|  |  |
| --- | --- |
|  | *Ministério da Educação* **Universidade Tecnológica Federal do Paraná**  *Pró-Reitoria de Pesquisa e Pós-Graduação* |

**Relatório Final de Atividades**

**Comparação do tempo de trabalho entre modelo relacional e modelo não relacional**

**vinculado ao projeto**

**<Título do Projeto>**

**Jean Carlos da Silva**

**Bolsista UTFPR**

**Ciência da computação**

**Israel Rodrigues**

**Voluntário**

**Ciência da computação**

**Data de ingresso no programa: 08/2019**

**Prof(ª). Dr(ª). Kelyn Schenatto**

Área do Conhecimento: (de acordo com a tabela de áreas de conhecimento do CNPq)

*CAMPUS SANTA HELENA*, 2019

**JEAN CARLOS DA SILVA**

**ISRAEL RODRIGUES**

**KELYN SCHENATTO**

**GIUVANI CONTI**

**COMPARAÇÃO DE TEMPO DE TRABALHO DO MODELO RELACIONAL E NÃO RELACIONAL**

Relatório do Programa de Iniciação Científica da Universidade Tecnológica Federal do Paraná.

*Universidade Tecnológica Federal do Paraná, Campus-Santa Helena*, 2019

|  |  |
| --- | --- |
| **SUMÁRIO**  [**INTRODUÇÃO 1**](#_Toc42033042)  [**METODOLOGIA 4**](#_Toc42033043)  [**RESULTADOS E DISCUSSÕES 12**](#_Toc42033044)  [**CONCLUSÕES 13**](#_Toc42033045)  [**REFERÊNCIAS 13**](#_Toc42033046) |  |
|  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**INTRODUÇÃO**

Com o constante crescimento da informação no mundo, o aumento das utilizações dos bancos de dados vem tornando-se a cada dia mais importante, tanto o modelo relacional quanto o modelo não-relacional, que por este estudo vem fazer um levantamento de informações de quais modelos oferecem vantagens e assim comparar ambos os modelos nas questões de inserção, atualização de registros e a exclusão. Também é levantado duvidas pelos profissionais da área tecnológica de qual modelo seria mais eficiente para futuras aplicações a serem desenvolvidas. O modelo relacional SQL (Structured Query Language) segue um paradigma de utilizar e atribuir as informações contidas nele de forma estruturada e armazenada em tabela.

O modelo não-relacional contem boas experiências comparado ao modelo relacional em diversas situações nos quais outros trabalhos já desenvolvidos resultaram em seu desempenho aprimorado em relação ao modelo relacional. Contudo o modelo não-relacional requer um pouco mais de atenção, pois requer um grau de conhecimento mais avançado pelo fato de que trabalha de forma diferente, pois traz consigo um paradigma novo no quesito de armazenamento de dados. Como exemplo, o armazenamento orientado a documentos que é o que o MongoDB trabalha e será utilizado neste trabalho.

O objetivo deste relatório é apresentar os resultados dentro do tempo de vigência da bolsa, com ênfase em mostrar os números comparativos do tempo em serviço sendo executado através de algoritmos desenvolvidos para realizar as funções normais dentro de qualquer banco de dados, mas com característica de inclusão, alteração e exclusão massiva de dados em determinado tempo. Oferecendo uma visão ampla de todo o processo realizado passo a passo, para assim no final conseguir direcionar uma resposta a qual de adéque para projetos futuros.

# METODOLOGIA

**Tecnologia.**

A tecnologia usada para o desenvolvimento do trabalho foi um notebook com o sistema operacional Windows 10 Home de 64 bits, processador Intel Core i5-7300HQ 2.50 GHz, 8 GB de RAM, 1T de HD.

O Software usado para a utilização dos sistemas gerenciadores de bancos de dados foi a versão do PostgreSQL 4, MySQL e o MongoDB.

**Programação.**

A programação foi desenvolvida utilizando a linguagem Java, onde todos as funções exercidas pelo programa para o modelo relacional, foram feitas com o auxílio do framework JPA (Java Persistence API) usado para persistir objetos dentro do banco de dados. Sendo assim não apenas um framework de mapeamento objeto-relacional, mas também tem uma gama de funcionalidades que facilita qualquer projeto.

Foram criadas duas classes para simular as atividades de inserção, atualização e a parte de deletar os dados dentro do banco de dados do modelo relacional, já para o modelo não relacional foi usado de forma direta através do Java sem a criação de classe até pelo fato de que o NoSQL trabalha de forma diferente sendo orientado a documentos, colunas, que diferente do relacional que trabalha com junções entre tabelas que foram ou são criadas dentro do banco de dados.

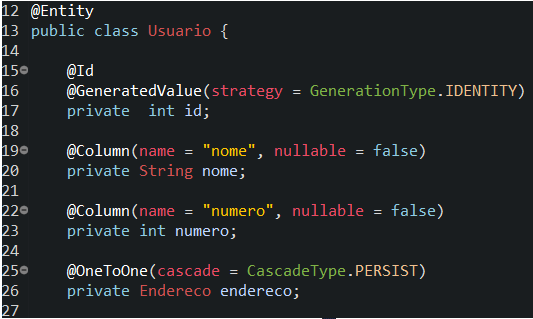


Figura 1. Demonstração da classe usuário.

Na figura 1 vemos a classe usuário com seus respectivos atributos e seus objetos para realizar o mapeamento através do JPA. Essas variáveis são criadas dentro da linguagem Java e com o JPA são inseridas de forma automática dentro do banco MySQL quanto do PostgreSQL para assim não ter a necessidade de ir dentro do banco de dados e efetuar a criação da tabela manualmente.

A variável Id usada para identificar o usuário do qual pertence o nome que será informado pelo usuário, o número gerado aleatoriamente para cada usuário dentro de uma range de 0 a 1 bilhão e a variável endereço sendo do tipo Endereço da qual desta forma irá instanciar a classe endereço utilizando o Id do mesmo para ter o referenciamento. Na figura a seguir poderá ser observado a classe endereço.

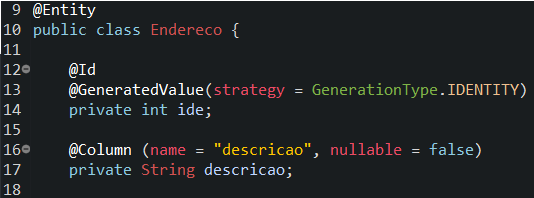


Figura 2. Demonstração da classe endereço.

A classe endereço tem como suas variáveis um ide que será gerado de forma também aleatória e a variável descrição onde será informado pelo usuário a localidade onde reside.

A anotação *@Entity* é usada para informar que uma classe é uma entidade que será relacionada a uma tabela no banco de dados através do mapeamento objeto relacional. Dentro de várias anotações utilizadas no projeto o *@Id* é utilizado para informar ao JPA que a variável que recebe esta anotação será uma chave primaria da respectiva tabela em que a classe está relacionada.

Logo após a anotação *@Id* deverá ter *@GeneratedValue* do qual utiliza-se para realizar a geração do valor único da entidade que será persistido pelo próprio provedor. Sem a utilização desta anotação implicará que o valor será gerado e gerenciado pelo próprio software.

**Conexão com banco de dados.**

Para realizar a conexão aos bancos de dados do modelo-relacional foi utilizado o arquivo chamado persistence.xml no qual foram feitas as configurações de acesso aos bancos MySQL e PostgreSQL.



Figura 3. Arquivo Persistence.xml

A elaboração da configuração do arquivo *persistence.xml* inicia-se atribuindo dentro do arquivo as classes que foram criadas com o efeito de criar as tabelas dentro do bando de dados. Após a esta etapa, as propriedades da conexão são configuradas com o driver do banco que no caso deste trabalho foi usado o MySQLe PostgreSQL.

Em seguida foi inserido o endereço do banco, sendo o endereço do local host ou IP dependendo da aplicação e o nome da base onde foram criadas as tabelas. Usuário e senha são informados para que haja uma conexão com o banco de dados.

Após a conexão inicia-se a parte do hibernate sendo que uma das etapas mais importante encontra-se na última linha, aonde têm a opção *create* que tem como função gerar o banco e as tabelas. Entre esse comando também existe outros no quais fazem suas respectivas funções. No trabalho foi usado o comando comum *create*(apenas gera o banco) , *create-drop* (gera o banco e logo em seguida o apaga), *update*( utilizado para a atualização do banco) e *none* que não muda e não interfere no banco de dados sem ser pela aplicação, mas usado no projeto para apagar os dados inseridos dentro do banco de dados.

No NoSQL a conexão com o banco de dados do MongoDB foi realizada de forma direta, sem a utilização do *persistence.xml*. Pois trata-se de ser um método mais simples e eficaz para a conexão.

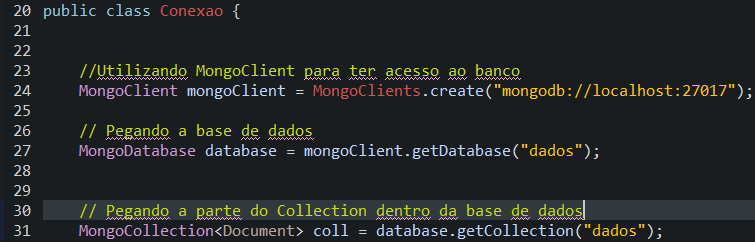


Figura 4**.** Classe conexão com o MongoDB

A classe conexão foi desenvolvida para realizar dentro dela a conexão direta com o MongoDB. A primeira parte da linha na qual começa com MongoClient que faz parte da biblioteca interna do mongo onde está relacionado ao objeto mongoClient do qual é atribuído o endereço de acesso ao servidor do banco que neste caso é em local host e a respectiva porta de acesso. Logo após no database é atribuído a base de dados que está dentro do mongo para ser adicionado todas as informações dentro dessa base de dados. Ao fim o MongoCollection que tem como um array na forma de documento para ser salvo dentro da base de dados do mongo. Collection é responsável pelas coleções pertencentes a base de dados e é uma função interna do Mongo que é instanciado e relacionado ao objeto coll.

**Função inserir (modelo relacional).**

Para o desenvolvimento do projeto foram dividas três funções como inserir, atualizar e deletar os dados para todos os bancos trabalhados.

Na função inserir iniciamos declarando as variáveis *i* e *x* que serão utilizadas no processo da construção da função. A variável *i* funcionará de forma que um id interno seja adicionado através da função **for** para obter o controle e garantir que o Id seja fornecido em forma crescente e organizada e que o Id não seja de forma aleatória.

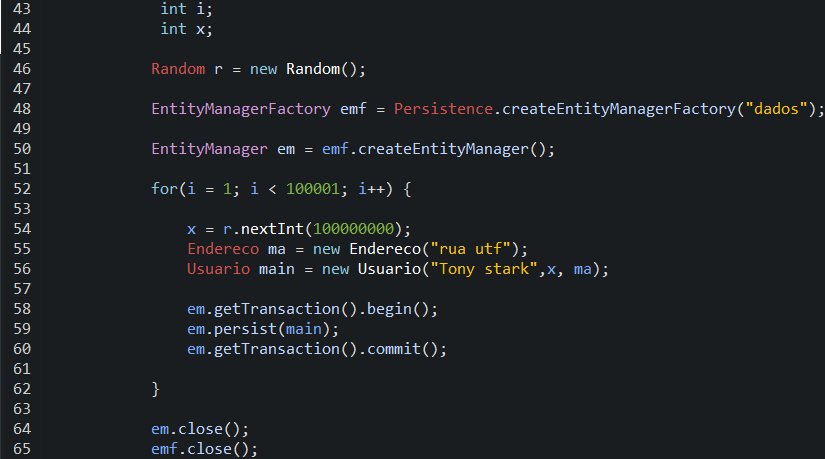


Figura 5. Função inserir.

A variável *x* assumirá o valor da variável *r* que será gerado de forma aleatória através de uma função randômica**,** que produz números aleatórios de um certo range estipulado pelo programa de zero a um bilhão. Dentro dos parâmetros da função para ocorrer a inserção nota-se de que o valor da variável *i* inicia-se com o número 1 e que a segunda parte do valor de *i* seria menor que o número de inserções.

As classes endereço e usuário são instanciadas e colocados seus respectivos parâmetros como na classe endereço a descrição do nome da rua e na classe usuário com o nome do indivíduo, o valor aleatório atribuído pela variável *x* e dentro como parâmetro da classe usuário foi utilizado uma variável endereço que tem como objetivo referenciar a classe endereço como parâmetro no usuário. Em seguida temos as funções do JPA sendo utilizadas para iniciar a persistência dentro do banco de dados do modelo relacional e após a função **for** ser totalmente executada é usada a função close para terminar a conexão com o banco de dados.

**Função atualizar (modelo relacional).**

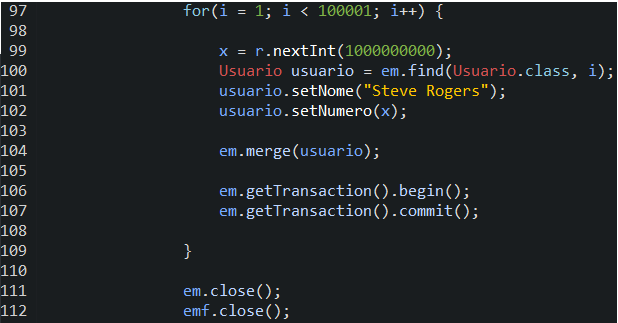


Figura 6. Função atualizar.

Nesta parte da programação foi utilizado a função **for** novamente para realizar a passagem em todos os usuários cadastrados no banco de dados e utilizando o comando **find** através da classe usuário identificar o id de cada e realizar a atualização do nome e alterar o número aleatório gerado anteriormente na função inserir.

O comando merge do JPA irá criar um novo documento que no caso será o usuário. Dessa forma o novo documento era mudar o nome e o número do usuário cadastrado com o respectivo id que foi encontrado pelo método **find**. Mas continuará a manter as outras informações persistidas no documento que já foi preenchido. Ao grosso modo ele atualizará mesclando os arquivos mantando a informação, mas atualizando com informações nova.

**Função deletar (modelo relacional).**

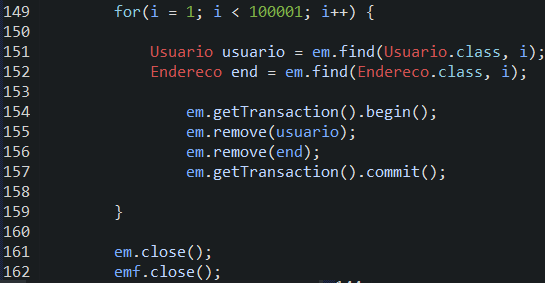


Figura 7. Função deletar.

Na função deletar com o for já em execução, as classes usuário e endereço são instanciadas e sendo utilizadas dentro código do JPA para encontrar o id de ambas as classes para começar a deletar o objeto das tabelas.

Antes de iniciar a remoção os objetos devem ser recuperados dessa forma usando a função **find** retornará à informação do objeto. Após encontrar o id requerido na função **for** pela variável *i*, a função remove é iniciada na qual será excluído fisicamente o objeto dentro da entidade e todos os dados removidos do banco e encerrando a conexão do JPA.

**Função inserir (modelo não-relacional).**

A função inserir no modelo não relacional é diferente das inserções feitas no MySQL e PostgreSQL pelo fato de ser mais dinâmico sem necessitar de chaves primarias e chaves estrangeiras, sendo assim a é baseada em documentos Json do qual é possível fazer uma inserção direta. Com a conexão já estabelecida o próprio mongoDB vai gerar um ID automático para cada documento adicionado sendo totalmente um inicial aleatório. Com isso foi usado um **for** para que gerasse uma variável *Id* para que o código também gerasse um número para o id de forma muito mais fácil para trabalhar com o crescimento de 0 até o valor estipulado pelo programa.



Figura 8. Função Inserir NoSQL.

Iniciamos com uma função própria do mongo, conhecida como **drop** essa função tem como objetivo no início apagar todo q qualquer dado já criado dentro do banco se existir para a inserção seja de forma sequencial. O método random também foi usado para gerar um valor aleatório para ser armazenado em um dos campos do documento.

O documento é instanciado e inserido dentro do document passando como parâmetro as informações que serão adicionadas dentro do banco. Após o preenchimento e do parâmetro a função interna do mongo **InsertOne** é chamada e que tem por parâmetro o documento chamado document para realizar a inserção dentro do banco e após o **for** parar se der utilizado é encerrada a conexão com o mongoDB. A variável *op* tem como objetivo ser especificada pelo usuário que se torna o número máximo de inserção que será feito dentro do banco de dados.

**Função atualizar (modelo não-relacional).**

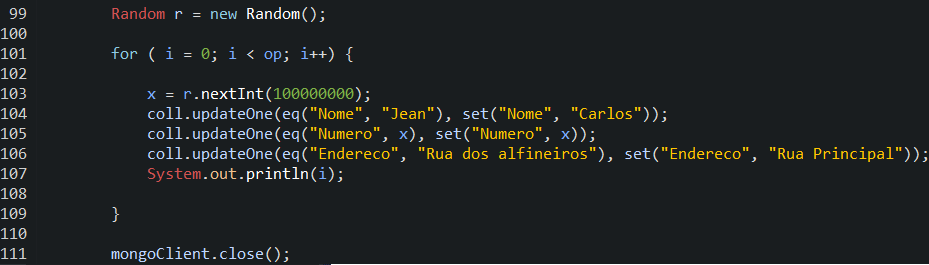


Figura 9. Função atualizar no NoSQL.

A função atualizar é utilizada com parâmetros assim como no modelo relacional ele irá atualizar o valor dentro do documento um por vez assim como é feito com as tabelas no modelo relacional. A função interna do mongo *updateOne* implica que apenas um item do documento seja alterado por vez. Assim dentro do **for** é possível alterar mais áreas do documento. Sendo como parâmetro da função update começa com o campo do qual deseja mudar e qual o valor que contem dentro dele, após usando ao **set** colocaremos o mesmo campo que foi inserido e o novo valor para ser atualizado. Assim é feito com o campo número e o endereço. No campo número ele vai substituir o valor que foi colocado de forma aleatória por outro também de forma aleatória só sendo especificado o range do valor. Após toda a operação do **for** ser concluída a conexão com o mongoDB é encerrada.

**Função deletar (modelo não-relacional).**

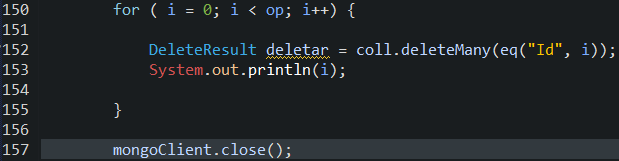


Figura 10. Função deletar no NoSQL.

Deletar é a parte da programação mais simples de todo o contexto NoSQL pois utilizando uma função própria do mongo ele tem a funcionalidade de excluir todo o banco ou apagar todos os dados do banco como foi o código usado para este trabalho e apenas excluir todos os documentos que estava dentro da base de dados do mongoDB sem alterar a base da coleção mas sim, apenas as informações. Desta forma fica igualitária ambos os parâmetros tanto para o modelo relacional quanto não relacional, que por sua vez teve apenas suas informações e dados apagados também.

**Tempo de execução.**

Dentro das funções inserir, atualizar e deletar foi utilizado uma função própria do Java *System.currentTimemillis()* que tem como objetivo calcular o tempo de execução do código do início ao fim.



Figura 11. Função de tempo do Java com a variável tempoInicial.

Iniciando uma variável do tipo Long com o nome de *tempoInicial* é iniciada no começo das funções e ao final de cada uma com o nome *tempoFinal.*

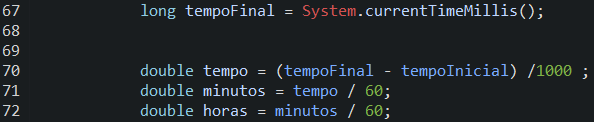


Figura 12. Função de tempo do Java com a variável tempoFinal.

Sendo no final criadas mais três variáveis do tipo double para realizar o cálculo simples do tempo para que dessa forma se os dados fossem alcançar números grandes eles fossem convertidos para segundos ou horas, pois a função *millis* apresenta apenas os dados em milissegundo.

# 

# RESULTADOS E DISCUSSÕES

Com a programação completa, foi iniciado os testes em cada banco de dados com as funções de inserir, atualizar e deletar massivamente, começando com poucas informações até uma expressão massiva de cem mil dados. Para poder ter uma criar uma média de tempo para comparar mais tarde a efetividade de cada banco para determinada função foi rodado o programa três vezes para realizar a média simples entre os três.

Na tabela inserir (tabela 1) é mostrado as medias de cada função em seu respectivo banco. A diferença enorme entre a efetividade entre os bancos começa a apresentar-se já na parte da inserção dos dez mil dados, onde o NoSQL mostrou-se nitidamente mais rápido. Onde sua diferença está no seu tempo enquanto o modelo relacional utilizou de minutos para completar todo o processo a modelo não-relacional utilizou apenas de segundos para a conclusão.

Tabela 1. Tempo médio das inserções feitas nos bancos de dados.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Cem dados | Mil dados | Dez mil dados | Cem mil dados |
| MySQL | 7,66 Seg | 49 Seg | 8,47 Min | 3,24 Horas |
| PostgreSQL | 3,33 Seg | 5,3 Seg | 2,53 Min | 2,35 Horas |
| MongoDB | 1,66 Seg | 2 Seg | 6 Seg | 16 Seg |

O NoSQL tende a ser mais rápido para esta situação, pois trata seus dados de forma diferente do modelo relacional onde, no caso já apresentado para o modelo relacional, foi criada duas tabelas onde continham uma ligação entre si de chave primaria e chave estrangeira. Que uma tabela detinha informação da outra em suas respectivas informações de cada id da tabela usuário relacionado com um id da tabela endereço. Sendo assim o modelo não relacional trabalha na forma orientado a documentos e não com tabelas. Então para a inserção as informações são colocadas em um único documento facilitando assim a inserção no banco sem necessitar preocupar com chaves estrangeiras.

Na parte da atualização dos dados dentro dos bancos de dados, os resultados não tem uma diferença grande, pois apresentam um comportamento previsível. Para o modelo relacional que levou mais tempo que o NoSQL, pelo fato de ter todo processo de buscar chaves estrangeiras relacionadas com cada *id* da tabela usuário que corresponde com um *id* interno dentro da tabela endereço, o que deve custar um esforço maior de tempo de cada banco.

Tabela 2. Tempo médio das atualizações feitas nos bancos de dados.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Cem dados | Mil dados | Dez mil dados | Cem mil dados |
| MySQL | 10,33 Seg | 1,02 Min | 10,1 Min | 3,3 Horas |
| PostgreSQL | 5,06 Seg | 10,33 Seg | 2,32 Min | 2,27 Horas |
| MongoDB | 1,33 Seg | 3 Seg | 1,10 Min | 1,34 Horas |

O NoSQL por se tratar de documentos tende a ser mais eficaz que o MySQL e PostgreSQL pois não tem que ficar realizando buscas através de tabelas. Por este fato o modelo não relacional apresentou-se mais eficaz na atualização dos dados mesmo que utilizando o método find para buscar cada *id*.

A função deletar apresentou um comportamento inesperado, mas dentro dos parâmetros, pois o banco postgreSQL foi mais rápido que MySQL e o MogoDB. O comportamento é facilmente explicado como o modelo relacional utilizou o recurso do JPA e o Hibernate que tendem a ganhar mais velocidade em suas funções internas diferente do que foi utilizado no NoSQL que é feito de forma manual as exclusões.

Tabela 3. Tempo médio da exclusão de dados nos bancos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Cem dados | Mil dados | Dez mil dados | Cem mil dados |
| MySQL | 9,33 Seg | 58 Seg | 9,55 Min | 1,32 Horas |
| PostgreSQL | 9,33 Seg | 10 Seg | 35 Seg | 13 Min |
| MongoDB | 2 Seg | 2 Seg | 31 Seg | 47 Min |

Na tabela acima (Tabela 3) nota-se que há uma discrepância nos dados até mesmo entre os próprios bancos MySQL e PostgreSQL que ambos pertencem ao modelo relacional. O PostgreSQL ele surgiu com uma forma mais dinâmica que o MySQL o que pode resultar em uma performance melhor.

# CONCLUSÕES

Conforme os resultados das médias dos testes apresentados nas tabelas anteriores, nota-se uma vantagem considerável do modelo não-relacional em relação ao modelo relacional, pois trata-se de que o NoSQL não veio para substituir o modelo relacional, mas sim para complementar e oferecer uma visão diferente do tradicional.

O NoSQL mostrou-se mais efetivo nas inserções feitas em massa apresentando números altamente produtivos perante o uso desta aplicação. Mas com tudo, para fazer o uso de um dos modelos, um projeto em si deve ser estudado e levantado as vantagens e desvantagens de cada um dos modelos. O modelo não-relacional mostrou-se vantajoso, porém não significa que ele tenha o mesmo comportamento efetivo dentro da aplicação.

A aplicação implementada atingiu os objetivos e resultados esperados. Demonstrando a forma de como age um modelo relacional com respectiva configuração em relação ao modelo não relacional que utilizou uma configuração simples. Uma forma para contribuir com este projeto, pensando em uma maneira de aprimorar a pesquisa seria utilizar configurações iguais para todos os modelos e incluir diferentes bancos de dados para o modelo não relacional orientado a colunas, grafos entre outras formas já que neste trabalho foi utilizado o modelo orientado a documento.

Os resultados mediantes apresentados neste trabalho podem não ser iguais se executado em outro meio, pois pode haver uma variação da utilização de cada aparelho como memória ram entre outros aspectos que podem interferir nos resultados finais.

# REFERÊNCIAS

[1] CARDOSO, V. M.; CARDOSO, G. C. **Sistema de banco de dados.** São Paulo: Saraiva, 2013. Acesso em: 10 junho 2020.

[2] MONGODB JAVA DRIVER DOCUMENTATION. MONGODB. Disponível em:<https://mongodb.github.io/mongo-java-driver/4.0/>. Acesso em: 10 junho 2020.

[3] KOBER, MARCEL. *Um estudo comparativo entre o uso de bases de dados relacionais e não relacionais para data warehouses.*2017. 106f. Trabalho de conclusão de curso – Univates, Lajeado, 2017. Disponível em: < https://www.univates.br/bdu/bitstream/10737/1674/1/2017MarcelKober. >. Acesso em: 10 junho 2020.

[4] SOLAGNA, EMANUELE. A. Um estudo comparativo entre o mongodb e o postgresql. 2016. 23f. Trabalho de conclusão de curso – Instituto Federal Sul-Rio-Grandense, Passo fundo,2016. Disponível em:< https://painel.passofundo.ifsul.edu.br/uploads/arq/201607111805501015914198.pdf >. Acesso em: 10 junho 2020.

|  |  |
| --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Nome Orientador e Assinatura | Nome Aluno e Assinatura |