em torno dos objetivos e questões de pesquisa definidos. E por fim, conclusões a cerca da pesquisa, sua relevância para a área e possibilidades de trabalhos futuros são levantadas.

4.2. Planejamento do Experimento

O estudo foi planejado como um experimento controlado [Wohlin et al. 2012], a fim de avaliar diferentes ORMs disponíveis no mercado. Para essa avaliação foram comparados os ORMs Prisma e Hibernate por meio da execução de consultas em bases de dados. A comparação foi conduzida por meio da análise da eficiência considerando o tempo de execução das consultas em milisegundos.

4.2.1. Definição do experimento

De acordo com a abordagem GQM (*Goal, Question, Metrics*) [Basili et al. 1994], a definição do objetivo do experimento relatado neste artigo pode ser resumida da seguinte forma:

"***Analisar*** *diferentes ORMs* ***com o propósito de*** *avaliar sua eficiência em relação ao tempo de execução de consultas* ***sob a perspectiva*** *do engenheiro de software,* ***no contexto*** *da gestão de bancos de dados.*"

4.2.2. Seleção das variáveis

A variável dependente do estudo é medida em unidades de tempo, considerando quantos milessegundos cada consulta demora pra ser executada em cada um dos frameworks. Quanto as variáveis independentes (ou seja, que podem ser manipuladas e controladas), além da variável relacionada às estruturas ORM, foram identificados dois outros potenciais fatores, a saber, o tipo de consulta realizada ao banco de dados e o tamanho da tabela sobre a qual a consulta é feita. Dessa maneira, optamos por controlar essas variáveis, considerando-as como elementos que podem influenciar os resultados. Isso levou à seleção dos seguintes fatores e abordagens experimentais:

- **Framework ORM**: foram analisados duas ferramentas ORMs diferentes: Prisma e Hibernate.
- **Tipo de Consulta:** as quatro funções básicas de armazenamento persistente foram utilizadas para gerar as medições: *Create, Read, Update and Delete*.
- **Tamanho da Tabela**:
  - **Tabela Departamento**: 9 registros.
  - **Tabela EmpregadosDepartamento**: 11050 registros.
  - **Tabela Empregados**: 10000 registros.

4.2.3. Design

O design do experimento descreve como os testes são organizados e executados. O experimento foi definido como um fator e dois tratamentos. Nesse caso, o mesmo grupo de consultas foram executadas comparando o tempo para os dois frameworks Hibernate e Prisma. As consultas foram distribuídas e balanceadas em dois grupos implementados para cada um dos ORMs.

4.2.4. Seleção da amostra

A amostra de consultas utilizadas foram extraídas da base relacional disponível na página do MySQL [MySQL 2023]. A amostra foi selecionada utilizando o método de seleção por conveniência, considerando uma base de dados amplamente utilizada pela comunidade e com características que poderiam ser replicadas no estudo. Uma seleção aleatório de consultas foi definida para ser aplicadas a essa base de dados.

4.2.5. Instrumentação

O experimento foi conduzido de Julho a Agosto de 2023, utilizando uma máquina virtual com as seguintes configurações, processador AMD Ryzen 5 3600X utilizando-se de 6 cores, 10GB de memória RAM, sistema operacional Windows 11 Pro.

Para a realização da avaliação de desempenho, o MySQL foi o SGBD escolhido para a realização do modelo relacional utilizado no presente trabalho. O mesmo utiliza linguaguem SQL que trata-se de um banco de dados muito difundido na comunidade, pela sua facilidade de uso e confiabilidade.

Como primeiro objeto de estudo e base de testes, foi utilizado o Hibernate, um framework ORM, que faz a abstração da camada de dados. Utilizando Java, o ORM transforma classes para tabela de dados. O Hibernate gera as chamadas SQL e libera o desenvolvedor do trabalho manual da conversão dos dados resultante, mantendo o programa portável para quaisquer bancos de dados SQL. Já para a realização do banco de dados utilizando TypeScript, foi utilizado o Prisma ORM, que apesar de seu pouco tempo de mercado, atinge uma grande quantidade de desenvolvedores que fazem de sua utilização, dada sua facilidade de uso e implementação.

Os tempos de execução foram obtidos por meio de um procedimento manual que empregou a ferramenta Postman. Nesse procedimento, foram criadas requisições HTTP específicas para avaliar o desempenho dos ORMs Prisma e Hibernate. Essas requisições foram organizadas e armazenadas em uma *"Collection"*. A fim de automatizar a execução das requisições, utilizou-se a funcionalidade *"Runner"* do Postman.

Por meio do "Runner", foi possível estabelecer uma sequência de execução para as requisições HTTP. Nesse processo, selecionaram-se as requisições relevantes e configurou-se o número de iterações desejado para a execução. Isso permitiu a realização de uma série de execuções repetidas e consistentes para cada ORM, proporcionando dados robustos para análise comparativa de desempenho.

Assim, ao empregar essa abordagem, foi possível coletar informações precisas e comparáveis sobre os tempos de execução dos ORMs Prisma e Hibernate, agregando credibilidade e confiabilidade aos resultados obtidos.

4.2.6. Desrição da Aplicação

Como **framework de modelagem e persistência**, objetivando uma simulação que represente um pouco dos paradigmas relacionados ao desenvolvimento web e programação orientada a objetos, foi escolhido um benchmark SQL disponibilizado pelo prórpio MySQL. Trata-se de um banco de dados que representa uma empresa com cadastros de funcionários. A Figura 2 representa o modelo de classe mencionado.

4.2.7. Hibernate

No Hibernate, as anotações do JPA (Java Persistence API) estão relacionadas ao paradigma de Programação Orientada a Objetos. O código fonte apresentado na Figura 3, exemplifica essa relação. No paradigma de Programação Orientada a Objetos, as classes são usadas para representar entidades ou objetos do mundo real. A anotação @Entity é aplicada à classe DepartmentsModel, indicando que ela é uma entidade persistente que será mapeada para uma tabela no banco de dados. Dessa forma, o conceito de classe como representação de uma entidade é mantido. Os atributos da classe DepartmentsModel são definidos como campos privados, seguindo o conceito de encapsulamento. Os métodos getters e setters são utilizados para acessar e modificar esses atributos, promovendo o acesso controlado às propriedades da classe. A anotação @Id é usada para definir a chave primária da entidade, mantendo o conceito de identidade de objetos. Cada objeto DepartmentsModel é identificado de forma única pela chave primária. A anotação @Column é

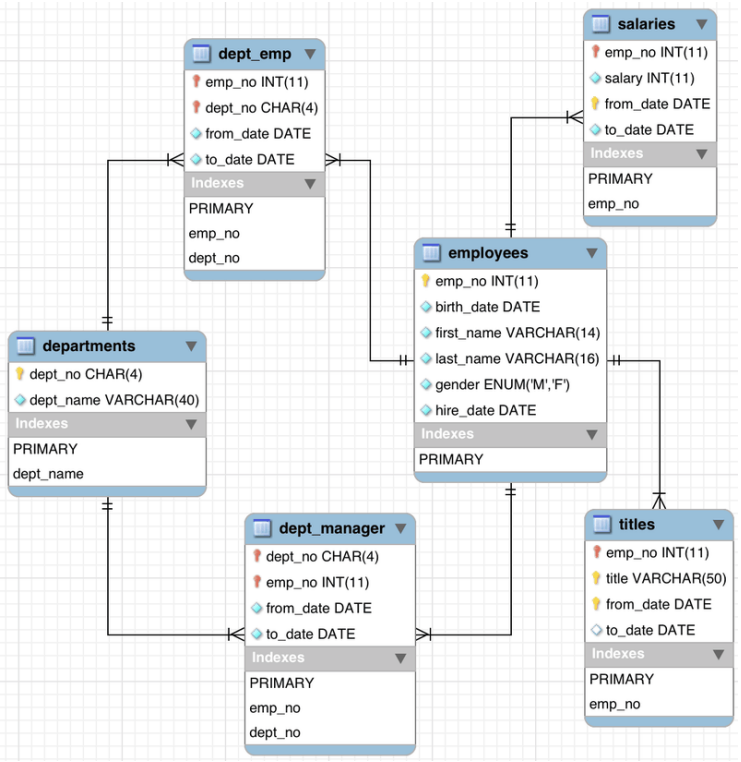

Figura 2. Diagrama Entidade Relacionamento do Benchmark [MySQL 2023]

utilizada para configurar as colunas do banco de dados correspondentes aos atributos da classe. Essa configuração inclui detalhes como tamanho, nulidade e unicidade das colunas. Isso reflete o conceito de mapeamento objeto-relacional, em que os atributos da classe são mapeados para as colunas do banco de dados. Assim, o uso das anotações do JPA no Hibernate, no contexto da Programação Orientada a Objetos, permite mapear objetos para o banco de dados, seguindo os conceitos fundamentais desse paradigma, como encapsulamento, identidade de objetos e mapeamento objeto-relacional.

```java
package com.example.tcci.models;
import jakarta.persistence.*;
import java.io.Serializable;

@Entity
@Table(name="departments")
public class DepartmentsModel implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @Column(length = 4,name = "deptNo")
    private String deptNo;
    @Column(nullable = false,length = 40, unique = true, name = "deptName")
    private String deptName;

    @Override
    public String toString() {
        return "DepartmentsModel{" +
                "deptNo='" + deptNo + '\'' +
                ", deptName='" + deptName + '\'' +
                '}';
    }
```

Figura 3. Classe ‘DepartmentsModel‘ em Java

4.2.8. Prisma

O código fonte apresentado na Figura 4, é uma definição de modelo usando o Prisma, um ORM para acesso a banco de dados. Ele define três modelos: Employee, Department e DeptEmp, que correspondem às tabelas employees, departments e deptEmp em um banco de dados MySQL. O bloco generator especifica o uso do prisma-client-js como provedor de código. O bloco datasource define a conexão com o banco de dados MySQL, especificando a URL de conexão. Os modelos são definidos usando a palavra-chave model. Cada modelo possui campos que correspondem às colunas da tabela no banco de dados. As anotações @id, @unique e @db.VarChar são usadas para definir as propriedades das colunas. As relações entre os modelos são definidas usando a anotação @relation, que especifica os campos relacionados e as referências nas tabelas relacionadas. A opção onDelete: Cascade indica que, ao excluir registros em uma tabela relacionada, as correspondências relacionadas na tabela deptEmp também serão excluídas. Em resumo, o código define os modelos Employee, Department e DeptEmp usando o Prisma para mapear as tabelas employees, departments e deptEmp em um banco de dados MySQL.

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = "mysql://root:admin@localhost:3306/prisma"
}

// Define the employees table
model Employee {
  emp_no     Int       @id
  birth_date DateTime
  first_name String @db.VarChar(14)
  last_name  String @db.VarChar(16)
  gender     String
  hire_date  DateTime
  DeptEmp    DeptEmp[]
}

// Define the departments table
model Department {
  dept_no   String      @id
  dept_name String      @unique @db.VarChar(40)
  DeptEmp   DeptEmp[]
}

// Define the dept_emp table
model DeptEmp {
  emp_no     Int
  dept_no    String
  from_date  DateTime
  to_date    DateTime
  employee   Employee   @relation(fields: [emp_no], references: [emp_no], onDelete:
      Cascade)
  department Department @relation(fields: [dept_no], references: [dept_no], onDelete:
      Cascade)

  @@id([emp_no, dept_no])
}
```

Figura 4. Arquivo 'schema.prisma'

4.2.9. Modelo de requisições web

Nos testes realizados para as requisições HTTP, foi utilizada Postman para simular as requisições e verificar as respostas dos *endpoints* da aplicação. Tanto no Hibernate quanto no Prisma ORM, o processo de realização dos testes seguiu uma abordagem semelhante.

O Postman é uma plataforma de colaboração que permite aos desenvolvedores criar, testar, documentar e compartilhar facilmente APIs. É uma ferramenta popular usada para enviar e receber solicitações HTTP e visualizar as respostas do servidor.

No caso do Hibernate, a aplicação foi desenvolvida utilizando o framework Java e as funcionalidades do Hibernate para mapeamento objeto-relacional. Após a implementação dos endpoints e lógica de negócio, o Postman foi utilizado para enviar requisições HTTP aos endpoints definidos. Com o Postman, foi possível especificar o método da requisição (como *GET*, *POST*, *PUT*, *DELETE*), os parâmetros, cabeçalhos e corpo da requisição, conforme necessário para cada teste. A resposta do servidor, contendo os dados retornados ou as mensagens de sucesso ou erro, foi verificada diretamente no Postman.

No caso do Prisma ORM, a aplicação foi desenvolvida utilizando JavaScript ou TypeScript, juntamente com o Prisma como ORM para acesso ao banco de dados. Da mesma forma, após a implementação dos endpoints e da lógica de negócio, o Postman foi utilizado para enviar as requisições HTTP aos endpoints definidos. Com o Postman, foi possível enviar as requisições com os métodos apropriados e os parâmetros necessários para cada teste. A resposta do servidor, que continha os dados retornados, mensagens de sucesso ou erro e também o tempo de execução, foi verificada diretamente no Postman.

5. Execução do Experimento

O primeiro passo para desenvolver os testes, foi definir a métrica de comparação. Para este trabalho, a métrica de tempo de execução das consultas foi escolhida, dado as necessidadas do trabalho, além do ambiete de testes previamente citada no trabalho. As métricas definidas, as questões de pesquisa a serem respondidas e o identificador de cada uma é apresentado na Tabela 3. Seguindo com o processo, foram definidas as métricas que serviriam de base para execução em cada um dos testes. As métricas foram focadas em executar as principais consultas existentes no banco de dados, sendo elas: *INSERT*, *SELECT*, *DELETE* e *UPDATE*.

5.1. Resultados

Com as métricas definidas, foram executadas as seções de teste, realizando as consultas nos diferentes frakeworks, em SQLQueries, Hybernate e por fim no Prisma. Os resultados com tempo de execução em cada uma das métricas em milissegundos são apresentados na Tabela 4.

Tabela 3. Questões de pesquisa e métricas definidads para o estudo

Métrica	Questão de pesquisa	Descrição
M1	Qual desempenho do ORM em consultas do tipo inserção de dados?	Tempo para executar inserção de 10.000 dados do tipo employee na base de dados.
M2	Qual desempenho do ORM em consultas do tipo seleção para apenas um dado?	Tempo para selecionar na tabela Employees um indivíduo com emp.no = 17869
M3	Qual desempenho do ORM em consultas do tipo seleção um conjunto de dados?	Tempo para selecionar na tabela Employees indivíduos com emp.no $>$ 12000
M4	Qual desempenho do ORM em consultas do tipo seleção um conjunto de dados com 'join table'?	Tempo para selecionar na tabela Employees indivíduos que trabalham no dep.no = "d002"
M5	Qual desempenho do ORM em consultas do tipo seleção em um conjunto de dados com 'join table' e cláusula 'where'?	Tempo para selecionar na tabela Employees indivíduos que trabalham no dep.no = "d002"e tenham emp.no $>$ 15000"
M6	Qual desempenho do ORM em consultas do tipo seleção um conjunto de dados através do nome	Tempo para selecionar na tabela Employees indivíduos cujo first.name contenha "da"
M7	Qual desempenho do ORM em consultas do tipo atualização em um conjunto de dados	Tempo para alterar fist.name dos Employees cujo emp.no $>$ 15000
M8	Qual desempenho do ORM em consultas do tipo deleção em um conjunto de dados	Tempo necessário para deletar da tabela Employees os indivíduos cujo first.name = "change.name".

Tabela 4. Média dos resultados para considerando as métricas definidas

Operação	Métrica	Tempo de resposta (ms)		
		Mysql	Hibernate	Prisma
CREATE	M1	120,4	139,84	135,90
READ	M2	0,62	2,00	1,63
	M3	2,58	34,15	16,10
	M4	7,16	6,70	9,2
	M5	8,85	5,51	8,00
	M6	17,88	8,66	10,50
UPDATE	M7	156,20	141,93	154,30
DELETE	M8	76,50	49,48	87,80

Foram feitos também testes visando avaliar o desempenho dos frameworks citados no contexto de desenvolvimento web. Com todo o ambiente configurado, foram feitas as requisições HTTP via Postman. O primeiro teste foi feito utilizando as mesmas métricas descritas antes, com exceção das métricas de Insert. O segundo teste foi feito visando obter resultados em relação a uma maior quantidade de dados. Para este teste foram realizados 1000 operações para cada uma das métricas definidas. Para estes testes, os

resultados são apresentados na Tabela 5.

Tabela 5. Média dos resultados para considerando o número de operações

Operação	Métrica	Nro. de Operações	Tempo de resposta (ms)	
			Hibernate	Prisma
READ	M2	10	9	7.8
		1000	5	4
	M3	10	2853.8	433.2
		1000	3385	203
	M4	10	229.1	32.8
		1000	228	26
	M5	10	124.6	18.4
		1000	121	17
	M6	10	133.9	25.9
		1000	123	24
UPDATE	M7	10	3180	149.6
		1000	3559	158
DELETE	M8	10	3029	89.6
		1000	3377	90

Através da análise dos resultados obtidos, fica claro que o Prisma ORM surge como uma poderosa ferramenta para operações de consultas a banco de dados através de requisições HTTP. De acordo com os dados coletados e o gráfico apresentado, o Prisma ORM demonstrou um desempenho impressionante, podendo ser até quatro vezes mais rápido do que o Hibernate em determinadas operações, como é possível analisar na Figura 5.

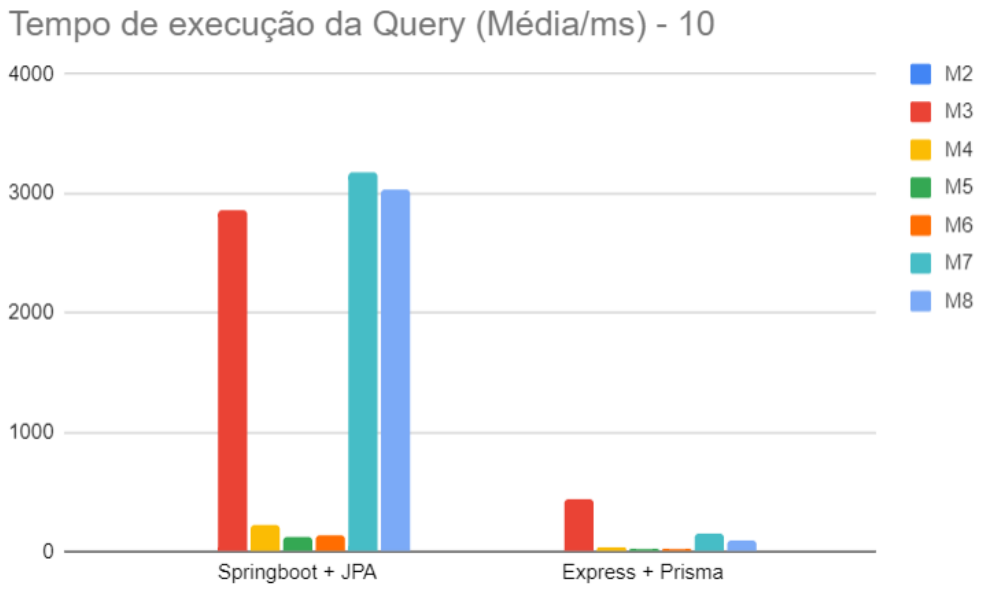

Figura 5. Hibernate x Prisma em milissegundos para as 10 operações de CRUD

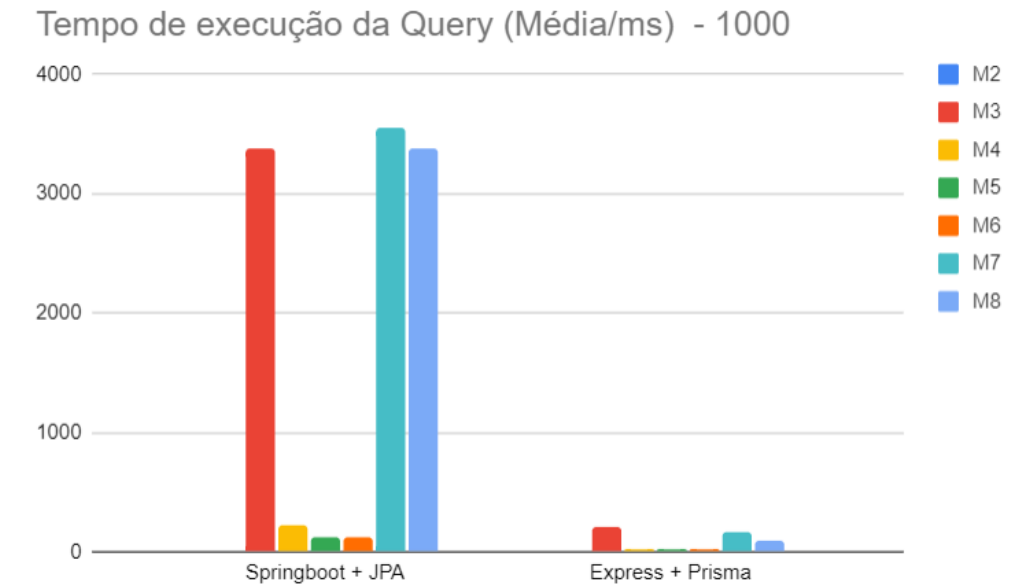

Figura 6. Hibernate x Prisma em milissegundos para as 1000 operações de CRUD

Com base nas representações gráficas evidenciadas na Figura 5 e Figura 6, é possível observar uma discrepância significativa nos intervalos de tempo de execução entre as soluções Hibernate e Prisma. Essa disparidade permanece notável mesmo quando consideramos contextos com um volume substancial de testes realizados.

Ao analisar os indicadores previamente apresentados, é possível discernir padrões notáveis. No caso de M2, não se verificou uma distinção significativa, com seu período de execução sendo tão exíguo que sequer se tornou visível nos gráficos. Contrastando com isso, em M3, destaca-se uma discrepância substancial nos tempos de execução, apresentando um salto considerável, aproximando-se de uma diferença média de 11 vezes entre os sistemas Prisma e Hibernate.

Nos cenários M4, M5 e M6, verifica-se uma redução gradual na discrepância dos tempos de execução, aproximando-se de uma média de 6.72 vezes entre os Sistemas Gerenciadores de Mapeamento de Objetos (ORMs). Entretanto, M7 e M8 assinalam um retorno a disparidades notáveis, evidenciando uma diferença média de 28.85 vezes nos tempos de execução entre os mencionados ORMs.

Esses resultados elucidam a complexa interação entre os ORMs, destacando variações notáveis nos tempos de execução com base em diferentes cenários e métricas, o que aponta para considerações cruciais ao selecionar uma solução apropriada para contextos específicos.

Essa diferença significativa de velocidade pode exercer um impacto direto e altamente positivo na aplicação como um todo. Ao utilizar o Prisma ORM, a aplicação ganha em eficiência e agilidade no acesso ao banco de dados, o que se traduz em uma melhor experiência para os usuários e um aumento notável no desempenho geral da aplicação.

A rapidez no processamento de consultas a bancos de dados é de suma importância em muitos contextos, especialmente em sistemas que lidam com grande volume de dados

e necessitam de respostas rápidas para atender às demandas dos usuários. Nesse sentido, o Prisma ORM se mostra como uma escolha promissora para otimizar o desempenho da aplicação, reduzindo o tempo de resposta e proporcionando uma experiência mais fluida e satisfatória aos usuários finais.

6. Conclusões

O trabalho abordou o problema de impedância de dados existentes na atualidade, no contexto de desenvolvimento web e como as os frameworks ORMs propõem uma solução para este problema através de uma camada de abstração entre o código e o banco de dados.

Um experimento controlado foi planejado e conduzido como uma avaliação de desempenho entre dois ORMs populares no mercado, o Hibernate e o Prisma, a eficiência dos ORMs foi avaliada por meio da realização de requisições HTTP em um benchmark diponibilizado pelo MySQL. Com base neste benchmark foi construída uma aplicação em ambos os ORMs e realizados os testes e coletados os tempos de execução para avaliação das questões de pesquisa.

A análise dos resultados leva a crer que o ORM Prisma se mostra um grande concorrente ao Hibernate na atualidade, dado seu ótimo desempenho na comparação do tempo de execução de consultas. Os resultados mostraram uma performance até 4 vezes maior em operações de update, delete e select para o Prisma em contrapartida ao Hibernate, considerando o benchmark utilizado, sob as condições especificadas.

Este trabalho visa contribuir para a comunidade de desenvolvimento web e banco de dados demonstrando o ganho de desempenho real com a utilização de frameworks ORMs na construção de aplicações. Os resultados dessa pesquisa podem servir como guia para pesquisadores e desenvolvedores que visam estudar e analisar as ferramentas ORMs disponíveis no mercado. Todo o material utilizado neste trabalho está disponível online no repositório de código livre em: `https://github.com/Matheus-nb/ORM-Efficiency-Comparison`, para acesso ou uso em novos estudos ou comparações com estudos existentes, contribuindo na área de pesquisa com resultados que podem ser comparados e replicados em estudos futuros.

Referências

[Aleksandra and Przemyslaw 2014] Aleksandra, G. and Przemyslaw, P. (2014). Performance analysis of .net based object-relational mapping frameworks.

[Basili et al. 1994] Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley.

[Derek et al. 2020] Derek, C., Clare, S., and Md, A. (2020). Investigating the effects of object-relational impedance mismatch on the efficiency of object-relational mapping frameworks.

[Doina et al. 2020] Doina, Z., Lucian-Laurentiu, P.-F., Cornelia, G., and Robert, G. (2020). Performance comparison of crud methods usingnet object relational mappers: A case study.

[Giuseppe et al. 2016] Giuseppe, P., Patricia, L., and Wouter, D. (2016). Energy efficiency of orm approaches: an empirical evaluation.

[Jing et al. 2016] Jing, P. Z., “Jason”, S., and Eli, T. (2016). Understanding the energy, performance, and programming effort trade-offs of android persistence frameworks.

[Michał et al. 2013] Michał, G., Piotr, W., and Krzysztof, S. (2013). Enhanced segment trees in object-relational mapping.

[MySQL 2023] MySQL (2023). Employees sample database. `https://dev.mysql.com/doc/employee/en/employees-introduction.html`.

[Pieter et al. 2006] Pieter, v. Z., Derrick, K., and Andrew, B. (2006). Comparing the performance of object databases and orm tools.

[Pieter et al. 2009] Pieter, v. Z., Derrick, K., and Louis, C. (2009). The influence of optimisations on the performance of an object relational mapping tool.

[Pieter et al. 2014] Pieter, v. Z., Derrick, K., and Louis, C. (2014). An evaluation of the hibernate object-relational mapping for processing interactive social networking actions.

[Prisma 2022] Prisma (2022). Best 11 orms for node.js, query builders & database libraries in 2022.

[Stevica and Dragan 2010] Stevica, C. and Dragan, J. (2010). A comparative study of the features and performance of orm tools in a .net en- vironment.

[Torres et al. 2017] Torres, A., Galante, R., Pimenta, M. S., and Martins, A. J. B. (2017). Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Information and Software Technology*, 82:1–18.

[Wohlin et al. 2012] Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., and Wessln, A. (2012). *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

[Zhiyu and Zhiang 2010] Zhiyu, Z. and Zhiang, C. (2010). Performance evaluation of transparent persistence layer in java applications.