INSTITUTO DOCTUM DE EDUCAÇÃO E TECNOLOGIA

FACULDADES INTEGRADAS DE CARATINGA

CIÊNCIA DA COMPUTAÇÃO

ANÁLISE COMPARATIVA ENTRE *FRAMEWORKS* DE MAPEAMENTO DE BANCO DE DADOS

MAURÍCIO GALDINO DOS SANTOS

Caratinga
2015

Maurício Galdino dos Santos

ANÁLISE COMPARATIVA ENTRE *FRAMEWORKS* DE MAPEAMENTO DE BANCO DE DADOS

Monografia apresentada ao Curso de Ciência da Computação das Faculdades Integradas de Caratinga como requisito parcial para obtenção do título de bacharel em Ciência da Computação orientada pelo Prof. Glauber Luiz da Silva Costa.

Caratinga

2015

FACULDADES INTEGRADAS DE CARATINGA

TRABALHO DE CONCLUSÃO DE CURSO

TERMO DE APROVAÇÃO

TÍTULO DO TRABALHO

ANÁLISE COMPARATIVA ENTRE FRAMEWORKS DE MAPEAMENTO DE BANCO DE DADOS

por

Maurício Galdino dos Santos

Este Trabalho de Conclusão de Curso foi apresentado perante a Banca de Avaliação composta pelos professores Glauber Luiz da Silva Costa, Bruno Vieira Becattini e Wanderson Miranda Nascimento, às 21 horas do dia 14 de dezembro de 2015 como requisito parcial para a obtenção do título de bacharel. Após a avaliação de cada professor e discussão, a Banca Avaliadora considerou o trabalho aprovado, com a qualificação: Ótimo.

Trabalho indicado para publicação: (X)SIM ( ) NÃO

Caratinga, 14 de DEZEMBRO de 2015

Professor Orientador e Presidente da Banca

Professor Avaliador 1

Bruno Vieira Becattini

Professor Avaliador 2

Maurício Galdino dos Santos

Aluno(a)

Coordenador (a) do Curso

DEDICATÓRIA

Dedico esse trabalho primeiramente a Deus, por ser essencial em minha vida, ao meu pai José Mauro e minha mãe Elvira Galdino, que não mediram esforços para que eu chegasse até essa etapa da minha vida.

AGRADECIMENTOS

Primeiramente agradeço a Deus por ter me abençoado, dando forças para superar as dificuldades e por estar presente, em todos os momentos da minha vida.

Um agradecimento em especial aos meus pais e familiares, pelo amor, carinho e apoio no decorrer desses quatro anos, que apesar das dificuldades sempre me incentivaram a continuar.

Aos professores do curso, que dividiram um pouco dos seus conhecimentos e em especial ao meu orientador, que me auxiliou na elaboração desse trabalho.

Agradeço também aos meus colegas e amigos da universidade e do trabalho que estiveram presente no decorrer desses anos, proporcionando momentos alegres e me auxiliando de forma direta e indireta, na conclusão desse trabalho.

“O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis.”

José de Alencar

“Senhor, mostre-me quais são os seus caminhos; ensine-me por onde devo andar. Ajude-me a andar na sua verdade; ensine-me o que é certo, pois o Senhor é Deus, o meu Salvador. Confiarei no Senhor por toda minha vida.”

Salmos 25: 4,5

RESUMO

Em virtude da diferença entre o modelo Orientado a Objetos, empregado na construção de *software* e o modelo Relacional dos Bancos de Dados, surgiram técnicas de Mapeamento Objeto-Relacional (ORM) que possibilitam uma melhor ligação entre os dois modelos. Contudo, existem inúmeras especificações e *frameworks* que implementam técnicas de ORM, por esse motivo, faz necessário identificar qual utilizar na construção de um *software*.

O trabalho proposto tem a finalidade de realizar uma análise comparativa entre dois *frameworks* existentes, para a linguagem PHP. Foram escolhidos os seguintes *frameworks*: Doctrine e Propel. A análise realizada teve como referência, a utilização da norma ISO/IEC 9126, que apresenta métricas de avaliações de *software*. Como critério de análise, foram escolhidas as seguintes métricas: eficiência, portabilidade e instabilidade.

Palavra-chave: *Frameworks*, Mapeamento Objeto-Relacional e métricas.

ABSTRACT

Due to the difference between the object-oriented model used in software construction, and the relational model of databases, techniques of object-relational Mapping (ORM) that enable a better linkage between the two models. However, there are numerous specifications and *frameworks* that implement ORM techniques, for this reason, it is necessary to identify which to use in the construction of a software.

The proposed work aims to carry out a comparative analysis between two existing *frameworks* for PHP language. Were chosen the following *frameworks*: Doctrine and Propel. The analysis had as reference the use of standard ISO/IEC 9126, which features reviews of software metrics. As a criterion of analysis were chosen the following metrics: efficiency, portability, resource usage and instability.

Keyword: Frameworks, Object-Relational Mapping and metrics.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE SIGLAS

BDOO - Banco de Dados Orientados a Objetos

CASE - *Computer – Aided Software Engineering*

DB – Banco de Dados

ER – Entidade/Relacionamento

IEC – Internacional Electrotechnical Commision

ISO – Internacional Organization for Standardization

OO – Orientado a Objetos

ORM – Mapeamento Objeto-Relacional

SGDB - Sistema de Gestão de Banco de Dados

SQL - Linguagem de Consulta Estruturada

SUMÁRIO

1. INTRODUÇÃO

Segundo a B2B Magazine (2015), o mercado de software e serviços de TI (Tecnologia da Informação) cresce em ritmo acelerado, devido as demandas do mercado por soluções inovadoras. No mundo atual, a informação se tornou algo de grande valor para as organizações, os bancos, por exemplo, possuem uma gigantesca quantidade de informações armazenadas, sobre seus clientes. Realizar a manipulação e gerenciamento dessas informações, torna-se algo crucial e de grande importância.

A necessidade de armazenar grande quantidade de dados, é algo que teve início por volta da década de 70, com o surgimento dos bancos de dados relacionais, que proporcionam uma estruturação e confiabilidade dos dados armazenados. Em contrapartida, na década de 90, com o crescimento do paradigma de orientação a objetos, que permitia construir sistemas mais robustos e rentáveis em menor quantidade de tempo, o número de *software* desenvolvido nesse modelo, teve um aumento significativo.

Devido ao modelo orientado a objetos (OO), ser muito diferente do modelo Entidade/Relacionamento (ER) utilizado pelos bancos de dados relacionais, os desenvolvedores encontravam dificuldades na hora de extrair dados, ou seja, informações armazenadas no banco de dados, para o modelo orientado a objetos, e no momento de armazenar os objetos no banco de dados (BD), sendo necessárias inúmeras linhas de código para realizar tais procedimentos, que demandavam grande tempo dos desenvolvedores, além do que, essas linhas de código, diminuíam significativamente a manutenibilidade e a capacidade de crescimento dos sistemas. (BAUER; KING, 2007).

Para solucionar esse problema de impedância, foram desenvolvidas técnicas para realizar a comunicação entre a aplicação e o banco de dados de maneira eficiente, fácil e com menor tempo, técnica essa conhecida como mapeamento objeto-relacional (Objetc Relacional Mapping - ORM). De acordo com Silva (2005) “O termo Mapeamento Objeto Relacional refere-se a técnica de mapear os registros do Banco de dados em objetos e persistir as informações contidas nos objeto em forma de linhas

e colunas”. Deste modo, possibilitando a interpretação dos dados de um modelo para o outro, de maneira explícita para o desenvolvedor.

Com o surgimento dessa técnica de ORM, inúmeros desenvolvedores adotaram sua utilização, proporcionando assim seu crescimento, com isso, surgiram várias especificações e *frameworks* para a linguagem PHP, deste modo, estabelecer qual utilizar se torna uma decisão crucial para os desenvolvedores.

O trabalho desenvolvido tem como finalidade, realizar análise comparativa entre o Doctrine e Propel, que são dois *frameworks* de ORM *open sources* para linguagem PHP. A escolha desses dois *frameworks,* foi realizada devido a pesquisa efetuada no Google Trends (Ferramenta do Google que relaciona termos mais pesquisados no Google Search), que apontou dentre os *frameworks open sources* pesquisados, quais apresentam maior percentual de busca pelos usuários.

Para realizar a análise comparativa entre os *frameworks*, foi utilizado como base as qualidades de *software* propostas pela ISO (*International Organization for Standardization*) com parceria da IEC (*International Electrotechnical Commision*), as quais elaboram uma norma conhecida como ISO/IEC 9126, que determina qualidades que um *software* deve possuir. Dentre as métricas de qualidade apresentadas pela norma ISO/IEC 9126, foram selecionadas para o desenvolvimento deste trabalho as seguintes métricas: eficiência, portabilidade e instabilidade.

Contudo o trabalho ainda tem como finalidade, beneficiar toda a comunidade científica, indicando quais fatores são mais relevantes para se analisar *frameworks* de ORM, demonstrar como implementar a aplicação de uma norma ISO e a realizar uma comparação de duas ferramentas, além de auxiliar desenvolvedores na escolha.

Esse trabalho foi divido em 6 capítulos, onde o segundo capítulo apresenta os referenciais teóricos que sustentam está pesquisa. O terceiro capítulo, informa a metodologia aplicada na realização da pesquisa e a construção do método de aplicação das métricas selecionadas. No quarto capítulo é apresentado os resultados da aplicação das métricas, subdivido de acordo com as métricas selecionadas. O quinto capítulo é apresentado as considerações finais e conclusão desse trabalho e por último, no sexto capítulo foi realizado algumas ideias para futuros trabalhos que podem ser realizados.

2. REFERENCIAL TEÓRICO

As próximas seções têm por objetivo, descrever os temas nos quais este trabalho científico se fundamenta, apresentando os principais conceitos referente a cada tópico.

2.1 BANCO DE DADOS

Um banco de dados (BD), é um repositório ou depósito onde é inserido dados. O BD tem como objetivo armazenar um conjunto de informações que posteriormente, podem ser acessadas de maneira fácil e ágil. Com o avanço da tecnologia nos últimos anos, surgiram novas aplicações dos sistemas de BD, possibilitando armazenar imagens, vídeos, extrair e analisar dados comerciais, controlar processos industriais, dentre outros. Devido esse avanço, o BD possui uma atuação em quase todas as áreas em que os computadores são utilizados. (ELMASRI; NAVATHE, 2010).

Uma definição para o termo banco de dados: “é uma coleção de dados estruturados, que pode ser desde uma simples lista de compras a uma galeria de imagens ou a grande quantidade de informação da sua rede corporativa.” (WILLEMANN; IBARRA, 2007). Os dados armazenados são eventos que podem ser armazenados e que fazem algum significado implícito.

> Um sistema de banco de dados é uma coleção de dados inter-relacionados e um conjunto de programas que permitem aos usuários acessar e modificar esses dados. Uma importante finalidade de um sistema de banco de dados é fornecer aos usuários uma visão abstrata dos dados, ou seja, o sistema oculta certos detalhes de como os dados são armazenados e mantidos. (SILBERSCHATZ, 2010).

Um BD pode ser criado e mantido manualmente ou computadorizado. Ao contrário de um bando de dados manualmente, o banco de dados computadorizado pode utilizar um conjunto de programas para criar e manter esse sistema, por meio de um sistema de gerenciador de bando de dados (SGBD – Sistema de Gestão de Banco de Dados). O SGBD é um conjunto de programas que possibilita aos usuários criar e

manter um BD, com o objetivo de facilitar o processo de definição, construção, manipulação e compartilhando de banco de dados, entre aplicações e usuários. (ELMASRI; NAVATHE, 2010).

Segundo Elmasri, Navathe (2010), segue a definição de cada processo do SGDB:

- Definir: Consiste em especificar os tipos, estruturas e restrições dos dados que serão inseridos (armazenados) no bando de dados.
- Construção: É armazenar os dados que são inseridos em algum meio controlado pelo SGBD.
- Manipulação: Possibilitar que os dados armazenados sejam alterados e recuperados por meio de funções de consultas.
- Compartilhamento: Permitir que diversos usuários e programas consigam acessar o BD simultaneamente.

A Figura 1, representa os processos descritos por Elmasri e Navathe (2010):

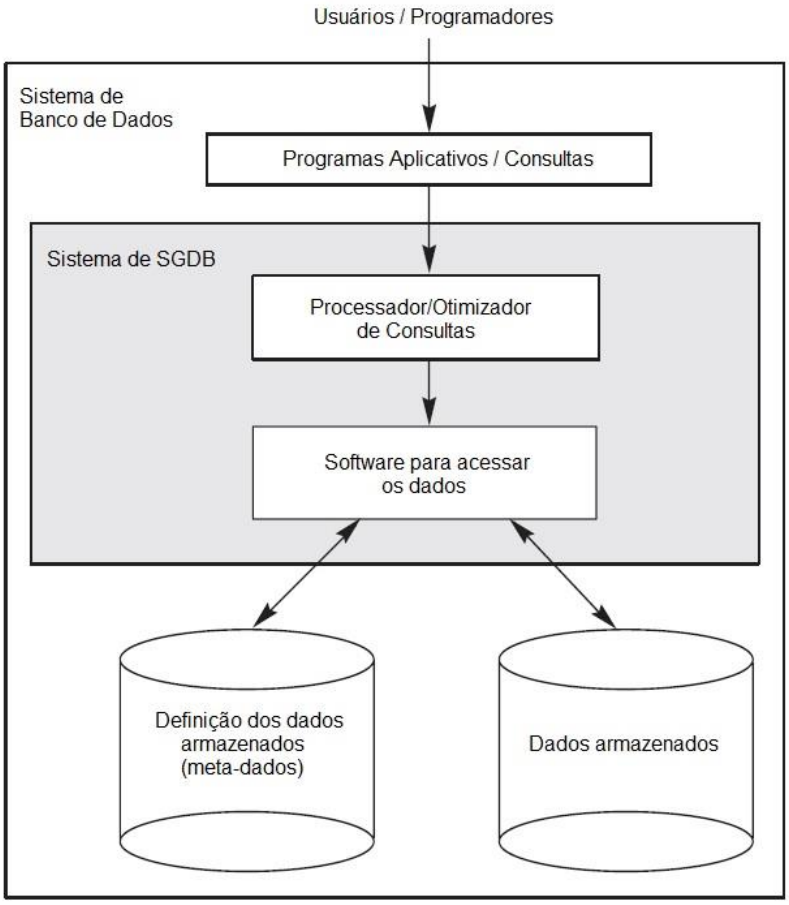

Figura 1 Diagrama simplificado de um ambiente de sistema de banco de dados.
Fonte: Elmasri e Navathe (2010).

O SGBD oferece outras funções relevantes, sendo elas: proteção, podendo ser contra falhas de *hardware* ou *software* e até mesmo de acesso não autorizado e a manutenção, que possibilita a evolução do sistema de banco de dados. (ELMASRI; NAVATHE, 2010).

Outra característica do SGBD, é garantir a persistência dos dados em um dispositivo físico, onde as informações armazenadas não serão perdidas, podendo ser recuperadas a qualquer momento. A seção 2.1.1, será abordado detalhadamente sobre persistência de dados.

2.1.1 Persistência de dados

Persistência de dados, consiste em garantir que as informações utilizadas em uma aplicação sejam duráveis, ou seja, mesmo com o encerramento da aplicação as informações são mantidas, por meio da utilização de um armazenamento que possibilita sua restauração posteriormente, sendo necessidade fundamental no sistema de computação (DATE, 2000).

A primeira forma de gerenciar a persistência dos dados, era a utilização de um simples arquivo binário e todo controle, ficava por conta do próprio sistema. Com o surgimento dos sistemas de gerenciadores de banco de dados (SGDBs), esse esforço em gerenciar foi retirado da aplicação e todo esforço necessário para controlar a persistência e suas regras é centralizado no SGDB. (SILBERSCHATZ, KORTH, SUDARSHAN, 2006).

Primordialmente, os BD relacionais foram criados para separar o armazenamento físico dos dados da sua representação conceitual, com isso os SGBDs, introduziram uma linguagem de consulta estruturada (SQL - Structured Query Language) de alto nível, facilitando e otimizando o acesso aos dados. Devido ao desempenho, facilidade e prática manutenção, a utilização de BD relacionais tornou-se comum nos sistemas até hoje. (ELMASRI, NAVATHE, 2005).

2.1.2 Linguagem de consulta estruturada

A linguagem SQL teve início nos anos 70, tendo como fundador a IBM (International Business Machines). Sua primeira versão ficou conhecida como SEQUEL (Structured Query English Language). Desde então, a linguagem passou por evoluções e seu nome foi mudado para SQL (Structured Query Language – Linguagem de Consulta Estrutura). (SILBERSCHATZ, KORTH, SUDARSHAN, 2003).

Devido ao sucesso da linguagem, o Instituto Americano Nacional de Padrões (ANSI) e a International Standards Organization (ISO) definiram padrões para realizar sua implementação. Com isso, a SQL tornou-se linguagem padrão para os bancos de dados relacionais, devido sua facilidade e simplicidade de utilização.

A SQL tem como objetivo definir, manipular e controlar os dados do BD, possibilitando desde a criação de tabelas, campos e tipos de dados, até a realização de consultas, inserções, exclusões e alterações em um ou mais registro de maneira simultânea.

2.2 MODELO DE REPRESENTAÇÃO DE DADOS

De acordo com Date (2004) o modelo de representação de dados, consiste na forma como os dados armazenados serão acessados e manipulados, pelos usuários. Dessa forma, esse modelo está ligado diretamente a qualidade do SGBD.

2.2.1 Modelo relacional

Segundo Date (2004), o modelo relacional é o mais utilizado nas aplicações atuais, teve seu início na década de 70, trazendo um conceito novo de modelagem de dados, nas quais entidades do mundo real poderiam ser representadas em forma de tabelas.

> O modelo relacional representa o banco de dados como uma coleção de *relações.* Informalmente, cada relação se parece com uma tabela de valores [...] No modelo relacional, cada linha na tabela representa um fato que corresponde a uma entidade ou relacionamento do mundo real. O nome da tabela e os nomes das colunas são usados para ajudar na interpretação do significado dos valores em cada linha. (ELMASRI, NAVATHE, 2005).

O modelo relacional é presentado por um conjunto de tabelas que se relacionam, por um nome ou várias colunas. A partir da criação do banco de dados, é modelado os itens do mundo real e os relacionamentos, onde cada classe de objeto modelado, é representado em uma tabela distinta. A tabela é formada por colunas e linhas, as colunas possuem um determinado tipo de valor, onde cada célula possui um valor único, as linhas são conhecidas como os registros ou tuplas. Para representar um registro em uma tabela, existe a chave (identificador) como garantia de valor único. (ELMASRI; NAVATHE, 2005).

A Tabela 1 é uma representação de uma tabela no modelo de dados relacional:

Tabela 1 Exemplo de um modelo relacional

Id	**nome_empregado**	**salario**
01	Roberto	1000.00
02	Marcos	1200.00
03	Pedro	1000.00

Fonte: Próprio autor.

2.2.2 Modelo orientados a objeto

Segundo Date (2004), o surgimento do modelo de BD orientados a objetos (OO), se deve a combinação da linguagem de programação orientada a objetos com ideias dos modelos de dados tradicionais. Nesse modelo de banco de dados orientados a objetos (BDOO) as tabelas do modelo relacional são representadas por objetos e seu acesso é realizado por meio de classes.

“Uma característica-chave dos bancos de dados orientados a objetos é o poder dado ao projetista para especificar tanto a estrutura de objetos complexos quanto as operações que podem ser aplicadas a esses objetos” (ELMASRI e NAVATHE, 2005).

O BDOO possibilita atender as necessidades das aplicações mais complexas, oferecendo melhor flexibilidade para trabalhar com os dados, eliminando limitações a tipos de dados e linguagens de consultas.

2.2.3 Impedância objeto-relacional

De acordo com Ambler (2003), impedância objeto-relacional se deve ao fato que o modelo ER é definido matematicamente como dados normalizados em tabelas ao contrário do modelo OO, que é estabelecido em classes, heranças e polimorfismos.

A Figura 2, representa a diferença entre ambos os modelos:

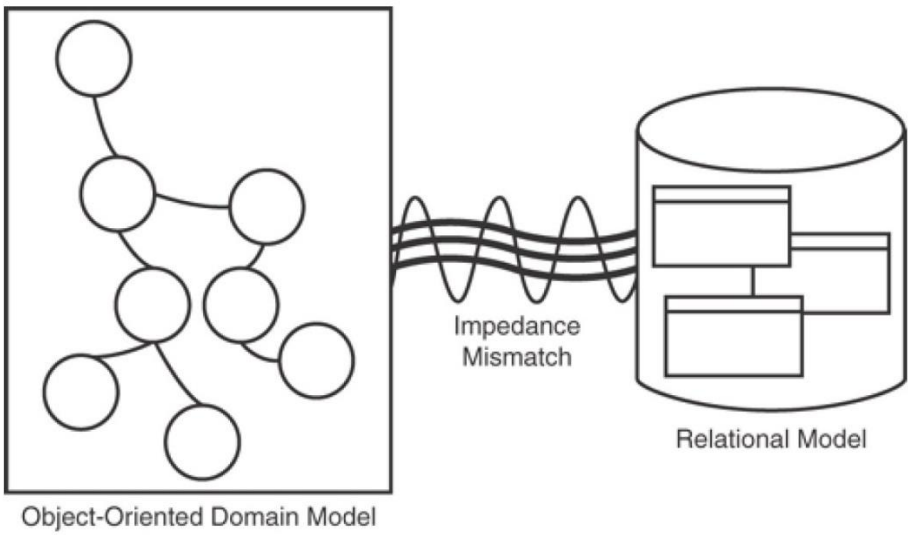

Figura 2 Impedância Objeto Relacional.
Fonte: Ambler (2003).

2.2.4 Mapeamento objeto-relacional

De acordo com Luckow e Melo (2010), a forma mais comum de armazenar dados é por meio da utilização de bancos relacionais, deste modo é necessário a utilização da linguagem orientada a objeto (OO) uma interação mais funcional. A partir

da utilização dessa linguagem, ocorre a necessidade de converter objetos em tabelas e tabelas em objetos, e em muitos casos a linguagem torna-se incompatível. Deste modo, com a utilização do mapeamento objeto-relacional (ORM), é realizado mapeamento dos objetos em uma tabela de banco de dados relacionais.

O ORM, permite reduzir a impedância que existe entre o modelo orientado a objetos (OO) e o modelo relacional, durante o desenvolvimento de *software.* Quando realiza-se o mapeamento dos dados do modelo OO para o modelo relacional, é criado um efeito de que o banco de dados (DB) seja orientado a objetos. Com isso, o desenvolvedor não precisa preocupar-se com a organização dos dados nas tabelas, podendo focar-se na manipulação de objetos e no problema de negócio (BAUER e KING, 2007).

Ainda de acordo com Bauer e King (2007), com a implementação do ORM, é realizado uma transformação (reversível) de dados de um modelo para outro, de forma transparente, para o usuário.

A Figura 3, representa a implementação do ORM, sendo possível enxergar a ligação estabelecida entra os dois modelos de representação de dados. O *framework* de ORM, atua na tradução dos objetos para a linguagem do BD.

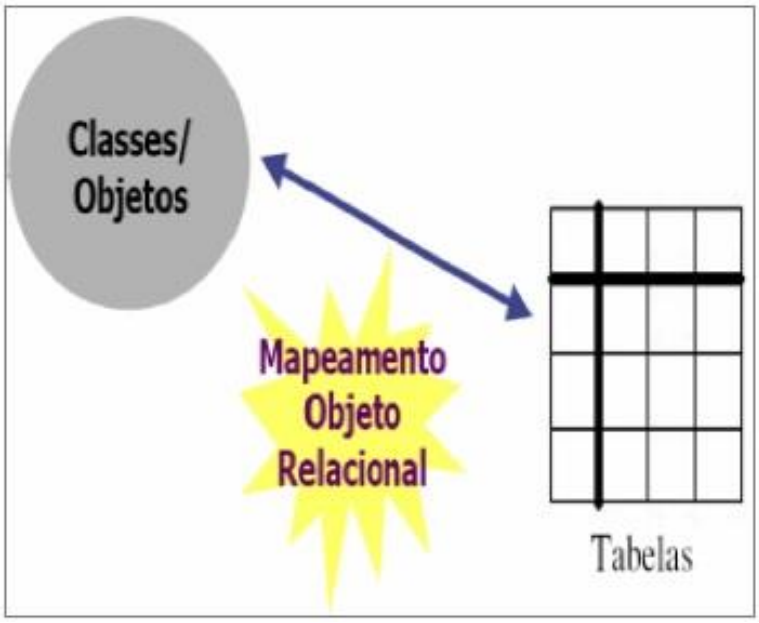

Figura 3 Exemplo de Mapeamento Objeto Relacional
Fonte: CORDEIRO (2011)

Os *frameworks* ORM são ferramentas que auxiliam os desenvolvedores a

realizarem atividades de mapeamento de banco de dados, deste modo são consideradas como ferramentas CASE. Na seção 2.3 será abordado detalhadamente o que são ferramentas CASE.

2.3 FERRAMENTAS CASE

O termo CASE (*Computer – Aided Software Engineering*), é definido como automação de desenvolvimento de *software*, sendo um conjunto de técnicas e ferramentas informatizadas que tem por objetivo auxiliar, o engenheiro de software ou desenvolvedor no desenvolvimento de aplicações, reduzindo o esforço e complexidade gastos no desenvolvimento. As ferramentas CASE são uma caixa de ferramentas, que proporciona uma automação das atividades, que antes eram feitas manualmente, dessa forma facilitando assim o desenvolvimento. (RAMOS, 2011).

> A Engenharia de Software Auxiliada por Computador (CASE – Computer - Aided Software Engineering) é o nome dado ao software usado para apoiar as atividades de processo de software, como engenharia de requisitos, projeto, desenvolvimento de programas e teste. As ferramentas CASE, portanto, incluem editores de diagramas, dicionário de dados, compiladores, debuggers, ferramentas de construção de sistemas etc. A tecnologia CASE fornece apoio ao processo de software pela automação de algumas atividades de processo e pelo fornecimento de informações sobre o software que está sendo desenvolvido (SOMMERVILLE, 2007, p. 56).

De acordo com Ramos (2011), as ferramentas CASE, são indispensáveis hoje em dia, para uma empresa desenvolvedora de *software.* Devido essas ferramentas auxiliarem em todo o ciclo de desenvolvimento: análise; projeto; implementação e teste, além de serem de grande importância para a manutenção de *software*.

A Figura 4, representa a utilização de ferramentas CASE para auxiliar na comunicação entre os dados enviados ou acessados pelo usuário do banco de dados.

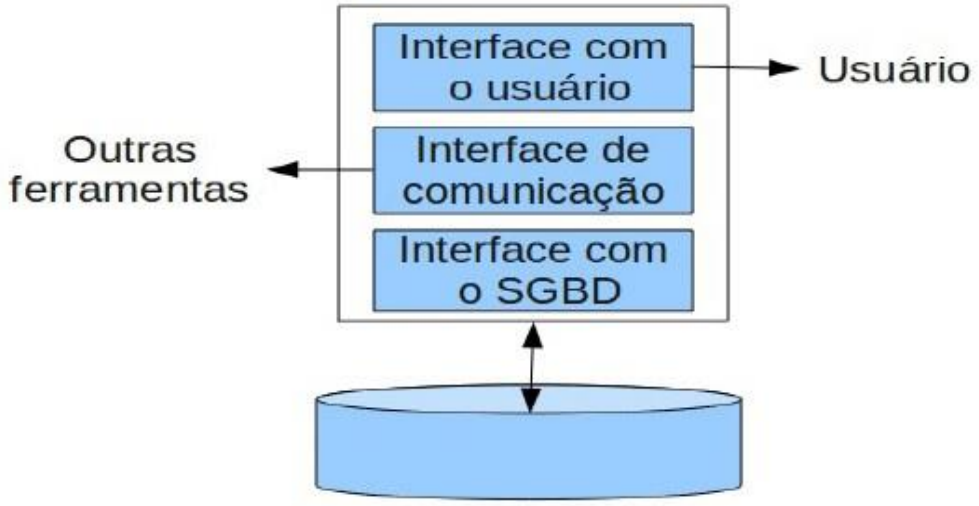

Figura 4 Composição básica de uma ferramenta CASE.
Fonte: Caliari (2010).

Como mostra a Figura 4, entre a camada de interface do sistema e o SGDB, existe a camada responsável por realizar a comunicação entre as duas camadas. Nessa camada que se concentra os *frameworks* ORM, que são responsáveis por realizar a comunicação. Na seção 2.3.1, será abordado mais detalhadamente o que são *frameworks* e sua determinada importância para desenvolvimento de sistema.

2.3.1 *Frameworks*

Frameworks é definido como um projeto, que pode ser reutilizável em um sistema ou somente parte dele, sendo montado por classes associadas que cooperam entre si. Inicialmente a expressão *framework,* estava associada ao conceito de bibliotecas de classes reutilizáveis, mas teve seu conceito ampliado para qualquer solução inacabada, que pode ser completada por meio de instanciação, proporcionando o desenvolvimento de mais uma aplicação. (SILVA, 2005)

> "Framework é uma infra-estrutura genérica, baseada em um domínio, que pode ser adaptada para solucionar problemas específicos desse domínio, servindo como um modelo para a construção de aplicações através da especificação das classes e das colaborações entre elas". (PAZIN, 2004, p.16)

Os desenvolvedores que apreciam o desenvolvimento orientado a objetos, destacam a principal vantagem em utilizar essa metodologia, sendo ela: A possibilidade de reutilizar em diferentes sistemas. Mas esse reuso orientado a objetos é melhor suportável, quando utilizado abstrações de alta granularidade, conhecidos como *frameworks.* (SOMMERVILLE, 2009).

Segundo Willemann e Ibarra (2007), os *frameworks* têm como objetivo auxiliar no desenvolvimento, proporcionando um ganho de tempo ao desenvolvedor além de reduzir custos, aumentar a reutilização de códigos, permitindo também desenvolver códigos mais seguros e com um menor número de defeitos.

> “Um framework de desenvolvimento é uma “base” de onde se pode desenvolver algo maior ou mais específico. É uma coleção de códigos-fonte, classes, funções, técnicas e metodologias que facilitam o desenvolvimento de novos softwares.” (MINETTO, 2007).

De acordo com Gamma et al. (2002, p.42), *framework* é definido como “um conjunto de classes que cooperam entre si provendo assim um projeto reutilizável para uma específica classe de *software*”. Ainda segundo Willemann e Ibarra (2007), os *frameworks* estão se tornando cada vez mais importantes e frequentes, fazendo com que os sistemas orientado a objeto, consigam uma melhor reutilização.

Segundo Gamma (2002), as principais vantagens de um *framework* são o aumento do reuso, redução do tempo gasto no desenvolvimento de aplicações, padronização, compatibilidade e não haver necessidade de implementar partes básicas de um sistema, devido os *frameworks* já possuírem essas configurações, deste modo, agiliza no processo de desenvolvimento.

Um modelo de *frameworks* são os de mapeamento objeto-relacional (ORM), que tem a finalidade de realizar a comunicação da aplicação com o banco de dados (DB), para se realizar consultas ou inserções de dados no DB. Na seção 2.3.3 serão apresentado os *frameworks* de ORM.

2.3.2 *Frameworks* de mapeamento objeto-relacional

Atualmente existe algumas ferramentas *open sources* (*software* livre) ou pagas, capazes de realizar mapeamento entre objeto e registro de tabelas de banco de dados relacionais, com objetivo de realizar operações básicas: armazenamento, atualizações, exclusão e consultas.

Segundo Silva (2005), as ferramentas responsáveis pelo mapeamento devem ser mais ou menos genéricas e deve possuir uma capacidade de adaptar-se aos recursos existentes. Realizar buscas no banco de dados pode ser tornar complexo se a ferramenta de mapeamento, possuir restrições de integridade sobre o modelo relacional utilizado.

Ainda de acordo com Silva (2005), a ferramenta deve prover funcionalidade de pesquisa, de montagem de objeto e de atualizações. Esses mecanismos devem ser capazes de adaptar-se a arquitetura do sistema de informação e devem gerenciar quaisquer problemas de transações de maneira correta.

Ferramenta de mapeamento quando implementadas em uma empresa de desenvolvimento, oferecem as seguintes vantagens descritas abaixo:

- Melhoria na manutenibilidade de código;
- Redução na quantidade de código relacionado a persistência;
- Redução no trabalho gasto, não havendo problema de persistência, pode-se concentrar em resolver problemas da regra de negócio;
- Aumento da portabilidade da aplicação. Devido os *frameworks* de mapeamento possuírem compatibilidade com a maioria dos bancos de dados relacionais existentes.

Nas seções 2.3.2.1 e 2.3.2.2, serão abordados os *frameworks* ORM selecionados para realização desse trabalho.

2.3.2.1 Doctrine

O doctrine é uma ferramenta *open source*, que tem como objetivo criar uma série de bibliotecas PHP, com a finalidade de tratar das funcionalidades de persistência de dados e funções, relacionadas a isto. (MINETTO, 2014).

Segundo Doctrine core team (2015), o *framework* Doctrine é utilizado em desenvolvimento de programas na linguagem PHP, é desenvolvido sobre uma poderosa camada de abstração de banco de dados. Sendo capaz de converter registros de um banco de dados em objetos, dessa forma, evitando que os dados sejam corrompidos, pois as características de relacionamento e restrições são mantidas. Oferece enormes benefícios em questão de desempenho, clareza e modalização do código fonte, mas é uma ferramenta complexa de se utilizar.

A arquitetura do Doctrine é baseada no padrão de projeto *Unit of Work*, que é responsável por controlar a persistência de dados no banco relacional. Em uma aplicação onde ocorrer inúmeras manipulações (seleção, inserção e atualizações) de dados, ocorre o excessivo controle de transações. Deste modo, o padrão tem a finalidade de gerenciar o uso das transações e persistência de dados. (MARQUES, JÚNIOR, 2014).

2.3.2.2 Propel

O propel é outro exemplo de *framework* para desenvolvimento de aplicações na linguagem PHP, que é responsável por realizar mapeamento objeto relacional. É uma ferramenta *open source*, que possibilita acessar o banco de dados utilizando um conjunto de objetos, fornecendo uma simples API que possibilita armazenar e recuperar os dados. (PROPEL, 2015).

Segundo Cézar (2007), a arquitetura do propel utiliza uma camada denominada Creole, que é responsável por realizar a comunicação entre o servidor de banco de dados e a aplicação, desse modo, uniformizando a comunicação. Nessa camada é utilizado arquivos de configuração para se criar objetos complexos, para isso é

utilizado arquivo XML, contendo as informações sobre as tabelas e campos do banco de dados.

Dentre os dois *frameworks* ORM apresentados, saber escolher qual implementar no desenvolvimento de *software* é crucial, devido isso, é necessário identificar qual possui melhor qualidade. Para isso, é de grande importância compreender o que é qualidade e metodologias de análises de qualidade. Por esse motivo, a seção 2.4 abordará sobre qualidade de *software* e metodologias de análises.

2.4 QUALIDADE DE PRODUTO DE *SOFTWARE*

Segundo Sommerville (2003), o principal objetivo da maioria das organizações é alcançar um alto grau de qualidade nos produtos ou serviços, desenvolver e entregar produtos de baixa qualidade, não é mais aceitável nos dias atuais. Por esse motivo, é necessário compreender melhor o que é qualidade e metodologias de análises, que será apresentado a seguir.

2.4.1 *Software*

Software é definido como uma sequência de instruções, que quando executadas ou seguidas pelo computador, tem o objetivo de realizar uma tarefa específica, possibilitando a manipulação de informações de forma adequada e correta. (PRESSMAN, 2011).

Segundo Sommerville (2013) o mundo atual, não conseguiria existir sem a existência de um *software*. Sendo que existe inúmeros tipos de sistema de *softwares*, responsáveis por controlar diversas áreas, como de infraestruturas, serviços nacionais, áreas de entretenimento: cinema, televisão, jogos de computador, dentre outros. Devido isso, é de grande importância desenvolver *softwares* de alta qualidade.

2.4.2 Qualidade de *software*

Segundo Sommerville (2009), o primeiro problema relacionado a qualidade de *software* surgiu em 1960, devido ao sistema entregue era difícil de se manter, pouco confiável e lento. Devido a insatisfação na época, adotou-se medidas de gerenciamentos de qualidade de *software*, que proporcionaram melhorias significativas no nível de qualidade de *software*.

O termo qualidade de *software* é algo difícil de se definir. Qualidade refere-se a algo que pode ser medido ou comparado, como por exemplo: tamanho; cor; peso, dentre outros. De forma mais geral, qualidade de *software* refere-se a uma gestão de qualidade, que quando aplicada, tem objetivo de criar um produto conveniente, que forneça um valor calculável para quem utiliza ou que desenvolvem tal produto. (PRESSMAN, 2011).

A importância da qualidade de *software* vem aumentando significativamente, devido as exigências cada vez maiores dos usuários, em relação a eficiência, eficácia e valor final do produto de *software* produzido.

> De um produto exige-se qualidade e preço. Portanto como produto, *software* tem que ter o nível de qualidade exigido e procurar ser desenvolvido no menor custo possível. Esta é exatamente a função da engenharia, procurar sistemas de melhor qualidade dentro de um custo compatível com essa qualidade, otimizando a redução de custos. (LEITE, 2010).

A satisfação dos usuários está ligada a satisfação do produto, que está relacionada ao desempenho e com ausências de falhas, defeitos ou erros. Assim sendo, a satisfação do produto é obtida quando as necessidades do cliente são supridas.

Com o avanço tecnológico e o aumento das exigências em se produzir *software* de qualidade, com menor custos e menor quantidade de defeitos, levou o surgimento de normas, com o objetivo de permitir realizar uma correta avaliação tanto do produto desenvolvido, quanto do processo de desenvolvimento desses produtos. (PERES, 2006).

2.4.3 Norma de qualidade de *software*

A International Organization for Standardization (ISO), é uma organização não governamental, fundada em 1947. Seu surgimento se deve a necessidade de referências internacionais para apresentar um esquema padronizado, bem definido e aceito, para analisar a qualidade dos produtos de *software*. (PERES, 2006)

> A ISO, que objetiva promover o desenvolvimento de padronização, no sentido de facilitar o intercâmbio de bens e serviços entre os povos, foi criada em razão da necessidade das empresas de exportarem seus produtos, por causa da chamada barreira técnica. Os padrões da ISO estabelecem especificações técnicas e definem características para garantir que produtos, serviços ou processos sejam adequados a seus propósitos. (GUERRA; COLOMBO, 2009)

A International Electrotechnical Commission (IEC), é uma organização mundial, fundada em 1906, que tem a finalidade de publicar normas internacionais relacionadas a padronização de tecnologias elétricas, eletrônicas e relacionadas. A ISO com parceria da IEC, juntas, elaboraram um conjunto de normas que tem a finalidade de padronizar a qualidade de produtos de *software*. (GUERRA; COLOMBO, 2009).

Ainda de acordo com Guerra; Colombro (2009) as normas indicam um modelo de qualidade e processos para realizar a avaliação do *software*, se tornando um auxílio para obtenção de produtos de *software* com uma maior qualidade e mais confiável. Essas normas, ainda possibilitam o surgimento de novas metodologias para se avaliar a qualidade de produto de *software*, com base em seus atributos de qualidades, reconhecidos internacionalmente.

2.4.4 Norma ISO/IEC 9126

A norma ISO/IEC 9126, é referência no que diz respeito a avaliação de *software*, sendo dividida em quatro documentos: modelo de qualidade, métricas externas, métricas internas e métricas de qualidade de uso (ABNT, 2003).

Segundo a Associação Brasileira de Normas Técnicas (ABNT) a ISO/IEC 9126, possibilita que a qualidade do *software* seja classificada e avaliada em diferentes

possibilidades pelos envolvidos com aquisição, requisitos, uso, desenvolvimento, apoio, manutenção, avaliação, auditoria de *software* e garantida da qualidade. Essa norma pode ser utilizada, por desenvolvedores, pessoal responsável pela garantia de qualidade ou até mesmo avaliadores independentes.

De acordo com Pressman (2011), a norma ISO/IEC 9126, foi elaborada com o intuito de identificar os principais atributos de qualidade de *software*, sendo identificados seis atributos fundamentais para qualidade. Almeida (2005), traz a seguinte definição acerca dos seis atributos:

- Funcionalidade: Refere-se ao nível com que o *software* é capaz de satisfazer as necessidades explicitas ou implícitas dos usuários e suas propriedades específicas;
- Confiabilidade: Refere-se a quantidade de tempo que o *software* consegue manter seu funcionamento sobre condições especificas;
- Usabilidade: Refere-se ao grau de facilidade da compreensão, utilização e aprendizagem do *software,* e também sua capacidade de agradar ao usuário, quando utilizado sobre condições especificas;
- Eficiência: É a capacidade que o *software* possui de prover os requisitos de performance, com relação a quantidade de recursos utilizados, sob condições definidas;
- Facilidade de manutenção: A facilidade que o *software* pode de ser modificado, corrigido, ou melhorado com relação a mudanças no ambiente ou nos requisitos;
- Portabilidade: Capacidade do *software* ser transferido de um sistema operacional para outro.

Dentre os atributos apresentados, foram aplicadas métricas relacionadas a eficiência, portabilidade e facilidade de manutenção. Essas métricas foram selecionadas para realizar a análise comparativa entre os *frameworks* ORM. A seção 2.4.5 a seguir, apresenta a definição de métricas de *software.*

2.4.5 Métricas de *software*

Métricas de *software* é definido como medidas quantitativas, ou seja, medidas que podem atribuir um valor, possibilitando ter uma ideia se os resultados obtidos alcançaram as metas definidas. Sendo utilizadas também para detectar grupos de problemas, possibilitando assim elaborar soluções que possam ser desenvolvidas, possibilitando também um melhor processo de *software*. (PRESSMAN, 2006).

> Medição é uma ferramenta de gestão. Se conduzida adequadamente, fornece conhecimento a um gerente de projetos. E, como resultado, apoia o gerente de projeto e a equipe de *software* na tomada de decisão que irão conduzir a um projeto de sucesso. (PRESSMAN, 2006).

Segundo Sommerville (2003), métricas é considerado como qualquer tipo de medição, processo ou documentação. Deste modo, permitindo que os *softwares* sejam quantificados, possibilitando realizar previsões gerais, realizar o controle do processo de *software* e até mesmo, para identificar alguma anomalia no processo de desenvolvimento.

O processo de realização de uma métrica, consiste primeiramente em estabelecer um passo a passo, auxiliando assim a coletar, calcular e analisar as métricas, deste modo, melhorando a obtenção dos resultados obtidos. A Figura 5 a seguir, faz referência de um passo a passo a ser seguindo para realização de uma medição:

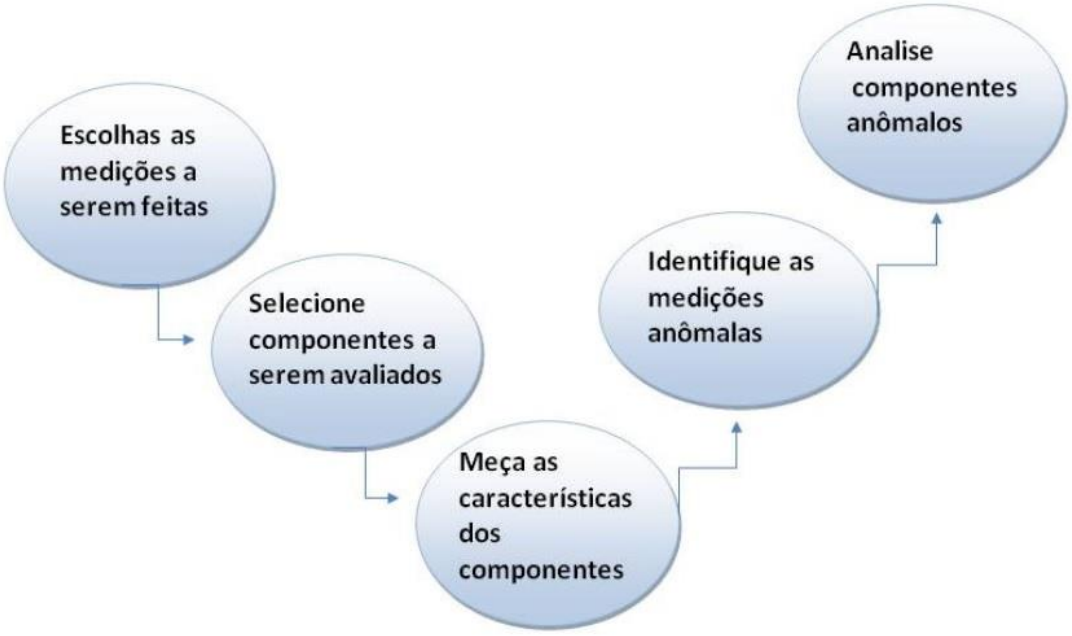

Figura 5 Processo de medição de um *software*.
Fonte: Sommerville (2003).

Ainda de acordo com Pressman (2006), aplicar métricas possui um duplo objetivo, sendo possível utilizar para minimizar o cronograma de desenvolvimento, evitando atrasos, problemas e até mesmo riscos e podem ser utilizadas para avaliar a qualidade do *software* durante seu processo de evolução, possibilitando melhorar a qualidade. Com isso, as métricas oferecem um resultado direto no aumento de qualidade e uma diminuição de defeitos e até mesmo, na redução do custo total do *software* final.

2.4.5.1 Categorização das métricas

Segundo a ISO/IEC 9126, métricas de *software* podem ser classificadas de maneira distintas, sendo elas: qualidades internas, externas e qualidade em uso.

Qualidade interna: São utilizadas para avaliar a especificação ou o código fonte de um *software.* Pode ser aplicado em partes especificas do *software,* para garantir a qualidade final. Seu objetivo é assegurar a qualidade externa e qualidade de uso.

> Qualidade interna é a totalidade das características do produto de *software* do ponto de vista interno. A qualidade interna é medida e avaliada com relação aos requisitos de qualidade interna. Detalhes da qualidade do produto de *software* podem ser melhorados durante a implementação do código, revisão e teste, mas a natureza fundamental da qualidade do produto de *software* representada pela qualidade interna mantém-se inalterada, a menos que seja reprojetada. (ISO 9126, 2003).

Qualidade externa: São medidas que são coletadas quando o software é executado. Seu objetivo é avaliar o *software* através das necessidades do usuário.

> Qualidade externa é a totalidade das características do produto de *software* do ponto de vista externo. É a qualidade quando o *software* é executado, o qual é tipicamente medido e avaliado enquanto está sendo testado num ambiente simulado, com dados simulados e usando métricas externas. (ISO 9126, 2003).

Qualidade em uso: Responsável por medirem o quanto o *software* atende as necessidades do usuário. As medidas são coletadas por meio de observações do uso do *software.*

> Qualidade em uso é a visão da qualidade do produto de *software* do ponto de vista do usuário, quando este produto é usado em um ambiente e um contexto de uso especificados. Ela mede o quanto usuários podem atingir seus objetivos num determinado ambiente e não as propriedades do *software* em si. (ISO 9126, 2003).

Dentre as classificações apresentadas, foram aplicadas para a análise dos *frameworks* as métricas de qualidade interna que são apresentadas pela norma ISO/IEC 9126-3, que é responsável por apresentar as métricas de avaliações internas.

Nesse capítulo, foi apresentado o que é qualidade e metodologias de análise por meio da utilização da norma ISO/IEC 9126. Com isso o capítulo 3, tem o objeto de apresentar como foi aplicado a utilização da ISO para realizar a análise proposta nesse trabalho.

3. METODOLOGIA

A pesquisa desenvolvida pode ser qualificada como: aplicada ou tecnológica, devido ao ponto inicial ter iniciado por meio de um estudo relacionado a teoria de Orientação a Objetos, Banco de dados, Mapeamento Objeto Relacional e Qualidade de *Software*, com a finalidade de prover uma análise de *frameworks* que são princípios sobre estes conceitos.

De acordo com os objetivos deste trabalho, essa pesquisa é definida como descritiva, pois tem como base a observação, registro e análise das coletas dos dados, que foram realizados por meio de uma pesquisa bibliográfica e avaliação de teste em um *software* desenvolvido, utilizando os *frameworks* ORM selecionados.

Com relação aos métodos de pesquisa pode ser qualificado como um estudo de caso, visto que foram selecionadas as soluções ORM, para serem utilizadas em uma aplicação desenvolvida para realização dos testes.

A seção a seguir, explica os critérios utilizados para se avaliar os *frameworks* ORM, como é realizada sua implementação e o modo como os testes foram conduzidos.

3.1 *Frameworks* selecionados

De acordo com Roebuck (2012, p. 32) existe uma variedade de *frameworks* ORM para linguagem PHP. Dentre os *frameworks* apresentados por Roebuck (2012), foram selecionados para desenvolver o trabalho proposto apenas os *open source*. Devido ao número de *frameworks open sources*, foi realizado uma pesquisa no *Google Trends*, com objetivo de selecionar quais *frameworks* são mais pesquisados pelos usuários.

A Figura 6 a seguir, apresenta a pesquisa realizada no *Google Trends* onde é possível observar que dentre os *frameworks* pesquisados, os que apresentaram maior busca pelos usuários são: Doctrine e Propel, devido isso os demais *frameworks* analisados foram desconsiderados para realização desse trabalho.

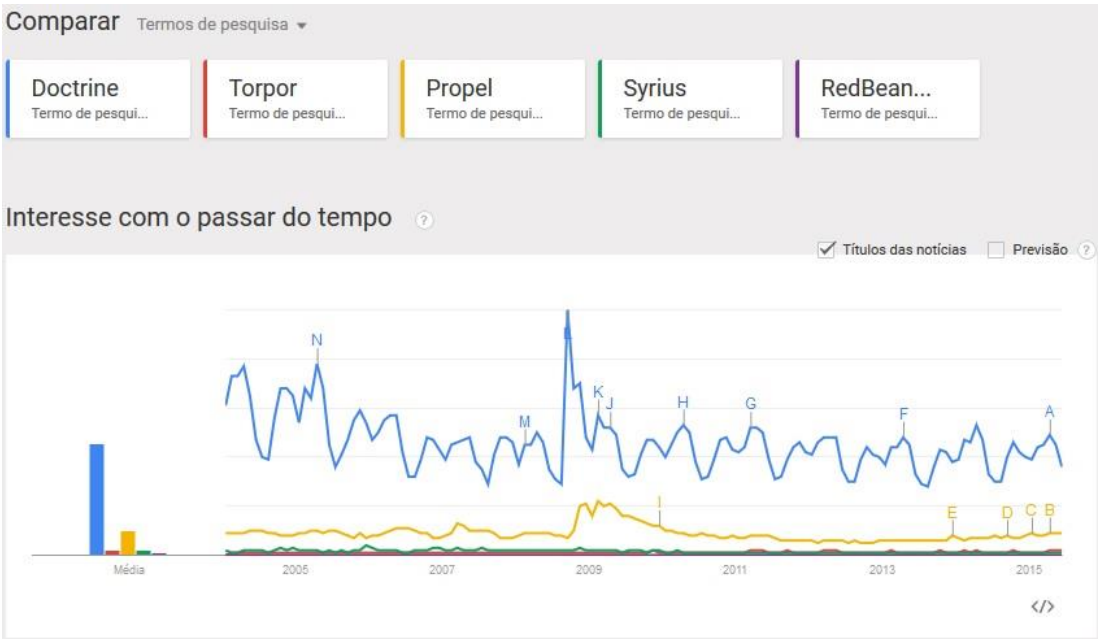

Figura 6 Pesquisa realizada no Google Trends indicando os *frameworks* mais pesquisados no Google Search.

Fonte: Próprio autor.

Os *frameworks* definidos foram analisados com o objetivo de indicar qual proporciona melhor eficiência, portabilidade, instabilidade e menor utilização de recursos da CPU, sendo esse os critérios de comparação definidos. Na seção 3.2 a seguir, será apresentado de forma detalhada os critérios de comparação.

3.2 Critérios de comparação

Para poder realizar a comparação entre os *frameworks* ORM analisados nesse trabalho, foi necessário estabelecer métricas definidas pela norma ISO 9126-3 que seriam relevantes para análise. Deste modo, a definição dessas métricas, foram realizadas tendo como base uma pesquisa bibliográfica relacionada ao tema, tendo como crucial, as orientações dos trabalhos de Bauer e King (2007) e Pothu (2008).

A seção 3.2.1 a seguir, apresenta a primeira métrica selecionada para realização da análise proposta.

3.2.1 Eficiência

A métrica de eficiência tem a finalidade de avaliar o tempo de resposta ou de processamento necessário para realizar determinadas tarefas. Para realizar a medição é analisado o tempo gasto pelos *frameworks* para apresentar a resposta ou resultado de uma determinada tarefa especifica. Medir a eficiência é crucial para indicar qual *framework* apresenta melhor tempo de resposta ao realizar tarefas especificas.

A Tabela 2 apresenta as métricas relacionadas a eficiência, apresentando seu propósito, metodologia, a fórmula e a interpretação dos dados.

Tabela 2 Descrição das métricas de Eficiência

Nome da Métrica	**Propósito**	**Metodologia**	**Fórmula**	**Interpretação do resultado**
Tempo de resposta	Qual é o tempo estimulado para completar consultas específica?	Avaliar a eficiência do da aplicação de acordo com o tempo gasto para completar uma determinada tarefa. Realizar a medição analisando partes especificas da aplicação.	**X** = tempo (calculado ao simulada)	Quanto menor tempo, melhor.
Tempo de processamento	Qual o número estimado de tarefas que podem ser	Avaliar a eficiência de manuseamento	**X** = Número de tarefas por unidade de tempo	Quanto maior, melhor.

	realizadas através de uma unidade de tempo?	de recursos da aplicação.		
Tempo de retorno	Qual é o tempo estimado para completar um grupo de tarefas especificadas?	Avaliar a eficiência do da aplicação de acordo com o tempo estimado de resposta para completar uma determinada tarefa. Realizar a medição analisando partes especificas da aplicação.	**X** = tempo (calculado ou simulada)	Quanto menor tempo melhor.
Tempo gasto para realizar importação de dados	Qual é o tempo estimado gasto para realizar a importação de dados?	Avaliar o tempo gasto para os *frameworks* realizarem a importação de dados.	**X** = tempo (calculado ao simulada)	Quanto menor tempo melhor.

Fonte: Tradução do próprio Autor: Fonte: ISO/IEC 9126-3, 2003.

As métricas de eficiência, têm o objetivo de avaliar o tempo gasto em cada *framework,* para realizar operações básicas de persistência de dados (CRUD).

3.2.2 Portabilidade

A métrica de portabilidade tem a finalidade de avaliar a capacidade dos *frameworks* funcionar em diferentes modelos de bancos de dados. Deste modo, foram analisados os números de banco de dados suportados pelos *frameworks* e no caso de substituição do modelo de banco de dados, foram verificados se os *frameworks* ainda funcionam corretamente verificando o número de funções que devem ser modificadas. Deste modo, a portabilidade avalia a adaptabilidade e a capacidade de substituições de modelos de dados.

A Tabela 3 apresenta a métrica relacionada a portabilidade, apresentando seu propósito, metodologia, a fórmula e a interpretação dos dados.

Tabela 3 Descrição das métricas de portabilidade.

Nome da Métrica	**Propósito**	**Metodologia**	**Fórmula**	**Interpretação do resultado**
Adaptabilidade das estruturas de dados	Como é o produto adaptável as mudanças de estrutura de dados	Contar o número de estruturas de dados, que são operáveis e não tem qualquer limitação depois da adaptação e compará-lo com o número total de estrutura de dados que exigem	X = A / B A = Número de estruturas de dados que são operáveis após a adaptação. B = Número de estruturas de dados que exige capacidade de adaptação.	0 <= X <= 1 Quanto mais próximo de 1, melhor.

		capacidade de adaptação.		
Compatibilidade com estruturas de dados diferentes	Contabilizar o número de banco de dados suportados pelos *frameworks*	Contar o número de banco de dados que são suportados pelos *frameworks*	X = número de estruturas de dados suportados.	Quanto maior número de banco de dados, melhor

Fonte: Tradução do próprio Autor: Fonte: ISO/IEC 9126-3, 2003.

As métricas de portabilidade têm o objetivo de avaliar o número de funções que são alteradas após uma mudança na estrutura de dados, e o número de banco de dados suportado por cada *framework*, sendo que a métrica de número de banco de dados suportado, foi adaptada para atender o meu objeto de estudo.

3.2.3 Instabilidade

A métrica de instabilidade tem a finalidade de avaliar a flexibilidade que os *frameworks* selecionados possuem para realizar a instalação. Essa métrica é relevante para análise dos *frameworks*, para avaliar a facilidade e opções disponíveis para realizar a instalação.

A Tabela 4 apresenta a métrica relacionada a instabilidade, apresentando seu propósito, metodologia, a fórmula e a interpretação dos dados.

Tabela 4 Descrição das métricas de utilização de instabilidade

Nome da Métrica	Propósito	Metodologia	Fórmula	Interpretação do resultado
Flexibilidade de instalação	Como flexível e personalizável é a capacidade de instalação?	Conte o número de operações de instalação podem ser utilizadas.	0 <= X >= 1 Quanto maior o valor de X, melhor.	Quanto maior, melhor.

Fonte: Tradução do próprio Autor: Fonte: ISO/IEC 9126-3, 2003.

A métrica de utilização de instabilidade, foi adaptada para atender o objeto de estudo a partir da métrica de flexibilidade de instalação. Essa métrica tem como objeto, indicar qual o número de opções possíveis para se realizar a implementação dos *frameworks* selecionados.

3.2.4 Utilização de recursos

A métrica de utilização de recurso tem a finalidade de avaliar a quantidade de recurso da CPU são utilizados para os *frameworks* conseguirem realizar uma determinada tarefa específica. Essa métrica é de grande importância, devido a análise realizada para identificar qual *framework* realiza menos utilização da CPU para realizar determinada tarefa especifica.

A Tabela 5 apresenta a métrica relacionada a utilização de recursos, apresentando seu propósito, metodologia, a fórmula e a interpretação dos dados.

Tabela 5 Descrição das métricas de utilização de recurso

Nome da Métrica	Propósito	Metodologia	Fórmula	Interpretação do resultado
Utilização de desempenho da CPU	Qual a porcentagem da utilização	Estimar a exigência de	X = número de porcentagem	Quanto menor, melhor.

	da CPU ao realizar carregamento da listagem de dados?	utilização da CPU.	da utilização da CPU	

Fonte: Tradução do próprio Autor: Fonte: ISO/IEC 9126-3, 2003.

A métrica de utilização de recurso tem o objetivo de avaliar a utilização da CPU, ao realizar o carregamento da página responsável por listar os dados do banco de dados, sendo que essa métrica foi adaptada para atender o objeto de estudo a partir da métrica de utilização de memória.

Foram apresentados os critérios de comparação, selecionados para realização da análise comparativa entre os *frameworks* ORM. A seção 3.3 a seguir, apresenta o ambiente utilizado para execução das métricas.

3.3 Ambiente para execução dos testes

Segundo Vieira, Durães e Madeira (2005), para realizar uma comparação valida entre os *frameworks* de persistência de banco de dados, existe a necessidade de implementar uma aplicação real. Devido isso, foi implementado duas aplicações de teste, sendo que cada uma foi implementada utilizando um *framework* ORM selecionado. O SGDB utilizado como teste, consiste em ser o mesmo para ambas as aplicações, mantendo assim a homogeneidade do ambiente.

Para desenvolver a aplicação e aplicar as métricas selecionadas, foi utilizado o computador que possui as seguintes configurações exibidas na Tabela 6 a seguir.

Tabela 6 Configuração da máquina utilizada nos testes

Processador	Intel Core i5 2.4 GHz
Memória RAM	4 GB DDR 3
Sistema Operacional	Linux Mint 64 bits
Banco de Dados (SGDB)	MySql

Fonte: Próprio autor.

Umas das escolhas cruciais para desenvolvimento desse trabalho, foi a necessidade de identificar qual banco de dados seria utilizado para desenvolvimento das aplicações. Foram selecionados alguns critérios para essa escolha: Documentação, fóruns, artigos e livros, com objetivo de facilitar o trabalho desenvolvido. Uma das opções que mais se destacou com relação aos critérios definidos foi o MySql, além de possuir um amplo reconhecimento na comunidade de *software open source*, utiliza o modelo relacional para armazenar os dados.

A seção 3.4 a seguir, apresenta o modelo de entidade e relacionamento da aplicação desenvolvida em forma de diagrama para melhor compreensão da aplicação proposta.

3.4 Diagrama de entidade e relacionamento

O diagrama de entidade e relacionamento, representado pela Figura 7, é a representação gráfica do sistema proposto para o desenvolvimento das aplicações, onde é apresentado de forma detalhada a organização das tabelas e seus respectivos atributos, que possuem relacionamento. Esse diagrama foi elaborado para ser utilizado, na aplicação de teste desenvolvida utilizando os *frameworks* ORM selecionados.

Com a construção do diagrama, é possível identificar o funcionamento da aplicação proposta, onde consiste em possuir um responsável que possui um ou mais alunos que possuem uma ou mais matrícula, que é responsável por indicar em qual curso o aluno está matriculado e na respectiva campanha em que o aluno foi matriculado. A aplicação proposta pode ser considerada como um modelo simples de uma escola.

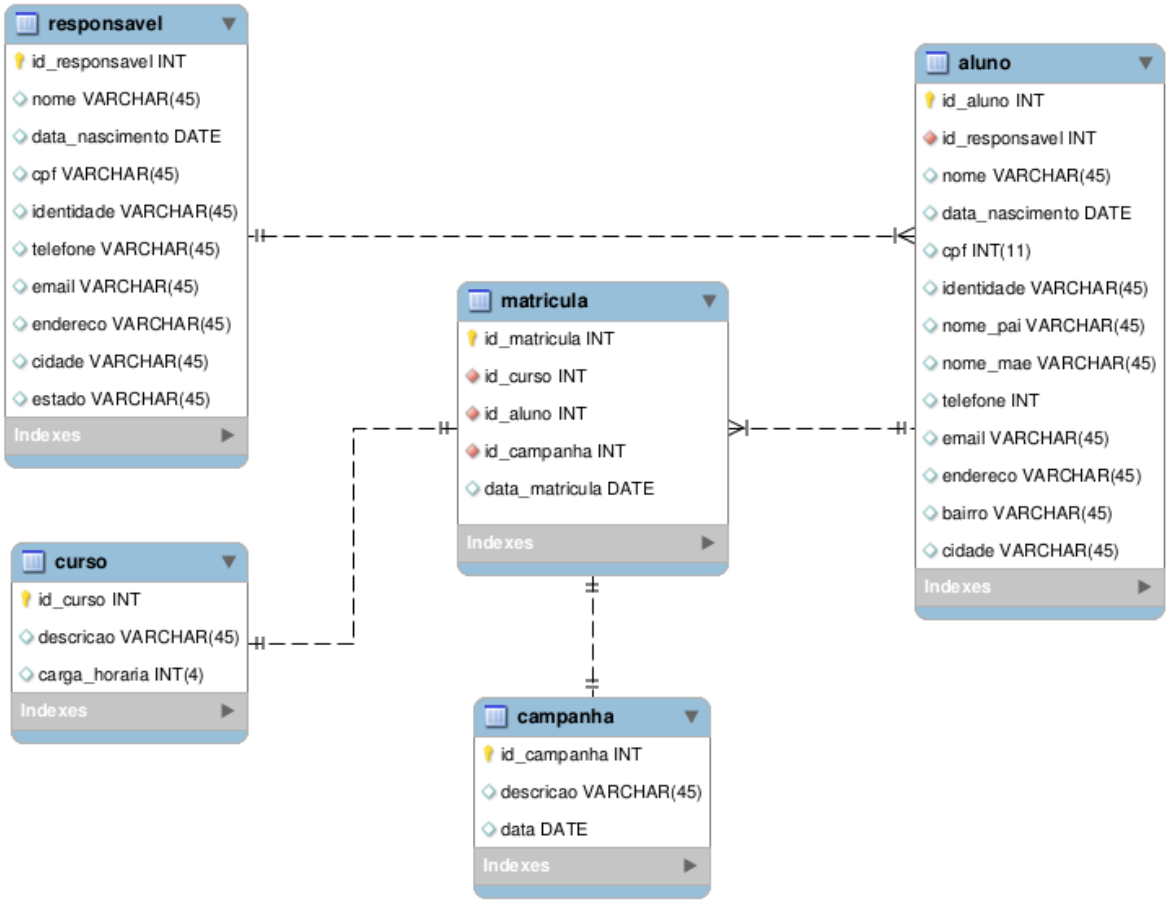

Figura 7 Diagrama de classes utilizado para realizado da aplicação.
Fonte: Próprio autor.

3.5 Execuções dos testes

Para representar um ambiente mais próximo de uma situação real, foram inseridos nas tabelas um conjunto significativo de dados, desde modo, atendendo o ambiente sugerido na metodologia. Foram inseridos nas tabelas aluno, matrícula e responsável um total de 50.000 registros com tamanhos variados, totalizando 150.000 registros, na tabela de curso e campanha foram inseridos um total de 30 registros, totalizando 60 registros de tamanhos variados.

Dentre as atividades elaboradas foi desenvolvido uma aplicação de teste, utilizando o modelo de banco de dados MySql, para realizar a utilização dos *frameworks* ORM. Foi realizado a preparação da aplicação para aplicar os testes, sendo executadas as rotinas de operações de persistência de dados e por último foi

realizado, a avaliação dos testes de acordo com as métricas e seus respectivos resultados.

Nesse capítulo foi apresentado a metodologia utilizada para desenvolver o trabalho, no capítulo 4, será apresentado os resultados obtidos com a aplicação das métricas selecionadas.

4. RESULTADOS

Os resultados que seguem, tem por objetivo apresentar os resultados dos dois *frameworks* ORM selecionados. Os dados coletados da análise, estão exibidos em gráficos e tabelas, onde foram apresentados os dados de cada métrica aplicada.

4.1 Eficiência

A eficiência mede o tempo gasto para os *frameworks* executaram determinadas tarefas que foram propostas, com o objeto de avaliar o tempo gasto para ambos concluírem tais tarefas.

4.1.1 Tempo de resposta

Para realizar a medição de tempo de resposta, foi proposta a realização de uma listagem de dados, envolvendo recuperar dados de todas as tabelas existente no banco de dados modelado. Deste modo, pode-se analisar o comportamento dos *frameworks* quando realizavam listagem de mil, dez mil, vinte e cinco mil e cinquenta mil registros.

Para medir o tempo gasto dos *frameworks*, foi medido o tempo inicial antes de realizar a chamada da função, que listava os dados, e o tempo posterior do retorno da função. A medida de tempo foi realizada em milissegundo, devido a rapidez em se listar os registros.

Os testes realizados, foram repetidos cem vezes, para se ter uma média da diferença de tempo gasto, para listar os respectivos registros. Com esse número de repetições, pode-se definir qual *framework* apresentou melhor eficiência no teste realizado.

O Gráfico 1, apresenta o tempo gasto para os *frameworks* realizarem a recuperação de mil registros inseridos no banco de dados, onde foi realizado relacionamento com todas as tabelas existentes, tornando a consulta executada complexa.

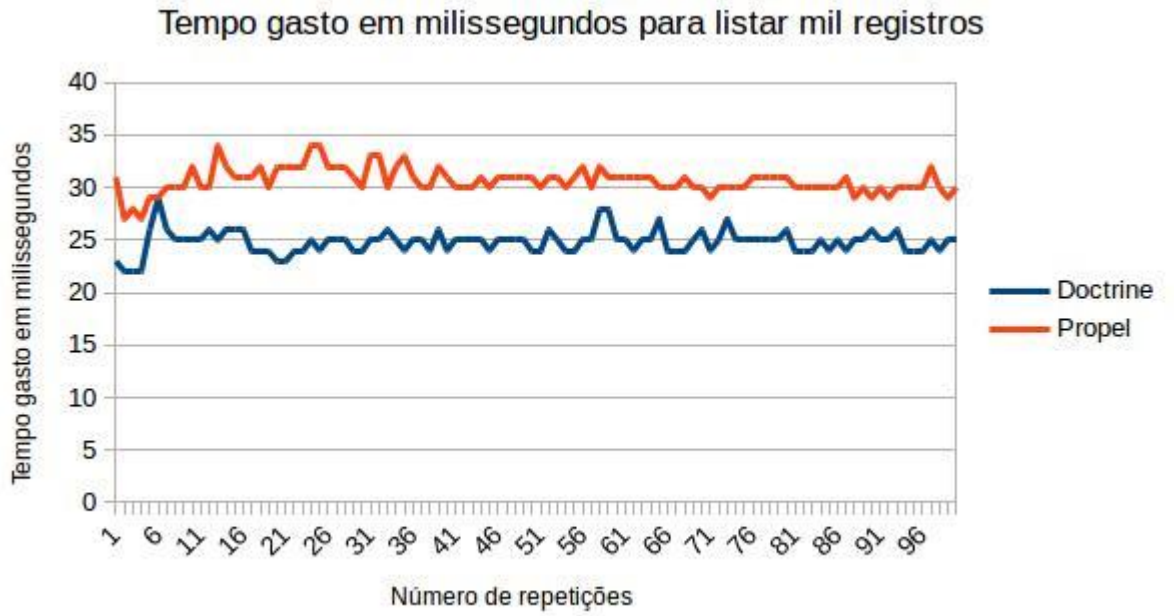

Gráfico 1 Teste de eficiência listando mil registros.

Fonte: Próprio autor.

No primeiro teste com listagem de mil registros, o Doctrine apresentou uma média de 25 milissegundos para realizar a listagem dos registros, ao contrário do Propel, que teve uma média de 30,5 milissegundos para realizar a mesma listagem. Deste modo, houve uma diferença de 5,5 milissegundos entre os dois *frameworks,* sendo que o Doctrine apresentou melhor eficiência.

O Gráfico 2, apresenta o tempo gasto para os *frameworks* realizarem a recuperação de dez mil registros inseridos no banco de dados, onde foi realizado

relacionamento com todas as tabelas existentes, tornando a consulta executada complexa.

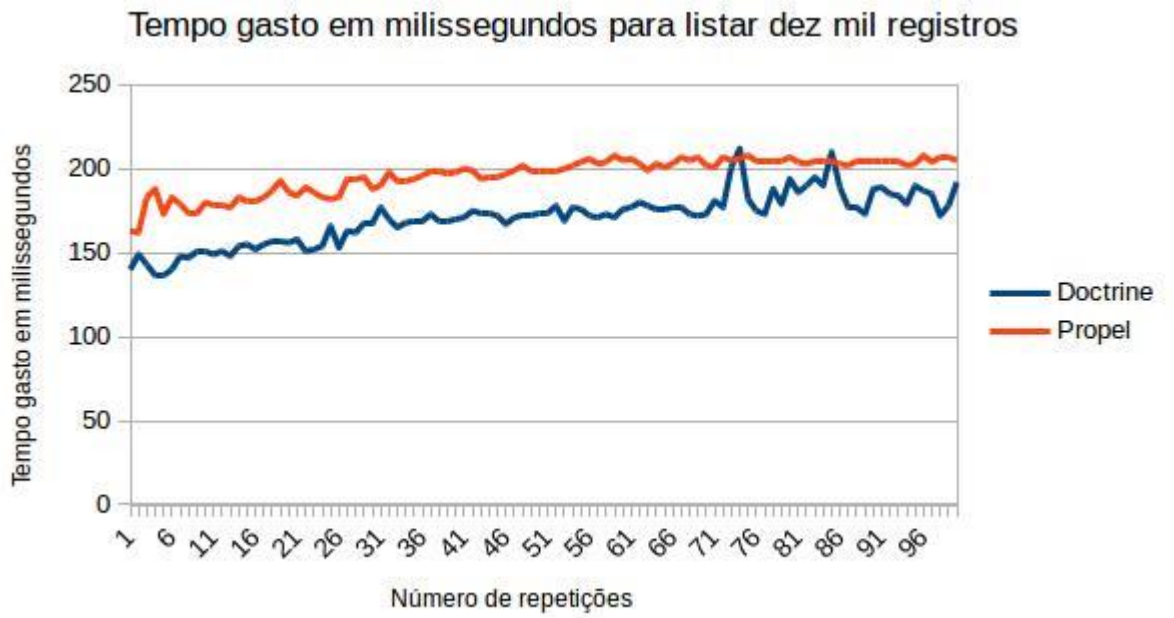

Gráfico 2 Teste de eficiência listando dez mil registros.
Fonte: Próprio autor.

O segundo teste com listagem de dez mil registros, o Doctrine apresentou uma média de 172,9 milissegundos para realizar a listagem dos registros, ao contrário do Propel, que teve uma média de 199 milissegundos para realizar a mesma listagem. Deste modo, houve uma diferença de 26,5 milissegundos entre os dois *frameworks,* sendo que o Doctrine apresentou melhor eficiência.

O Gráfico 3, apresenta o tempo gasto para os *frameworks* realizarem a recuperação de vinte e cinco mil registros inseridos no banco de dados, onde foi realizado relacionamento com todas as tabelas existentes, tornando a consulta executada complexa.

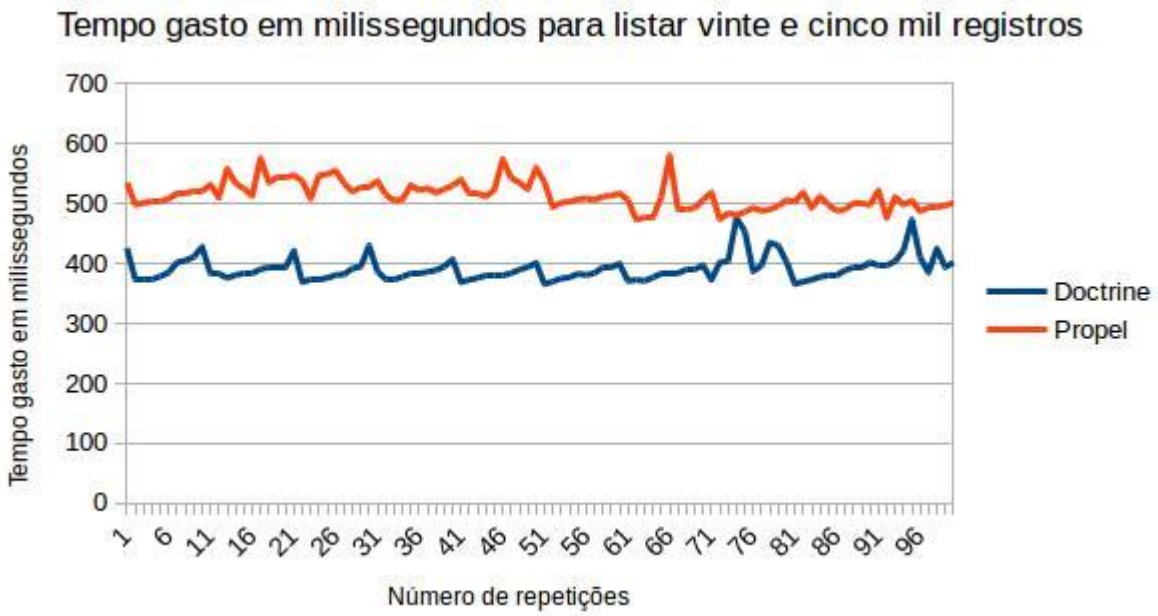

Gráfico 3 Teste de eficiência listando vinte e cinco mil registros.

Fonte: Próprio autor

O terceiro teste com listagem de vinte e cinco mil registros, o Doctrine apresentou uma média de 385 milissegundos para realizar a listagem dos registros, ao contrário do Propel, que teve uma média de 511,5 milissegundos para realizar a mesma listagem. Deste modo, houve uma diferença de 126,5 milissegundos entre os dois *frameworks,* sendo que o Doctrine apresentou melhor eficiência.

O Gráfico 4, apresenta o tempo gasto para os *frameworks* realizarem a recuperação de cinquenta mil registros inseridos no banco de dados, onde foi realizado relacionamento com todas as tabelas existentes, tornando a consulta executada complexa.

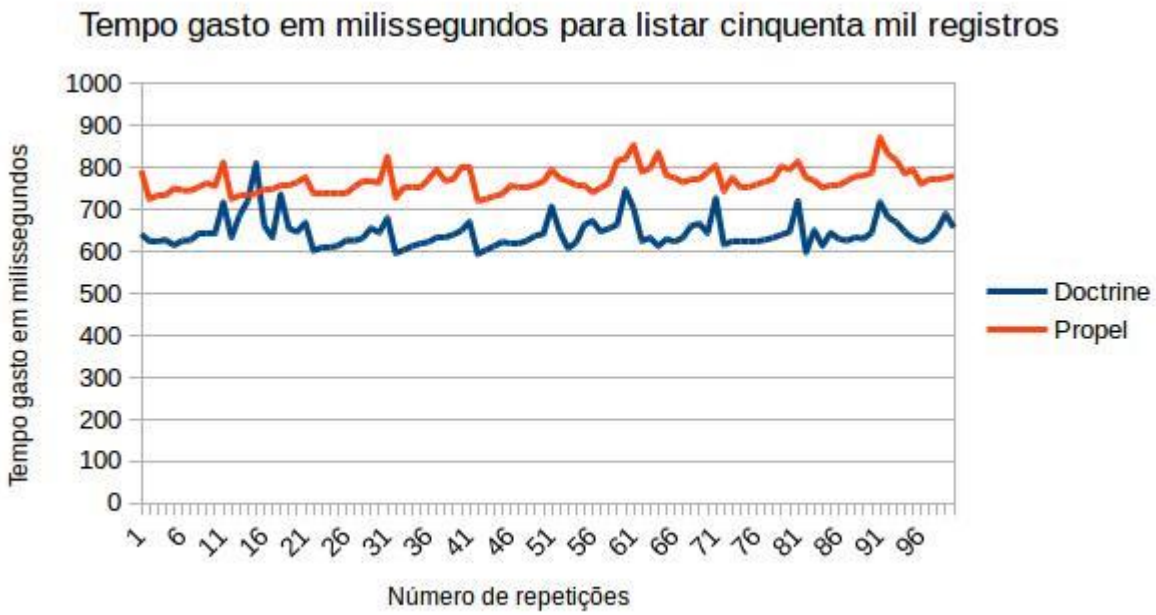

Gráfico 4 Teste de eficiência listando cinquenta mil registros.
Fonte: Próprio autor.

O quarto teste com listagem de cinquenta mil registros, o Doctrine apresentou uma média de 634 milissegundos para realizar a listagem dos registros, ao contrário do Propel, que teve uma média de 765 milissegundos para realizar a mesma listagem. Deste modo, houve uma diferença de 131 milissegundos entre os dois *frameworks,* sendo que o Doctrine apresentou melhor eficiência.

4.1.2 Tempo de processamento

Para realizar a medição do tempo de processamento, foi proposto medir quantos registros eram inseridos e atualizados em um segundo. Deste modo, pode-se analisar o comportamento dos *frameworks* quando realizavam inserções e atualizações de registros. Para realizar esse teste, utilizou-se a tabela aluno, sendo que a mesma possui um relacionamento com tabela responsável.

Para analisar o tempo gasto pelos *frameworks*, foi medido o tempo inicial, e inserido um laço onde estava contido a função responsável por inserir o registro em uma tabela do banco de dados, após a inserção, era contabilizado um registro

inserido, e era medido o tempo, se a diferença entre o tempo inicial e o final fosse de um segundo, o laço era interrompido. O mesmo procedimento foi realizado com atualização de registro. Com isso, pode-se analisar quando registros eram inseridos e atualizados em um segundo.

Os testes realizados, foram repetidos cem vezes, para se ter uma média da diferença de tempo gasto para inserir e atualizar dados do banco de dados. Com esse número de repetições, pode-se definir qual *framework* apresentou melhor eficiência no teste realizado.

O Gráfico 5, apresenta o número de inserções que cada *framework* realizou em um segundo.

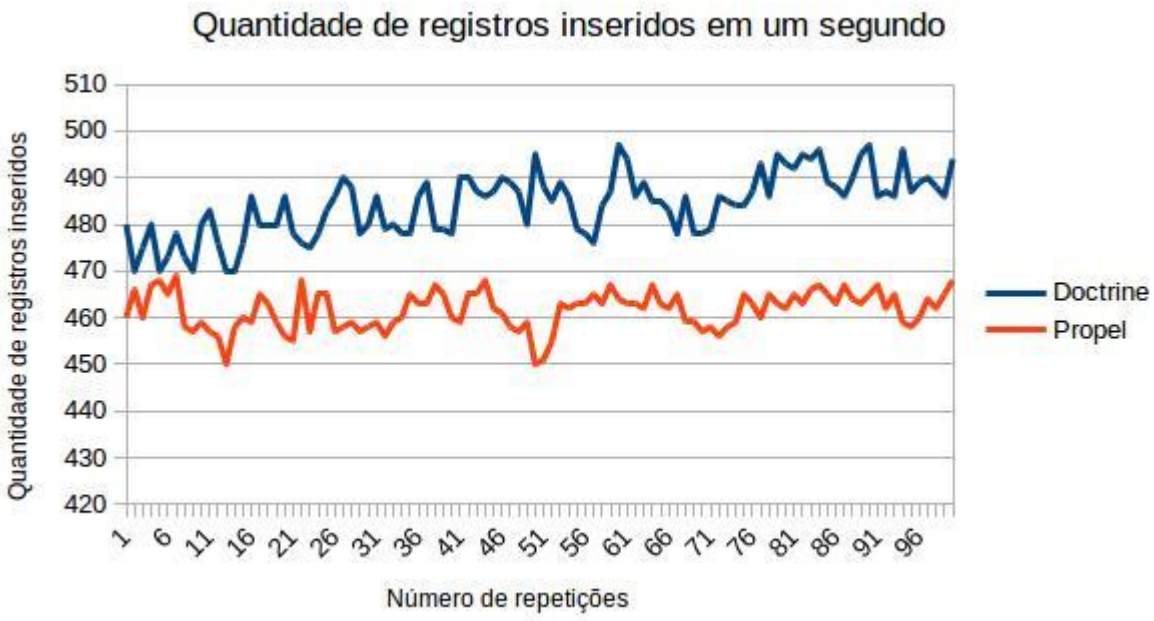

Gráfico 5 Número máximo de registro inseridos em um segundo.
Fonte: Próprio autor.

No teste de inserção, o Doctrine apresentou uma média de 486 registros inseridos com um segundo, ao contrário do Propel que apresentou uma média de 462 registros inseridos em um segundo. Deste modo, houve uma diferença de 24 registros

inseridos entre os dois *frameworks,* sendo que o Doctrine apresentou melhor eficiência.

O Gráfico 6, apresenta o número de atualizações que cada *framework* realizou em um segundo.

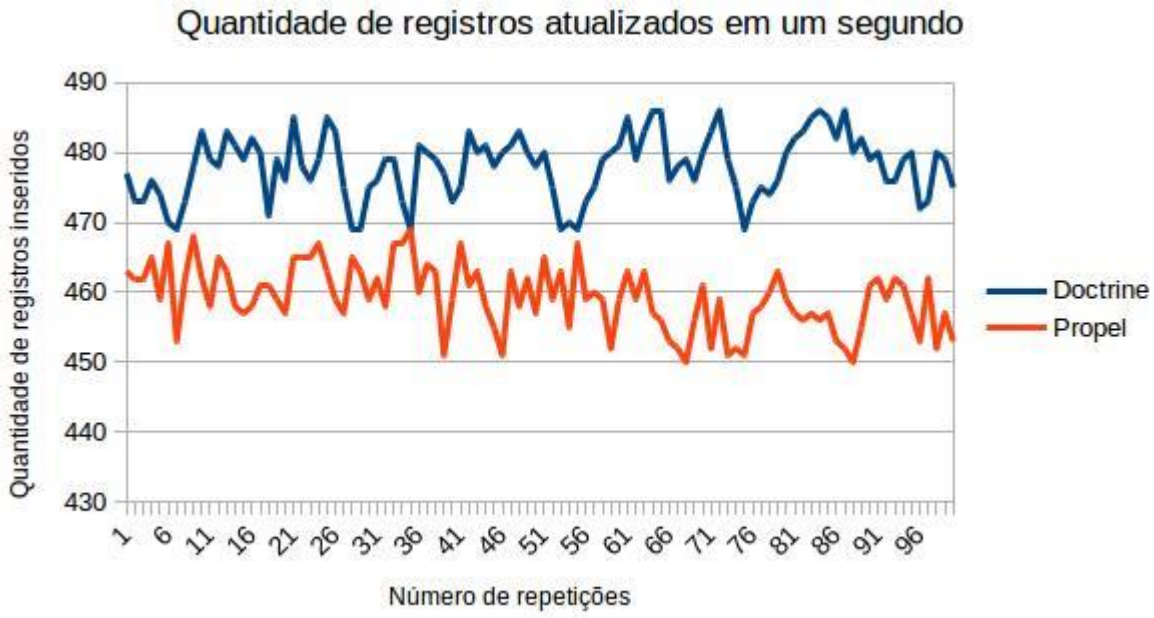

Gráfico 6 Número máximo de registro atualizados em um segundo.
Fonte: Próprio autor.

No teste de atualização, o Doctrine apresentou uma média de 479 registros atualizados com um segundo, ao contrário do Propel que apresentou uma média de 459 registros atualizados em um segundo. Deste modo, houve uma diferença de 20 registros atualizados entre os dois *frameworks,* sendo que o Doctrine apresentou melhor eficiência.

4.1.3 Tempo de retorno

Para realizar a medição do tempo de retorno, foi proposto medir quantos milissegundos cada *framework* utilizou para realizar uma inserção, busca, atualização e exclusão de um registro na tabela aluno.

Para medir o tempo gasto dos *frameworks*, foi medido o tempo inicial antes de realizar a chamada das funções de inserção, busca, atualização e exclusão, assim que era realizado a exclusão, era medido o tempo final.

Os testes realizados, foram repetidos cem vezes, para se ter uma média da diferença de tempo gasto para realizar as quatro funções.

O Gráfico 7, apresenta o tempo gasto para os *frameworks* realizarem as quatro funções estabelecidas:

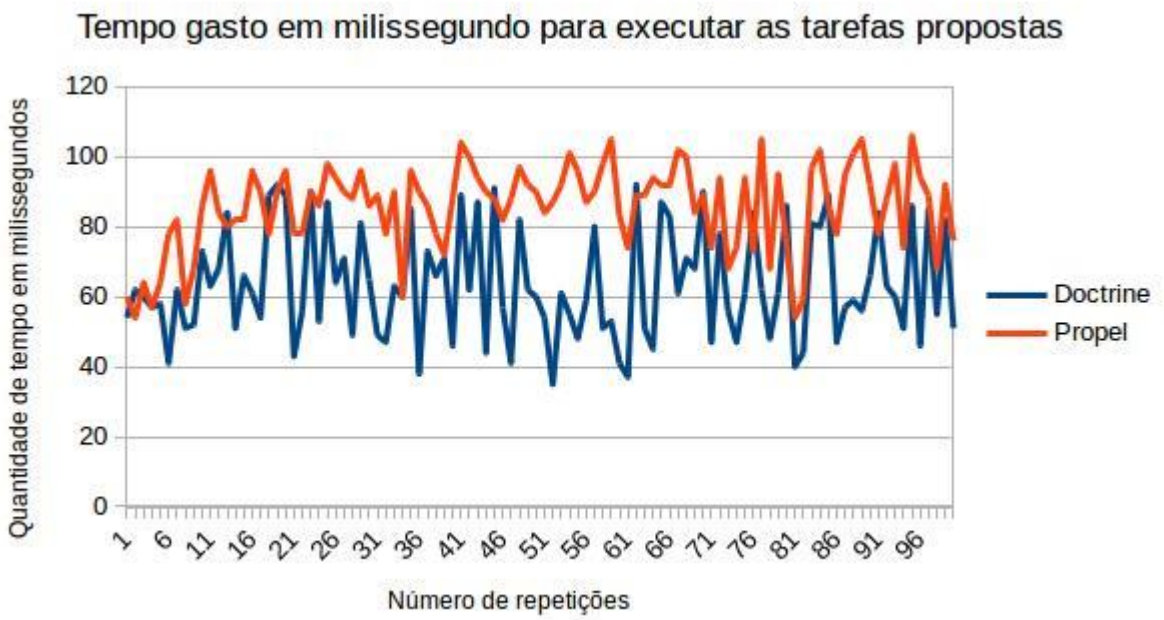

Gráfico 7 Tempo gasto para executar determinadas tarefas.
Fonte: Próprio autor.

No teste realizado, o Doctrine apresentou uma média de 61 milissegundos para realizar os quatro procedimentos especificados, ao contrário do Propel que apresentou uma média de 88,5 milissegundos para realizar os quatro procedimentos. Deste modo, houve uma diferença de 27,5 milissegundos entre os dois *frameworks,* sendo que o Doctrine apresentou melhor eficiência.

4.1.4 Tempo gasto para realizar importação de dados

A métrica de tempo gasto para realizar importação de dados, tem como finalidade medir o tempo gasto para os *frameworks* ORM realizarem a importação de dados para a tabela aluno.

Para analisar o tempo gasto, foi necessário criar um script para executar o comando de importação de cada *framework,* nesse script é medido o tempo inicial, e o tempo final após a importação.

O Gráfico 8 a seguir, apresenta o tempo em milissegundo gasto para realizar a importação de cinquenta mil registro.

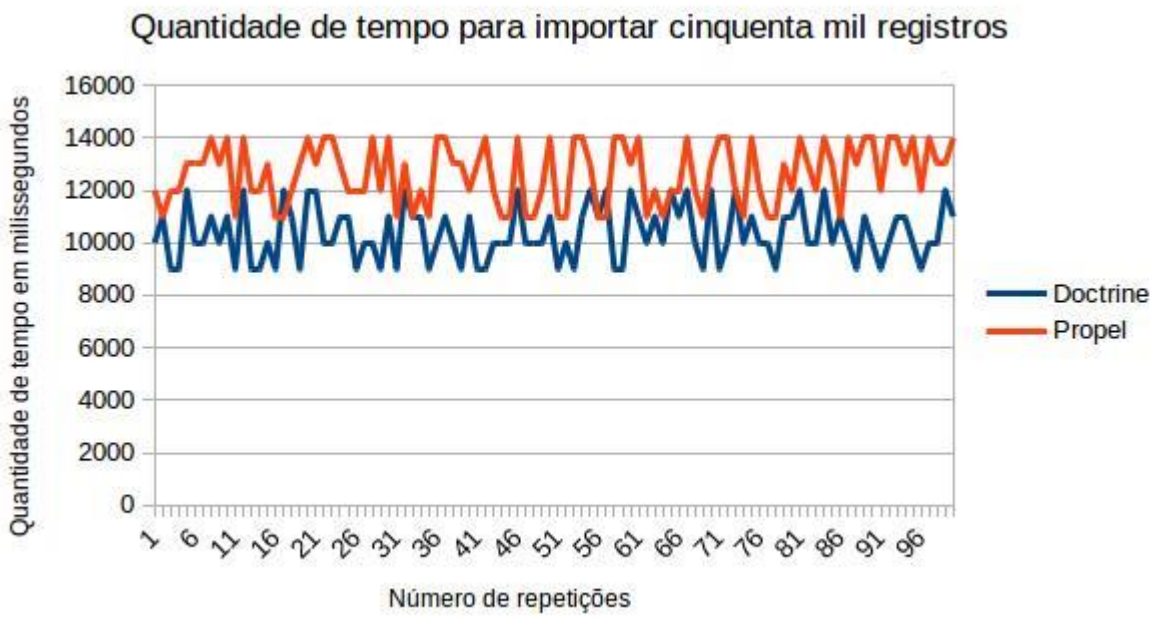

Gráfico 8 Tempo gasto para realizar importação de cinquenta mil registros.
Fonte: Próprio autor.

No teste realizado, o Doctrine apresentou uma média de 10000 milissegundos, que são equivalentes a 10 segundos para realizar a importação de cinquenta mil registros, ao contrário do Propel que apresentou uma média de 13000 milissegundos, que são equivalentes a 13 segundos a mesma importação. Deste modo, houve uma

diferença de 3000 milissegundos, que são equivalentes a 3 segundos entre os dois *frameworks,* sendo que o Doctrine apresentou melhor eficiência.

O Gráfico 9 a seguir, apresenta o tempo em milissegundo gasto para realizar a importação de toda a base de dados.

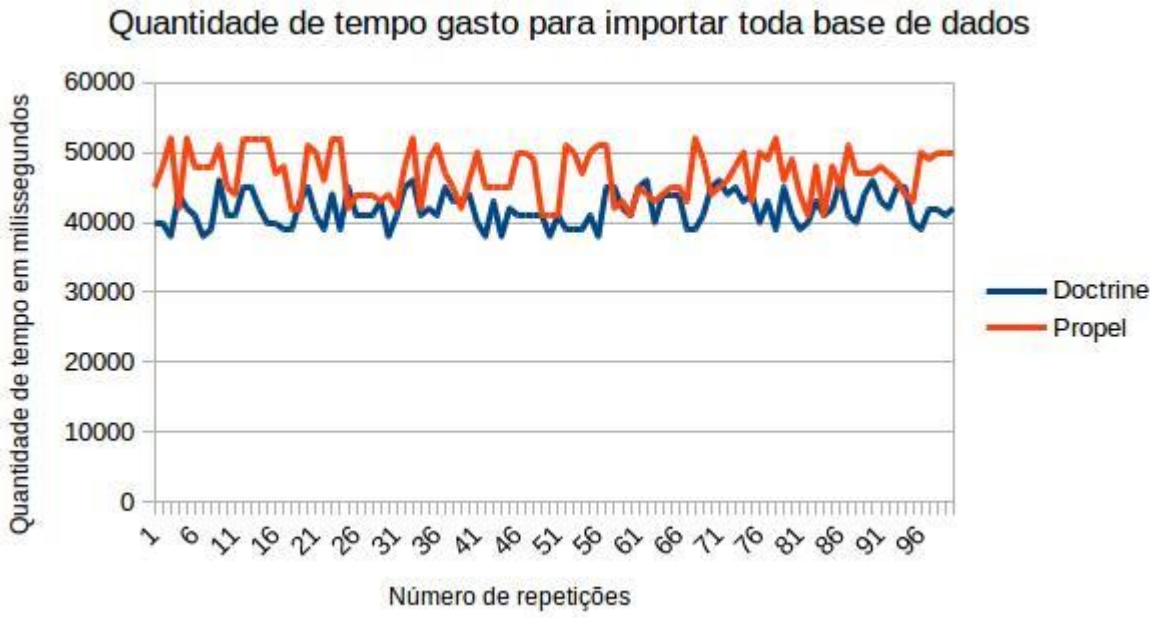

Gráfico 9 Tempo gasto para realizar importação de toda base de dados utilizada.
Fonte: Próprio autor.

No teste realizado, o Doctrine apresentou uma média de 41000 milissegundos, que são equivalentes a 41 segundos para realizar a importação de cinquenta mil registros, ao contrário do Propel que apresentou uma média de 47000 milissegundos, que são equivalentes a 47 segundos a mesma importação. Deste modo, houve uma diferença de 6000 milissegundos, que são equivalentes a 6 segundos entre os dois *frameworks,* sendo que o Doctrine apresentou melhor eficiência.

4.2 Portabilidade

A portabilidade mede o quão os *frameworks* são compatíveis com diferentes ambientes especificados.

4.2.1 Adaptabilidade das estruturas de dados

Para realizar a medição de adaptabilidade de estruturas de dados, foi realizada a alteração no SGDB utilizado, passando a se utilizar o PostgreSQL, podendo assim contabilizar o número de funções que tiveram que ser modificadas, para atender as modificações.

A Tabela 7, apresenta o número de funções que foram alteradas na aplicação para atender as modificações de estruturas de dados.

Tabela 7 Número de funções alteras em cada framework

Frameworks	**Modificações necessárias**
Doctrine	1
Propel	1

Fonte: Próprio autor.

Ambos os *frameworks* analisados precisou alterar a configuração do arquivo responsável por realizar a conexão em caso de alterações de estrutura de dados. Segundo suas documentações e o teste realizado, foi necessário alterar apenas qual o modelo do banco de dados que estava utilizando. Entretanto ouve a necessidade de gerar novamente as classes responsáveis por comunicar com o novo modelo de banco de dados. Com as devidas alterações de estrutura de dados e do arquivo de configuração, a aplicação de teste ainda operou de maneira estável sem apresentar problemas.

4.2.2 Compatibilidade com estruturas de dados diferentes

Para realizar a métrica de compatibilidade com estruturas de dados diferentes, foi realizado uma análise nas documentações de ambos os *frameworks* e foram levantados os modelos de bancos de dados suportados por cada *framework.*

A Tabela 8, apresenta o número de banco de dados suportados pelos *frameworks* de acordo com suas documentações.

Tabela 8 Número de banco de dados suportado

Frameworks	**Total de banco de dados suportados**
Doctrine	6
Propel	5

Fonte: Próprio autor.

Desenvolver uma aplicação com suporte para N banco de dados (DB) é relevante para a escolha de um *framework*, caso seja necessário migrar de DB, os *frameworks* ainda conseguem trabalhar de maneira adequada, evitando assim manutenções trabalhosas. O Doctrine oferece suporte para os seguintes banco de dados:

- MySQL;
- Oracle;
- PostgreSQL;
- SQLite;
- SAP SQL Anywhere;
- Microsoft SQL Server.

O Propel por sua vez, oferece suporte para os seguintes banco de dados:

- MySQL;
- PostgreSQL;
- SQLite;
- Microsoft SQL Server;
- Oracle.

Ambos *frameworks* oferecem suporte para os principais modelos de banco de dados do mercado, mais o doctrine tem a vantagem de oferecer um modelo a mais, sendo ele: SAP SQL Anywhere.

4.3 Instabilidade

Métrica de instabilidade tem por objetivo medir o número de possibilidades de instalação que cada *framework* oferece, de acordo com sua documentação.

4.3.1 Flexibilidade de instalação

Para realizar a aplicação da métrica de flexibilidade de instalação, foi realizado uma pesquisa na documentação de cada *framework* para fazer o levantamento do número de possibilidades de instalação.

A Tabela 9, apresenta o número de flexibilidade de instalação dos *frameworks* analisados

Tabela 9 Número de flexibilidade de instalação

Frameworks	**Quantidade de formas de instalação**
Doctrine	1
Propel	3

Fonte: Próprio autor.

De acordo com a documentação dos *frameworks* selecionados ambos podem ser instalados através da utilização do composer. O composer é uma ferramenta que gerencia as dependências em PHP, ele permite que seja declarado as bibliotecas dependentes em seu projeto para serem instaladas.

O propel oferece a vantagem de ser adquirido pelo gitHub, que é um serviço de compartilhamento para projetos, ou por download de maneira normal. O doctrine também pode ser adquirido por meio da utilização do gitHub, mais sua documentação não especifica essa utilização, por esse motivo, não foi considerado.

4.4 Utilização de recursos

A métrica de utilização de recursos, tem a finalidade de medir o consumo da utilização da CPU, ao realizar o carregamento da aplicação listando uma determinada quantidade de dados.

4.4.1 Utilização de desempenho da CPU

Para realizar a aplicação da métrica de utilização de desempenho da CPU, foi necessário a utilização do programa Htop. Esse programa realiza a listagem de todos os processos que estão sendo executados pelo processador, sendo possível mensurar o gasto de utilização de recursos da CPU.

Para medir a porcentagem da utilização da CPU, a cada teste a aplicação era encerrada e carregada novamente realizando a listagem dos registros. No momento do carregamento da aplicação era analisado a utilização da CPU exibido pelo Htop em porcentagem.

O Gráfico 10, apresenta em porcentagem a utilização da CPU para realizar o carregamento da aplicação com listagem de mil registros.

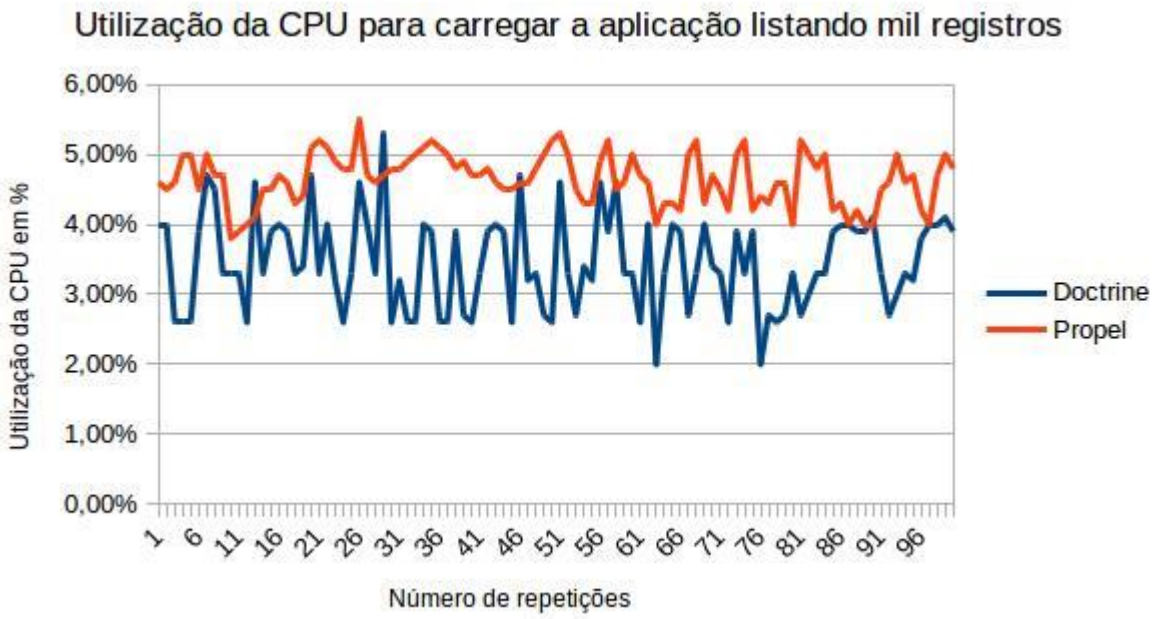

Gráfico 10 Utilização da CPU ao carregar aplicação listando mil registros.
Fonte: Próprio autor.

No teste realizado, a aplicação desenvolvida com o Doctrine apresentou uma média de 3,45% de utilização da CPU ao realizar a listagem de mil registros, ao contrário do Propel que apresentou uma média de 4,65% de utilização da CPU. Deste modo, houve uma diferença de 1,2%, sendo que a aplicação desenvolvida com Doctrine apresentou menor utilização da CPU.

O Gráfico 11, apresenta em porcentagem a utilização da CPU para realizar o carregamento da aplicação com listagem de dez mil registros.

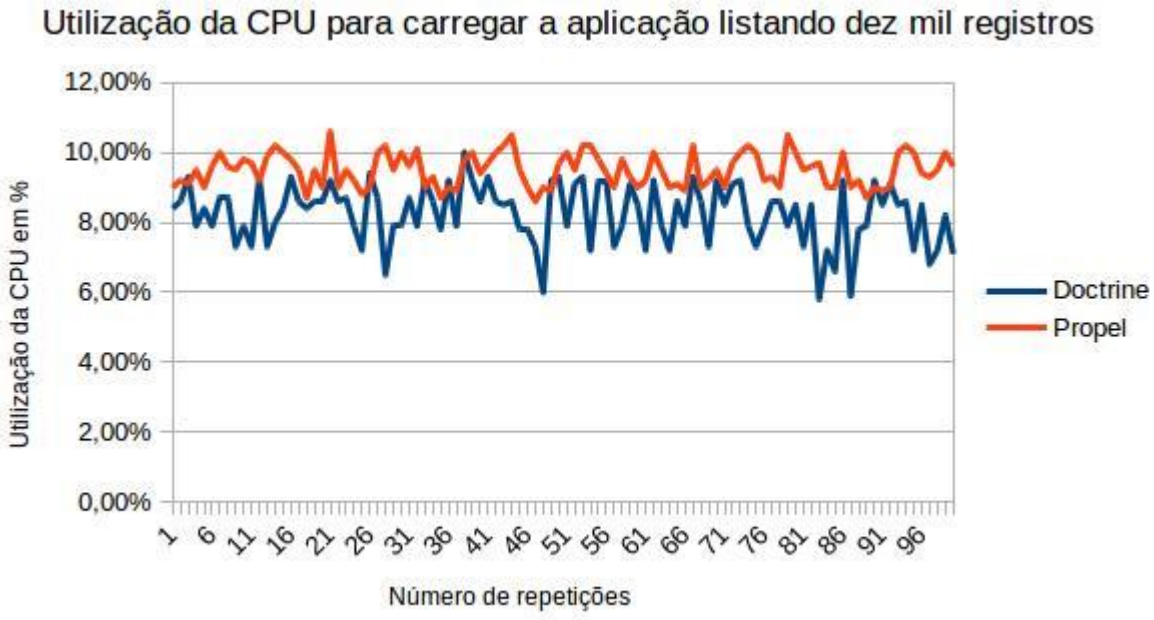

Gráfico 11 Utilização da CPU ao carregar aplicação listando dez mil registros.
Fonte: Próprio autor.

No teste realizado, a aplicação desenvolvida com o Doctrine apresentou uma média de 8,24% de utilização da CPU ao realizar a listagem de dez mil registros, ao contrário do Propel que apresentou uma média de 9,5% de utilização da CPU. Deste modo, houve uma diferença de 1,26%, sendo que a aplicação desenvolvida com Doctrine apresentou menor utilização da CPU.

O Gráfico 12, apresenta em porcentagem a utilização da CPU para realizar o carregamento da aplicação com listagem de vinte e cinco mil registros.

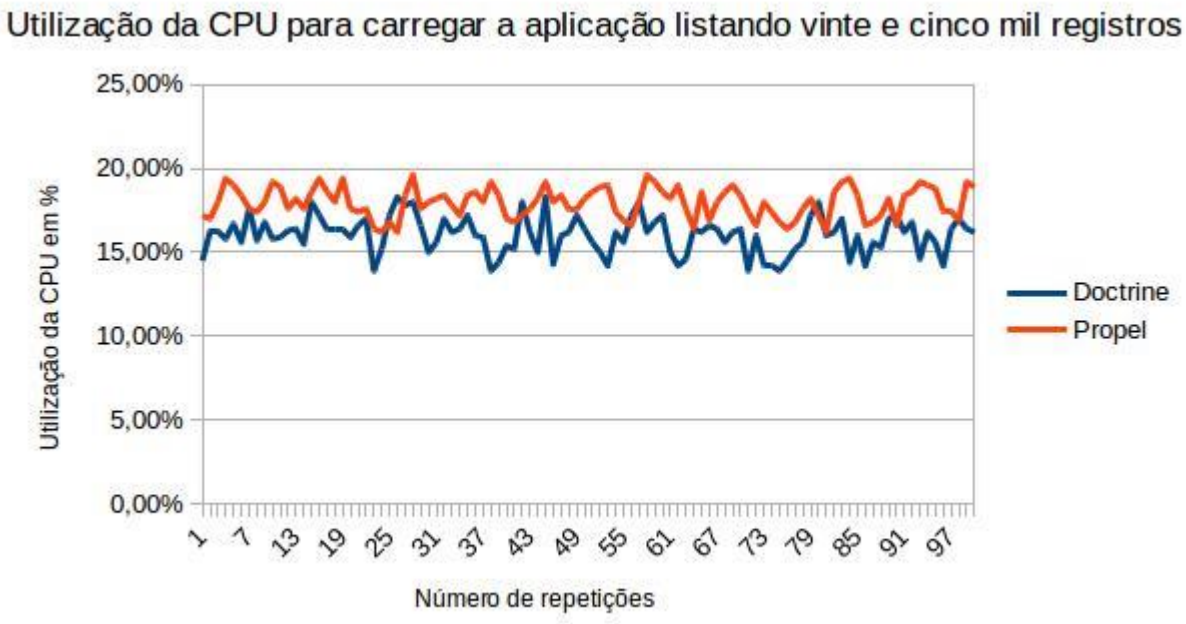

Gráfico 12 Utilização da CPU ao carregar aplicação listando vinte e cinco mil registros.
Fonte: Próprio autor.

No teste realizado, a aplicação desenvolvida com o Doctrine apresentou uma média de 16,03% de utilização da CPU ao realizar a listagem de vinte e cindo mil registros, ao contrário do Propel que apresentou uma média de 17,96% de utilização da CPU. Deste modo, houve uma diferença de 1,93%, sendo que a aplicação desenvolvida com Doctrine apresentou menor utilização da CPU.

O Gráfico 13, apresenta em porcentagem a utilização da CPU para realizar o carregamento da aplicação com listagem de cinquenta mil registros.

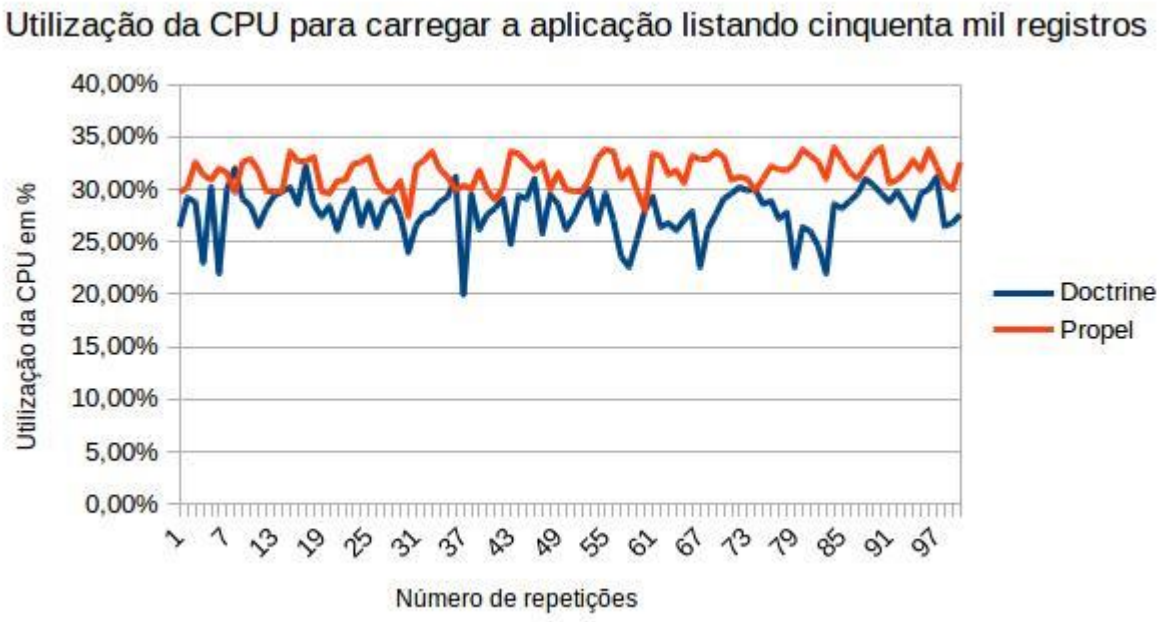

Gráfico 13 Utilização da CPU ao carregar aplicação listando cinquenta mil registros.
Fonte: Próprio autor.

No teste realizado, a aplicação desenvolvida com o Doctrine apresentou uma média de 27,85% de utilização da CPU ao realizar a listagem de cinquenta mil registros, ao contrário do Propel que apresentou uma média de 31,6% de utilização da CPU. Deste modo, houve uma diferença de 3,75%, sendo que a aplicação desenvolvida com Doctrine apresentou menor utilização da CPU.

4.5 Resultados das métricas aplicadas

No total foram aplicados quatro tipos de métricas. Na métrica de eficiência o Doctrine se apresentou mais eficiente do que o Propel em todos critérios avaliados, o mesmo ocorreu na métrica de utilização de recursos. Na métrica de instabilidade, onde foi analisado a flexibilidade de instalação, o Propel levou a vantagem por oferecer mais opções de instalação de acordo com sua documentação. Na métrica de portabilidade foi proposta a métrica de adaptabilidade das estruturas de dados, onde ambos os *frameworks* necessitam de alterar o mesmo número de funções caso seja alterado o modelo de banco de dados, deste modo ocorreu um empate, mas na

métrica de compatibilidade com outras estruturas de dados, que é referente a métrica de portabilidade o Doctrine levou vantagem, pois apresentou um modelo a mais de banco de dados suportado.

O Gráfico 14, a seguir apresenta os resultados das métricas aplicas.

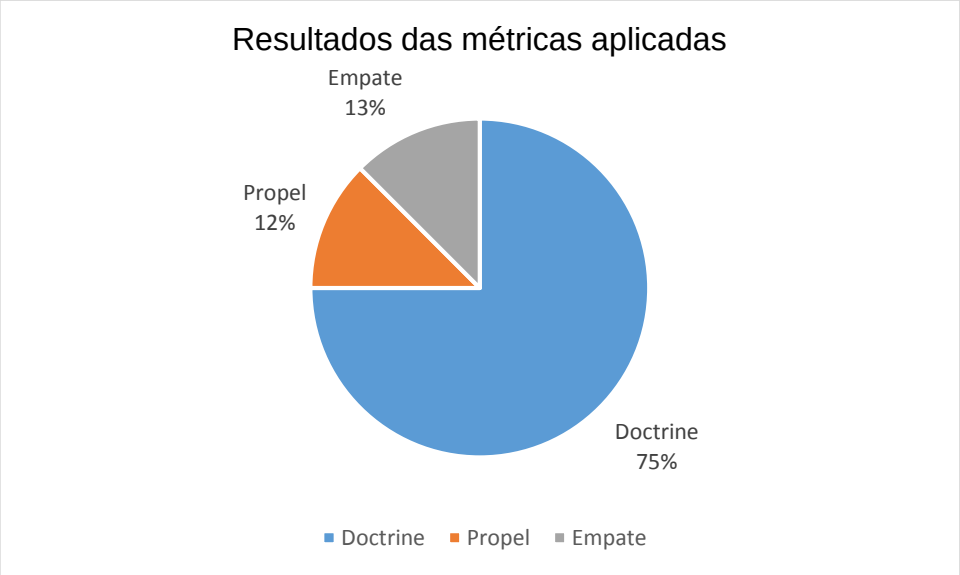

Gráfico 14 Resultados das métricas aplicadas.

Fonte: Próprio autor.

Contudo o Doctrine sobressaiu em relação as métricas aplicadas, sendo mais eficiente em relação ao Propel. No capítulo 5 a seguir, será apresentado as conclusões sobre o trabalho realizado.

5. CONCLUSÃO

No trabalho apresentado foram esclarecidos alguns conceitos fundamentais de persistência de dados, que foram fundamentais para compreender a importância dos *frameworks* de mapeamento de banco de dados, que são responsáveis por realizar a comunicação entre a aplicação e o banco de dados, diminuindo a impedância entre o modelo orientado a objetos utilizado no desenvolvimento da aplicação e o modelo entidade relacionamento do banco de dados.

Outra questão importante apresentada, foi a utilização de medidas de qualidade de *software*, apresentadas por meio da utilização da norma ISO/IEC 9126, que tem a finalidade de avaliar e qualificar o produto de *software*. Sendo relevante para realizar o estudo comparativo entre os *frameworks* Doctrine e Propel.

Com a utilização da norma ISO/IEC 9126, foram aplicadas métricas que possibilitaram fazer uma inferência de resultados para esta pesquisa. Com relação a métrica de eficiência e utilização de recursos, é possível concluir que de acordo com o número de registros listados pelos *frameworks* ocorria um aumento na utilização da CPU. De acordo com essas duas métricas, o Doctrine se apresentou melhor em relação ao Propel, devido conseguir realizar a listagem em menor tempo e com menor utilização da CPU.

De acordo com os resultados da métrica de portabilidade e instabilidade é possível realizar outra inferência a partir dos resultados das métricas, sendo que, dentre os resultados obtidos ambos os *frameworks* oferecem uma flexibilidade na instalação e suporte aos principais bancos de dados do mercado.

Outra inferência de grande relevância para este trabalho é a principal vantagem na utilização dos *frameworks* ORM, que consistem em facilitar o desenvolvimento, melhoria na manutenibilidade e redução de código escrito para realizar a persistência de dados. Sendo possível observar essas vantagens na aplicação da métrica de portabilidade, na qual analisou a compatibilidade dos *frameworks* com outros modelos de banco de dados, e em caso de alteração do modelo de banco de dados utilizado apresentou o número de alterações necessárias para o funcionamento correto da aplicação. Em ambas as métricas os *frameworks* se coincidem nos resultados obtidos.

Contudo, mesmo os *frameworks* analisados se coincidirem em alguns critérios analisados, o Doctrine se apresentou mais eficiente em relação ao Propel, levando vantagens principalmente nas métricas de eficiência e de utilização de recursos. Vale ressaltar também, que o Doctrine apresenta uma gama maior de conteúdos para auxiliar em dúvidas, esses conteúdos são desde artigos, fóruns, blogs e tutoriais sobre sua utilização, o Propel por sua vez possui uma quantidade bem menor, sendo um dos problemas encontrados para o desenvolvimento desse trabalho.

6. TRABALHOS FUTURO

Para trabalhos futuros, é interessante aplicar um questionário com o objetivo de avaliar as características de qualidade dos *frameworks* com relação ao ponto de vista dos desenvolvedores e empresas que fazem a sua utilização, esse questionário teria a finalidade de aplicar as demais métricas da norma ISO/IEC 9126.

Outra proposta de trabalho, seria a aplicação de um dos *frameworks* ORM apresentado neste trabalho, em uma empresa de desenvolvimento que não faz utilização de *frameworks* de mapeamento de banco de dados. Após sua aplicação, mensurar os ganhos que a organização obteve com a implementação do *framework* escolhido.

REFERÊNCIAS

ALMEIDA, Vânia Paula de. **Estrátegias Cognitivas para o Aumento da Qualidade do Hiperdocumento para Educação a Distância**. 170 folhas. Curso de Ciência da Computação, Universidade Federal de São Carlos, São Carlos, 2005.

BAUER, C.; KING, G. **Java Persistence com Hibernate**, 1ª Edição, Editora Ciência Moderna Ltda.,2007.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML** . Rio de Janeiro: Campus, 2002.

CÉSAR, Fábio. **Propel Um framework para ORM em PHP**. Php Magazine A Sua Revista Digital de Php. 2ª Edição, Março 2007.

CORDEIRO, Jader dos Santos Teles. **ESTUDO COMPARATIVO ENTRE OS FRAMEWORKS DE MAPEAMENTO OBJETO-RELACIONAL HIBERNATE E TOPLINK.** 2011. 55 f. Curso de Ciência da Computação, Universidade Estadual de Maringá, Maringá, 2011.

DALL'OGLIO, P. **PHP: programando com orientação a objetos**. 2. ed. São Paulo: Novatec, 2009.

DATE, C.J. **Introdução a Sistemas de Banco de Dados**. 7. ed. Rio de Janeiro: Campus, 2000.

DATE, C.J. **Introdução a Sistemas de Banco de Dados**. 8. ed. Rio de Janeiro: Campus, 2004.

DOCTRINE CORE TEAM. **About Doctrine ORM**. Disponível em: http://www.doctrine-project.org/. Acesso em: 28 Set, 2015.

DOEDERLEIN, Osvaldo Pinali. **Dados e Mapeamento Explorando técnicas e tecnologias para persistência de dados**. JAVA magazine, Ed.42, ano.V, p. 22-30, 2006.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 6. ed. Addison-Wesley, 2010.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de bancos de dados**. 4. ed. São Paulo: Pearson, 2005.

GAMMA, Erich et al**. Padrões de projeto: soluções reutilizáveis de *software* orientado a objetos**. Porto Alegre: Bookman, 2002.

GUERRA, Ana Cervigni; COLOMBO, Regina Maria Thienne. **Qualidade de Produto de *Software***. 2009. 259 p.

LEITE, Julio Cesar Sampaio do Prado. **Gerenciando a Qualidade de Software com Base em Requisitos**. Rio de janeiro: PUC RJ. 2010.

LUCKOW, D.H; MELO, A. A. de. **Programação Java para a Web**. São Paulo: Novatec, 2010.

MARQUES, Lázaro Henrique C.; R. JÚNIOR, Dilson José L. **Aplicação do ORM Doctrine para abstração de banco de dados no desenvolvimento de *software* em PHP na UNASUS/ UFMA**. São Luís, 2014.

MINETTO, E. L. **Frameworks para Desenvolvimento em PHP**. São Paulo: Novatec (2007).

NBR ISO/IEC 9126-1:2003. **Tecnologia de informação: Engenharia de *software* – Qualidade de produto Parte 1: Modelo de qualidade**. Esta norma cancela e substitui a NBR 13596. Julho 2003.

Pothu, S. A **Comparative Analysis of Object-relational mappings for Java**, MSc. Thesis, University of Applied Sciences at Braunschweig/Wolfenbuettel, 2008

PRESSMAN, Roger S**. Engenharia de *software*, uma abordagem profissional**. 7ª ed. New York: The McGraw-Hill Companies, 2011.

PRESSMAN, Roger S., **Engenharia de *Software*** - (6ª edição), São Paulo, Ed. McGrawHill, 2006.

RAMOS, Dênis Paiva. **AMBIENTES DE DESENVOLVIMENTO DE *SOFTWARE* (ADS) & FERRAMENTAS CASE: IMPORTÂNCIA E APLICAÇÕES.** 2011. 61 f. Curso de Ciências da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, 2011.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S.; **Database System Concepts**, 6ª Edição, McGraw-Hill, 2010

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S.; **Sistema de bancos de dados.** 3. Ed. São Paulo: Ed. Makron, 2006.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S.; **Sistemas de Banco de dados**, 5ª edição, Tradução de Daniel Vieira. Rio de Janeiro: Editora Elsevier 2006.

SILVA, Anderson Gomes. **Estudo Comparativo de Ferramentas de Mapeamento Objeto-Relacional**. 2005. Curso de Ciências da Computação da Faculdade de Jaguariúna, Jaguariúna.

VIEIRA, M.; DURÃES, J.; MADEIRA, H. **Especificação e Validação de Benchmarks de Confiabilidade para Sistemas Transaccionais**. IEEE Latin America Transactions, Jun. 2005.

WILLEMANN, David Pedro; IBARRA, Gustavo Bestetti. **Framework Java de Apoio ao Desenvolvimento de Aplicações Web com Banco de Dados, utilizando Struts, Tiles e Hibernate.** 2007. 155 f. Monografia - Curso de Ciências da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2007.

PAZIN, A. **GawCRe: Um gerador de aplicações baseadas na web para o domínio de clinicas de reabilitação**. 2004. Dissertação (Mestrado em Ciências da Computação) – Universidade Federal de São Carlos, Centro de Ciências Exatas e de Tecnologia, São Carlos.

APÊNDICE 01 – CLASSE UTILIZADA PELO DOCTRINE

Classe Aluno

```
namespace Escola\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Aluno
 *
 * @ORM\Table(name="aluno",
indexes={@ORM\Index(name="fk_aluno_responsavel_idx",
columns={"id_responsavel"})})
 * @ORM\Entity(repositoryClass="Escola\Entity\AlunoRepository")
 */
class Aluno
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id_aluno", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $idAluno;

    /**
     * @var string
     *
     * @ORM\Column(name="nome", type="string", length=45, nullable=true)
     */
    private $nome;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="data_nascimento", type="date", nullable=true)
     */
    private $dataNascimento;

    /**
     * @var integer
     *
     * @ORM\Column(name="cpf", type="integer", nullable=true)
     */
    private $cpf;

    /**
     * @var string
     *
     * @ORM\Column(name="identidade", type="string", length=45,
nullable=true)
     */
    private $identidade;
```

```
    /**
     * @var string
     *
     * @ORM\Column(name="nome_pai", type="string", length=45,
nullable=true)
     */
    private $nomePai;

    /**
     * @var string
     *
     * @ORM\Column(name="nome_mae", type="string", length=45,
nullable=true)
     */
    private $nomeMae;

    /**
     * @var integer
     *
     * @ORM\Column(name="telefone", type="integer", nullable=true)
     */
    private $telefone;

    /**
     * @var string
     *
     * @ORM\Column(name="email", type="string", length=45, nullable=true)
     */
    private $email;

    /**
     * @var string
     *
     * @ORM\Column(name="endereco", type="string", length=45,
nullable=true)
     */
    private $endereco;

    /**
     * @var string
     *
     * @ORM\Column(name="bairro", type="string", length=45, nullable=true)
     */
    private $bairro;

    /**
     * @var string
     *
     * @ORM\Column(name="cidade", type="string", length=45, nullable=true)
     */
    private $cidade;

    /**
     * @var \Responsavel
     *
     * @ORM\ManyToOne(targetEntity="Responsavel")
     * @ORM\JoinColumns({
     *   @ORM\JoinColumn(name="id_responsavel",
referencedColumnName="id_responsavel")
     * })
```

```
     */
    private $idResponsavel;

    function __construct($idAluno, $nome, \DateTime $dataNascimento, $cpf,
$identidade, $nomePai, $nomeMae, $telefone, $email, $endereco, $bairro,
$cidade, \Responsavel $idResponsavel) {
        $this->idAluno = $idAluno;
        $this->nome = $nome;
        $this->dataNascimento = $dataNascimento;
        $this->cpf = $cpf;
        $this->identidade = $identidade;
        $this->nomePai = $nomePai;
        $this->nomeMae = $nomeMae;
        $this->telefone = $telefone;
        $this->email = $email;
        $this->endereco = $endereco;
        $this->bairro = $bairro;
        $this->cidade = $cidade;
        $this->idResponsavel = $idResponsavel;
    }


    function getIdAluno() {
        return $this->idAluno;
    }

    function getNome() {
        return $this->nome;
    }

    function getDataNascimento() {
        return $this->dataNascimento;
    }

    function getCpf() {
        return $this->cpf;
    }

    function getIdentidade() {
        return $this->identidade;
    }

    function getNomePai() {
        return $this->nomePai;
    }

    function getNomeMae() {
        return $this->nomeMae;
    }

    function getTelefone() {
        return $this->telefone;
    }

    function getEmail() {
        return $this->email;
    }

    function getEndereco() {
        return $this->endereco;
    }
```

```
function getBairro() {
    return $this->bairro;
}

function getCidade() {
    return $this->cidade;
}

function getIdResponsavel() {
    return $this->idResponsavel;
}

function setIdAluno($idAluno) {
    $this->idAluno = $idAluno;
}

function setNome($nome) {
    $this->nome = $nome;
}

function setDataNascimento(\DateTime $dataNascimento) {
    $this->dataNascimento = $dataNascimento;
}

function setCpf($cpf) {
    $this->cpf = $cpf;
}

function setIdentidade($identidade) {
    $this->identidade = $identidade;
}

function setNomePai($nomePai) {
    $this->nomePai = $nomePai;
}

function setNomeMae($nomeMae) {
    $this->nomeMae = $nomeMae;
}

function setTelefone($telefone) {
    $this->telefone = $telefone;
}

function setEmail($email) {
    $this->email = $email;
}

function setEndereco($endereco) {
    $this->endereco = $endereco;
}

function setBairro($bairro) {
    $this->bairro = $bairro;
}

function setCidade($cidade) {
    $this->cidade = $cidade;
}
```

```
    function setIdResponsavel(\Responsavel $idResponsavel) {
        $this->idResponsavel = $idResponsavel;
    }

}
```

Classe Campanha

```
namespace Escola\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Campanha
 *
 * @ORM\Table(name="campanha")
 * @ORM\Entity(repositoryClass="Escola\Entity\CampanhaRepository")
 */
class Campanha
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id_campanha", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $idCampanha;

    /**
     * @var string
     *
     * @ORM\Column(name="descricao", type="string", length=45,
nullable=true)
     */
    private $descricao;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="data", type="date", nullable=true)
     */
    private $data;

    function __construct($idCampanha, $descricao, \DateTime $data) {
        $this->idCampanha = $idCampanha;
        $this->descricao = $descricao;
        $this->data = $data;
    }

    function getIdCampanha() {
        return $this->idCampanha;
    }

    function getDescricao() {
        return $this->descricao;
```

```
    }

    function getData() {
        return $this->data;
    }

    function setIdCampanha($idCampanha) {
        $this->idCampanha = $idCampanha;
    }

    function setDescricao($descricao) {
        $this->descricao = $descricao;
    }

    function setData(\DateTime $data) {
        $this->data = $data;
    }

}
```

Classe Curso

```
namespace Escola\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Curso
 *
 * @ORM\Table(name="curso")
 * @ORM\Entity(repositoryClass="Escola\Entity\CursoRepository")
 */
class Curso
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id_curso", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $idCurso;

    /**
     * @var string
     *
     * @ORM\Column(name="descricao", type="string", length=45,
nullable=true)
     */
    private $descricao;

    /**
     * @var integer
     *
     * @ORM\Column(name="carga_horaria", type="integer", nullable=true)
     */
```

```
    private $cargaHoraria;

    /**
     * @var integer
     *
     * @ORM\Column(name="valor", type="integer", nullable=false)
     */
    private $valor;

    function __construct($idCurso, $descricao, $cargaHoraria, $valor) {
        $this->idCurso = $idCurso;
        $this->descricao = $descricao;
        $this->cargaHoraria = $cargaHoraria;
        $this->valor = $valor;
    }


    function getIdCurso() {
        return $this->idCurso;
    }

    function getDescricao() {
        return $this->descricao;
    }

    function getCargaHoraria() {
        return $this->cargaHoraria;
    }

    function getValor() {
        return $this->valor;
    }

    function setIdCurso($idCurso) {
        $this->idCurso = $idCurso;
    }

    function setDescricao($descricao) {
        $this->descricao = $descricao;
    }

    function setCargaHoraria($cargaHoraria) {
        $this->cargaHoraria = $cargaHoraria;
    }

    function setValor($valor) {
        $this->valor = $valor;
    }

}
```

Classe Matricula

```
namespace Escola\Entity;

use Doctrine\ORM\Mapping as ORM;
```

```
/**
 * Matricula
 *
 * @ORM\Table(name="matricula",
indexes={@ORM\Index(name="fk_matricula_curso1_idx", columns={"id_curso"}),
@ORM\Index(name="fk_matricula_aluno1_idx", columns={"id_aluno"}),
@ORM\Index(name="fk_matricula_campanha1_idx", columns={"id_campanha"})})
 * @ORM\Entity(repositoryClass="Escola\Entity\MatriculaRepository")
 */
class Matricula
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id_matricula", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $idMatricula;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="data_matricula", type="date", nullable=true)
     */
    private $dataMatricula;

    /**
     * @var \Aluno
     *
     * @ORM\ManyToOne(targetEntity="Aluno")
     * @ORM\JoinColumns({
     *   @ORM\JoinColumn(name="id_aluno", referencedColumnName="id_aluno")
     * })
     */
    private $idAluno;

    /**
     * @var \Campanha
     *
     * @ORM\ManyToOne(targetEntity="Campanha")
     * @ORM\JoinColumns({
     *   @ORM\JoinColumn(name="id_campanha",
referencedColumnName="id_campanha")
     * })
     */
    private $idCampanha;

    /**
     * @var \Curso
     *
     * @ORM\ManyToOne(targetEntity="Curso")
     * @ORM\JoinColumns({
     *   @ORM\JoinColumn(name="id_curso", referencedColumnName="id_curso")
     * })
     */
    private $idCurso;

    function __construct($idMatricula, \DateTime $dataMatricula, \Aluno
$idAluno, \Campanha $idCampanha, \Curso $idCurso) {
        $this->idMatricula = $idMatricula;
```

```
        $this->dataMatricula = $dataMatricula;
        $this->idAluno = $idAluno;
        $this->idCampanha = $idCampanha;
        $this->idCurso = $idCurso;
    }


    function getIdMatricula() {
        return $this->idMatricula;
    }

    function getDataMatricula() {
        return $this->dataMatricula;
    }

    function getIdAluno() {
        return $this->idAluno;
    }

    function getIdCampanha() {
        return $this->idCampanha;
    }

    function getIdCurso() {
        return $this->idCurso;
    }

    function setIdMatricula($idMatricula) {
        $this->idMatricula = $idMatricula;
    }

    function setDataMatricula(\DateTime $dataMatricula) {
        $this->dataMatricula = $dataMatricula;
    }

    function setIdAluno(\Aluno $idAluno) {
        $this->idAluno = $idAluno;
    }

    function setIdCampanha(\Campanha $idCampanha) {
        $this->idCampanha = $idCampanha;
    }

    function setIdCurso(\Curso $idCurso) {
        $this->idCurso = $idCurso;
    }

}
```

Classe Responsavel

```
namespace Escola\Entity;

use Doctrine\ORM\Mapping as ORM;
```

```
/**
 * Responsavel
 *
 * @ORM\Table(name="responsavel")
 * @ORM\Entity(repositoryClass="Escola\Entity\ResponsavelRepository")
 */
class Responsavel
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id_responsavel", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $idResponsavel;

    /**
     * @var string
     *
     * @ORM\Column(name="nome", type="string", length=45, nullable=true)
     */
    private $nome;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="data_nascimento", type="date", nullable=true)
     */
    private $dataNascimento;

    /**
     * @var integer
     *
     * @ORM\Column(name="cpf", type="integer", nullable=false)
     */
    private $cpf;

    /**
     * @var string
     *
     * @ORM\Column(name="identidade", type="string", length=45,
nullable=true)
     */
    private $identidade;

    /**
     * @var string
     *
     * @ORM\Column(name="telefone", type="string", length=45,
nullable=true)
     */
    private $telefone;

    /**
     * @var string
     *
     * @ORM\Column(name="email", type="string", length=45, nullable=true)
     */
    private $email;
```

```
    /**
     * @var string
     *
     * @ORM\Column(name="endereco", type="string", length=45,
nullable=true)
     */
    private $endereco;

    /**
     * @var string
     *
     * @ORM\Column(name="cidade", type="string", length=45, nullable=true)
     */
    private $cidade;

    /**
     * @var string
     *
     * @ORM\Column(name="estado", type="string", length=45, nullable=true)
     */
    private $estado;

    function __construct($idResponsavel, $nome, \DateTime $dataNascimento,
$cpf, $identidade, $telefone, $email, $endereco, $cidade, $estado) {
        $this->idResponsavel = $idResponsavel;
        $this->nome = $nome;
        $this->dataNascimento = $dataNascimento;
        $this->cpf = $cpf;
        $this->identidade = $identidade;
        $this->telefone = $telefone;
        $this->email = $email;
        $this->endereco = $endereco;
        $this->cidade = $cidade;
        $this->estado = $estado;
    }

    function getIdResponsavel() {
        return $this->idResponsavel;
    }

    function getNome() {
        return $this->nome;
    }

    function getDataNascimento() {
        return $this->dataNascimento;
    }

    function getCpf() {
        return $this->cpf;
    }

    function getIdentidade() {
        return $this->identidade;
    }

    function getTelefone() {
        return $this->telefone;
    }

    function getEmail() {
```

```
        return $this->email;
    }

    function getEndereco() {
        return $this->endereco;
    }

    function getCidade() {
        return $this->cidade;
    }

    function getEstado() {
        return $this->estado;
    }

    function setIdResponsavel($idResponsavel) {
        $this->idResponsavel = $idResponsavel;
    }

    function setNome($nome) {
        $this->nome = $nome;
    }

    function setDataNascimento(\DateTime $dataNascimento) {
        $this->dataNascimento = $dataNascimento;
    }

    function setCpf($cpf) {
        $this->cpf = $cpf;
    }

    function setIdentidade($identidade) {
        $this->identidade = $identidade;
    }

    function setTelefone($telefone) {
        $this->telefone = $telefone;
    }

    function setEmail($email) {
        $this->email = $email;
    }

    function setEndereco($endereco) {
        $this->endereco = $endereco;
    }

    function setCidade($cidade) {
        $this->cidade = $cidade;
    }

    function setEstado($estado) {
        $this->estado = $estado;
    }

}
```

APÊNDICE 02 – CLASSE UTILIZADA PELO PROPEL

```
<database name="default" defaultIdMethod="native"
defaultPhpNamingMethod="underscore">
  <table name="aluno" idMethod="native" phpName="Aluno">
    <column name="id_aluno" phpName="IdAluno" type="INTEGER"
primaryKey="true" required="true"/>
    <column name="id_responsavel" phpName="IdResponsavel" type="INTEGER"
required="true"/>
    <column name="nome" phpName="Nome" type="VARCHAR" size="45"/>
    <column name="data_nascimento" phpName="DataNascimento" type="DATE"/>
    <column name="cpf" phpName="Cpf" type="INTEGER"/>
    <column name="identidade" phpName="Identidade" type="VARCHAR"
size="45"/>
    <column name="nome_pai" phpName="NomePai" type="VARCHAR" size="45"/>
    <column name="nome_mae" phpName="NomeMae" type="VARCHAR" size="45"/>
    <column name="telefone" phpName="Telefone" type="INTEGER"/>
    <column name="email" phpName="Email" type="VARCHAR" size="45"/>
    <column name="endereco" phpName="Endereco" type="VARCHAR" size="45"/>
    <column name="bairro" phpName="Bairro" type="VARCHAR" size="45"/>
    <column name="cidade" phpName="Cidade" type="VARCHAR" size="45"/>
    <foreign-key foreignTable="responsavel" name="fk_aluno_responsavel">
      <reference local="id_responsavel" foreign="id_responsavel"/>
    </foreign-key>
    <index name="fk_aluno_responsavel_idx">
      <index-column name="id_responsavel"/>
    </index>
    <vendor type="mysql">
      <parameter name="Engine" value="InnoDB"/>
    </vendor>
  </table>
  <table name="campanha" idMethod="native" phpName="Campanha">
    <column name="id_campanha" phpName="IdCampanha" type="INTEGER"
primaryKey="true" required="true"/>
    <column name="descricao" phpName="Descricao" type="VARCHAR" size="45"/>
    <column name="data" phpName="Data" type="DATE"/>
    <vendor type="mysql">
      <parameter name="Engine" value="InnoDB"/>
    </vendor>
  </table>
  <table name="curso" idMethod="native" phpName="Curso">
    <column name="id_curso" phpName="IdCurso" type="INTEGER"
primaryKey="true" autoIncrement="true" required="true"/>
    <column name="descricao" phpName="Descricao" type="VARCHAR" size="45"/>
    <column name="carga_horaria" phpName="CargaHoraria" type="INTEGER"
size="4"/>
    <column name="valor" phpName="Valor" type="INTEGER" required="true"/>
    <vendor type="mysql">
      <parameter name="Engine" value="InnoDB"/>
    </vendor>
  </table>
  <table name="matricula" idMethod="native" phpName="Matricula">
    <column name="id_matricula" phpName="IdMatricula" type="INTEGER"
primaryKey="true" required="true"/>
    <column name="id_curso" phpName="IdCurso" type="INTEGER"
required="true"/>
    <column name="id_aluno" phpName="IdAluno" type="INTEGER"
required="true"/>
```

```
    <column name="id_campanha" phpName="IdCampanha" type="INTEGER"
required="true"/>
    <column name="data_matricula" phpName="DataMatricula" type="DATE"/>
    <foreign-key foreignTable="aluno" name="fk_matricula_aluno1">
      <reference local="id_aluno" foreign="id_aluno"/>
    </foreign-key>
    <foreign-key foreignTable="campanha" name="fk_matricula_campanha1">
      <reference local="id_campanha" foreign="id_campanha"/>
    </foreign-key>
    <foreign-key foreignTable="curso" name="fk_matricula_curso1">
      <reference local="id_curso" foreign="id_curso"/>
    </foreign-key>
    <index name="fk_matricula_curso1_idx">
      <index-column name="id_curso"/>
    </index>
    <index name="fk_matricula_aluno1_idx">
      <index-column name="id_aluno"/>
    </index>
    <index name="fk_matricula_campanha1_idx">
      <index-column name="id_campanha"/>
    </index>
    <vendor type="mysql">
      <parameter name="Engine" value="InnoDB"/>
    </vendor>
  </table>
  <table name="responsavel" idMethod="native" phpName="Responsavel">
    <column name="id_responsavel" phpName="IdResponsavel" type="INTEGER"
primaryKey="true" required="true"/>
    <column name="nome" phpName="Nome" type="VARCHAR" size="45"/>
    <column name="data_nascimento" phpName="DataNascimento" type="DATE"/>
    <column name="cpf" phpName="Cpf" type="INTEGER" required="true"/>
    <column name="identidade" phpName="Identidade" type="VARCHAR"
size="45"/>
    <column name="telefone" phpName="Telefone" type="VARCHAR" size="45"/>
    <column name="email" phpName="Email" type="VARCHAR" size="45"/>
    <column name="endereco" phpName="Endereco" type="VARCHAR" size="45"/>
    <column name="cidade" phpName="Cidade" type="VARCHAR" size="45"/>
    <column name="estado" phpName="Estado" type="VARCHAR" size="45"/>
    <vendor type="mysql">
      <parameter name="Engine" value="InnoDB"/>
    </vendor>
  </table>
</database>
```