https://eventos.utfpr.edu.br//sicite/sicite2020

# Estudo e análise de desempenho dos bancos de dados relacionais e NoSQL

**Jean Carlos da Silva**
jeansilva@alunos.utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Santa Helena, Paraná, Brasil

**Kelyn Schenatto**
kschenatto@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Medianeira, Paraná, Brasil

**Giuvane Conti**
giuvaneconti@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Santa Helena, Paraná, Brasil

**Recebido:** 19 ago. 2020.
**Aprovado:** 01 out. 2020.



## RESUMO

Este trabalho abordará um problema no qual é discutido por vários profissionais da área da computação, que é o desempenho comparativo de cada sistema gerenciador de banco de dados entre o modelo relacional e não-relacional. Os sistemas escolhidos para o processo do desenvolvimento do projeto foram o MySQL e PostgreSQL, ambos do modelo relacional e o NoSQL do modelo não-relacional. Sendo criada uma base de dados para preencher todos os campos dos bancos de dados para averiguar qual modelo teria um melhor desempenho durante a execução do projeto. Utilizando a ferramenta de linguagem de programação Java para realizar todos os objetivos como inserção, atualização e exclusão de dados. O modelo não-relacional obteve um desempenho acima do relacional para este projeto, mas dentro de cada aplicação pode ser alterado dependendo do objetivo. Deste modo, o desempenho de cada modelo pode variar, mesmo sendo rápido pode não ser eficaz para determinado projeto.

**PALAVRAS-CHAVE:** Comparação entre banco de dados. Modelo relacional. Modelo não-relacional.

## ABSTRACT

This work will address a problem in which it is discussed by several professionals in the field of computing, which is the comparative performance of each database management system between the relational and non-relational models. The systems chosen for the project development process were MySQL and PostgreSQL, both from the relational and NoSQL from the non-relational model. A database was created to fill all the fields of the databases to find out which model would perform better during the execution of the project. Using the Java programming language tool to accomplish all objectives such as inserting, updating and deleting data. The non-relational model achieved performance above the relational for this project but it can be changed depending on each application's objective. Thus, each model's performance may vary even though it may be fast and may not be useful for specific projects.

**KEYWORDS:** Database comparison. Relational model. No-relational model.

## INTRODUÇÃO

Com o crescimento da produção de informação, 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, pelo qual este estudo faz um levantamento de informações entre os modelos, analisando suas vantagens, e, assim, comparando os modelos na questão de inserção, atualização e exclusão. O modelo relacional SQL (*Structured Query Language*) segue um paradigma de atribuir as informações contidas nele de forma estruturada e armazenada em tabela. Segundo Silberschatz (2006, p.25), o modelo relacional é uma coleção de tabelas e cada uma tem atributo único. A linha da tabela é a relação de conjunto de valores, onde a tabela é um conjunto de entidades e a linha é uma entidade.

A partir da definição de Silberschatz, nota-se que o modelo relacional é um grupo de tabelas na qual tende a apresentar os dados e suas relações entre eles. O modelo não-relacional contém boas experiências assim como o modelo relacional em diversas situações. O NoSQL requer um pouco mais de atenção, pois trabalha com um paradigma novo. Algumas características apresentadas sobre o NoSQL é que ele se difere do modelo relacional. Segundo Sadalage e Fowler (2013, p.36), o NoSQL atua sem esquema, permitindo que sejam adicionados nos campos de registros no banco sem ter de definir primeiro qualquer mudança na estrutura, facilitando ao lidar com dados não uniformes.

Deste modo o NoSQL foi desenvolvido com o objetivo de suprir alguns defeitos que consiste no modelo relacional.

## METODOLOGIA

O equipamento utilizado no desenvolvimento do projeto é um notebook Acer nitro5 com sistema operacional Windows 10 home 64 bits, processador intel core i5-7300HQ 2.50 GHz, 8 GB de memória ram e 1T de HD. Sistemas gerenciadores de banco de dados que foram utilizados são: PostgresSQL 4, MySQL e MongoDB.

A programação foi desenvolvida utilizando a linguagem Java, onde todas as funções exercidas pelo programa para o modelo relacional foram feitas do framework JPA (Java Persistence API) usado para persistir objetos dentro do banco de dados.

Foram criadas duas classes para simular as atividades de inserção, atualização e exclusão dos dados dentro do modelo relacional. Para o modelo não-relacional foi feito através do Java sem a criação de classe pois o NoSQL trabalha orientado a documentos e colunas, enquanto o modelo relacional trabalha com junções entre tabelas.

Figura 1 - Demonstração da classe usuário.

```
12 @Entity
13 public class Usuario {
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private  int id;
18
19     @Column(name = "nome", nullable = false)
20     private String nome;
21
22     @Column(name = "numero", nullable = false)
23     private int numero;
24
25     @OneToOne(cascade = CascadeType.PERSIST)
26     private Endereco endereco;
27
```

Fonte: Autor (2020).

Na figura 1 a variável *id* é usada para identificar o usuário do qual pertence o nome, o número gerado aleatoriamente para cada usuário e a variável endereço sendo do tipo endereço que desta forma irá instanciar a própria classe para ter o referenciamento. Na figura 2 pode ser observado a classe endereço.

Figura 2 - Demonstração da classe endereço.

```
 9 @Entity
10 public class Endereco {
11
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private int ide;
15
16     @Column (name = "descricao", nullable = false)
17     private String descricao;
18
```

Fonte: Autor (2020).

A anotação *@Entity* é usada para informar que classe é uma entidade e será relacionada a uma tabela no banco. O *@Id* é utilizado para informar ao JPA que a variável será uma chave primária da tabela. A anotação *@Id* vem acompanhada com a anotação *@GeneratedValue* que é utilizada para realizar a geração de um valor único da entidade em que será persistida. No desenvolvimento do projeto foram divididas três funções: inserir, atualizar e deletar.

Na função inserir é declarada 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.

Figura 3 - Função inserir.

```
43             int i;
44             int x;
45
46             Random r = new Random();
47
48             EntityManagerFactory emf = Persistence.createEntityManagerFactory("dados");
49
50             EntityManager em = emf.createEntityManager();
51
52             for(i = 1; i < 100001; i++) {
53
54                 x = r.nextInt(100000000);
55                 Endereco ma = new Endereco("rua utf");
56                 Usuario main = new Usuario("Tony stark",x, ma);
57
58                 em.getTransaction().begin();
59                 em.persist(main);
60                 em.getTransaction().commit();
61
62             }
63
64             em.close();
65             emf.close();
```

Fonte: Autor (2020).

A variável *x* assumirá o valor da variável *r* que será gerado de forma aleatória através de uma função randômica. 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 colocadas em seus parâmetros. Na classe endereço a descrição do nome da rua e na classe usuário com o nome do indivíduo, o valor aleatório de x e a variável endereço que tem como objetivo referenciar a classe endereço como parâmetro no usuário. Logo as funções do JPA iniciam a persistência no banco de dados.

Figura 4 - Função atualizar.

```
97                  for(i = 1; i < 100001; i++) {
98
99                      x = r.nextInt(1000000000);
100                     Usuario usuario = em.find(Usuario.class, i);
101                     usuario.setNome("Steve Rogers");
102                     usuario.setNumero(x);
103
104                     em.merge(usuario);
105
106                     em.getTransaction().begin();
107                     em.getTransaction().commit();
108
109                 }
110
111                 em.close();
112                 emf.close();
```

Fonte: Autor (2020).

Na figura 4 a programação utiliza a função *for* para realizar a passagem em todos os usuários cadastrados no banco e o comando *find* procura através da classe para identificar cada id cadastrado.

O comando merge irá criar um novo objeto que será o usuário e dessa forma o novo objeto irá mudar o nome e o número do usuário cadastrado com o respectivo *id* encontrado pelo método *find*. Assim irá manter as outras informações persistidas no banco que já foi preenchido.

Figura 5 - Função deletar.

```
149         for(i = 1; i < 100001; i++) {
150
151             Usuario usuario = em.find(Usuario.class, i);
152             Endereco end = em.find(Endereco.class, i);
153
154                 em.getTransaction().begin();
155                 em.remove(usuario);
156                 em.remove(end);
157                 em.getTransaction().commit();
158
159         }
160
161         em.close();
162         emf.close();
```

Fonte: Autor (2020).

Antes de iniciar a remoção, os objetos devem ser encontrados usando a função *find* que retornará à informação do objeto. Após encontrar o *id*, a função *remove* é iniciada excluindo fisicamente o objeto dentro da entidade e todos os dados removidos do banco.

A função inserir do modelo não-relacional é baseado em um documento *json*. O mongoDB irá gerar um *id* automático para cada documento. Neste caso usando um *for* que crie uma variável *id* onde o algoritmo fornece um número.

Figura 6 - Função Inserir NoSQL.

```
46          coll.drop();
47
48          Random r = new Random();
49
50          for ( i = 0; i < op; i++) {
51
52              x = r.nextInt(100000000);
53              Document doc = new Document("Id", i)
54                      .append("Nome", "nomeusuario")
55                      .append("Numero", x)
56                      .append("Endereco", "Rua Brasil");
57
58              coll.insertOne(doc);
59
60          }
61
62          mongoClient.close();
```

Fonte: Autor (2020).

Iniciando a função *drop* do mongoDB ele proporciona que não tenha nenhum dado antes da inserção que modifique a sequência. O método *random* também foi usado para gerar um valor aleatório e armazenar no documento.

O documento é inserido como parâmetro onde as informações serão adicionadas dentro do banco. Após o preenchimento do parâmetro a função interna do mongoDB *InsertOne* é chamada e tem por parâmetro o *document* para realizar a inserção.

Figura 7. Função atualizar no NoSQL.

```
99          Random r = new Random();
100
101         for ( i = 0; i < op; i++) {
102
103             x = r.nextInt(100000000);
104             coll.updateOne(eq("Nome", "Jean"), set("Nome", "Carlos"));
105             coll.updateOne(eq("Numero", x), set("Numero", x));
106             coll.updateOne(eq("Endereco", "Rua dos alfineiros"), set("Endereco", "Rua Principal"));
107             System.out.println(i);
108
109         }
110
111         mongoClient.close();
```

Fonte: Autor (2020).

A função atualizar utiliza parâmetros para realizar a atualização, assim como para a função do modelo relacional. A função interna do mongoDB *updateOne* implica que apenas um item pode ser alterado por vez. Assim utilizando a função *for* para que possa fazer alteração para todos os documentos existentes com as informações passadas como parâmetro.

Figura 8. Função deletar no NoSQL.

```
150         for ( i = 0; i < op; i++) {
151
152             DeleteResult deletar = coll.deleteMany(eq("Id", i));
153             System.out.println(i);
154
155         }
156
157         mongoClient.close();
```

Fonte: Autor (2020).

Deletar é a parte simples de todo o contexto NoSQL, utilizando uma função do mongoDB que tem a funcionalidade de excluir todo o banco de dados ou apagar apenas a informação contida nele. Dentro das funções foi utilizado uma função interna do Java chamada de *System.currentTimemillis*, que calcula o tempo de execução do algoritmo do início ao fim. Criando duas variáveis tempoInicial e tempoFinal para calcular o tempo de execução do algoritmo. No final, três variáveis entram para realizar a conversão do resultado da média do tempo para segundos, minutos ou horas.

## RESULTADOS E DISCUSSÕES

Com o algoritmo completo, foi iniciado os testes em cada banco de dados. Iniciando com poucos dados até a cem mil dados. Para comparar a efetividade dos bancos para as funções foi rodado o algoritmo três vezes em cada banco para realizar a média simples.

Tabela 1 - Tempo médio das inserções.

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

Fonte: Autor (2020).

Na tabela 1 o NoSQL apresentou mais eficiência pelo fato de não precisar consultar tabelas, como ocorre no modelo relacional. Pois o modelo não-relacional preenche todos os dados juntos diferente do modelo relacional que precisa criar uma relação entre tabelas e os dados nelas persistidos. Na atualização o modelo relacional utilizou mais tempo que o não-relacional, pelo fato de ter todo processo de buscar chaves estrangeiras relacionadas com a chave primária de cada tabela.

Tabela 2 - Tempo médio das atualizações.

| SGBDS | Cem dados | Mil dados | Dez mil dados | Cem mil dados |
|---|---|---|---|---|
| MySQL | 10,33 s | 1,02 min | 10,1 min | 3,3 horas |
| PostgreSQL | 5,06 s | 10,33 s | 2,32 min | 2,27 horas |
| MongoDB | 1,33 s | 3 s | 1,10 min | 1,34 horas |

Fonte: Autor (2020).

Como mostra na tabela 2 o NoSQL por 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*.

Tabela 3 - Tempo médio da exclusão.

| SGBDS | Cem dados | Mil dados | Dez mil dados | Cem mil dados |
|---|---|---|---|---|
| **MySQL** | 9,33 s | 58 s | 9,55 min | 1,32 horas |
| **PostgreSQL** | 9,33 s | 10 s | 35 s | 13 min |
| **MongoDB** | 2 s | 2 s | 31 s | 47 min |

Fonte: Autor (2020).

Na tabela 3 temos a função deletar que demonstra um comportamento inesperado, pois o PostgreSQL foi mais rápido na exclusão dos 100 mil dados que os outros bancos. O comportamento é facilmente explicado pois 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.

## CONCLUSÃO

Os resultados dos testes nos quadros anteriores, nota-se uma vantagem do modelo não-relacional em relação ao modelo relacional. O NoSQL mostrou-se mais efetivo nas funções, obtendo números produtivos. O efeito causado pelo NoSQL é que ele faz todo o processo sem necessitar de consultas dentro de si mesmo como ocorre no modelo relacional entre chave primária e estrangeira.

Para realizar o uso dos modelos apresentados, é necessário realizar um estudo de vantagem e desvantagem de cada banco para determinada realidade. Mesmo que neste projeto o NoSQL demonstrou maior eficiência, não significa que ele terá o mesmo comportamento. Trata-se de que o modelo não-relacional não veio para substituir, mas sim para complementar.

A aplicação implementada atingiu os objetivos e resultados esperados. Demonstrando a forma de como o modelo relacional agiu em relação ao não-relacional. Uma forma de contribuir com o projeto para aprimorar a pesquisa seria usar configurações iguais a todos os modelos, buscando diferentes bancos além dos tradicionais e testar em diferentes sistemas operacionais.

## AGRADECIMENTOS

Diante deste momento agradeço a Dra. Professora Kelyn Schenatto pela oportunidade deste projeto em conjunto com a Universidade Tecnológica Federal do Paraná pelo apoio e concessão de bolsa e a um amigo em especial Felipe Nazário que concedeu todo apoio no início e durante todo o projeto.

## REFERÊNCIAS

SILBERSCHATZ, A.; KORTH, H. F., SUDARSHAN. S. Sistema de banco de dados, Rio de Janeiro: Editora Elsevier, 2006. Disponível em: https://mirrors.xtom.com/osdn//sfnet/t/te/tecnologia/Sistema.de.Banco.de.Dados.Abrahan.Silberschatz.pdf. Acesso em: 31 ago. 2020.

PRAMOD J. SADALAGE, MARTIN FOWLER. NoSQL Essencial - Um guia conciso para o mundo emergente da persistência poliglota. São Paulo: Novatec Editora Ltda.

2013. Disponível em: https://s3.novatec.com.br/capitulos/capitulo-9788575223383.pdf. Acesso em: 31 ago. 2020.