UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS DE DOIS VIZINHOS
CURSO DE ESPECIALIZAÇÃO EM CIÊNCIA DE DADOS

JHONATAN MARTINELLO

COMPARAÇÃO DE PERFORMANCE ENTRE BANCO DE DADOS RELACIONAL POSTGRESQL E NOSQL MONGODB

TRABALHO DE CONCLUSÃO DE CURSO

DOIS VIZINHOS
2021

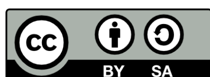
JHONATAN MARTINELLO

COMPARAÇÃO DE PERFORMANCE ENTRE BANCO DE DADOS RELACIONAL POSTGRESQL E NOSQL MONGODB

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados da Universidade Tecnológica Federal do Paraná, como requisito para a obtenção do título de Especialista em Ciência de Dados.

Orientador: Prof. Dr. Ives Renê Venturini Pola

DOIS VIZINHOS
2021



4.0 Internacional

Esta licença permite remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es) e que licenciem as novas criações sob termos idênticos. Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

JHONATAN MARTINELLO

COMPARAÇÃO DE PERFORMANCE ENTRE BANCO DE DADOS RELACIONAL POSTGRESQL E NOSQL MONGODB

Trabalho de Conclusão de Curso de Especialização apresentado ao Curso de Especialização em Ciência de Dados da Universidade Tecnológica Federal do Paraná, como requisito para a obtenção do título de Especialista em Ciência de Dados.

Data de aprovação: 27/novembro/2021

Ives Renê Venturini Pola
Doutorado
Universidade Tecnológica Federal do Paraná - Câmpus Pato Branco

Anderson Chaves Carniel
Doutorado
Universidade Federal de São Carlos

Marcelo Teixeira
Doutorado
Universidade Tecnológica Federal do Paraná - Câmpus Pato Branco

DOIS VIZINHOS
2021

RESUMO

MARTINELLO, Jhonatan. Comparação de performance entre Banco de Dados relacional PostgreSQL e NoSQL MongoDB. 2021. 44 f. Trabalho de Conclusão de Curso – Curso de Especialização em Ciência de Dados, Universidade Tecnológica Federal do Paraná. Dois Vizinhos, 2021.

O armazenamento de dados sempre foi de suma importância em qualquer projeto ou aplicação, e para isso os bancos de dados relacionais são amplamente utilizados. Porém com a popularização da internet é crescente o aumento do volume de dados oriundos de redes sociais, aplicações mobile e web, dando força ao uso de banco de dados NoSQL por novos projetos. Entretanto, ainda se questiona muito sobre qual seria o desempenho do banco de dados relacional ao trabalhar com essas grandes quantidades de dados. Este trabalho tem como objetivo comparar a *performance* entre o banco de dados relacional PostgreSQL e o NoSQL MongoDB, utilizando grandes quantidades de dados usualmente armazenados em bancos NoSQL. Foram realizadas consultas no Banco de dados relacional PostgreSQL e no banco de dados NoSQL MongoDB a fim de comparar a velocidade de execução, utilizando-se de recursos nativos de ambos os bancos a fim de melhorar suas performances. Testes realizados comprovam que o banco de dados PostgreSQL com o correto uso de seus recursos teve resultados superiores ao MongoDB.

Palavras-chave: Banco de dados Relacional. NoSQL. PostgreSQL. MongoDB.

ABSTRACT

MARTINELLO, Jhonatan. PostgreSQL Relational Database and NoSQL MongoDB: Performance Comparison. 2021. 44 f. Trabalho de Conclusão de Curso – Curso de Especialização em Ciência de Dados, Universidade Tecnológica Federal do Paraná. Dois Vizinhos, 2021.

Data storage has always been of paramount importance in any project or application, and for this, relational databases are widely used. However, with the popularization of the internet, the increase in the volume of data from social networks, mobile and web applications is increasing, giving strength to the use of NoSQL database by new projects. However, there is still a lot of questioning about how the relational database would perform when working with these large amounts of data. This article aims to compare the performance between PostgreSQL relational database and NoSQL MongoDB, query performance of data usually stored in NoSQL databases, manipulating large amounts of data. Queries were performed on the PostgreSQL relational database and on the NoSQL MongoDB database in order to compare the execution speed, using native resources from both databases to improve their permanence. Tests carried out prove that the PostgreSQL database with the correct use of its resources had better results than MongoDB.

Keywords: Relational Database. NoSQL. PostgreSQL. MongoDB.

LISTA DE FIGURAS

Figura 1 – Consulta PostgreSQL sem Filtros. 22
Figura 2 – Consulta PostgreSQL em Campo Texto. 22
Figura 3 – Consulta PostgresSQL em Campo Numérico. 23
Figura 4 – Consulta PostgresSQL em Campo Data. 24
Figura 5 – Consulta MongoDB sem Filtros. 25
Figura 6 – Consulta MongoDB em Campo Texto. 26
Figura 7 – Consulta MongoDB em Campo Numérico. 27
Figura 8 – Consulta MongoDB em Campo Data. 28
Figura 9 – Consulta PostgreSQL com Índice em Campo Texto. 29
Figura 10 – Consulta PostgreSQL com Índice em Campo Númerico. 30
Figura 11 – Consulta PostgreSQL com Índice em Campo Data. 30
Figura 12 – Consulta MongoDB com Índice em Campo Texto. 32
Figura 13 – Consulta MongoDB com Índice em Campo Númerico. 33
Figura 14 – Consulta MongoDB com Índice em Campo Data. 34
Figura 15 – Consulta PostgreSQL com JSON Índexado em Campo Texto. 35
Figura 16 – Consulta PostgreSQL com JSON Índexado em Campo Númerico. 36
Figura 17 – Consulta PostgreSQL com JSON Índexado em Campo Númerico. 37

LISTA DE TABELAS

Tabela 1 – Desempenho de Execução das Consultas 39

SUMÁRIO

1 – INTRODUÇÃO 9
1.1 PROBLEMA DE PESQUISA 10
1.2 OBJETIVOS 10
1.2.1 Objetivo Geral 10
1.2.2 Objetivos Específicos 10
1.3 JUSTIFICATIVA 10
1.4 MATERIAS E MÉTODOS 11
1.5 ORGANIZAÇÃO DO TRABALHO 11

2 – REVISÃO DE LITERATURA 12

3 – MATERIAS E MÉTODOS 16
3.1 JSON 16
3.2 POSTGRESQL 16
3.3 MONGODB 17
3.4 DBEAVER 17
3.5 ROBO 3T 18

4 – RESULTADOS 19
4.1 CONSULTAS POSTGRESQL SEM A UTILIZAÇÃO DE ÍNDICES 21
4.1.1 Consulta Simples sem Filtro 21
4.1.2 Consulta com Filtro em Campo Texto 22
4.1.3 Consulta com Filtro em Campo Numérico 23
4.1.4 Consulta com Filtro em Campo de Data 23
4.2 CONSULTAS MONGODB SEM UTILIZAÇÃO DE ÍNDICES 24
4.2.1 Consulta Simples sem Filtro 24
4.2.2 Consulta com Filtro em Campo Texto 25
4.2.3 Consulta com Filtro em Campo Numerico 26
4.2.4 Consulta com Filtro em Campo de Data 27
4.3 CONSULTAS POSTGRESQL UTILIZANDO ÍNDICES 28
4.3.1 Consulta com Filtro em Campo Texto 29
4.3.2 Consulta com Filtro em Campo Numerico 29
4.3.3 Consulta com Filtro em Campo de Data 30
4.4 CONSULTAS MONGODB UTILIZANDO ÍNDICES 31
4.4.1 Consulta com Filtro em Campo Texto 31
4.4.2 Consulta com Filtro em Campo Numerico 32

4.4.3 Consulta com Filtro em Campo de Data 33
4.5 UTILIZANDO JSON INDEXADO NO POSTGRESQL 34
4.5.1 Consulta com Filtro em Campo Texto 35
4.5.2 Consulta com Filtro em Campo Numerico 35
4.5.3 Consulta com Filtro em Campo de Data 36
4.6 ANÁLISE DOS RESULTADOS OBTIDOS 37

5 – CONCLUSÃO 41
5.1 LIMITAÇÕES 41
5.2 CONSIDERAÇÕES FINAIS 41

Referências 43

1 INTRODUÇÃO

O armazenamento e gerenciamento eficiente de dados sempre foi de suma importância em qualquer projeto ou aplicação. Para isso os bancos de dados relacionais são amplamente utilizados. Segundo (SOARES; BOSCARIOLI, 2013) bancos de dados relacionais são de fácil manipulação e possuem recursos que garantem a integridade, segurança e confiabilidade dos dados, além da durabilidade. No entanto, com a crescente popularização da internet, grandes quantidades de dados oriundos de diversas fontes como redes sociais, aplicações mobile e web surgiram, trazendo consigo um novo desafio, de tratar e manipular esses diferentes tipos de dados em grande escala. A partir deste desafio vieram à tona os bancos de dados não somente relacionais, que passaram a ser conhecidos como NoSQL.

Segundo (Pramod J.; Martin, 2019) o conceito de banco de dados NoSQL foi criado devido a uma necessidade de manipulação de volumes maiores de dados, e tem como vantagem possuir uma arquitetura diferenciada, sem esquema predefinido, trocando a ideia de consistência, própria do modelo relacional, por propriedades que desenvolvem melhor desempenho e escalabilidade. De acordo com (STRAUCH; SITES; KRIHA, 2011) a maioria dos bancos de dados NoSQL, em contraste com os sistemas de gerenciamento de banco de dados relacionais, são projetados para trabalhar em escalabilidade horizontal e não dependem de hardware altamente disponível, além de serem projetados para armazenar estruturas de dados mais semelhantes às das linguagens de programação orientadas a objetos.

Na prática, a utilização de bancos de dados NoSQL para o armazenamento e manipulação de grandes quantidades de dados tende a ser mais efetiva, já que ele otimiza as consultas e facilita o tratamento dos dados, de acordo com (SOARES; BOSCARIOLI, 2013). Mesmo com as aparentes vantagens e facilidades apresentadas pelo modelo NoSQL, podemos nos questionar: qual seria o desempenho do banco de dados relacional ao trabalhar com essas grandes quantidades de dados? Na visão de (WELBORN, 2019), fazendo-se uso de ferramentas como índices, cujo objetivo é acelerar a execução de consultas, podemos apresentar melhorias no desempenho dos bancos de dados relacionais.

Assim, neste trabalho estimamos que ao se fazer o uso de uma solução JSON indexada no PostgreSQL, inclinaríamos a obter vantagens que aproximam o banco de dados relacional à eficiência apresentada em um NoSQL baseado em documentos porém com as vantagens oferecidas pelo Sistema Gerenciador de Bancos de Dados (SGBD) relacional.

Conforme (HALLIDAY, 2018) o PostgreSQL tem adicionado suporte JSON nativo em suas novas versões, aumentando a capacidade de armazenar JSON como binário JSON ou JSONB, o qual adiciona um pouco de sobrecarga ao inserir dados, mas oferece um grande benefício quando esses dados são consultados por índices.

1.1 PROBLEMA DE PESQUISA

A utilização de bancos de dados NoSQL para o armazenamento e manipulação de grandes quantidades de dados tende a ser mais efetiva. Porém mesmo com as aparentes vantagens apresentadas pelo modelo NoSQL, podemos nos questionar: qual seria o desempenho do banco de dados relacional ao trabalhar com essas grandes quantidades de dados?

1.2 OBJETIVOS

O objetivo deste trabalho é fazer uma comparação de desempenho entre o banco de dados relacional PostgreSQL e o NoSQL MongoDB, em condições equivalentes de manipulação, em única instância, beneficiando-se da utilização de ferramentas de indexação disponibilizadas pelo PostgreSQL, como JSON indexados, a fim de verificar o comportamento relativo de eficiência entre as ferramentas nestas condições. Buscou-se medir o desempenho do banco de dados relacional PostgreSQL em comparação ao NoSQL MongoDB, executando consultas em uma mesma base de dados que foi convertida a fim de ser utilizada em ambos sistemas, tendo assim resultados mais confiáveis trabalhando com dados semelhantes. Foram utilizados recursos nativos dos PostgreSQL como índices e suporte a JSON visando melhorar a sua *performance*, e de mesmo modo utilizado índices nativos do MongoDB. Experimentos na forma de consultas foram realizadas em diversos campos dos dados mostrando as suas estatísticas e tempo de execução, apresentando um comparativo do desempenho nos cenários observados.

1.2.1 Objetivo Geral

Fazer uma análise comparatória entre o desempenho do banco de dados relacional PostgreSQL e o NoSQL MongoDB, trabalhando e manipulando dados em condições equivalentes (sem *sharding*).

1.2.2 Objetivos Específicos

- Análise de recuperação de dados simples no PostgreSQL frente ao MongoDB.
- Análise de recuperação de dados JSON comparando a eficiência em relação as buscas em ambos sistemas;
- Utilizar e medir eficiência de índices específicos no PostgreSQL para agilizar o resultado das consultas.

1.3 JUSTIFICATIVA

Fazendo-se uso de ferramentas como JSON indexados e índices específicos no PostgreSQL podemos apresentar melhorias no desempenho do banco de dados relacional. Tendo isso em vista, neste trabalho estimamos que ao se fazer o uso de uma solução JSON indexada

no PostgreSQL, inclinaríamos a obter vantagens que aproximam ou até superam o banco de dados relacional a eficiência apresentada em um NoSQL.

1.4 MATERIAS E MÉTODOS

Para o sesemvolvimento do projeto é feita a utilização da ferramenta gratuita de banco de dados multiplataforma para desenvolvedores DBeaver, onde serão feitas todas as manipulações no banco de dados relacional PostegreSQL, e a ferramenta Robo 3T que oferece suporte ao MongoDB, fazendo-se assim uso dessa ferramenta para manipulação dos dados do banco NoSQL.

PostgreSQL: PostgreSQL é um poderoso sistema de banco de dados relacional objeto de código aberto com mais de 30 anos de desenvolvimento ativo que lhe rendeu uma forte reputação de confiabilidade, robustez de recursos e desempenho (POSTGRESQL, 2021).

MongoDB: MongoDB é um banco de dados de propósito geral, baseado em documentos, criado para desenvolvedores de aplicativos modernos (MONGODB, 2021).

Dbeaver: Ferramenta gratuita de banco de dados multiplataforma para desenvolvedores, administradores de banco de dados, analistas e todas as pessoas que precisam trabalhar com bancos de dados (DBEAVER, 2021). No projeto será utilizado para manipulação e controle dos dados do SGBD PostgreSQL.

Robo 3T: Uma interface gratuita para os amantes do MongoDB. Leve e de código aberto, oferece suporte a várias plataformas e incorpora a linha de comando mongo em sua aplicação para fornecer interação para os usuários mais avançados.(FERNANDES, 2021).

1.5 ORGANIZAÇÃO DO TRABALHO

A seguir são apresentadas as seções de revisão de literatura, onde se detalha o que há de estudos na área do trabalho apresentado, trazendo também a visão de outros autores sobre o assunto. Serão apresentadas as seções de materiais e métodos, onde se explica as ferramentas e a metodologia utilizada no desenvolvimento do trabalho, seguida da seção de resultados, onde tudo o que foi demostrado é posto em prática a fim de obter dados representativos da eficácia do tema proposto no trabalho, e por fim as conclusões finais que podemos ter com o desenvolvimento do projeto.

2 REVISÃO DE LITERATURA

Esta seção provê uma visão geral dos trabalhos relacionados a comparação de desempenho entre banco de dados relacionais e bancos de dados NoSQL, focando nas técnicas utilizadas para avaliar a eficácia de cada banco, e nos resultados obtidos.

Bancos de dados são de suma importância para o armazenamento de informações, segundo (ELMASRI; NAVATHE, 2011) os bancos de dados e os sistemas de bancos de dados se tornaram componentes essenciais no cotidiano da sociedade moderna, tanto que no decorrer do dia, a maioria de nós se depara com atividades que envolvem alguma interação com os bancos de dados.

Os modelos de bancos de dados relacionais surgiram para sanar essa necessidade de armazenamento de informações, de acordo com (ROB; CORONEL, 2011) os bancos de dados relacionais trouxeram a capacidade de armazenar, acessar e alterar dados de forma rápida e fácil em computadores de baixo custo. Vários SGBDs foram desenvolvidos com o passar dos anos, como: MySQL, MariaDB, Oracle, SqlServer, Firebird e PostgreSQL. Neste trabalho será utilizado o PostgreSQL, por ser um SGBD de código aberto que conta com diversos recursos avançados.

Com a evolução internet e o aumento das grandes quantidades de dados, os modelos relacionais se tornaram um tanto quanto limitados, foi então que surgiram os bancos de dados NoSQL, que segundo (MONIRUZZAMAN; HOSSAIN, 2013) são bancos de dados não relacionais e distribuídos, projetados para armazenamento de dados em grande escala. De acordo com (NAYAK; PORIYA; POOJARY, 2013) os Bancos de dados NoSQL são divididos em 5 categorias, são elas: orientado a Chave-Valor, orientado a Colunas, orientado a Documentos, orientado a Grafos e orientado a Objetos. Neste trabalho será utilizado o modelo orientado a Documentos, mais especificamente o banco de dados MongoDB, que é um banco de dados de código aberto, gratuito, de alta *performance*, e muito difundido entre os modelos NoSQL.

Apresentando um conceito mais geral, os autores (NAYAK; PORIYA; POOJARY, 2013) discutem sobre as principais vantagens e desvantagens do modelo NoSQL comparado ao relacional. Como principais vantagens do NoSQL eles trazem a ampla gama de modelos de dados, a escalabilidade, a não necessidade de administradores de banco de dados, o fato de ser mais rápido, eficiente e flexível, e a evolução em ritmo muito alto desse modelo. Já suas desvantagens apresentadas são o fato de ser ainda imaturo, sem uma linguagem de consulta padrão, sem interface padrão e de difícil manutenção. Eles também trazem a tona o fato de alguns modelos NoSQL não garantirem o conceito ACID, que representa às quatro propriedades de transação de um sistema de banco de dados: Atomicidade, Consistência, Isolamento e Durabilidade. As principais vantagens do NoSQL se dão quanto ao desempenho, a diversidade de dados que podem ser armazenados e a evolução rápida. Porém quando comparado ao modelo relacional, perde principalmente em confiabilidade e segurança, além de ter maior complexidade

de manutenção.

Para (KUNDA; PHIRI, 2017) as demandas dos aplicativos modernos, especialmente na web 2.0, 3.0 e big data, tornaram o NoSQL um banco de dados de popular escolha. Bancos de dados relacionais são baseados no modelo ACID. Atomicidade garante a integridade das transações, Consistência fornece estabilidade em um banco de dados, Isolamento garante a independência de várias transações que são executadas ao mesmo tempo e a Durabilidade garante que transações armazenadas não mudam de estado, mesmo na presença de fracasso. ACID fornece consistência e disponibilidade tão fortes propriedades que tornaram os bancos de dados relacionais populares. NoSQL, em outro lado é baseado no BASE, que traz possibilidades de os dados serem parcialmente disponível quando algumas partes do banco de dados distribuído não estão em operação ou não podem ser alcançados, portanto, o termo basicamente disponível. O Soft State permite que os dados variem ao longo do tempo com ou sem entrada. Eventualmente, a Consistent garante que os dados se tornarão consistentes no futuro e não imediatamente após suma operação. O BASE dá ao NoSQL a capacidade de escalar facilmente, oferece melhor desempenho e maiores níveis de disponibilidade. Os autores comparam o banco de dados relacional e o modelo NoSQL em 10 Itens:

A. Fechado e de código aberto: Existem bancos de dados relacionais em código aberto como MySQL, MariaDB e PostreSQL e plataformas proprietárias como Oracle e SQLServer. Muitos modelos NoSQL são de código aberto, como MongoDB, CouchDB e Cassandra, oferecendo maiores oportunidades para pesquisadores investigarem recursos de um banco de dados e oferecendo mais economia no valor de armazenamento para usuários que não podem pagar banco de dados proprietários.

B. Escalabilidade: Bancos de dados relacionais geralmente necessitam de um servidor para torná-los mais eficientes, isso aumenta a quantidade de esforço necessária de administradores na atualização de bancos de dados relacionais. Esse modelo também enfrenta desafios em termos de limitações de hardware, por exemplo a quantidade máxima de RAM ou armazenamento secundário que é suportado pelo hardware tem um valor que é determinado pelos fabricantes do hardware. Isso significa que os bancos de dados relacionais têm a capacidade de escalar mas sempre haverá um limite no nível de escalabilidade, uma vez que é determinado pelo hardware. Para oferecer escalabilidade, NoSQL exigem o uso de servidores convencionais, ou seja, escalonamento horizontal. O dimensionamento horizontal não é significativamente afetado por limitações de hardware, porque menores, mais baratas e menos poderosas máquinas servidoras podem ser combinadas para oferecer maior níveis de escalabilidade em vez de ter um servidor caro. Essa capacidade torna a implementação fácil e escalável.

C. Custo: Bancos de dados relacionais melhores são pagos e portanto, requerem grandes quantidades de investimento de organizações e indivíduos que desejam se beneficiar de suas características avançadas, isso faz com que os bancos de dados relacionais sejam uma abordagem cara para armazenamento de dados. NoSQL é principalmente código aberto, o que o torna uma alternativa mais barata em relação aos Bancos de dados relacionais. A capacidade de

usar máquinas virtuais como servidores commodity reduzem ainda mais o custo de manutenção de um Banco de dados NoSQL, tornando o NoSQL um banco de dados atraente e barato.

D. Volume e variedade de dados: Os aplicativos da Internet aumentaram o volume de informações que os bancos devem manipular, com o surgimento da web 2.0 e 3.0 e a chegada de big data o volume e a variedade de dados que devem ser armazenados cresceu consideravelmente. Os bancos relacionais não conseguiram lidar com os grandes volumes de dados provenientes dessas fontes. NoSQL é excelente em lidar com essa situação, tornando-o adequado para aplicações intensivas de Internet. Isso pode ser visto em empresas como Google, Facebook e Yahoo que têm migrado para NoSQL.

E. Disponibilidade: Bancos de dados relacionais geralmente tem a disponibilidade mais limitada porque tem maior escalabilidade. A natureza distribuída do NoSQL torna uma melhor escolha de fornecer disponibilidade aos usuários o tempo todo, mesmo na presença de falhas de hardware. Os usuários estão garantidos de acesso contínuo ao banco de dados, independentemente das falhas com o sistema.

F. Desempenho: Bancos de dados relacionais requerem muito mais tempo para processar as informações tornando-os lentos em comparação com NoSQL, que são rápidos no processamento. Outra vantagem do NoSQL é que ele recupera dados da memória volátil, ao contrário Bancos de dados relacionais que recuperam dados de memórias não voláteis. Experimentos foram conduzidos para testar desempenho de bancos de dados NoSQL e relacionais. Uma comparação do banco de dados relacional MongoDB mostrou que seu desempenho foi melhor para leitura e consultas, enquanto o SQL só teve um bom desempenho na atualização.

G. Complexidade: Bancos de dados relacionais criam dados complexos em circunstâncias em que os dados a serem armazenados pelos usuários são difíceis de converter em tabelas. A ênfase no armazenamento estruturado dos dados em bancos de dados relacionais trazem essa complexidade. NoSQL pode armazenar dados semiestruturados e não estruturados, sendo assim fornecem flexibilidade necessária para apoiar múltiplas variedades de dados em seu estado bruto, sem perda de formação. Por exemplo, converter uma gravação de áudio de reclamação do cliente para texto a fim de armazenamento em bancos de dados relacionais leva a uma perda de informação sobre o humor do cliente. Tais informações podem ser preservadas no NoSQL, pois a gravação pode ser armazenada em seu estado sem conversão.

H. Linguagem de consulta: Bancos de dados relacionais têm uma base sólida e bem documentada sobre SQL que é a única linguagem de manipulação que todas as bases relacionais usam. No entanto, existem pequenas variações de implementações de SQL para os vários bancos de dados em uso. Os bancos relacionais são mais populares entre os desenvolvedores por causa do aprendizado mais curto em qualquer implementação, por ter uma linguagem padrão. Esse fundamento é precário no NoSQL, pois cada aplicação tem sua própria linguagem de manipulação de dados, que requer que desenvolvedores gastem tempo aprendendo a desenvolver em tipos diferentes de modelos NoSQL daqueles que eles são acostumados. Cada implementação NoSQL, fornece suas próprias consultas exclusivas.

I. Consistência: Bancos de dados relacionais oferecem maior consistência porém para isso precisam sacrificar a disponibilidade. Forte consistência é boa para fornecer uma visão uniforme dos dados imediatamente após a execução das operações. NoSQL fornece maior disponibilidade, mas tem baixa consistência.

J. Segurança: Bancos de dados relacionais enfrentam alguns desafios de segurança, como injeção de SQL em scripts e sites. Apesar destes desafios SQL tem fortes mecanismos de segurança que são usados para proteção, como autenticação, autorização, criptografia, integridade e auditoria. Em NoSQL, a segurança não é parte do banco de dados, mas é gerenciado por middleware, isso o deixa mais vulnerável a ataques. Além disso, os mecanismos de segurança implementados em middleware devem ser implementados de uma forma que não comprometa a escalabilidade e desempenho.

Os autores (SECCO, 2017) apresentam uma análise comparativa realizada entre os bancos Apache NoSQL Cassandra e o banco de dados relacional PostgreSQL, verificando a eficácia ao utilizá-los para armazenar dados de uma aplicação para controle de estoque de duas empresas distintas de pequeno porte. Concluíram que o PostgreSQL é um banco destacado pelo código aberto e por ser um dos mais avançados em termos de recursos, com administração fácil e dinâmica, entretanto conta com muitas regras de relacionamento entre as tabelas, ocasionando demora na busca das informações. Já o NoSQL Cassandra oferece uma alternativa flexível e escalável, por possuir disponibilidade, rendimento de gravação alto, bom rendimento de leitura, suporte para procura por índices secundários, consistência ajustável e suporte para replicação. Como ponto negativo destacam a dificuldade na consulta de dados, pelo fato de todas as linhas estarem armazenadas em um único nó.

Percebe-se que os autores utilizaram ambientes equivalentes para fazer a análise comparativa dos bancos de dados, e observaram consideráveis vantagens na utilização do NoSQL Cassandra, tendo como maior empecilho sua complexidade na consulta de dados. Quanto ao banco de dados PostgreSQL, tiveram problemas quanto sua maior lentidão no processamento dos dados. Uma possível melhoria no desempenho do banco de dados relacional PostgreSQL seria fazer a utilização das vantagens que índices e JSON indexados oferecem para aceleração das consultas.

Fica claro que os autores frisam muitas vantagens ao modelo NoSQL, trazendo a tona sua maior escalabilidade, disponibilidade e flexibilidade, porém também apresentam que itens como linguagem de consultas, consistência e segurança ainda são muito mais vantajosas no modelo relacional. Nota-se que nem todos os recursos do modelo relacional são utilizados, e que aplicando o uso das ferramentas disponibilizadas pelo modelo relacional os resultados poderiam ser diferentes.

3 MATERIAS E MÉTODOS

Para realizar a análise comparativa entre o banco de dados Relacional PostgreSQL e o NoSQL MongoDB, foi utilizado uma base de dados contendo logs de conexão e desconexão de internet. A mesma base de dados foi modificada a fim de ser utilizada tanto no banco de dados relacional PostgreSQL como também no banco NoSQL MongoDB, tendo assim resultados mais confiáveis trabalhando com dados semelhantes.

Foram feitas consultas de teste divididas em três etapas, a primeira sem a utilização de índices nos bancos de dados, posteriormente feita a criação e índices e executadas novas consultas de teste, e por fim utilizando-se de JSON indexado no PostgreSQL a fim de verificar novamente sua performance.

Os seguintes materiais foram utilizados para a execução da análise de performance:

3.1 JSON

A linguagem JSON é, de acordo com (JSON, 2021) leve de troca de informações entre sistemas, fácil para humanos lerem e escreverem, e fácil para as máquinas analisarem e gerarem. É baseado em um subconjunto do Padrão de Linguagem de Programação JavaScript, porém é um formato de texto completamente independente da linguagem.

A utilização de JSON permite uma maior velocidade na execução e transporte de dados, com um arquivo com tamanho reduzido. é utilizado atualmente por empresas como: Google, Facebook, Yahoo!, Twitter.

3.2 POSTGRESQL

Conforme (MILANI, 2008) o PostgreSQL é um SGBD relacional, utilizado para armazenar informações de soluções de informática em todas as áreas de negócios existentes, bem como administrar o acesso a estas informações.

Afirma (MILANI, 2008) que o postgres é um SGBD completo, com suporte a operações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), propriedades que garantem uma parte da qualidade dos serviços disponibilizados pelos bancos de dados.

O PostgreSQL possibilita a criação e uso de índices em suas tabelas, cujo objetivo conforme (WELBORN, 2019) é acelerar a execução de consultas, apresentando melhorias no desempenho. Conforme (HALLIDAY, 2018) o PostgreSQL tem suporte JSON nativo em suas novas versões, aumentando a capacidade de armazenar JSON como binário JSON ou JSONB, o qual oferece um grande benefício quando esses dados são consultados por índices.

3.3 MONGODB

Segundo (MONGODB, 2014) o MongoDB foi desenvolvido pela empresa 10gen no ano de 2007, e foi criado para o processamento de alto desempenho de grandes quantidades de dados. Ainda de acordo com (MONGODB, 2014) os dados são armazenados em documentos no MongoDB, o documento consiste em um conjunto ordenado de propriedades e não têm esquemas ou estruturas fixas, sendo assim, são preenchidos dinamicamente com os dados que são necessários.

O formato em que os documentos são salvos é BSON, JSON binário. BSON é caracterizado pela eficiência e economia espaço, podendo ser codificado e decodificado com alto desempenho. Independentemente do driver usado cada documento é mapeado internamente para uma construção BSON.

Em conformidade com (JUNIOR, 2017) pelo fato de ser orientado a documentos JSON o MongoDB pode modelar informações de modo muito mais natural, pois os dados podem ser aninhados em hierarquias complexas e ainda serem indexáveis e fáceis de buscar.

Dentre os bancos NoSQL o MongoDB é o mais utilizado atualmente, e fica em quinto lugar entre todos os banco de dados, incluindo relacionais, tendo inclusive maior representatividade do que o Oracle, um banco de dados relacional dos mais tradicionais e completos, conforme apresentado por (JUNIOR, 2017).

3.4 DBEAVER

De acordo com (DBEAVER, 2021) o DBeaver é uma ferramenta gratuita de banco de dados multiplataforma para desenvolvedores, administradores de banco de dados, analistas e todas as pessoas que precisam trabalhar com bancos de dados. Suporta todos os bancos de dados populares: MySQL, PostgreSQL, SQLite, Oracle, DB2, SQL Server, Sybase, MS Access, Teradata, Firebird, Apache Hive, Phoenix, Presto, etc. No projeto será utilizado para manipulação e controle dos dados do SGBD PostgreSQL.

DBeaver foi iniciado em 2010 como um projeto de hobby. Gratuito e de código aberto, incluindo recursos usados com frequência para desenvolvedores de banco de dados. O primeiro lançamento oficial foi em 2011 no Freecode. Rapidamente se tornou uma ferramenta popular na comunidade de código aberto.

DBeaver 3.x anunciou o suporte de bancos de dados NoSQL (Cassandra e MongoDB na versão inicial). Desde então, o DBeaver foi dividido nas edições Community e Enterprise. Enterprise Edition tem suporte para bancos de dados NoSQL, gerenciador de consultas persistentes e alguns outros recursos de nível corporativo. A versão EE não é de código aberto e requer a compra de uma licença.

3.5 ROBO 3T

Antes conhecido como Robomongo, o Robo 3T é um software gratuito multiplataforma para gerenciamento de banco de dados MongoDB de forma gráfica (ALEXANDRE, 2021).

Segundo (FERNANDES, 2021) é uma das mais populares interfaces gráficas gratuitas para MongoDB. Leve e de código aberto, oferece suporte a várias plataformas (Mac, Linux e Windows) e incorpora também a linha de comando mongo em sua aplicação para fornecer interação para os usuários mais avançados. É desenvolvido pela 3T Software..

Tem como principais recursos, a linha de comando incorporada, a interface assíncrona e sem bloqueios, além de suporte para MongoDB 4.0 ou superior (FERNANDES, 2021).

4 RESULTADOS

A seguir serão apresentados os resultados obtidos com a aplicação de operações de consultas em bases de dados no modelo de banco Relacional e NoSQL.

Antes de analisar a *performance* e trazer os resultados, torna-se imprescindível atentar-se a questões como critérios de escalonamento, consistência e disponibilidade de dados entre o PostgreSQL e NoSQL MongoDB. Quando se fala de escalabilidade o NoSQL MongoDB oferece uma grande vantagem em relação ao banco de dados PostgreSQL, pois tem mais flexibilidade, quanto se trata em um número acentuado de acesso de dados. Quando se trata em distribuir o banco de dados em várias máquinas, particionando os dados, o banco PostgreSQL deixa a desejar, pois obedece os critérios de normalização. Existem diferenças entre os bancos, enquanto o PostgreSQL prioriza a concentração dos dados em um servidor, o NoSQL MongoDB tem como objetivo a distribuição dos dados em diversos servidores. Outro aspecto importante entre os bancos de dados é o controle de concorrência, o PostgreSQL utiliza bloqueios para que um registro seja acessado somente por um usuário, o NoSQL MongoDB faz uso de outros recursos que permitem maior grau de concorrência.

A base de dados utilizada para os testes conta com pouco mais de um milhão e duzentos mil registros, oriundos de dados coletados de logs de conexão e desconexão de internet. A mesma base de dados foi modificada a fim de ser utilizada tanto no banco de dados relacional PostgreSQL como também no banco NoSQL MongoDB, tendo assim resultados mais confiáveis trabalhando com dados semelhantes. Antes de todas as consultas no PostgreSQL foi utilizado a função *explain analyse* que demostra uma visão geral do planejamento da consulta e seu tempo de execução, já no MongoDB foi utilizado a função *explain("executionStats")* que traz todas as estatísticas da consulta executada.

A seguir é apresentada a estrutura da tabela utilizada no banco de dados relacional PostgreSQL, podemos verificar que temos colunas com tipos de dados texto(varchar), numérico(int) e data(date).

```sql
CREATE TABLE radacct (
        radacctid serial4 NOT NULL,
        acctsessionid varchar(64) NULL,
        username varchar(64) NULL,
        acctdate date NULL,
        accttime time NULL,
        guid_sessao varchar(50) NULL,
        acctstartdate date NULL,
        acctstarttime time NULL,
        acctstopdate date NULL,
```

```
        acctstoptime time NULL,
        acctinputoctets int8 NULL,
        acctoutputoctets int8 NULL,
        nasportid int8 NULL,
        nasportidname varchar(100) NULL,
        nasipaddress varchar(25) NULL,
        framedipaddress inet NULL,
        calledstationid varchar(100) NULL,
        callingstationid varchar(50) NULL,
        acctterminatecause varchar(32) NULL,
        acctcount int4 NULL,
        acctfrequency time NULL,
        sessao_info varchar(200) NULL,
        framedipv6prefix inet NULL,
        delegatedipv6prefix inet NULL,
       CONSTRAINT radacct_pkey PRIMARY KEY (radacctid)
);
```

Podemos verificar abaixo também a formatação dos dados inseridos no banco de dados NoSQL MongoDB em JSON, onde não há uma tipagem de dados e sim todo o conteúdo é armazenado em um BSON (JSON Binário).

```
_id: objectId("617b33a36d0a34d8bd3340a1")
radacctid: 1
acctsessionid: "81a00171"
username: "adalberg-queluz182"
acctdate: 2018-05-07T00:00:00.000+00:00
accttime: 20
guid_sessao: "b2430d8b5f0c9e4b"
acctstartdate: 2018-05-07T00:00:00.000+00:00
acctstarttime: 17
acctstopdate: 2018-05-07T00:00:00.000+00:00
acctstoptime: 20
acctinputoctets: "33976149"
acctoutputoctets: "496163919"
nasportid: "15733229"
nasportidname: "ether4"
nasipaddress: "45.229.52.6"
framedipaddress: "100.64.32.142"
calledstationid: "service4"
callingstationid: "E0:3F:49:47:81:C3"
```

```
acctterminatecause: "NAS—Request"
acctcount: 10
acctfrequency: "00:15:01"
sessao_info: "Sessao inserida por pacote de acct STOP."
```

As operações foram feitas em um primeiro momento utilizando o banco de dados relacional nativo, sem nenhuma alteração ou adição de índices, e da mesma forma foi realizado com o banco de dados NoSQL, sem nenhuma modificação. Posteriormente foram utilizados índices específicos no PostgreSQL e no MongoDB, com intuito de otimizar as consultas. Por fim foram utilizados os dados armazenados em uma tabela com tipo de dados JSONB no PostgreSQL, e criados JSON indexados a fim de comparar a eficiência em relação as buscas no banco de dados relacional. Todas as consultas foram executadas localmente em uma máquina contendo um processador core i7 de 2.60GHz e 16GB de memória RAM. Todos os testes foram feitos em condições equivalentes de processamento da máquina evitando qualquer tipo de alteração nos resultados das consultas.

4.1 CONSULTAS POSTGRESQL SEM A UTILIZAÇÃO DE ÍNDICES

Sem a utilização de nenhum índice dentro do banco de dados relacional PostgreSQL e utilizando as consultas apresentadas abaixo na tabela do banco obtivemos os seguintes resultados em consultas:

4.1.1 Consulta Simples sem Filtro

Os resultados obtidos estão apresentados na Figura 1 abaixo, percebe-se que a consulta executou por *Seq Scan*, o que significa que a consulta precisou percorrer todos os um milhão duzentos e vinte mil registros em tela, o que para essa consulta sem filtro não tem tanto prejuízo, porém como poderemos verificar afeta em perda de desempenho. A consulta teve um tempo de planejamento de **15 milésimos de segundo** e levou cerca de **2 segundos e 621 milésimos de segundo** para ser executada.

```sql
select radacctid, username, acctstartdate, acctstopdate
from radacct
```

```
explain analyse
select radacctid, username, acctstartdate, acctstopdate
from radacct
```

```
QUERY PLAN
Seq Scan on radacct  (cost=0.00..57426.00 rows=1220000 width=26) (actual time=0.761..2556.334 rows=1220000 loops=1)
Planning Time: 15.186 ms
Execution Time: 2621.137 ms
```

Figura 1 – Consulta PostgreSQL sem Filtros.

4.1.2 Consulta com Filtro em Campo Texto

Fazendo uma consulta utilizando o operador *like* em um campo texto no PostgreSQL verificamos que como a consulta anterior foi utilizado *Seq Scan* para execução da consulta, neste caso a *performance* foi afetada pois foi necessário novamente percorrer todos os dados para apresentar os pouco mais de quarenta e oito mil registros em tela, a consulta se deu no tempo de **351 milésimos de segundo** e teve um tempo de panejamento de **0,2 milésimos de segundo**, como pode ser visto na Figura 2.

```
select radacctid, username, acctstartdate, acctstopdate
from radacct r
where username like '%maria%'
```

```
explain analyse
select radacctid, username, acctstartdate, acctstopdate
from radacct r
where username like '%maria%'
```

```
QUERY PLAN
Gather  (cost=1000.00..57610.57 rows=50304 width=26) (actual time=1.124..319.764 rows=48626 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on radacct r  (cost=0.00..51580.17 rows=20960 width=26) (actual time=0.340..219.839 rows=16209 loops=3)
        Filter: ((username)::text ~~ '%maria%'::text)
        Rows Removed by Filter: 390458
Planning Time: 0.235 ms
Execution Time: 321.518 ms
```

Figura 2 – Consulta PostgreSQL em Campo Texto.

4.1.3 Consulta com Filtro em Campo Numérico

A consulta utilizando o operador de maior no campo numérico radacctid no PostgreSQL foi executada atráves de *Index Scan* pois o campo é *primary key* da tabela, sendo assim já tem um índice que automaticamente é criado para ele. O resultado para execução e apresentação seiscentos e oitenta e quatro mil registros em tela se deu no tempo de **1 segundo e 222 milésimos de segundo** e com planejamento de **0,1 milésimos de segundo**, destaca-se que o tempo de planejamento e execução da consulta diminuiu por usar *Index Scan* e não precisar percorrer todos os registros da tabela para encontrar o resultado esperado, segue Figura 3.

```
select radacctid, username, acctstartdate, acctstopdate
from radacct r
where radacctid > 700000
```

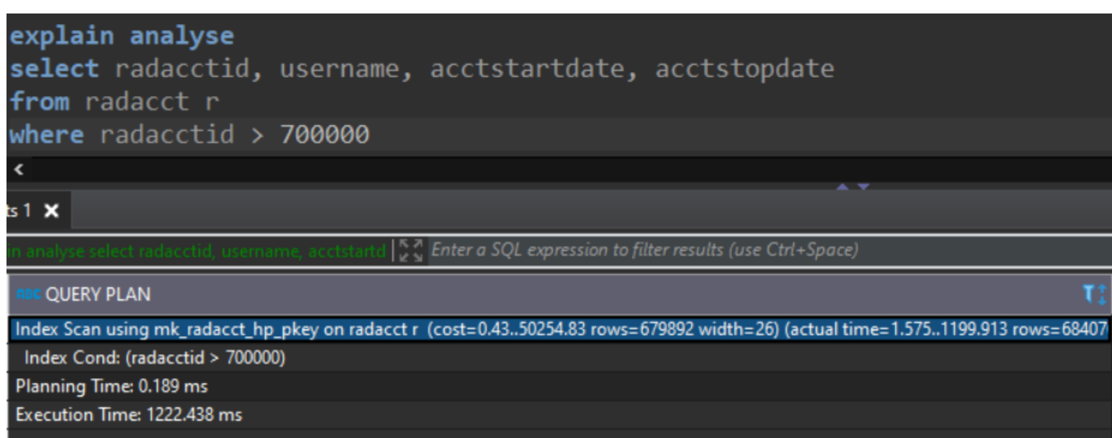


Figura 3 – Consulta PostgresSQL em Campo Numérico.

4.1.4 Consulta com Filtro em Campo de Data

Assim como as primeiras consultas utilizando o operador between em um campo data o PostgreSQL utilizou *Seq Scan* para execução, e o resultado para e apresentação de quinhentos mil registros em tela se deu no tempo de **426 milésimos de segundo** com planejamento de **0,2 milésimos de segundo**, conforme Figura 4.

```
select radacctid, username, acctstartdate, acctstopdate
from radacct r
where acctstartdate between '2018-01-01' and '2018-12-31'
```

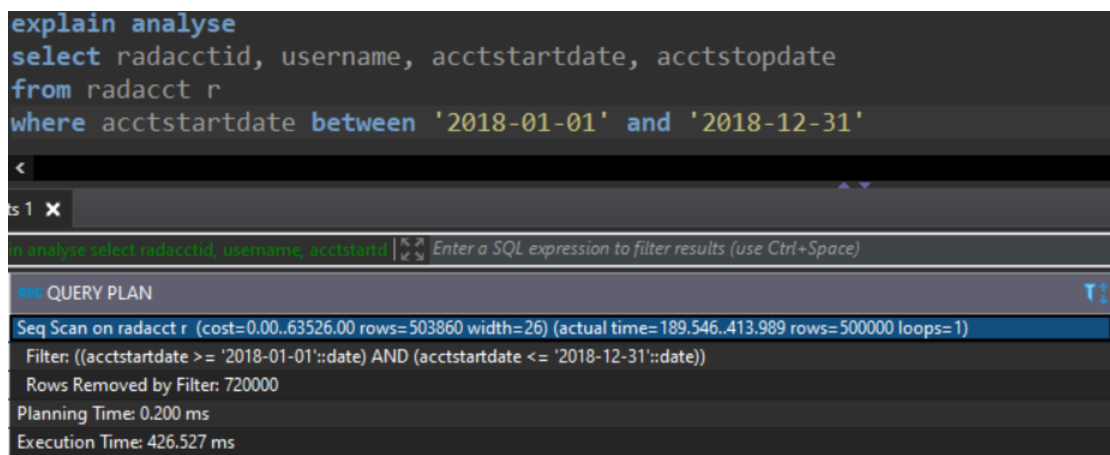


Figura 4 – Consulta PostgresSQL em Campo Data.

4.2 CONSULTAS MONGODB SEM UTILIZAÇÃO DE ÍNDICES

As consultas relativas as mesmas tratativas de dados feitas no PostgreSQL tiveram os seguintes resultados no NoSQL Mongo DB:

4.2.1 Consulta Simples sem Filtro

O resultado no MongoDB teve um desempenho superior ao PostgreSQL, a consulta levou **538 milésimos de segundo** para ser executada e apresentar todos os um milhão duzentos e vinte mil registros em tela, como pode ser visto na Figura 5.

```
db.radacct_nosql.find()
```

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

robo3t

0.541 se

```
1 /* 1 */
2 {
3     "explainVersion" : "1",
4     "queryPlanner" : {
5         "namespace" : "projeto_tcc_mongo.radacct_nosql",
6         "indexFilterSet" : false,
7         "parsedQuery" : {},
8         "maxIndexedOrSolutionsReached" : false,
9         "maxIndexedAndSolutionsReached" : false,
0         "maxScansToExplodeReached" : false,
1         "winningPlan" : {
2             "stage" : "COLLSCAN",
3             "direction" : "forward"
4         },
5         "rejectedPlans" : []
6     },
7     "executionStats" : {
8         "executionSuccess" : true,
9         "nReturned" : 1220000,
0         "executionTimeMillis" : 538,
1         "totalKeysExamined" : 0,
2         "totalDocsExamined" : 1220000,
```

Figura 5 – Consulta MongoDB sem Filtros.

4.2.2 Consulta com Filtro em Campo Texto

A Figura 6 mostra que a consulta no banco de dados MongoDB teve um resultado inferior em campo texto, percebe-se que a consulta percorreu todos os um milhão e duzentos e vinte mil registros da tabela para encontrar as linhas que se encaixavam com o parâmetro repassado, executando a consulta e apresentando os pouco mais de quarenta e oito mil registros em tela com o tempo de **867 milésimos de segundo**.

```
db.radacct_nosql.find({username:{$regex: "maria"}})
```

```
db.radacct_nosql.find({username:{$regex: "maria"}}) .explain("executionStats")
```

obo3t

0.869 se

```
    "queryPlanner" : {
        "namespace" : "projeto_tcc_mongo.radacct_nosql",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "username" : {
                "$regex" : "maria"
            }
        },
        "maxIndexedOrSolutionsReached" : false,
        "maxIndexedAndSolutionsReached" : false,
        "maxScansToExplodeReached" : false,
        "winningPlan" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "username" : {
                    "$regex" : "maria"
                }
            },
            "direction" : "forward"
        },
        "rejectedPlans" : []
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 48626,
        "executionTimeMillis" : 867,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 1220000,
```

Figura 6 – Consulta MongoDB em Campo Texto.

4.2.3 Consulta com Filtro em Campo Numerico

Trabalhando com dados do tipo numérico o MongoDB teve um desempenho melhor comparado ao PostgreSQL, analisando a consulta verificamos que como anteriormente percorreu todos os um milhão e duzentos e vinte mil registros da tabela para encontrar as linhas desejadas, porém executando a consulta e mostrando os seiscentos e oitenta e quatro mil registros em tela no tempo de **758 milésimos de segundo**, conforme Figura 7.

```
db.radacct_nosql.find({radacctid:{$gte: 700000}})
```

```
db.radacct_nosql.find({radacctid:{$gte: 700000}}). explain("executionStats")
```

obo3t

0.759 se

```
/* 1 */
{
    "explainVersion" : "1",
    "queryPlanner" : {
        "namespace" : "projeto_tcc_mongo.radacct_nosql",
        "indexFilterSet" : false,
        "parsedQuery" : {
            "radacctid" : {
                "$gte" : 700000.0
            }
        },
        "maxIndexedOrSolutionsReached" : false,
        "maxIndexedAndSolutionsReached" : false,
        "maxScansToExplodeReached" : false,
        "winningPlan" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "radacctid" : {
                    "$gte" : 700000.0
                }
            },
            "direction" : "forward"
        },
        "rejectedPlans" : []
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 684071,
        "executionTimeMillis" : 758,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 1220000,
```

Figura 7 – Consulta MongoDB em Campo Numérico.

4.2.4 Consulta com Filtro em Campo de Data

O MongoDB teve o tempo de execução em **1 segundo e 116 milésimos de segundo** na consulta em campo Data, percorrendo todos os um milhão e duzentos e vinte mil registros da tabela e apresentando os quinhentos mil registros em tela nesse tempo, como pode ser visto na Figura 8.

```
db.radacct_nosql.find({acctstartdate:{
$gte: ISODate("2018-01-01T00:00:00.000Z"),
$lt: ISODate("2018-12-31T00:00:00.000Z")}})
```

```
db.radacct_nosql.find({acctstartdate:{$gte: ISODate("2018-01-01T00:00:00.000Z"),
    $lt: ISODate("2018-12-31T00:00:00.000Z")}}).explain("executionStats")
```

obo3t

1.12 se

```
        "maxIndexedAndSolutionsReached" : false,
        "maxScansToExplodeReached" : false,
        "winningPlan" : {
            "stage" : "COLLSCAN",
            "filter" : {
                "$and" : [
                    {
                        "acctstartdate" : {
                            "$lt" : ISODate("2018-12-31T00:00:00.000Z")
                        }
                    },
                    {
                        "acctstartdate" : {
                            "$gte" : ISODate("2018-01-01T00:00:00.000Z")
                        }
                    }
                ]
            },
            "direction" : "forward"
        },
        "rejectedPlans" : []
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 500000,
        "executionTimeMillis" : 1116,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 1220000,
```

Figura 8 – Consulta MongoDB em Campo Data.

4.3 CONSULTAS POSTGRESQL UTILIZANDO ÍNDICES

Na seção acima verificamos que o MongoDB teve uma *performance* superior em algumas das consultas realizados nos mais diversos tipos de dados armazenados, chegando a ter cerca de 1 segundo de desempenho superior em relação ao PostgreSQLs, o que é bastante relevante para a quantidade de dados consultada.

Para tentar melhorar o desempenho do PostgreSQL foi feita a criação de 3 índices específicos para coluna numérica, textual e de data, a fim de dar mais agilidade as consultas, segue abaixo as consultas de criação dos índices que foram utilizadas:

Índice *GIN* para coluna texto username.

```
CREATE INDEX busca_username_idx ON radacct
USING gin (username gin_trgm_ops);
```

Índice *Single-column index* para coluna numérica radacctid.

```
CREATE INDEX index_radacctid ON radacct (radacctid);
```

Índice *Single-column index* para coluna data acctstartdate.

```
CREATE INDEX data_conexao_idx
ON radacct ((acctstartdate::DATE));
```

Com a utilização de índices específicos para tipos de dados texto, numérico e de data no banco de dados relacional PostgreSQL os resultados obtidos nas consultas foram os seguintes:

4.3.1 Consulta com Filtro em Campo Texto

Executando a mesma consulta utilizada anteriormente no campo texto username, agora com o índice criado podemos verificar que o PostgreSQL executou a consulta não mais por *Seq Scan* e sim agora por *Bitmap Index Scan*, apresentando um menor custo de desempenho e apresentando os dados no tempo de **254 milésimos de segundo** e com planejamento em **0,3 milésimos de segundo**, baixando o tempo obtido sem índices conforme Figura 9.

```
select radacctid, username, acctstartdate, acctstopdate
from radacct r
where username like '%maria%'
```

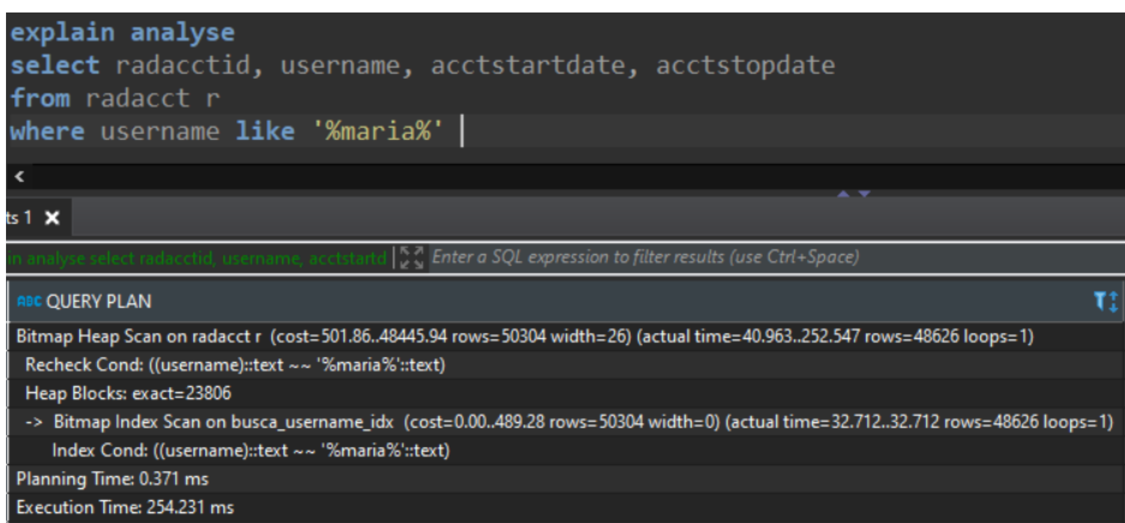


Figura 9 – Consulta PostgreSQL com Índice em Campo Texto.

4.3.2 Consulta com Filtro em Campo Numerico

A mesma consulta em campo numérico executada no PostgreSQL, que antes levou mais de 1 segundo para ser executada, agora com a criação do novo índice e utilizando *Index Scan* executou no tempo de **469 milésimos de segundo**, pois não precisou percorrer todos os registros da tabela para encontrar os dados desejados, isso fica claro pois o tempo de planejamento subiu para quase **3 milésimos de segundo**, o que significa que a consulta tece um melhor planejamento a fim de diminuir o tempo de execução, conforme Figura 10.

```
select radacctid, username, acctstartdate, acctstopdate
```

```
from radacct r
where radacctid > 700000
```

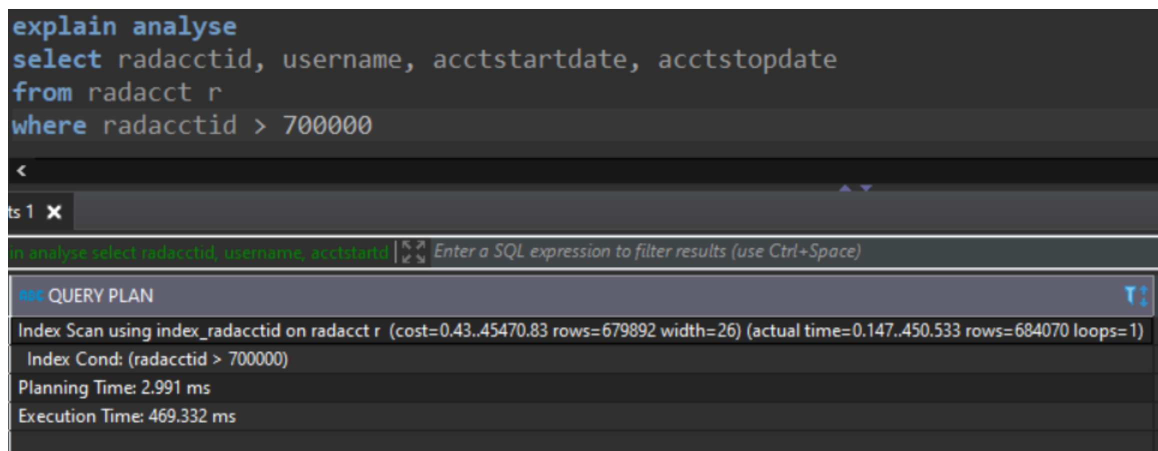


Figura 10 – Consulta PostgreSQL com Índice em Campo Númerico.

4.3.3 Consulta com Filtro em Campo de Data

A consulta em campo data teve grande melhora na execução após a criação do índice, levando apenas **287 milésimos de segundo** para executar, podemos verificar que essa consulta executou por seu índice criado em *Bitmap Index Scan*, e assim como na consulta em campo numérico acima teve um tempo de planejamento maior com o tempo de **6 milésimos de segundo**, melhorando a *performance* da sua execução, como podemos ver na Figura 11.

```
select radacctid, username, acctstartdate, acctstopdate
from radacct r
where acctstartdate between '2018-01-01' and '2018-12-31'
```

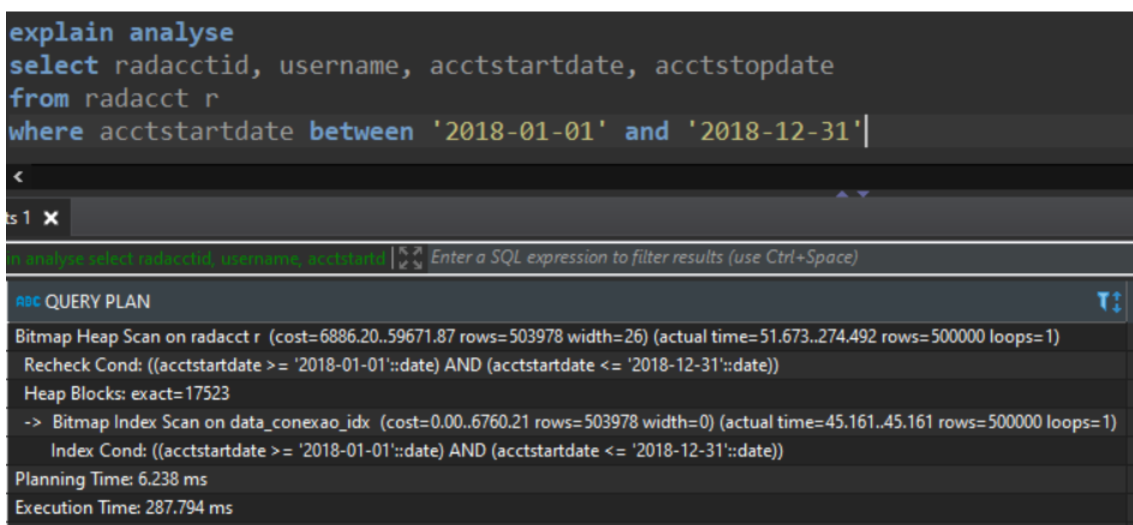


Figura 11 – Consulta PostgreSQL com Índice em Campo Data.

4.4 CONSULTAS MONGODB UTILIZANDO ÍNDICES

É de clara observação que o desempenho das consultas no PostgreSQL melhorou e baixou quase pela metade seu tempo de execução utilizando-se de índices, chegando a ser superior ao MongoDB em todas as consultas. A fim de melhorar os resultados do NoSQL MongoDB e beneficiar-se também do uso de índices foram criados os seguintes índices no banco de dados MongoDB, a fim de analisar qual a *performance* de execução através deles.

Os índices no MongoDB funcionam de maneira parecida com os índices dos bancos de dados relacionais, pois permitem que o MongoDB processe e atenda consultas rapidamente através da criação de pequenas e eficientes representações dos documentos em uma coleção. Os seguintes índices foram criados através do método do MongoDB *createndex*:

Índice para coluna texto username.

```
db.radacct_nosql.createIndex({username: 1})

db.radacct_nosql.createIndex({"username": "text"},
    { default_language: "portuguese" })
```

Índice para coluna numérica radacctid.

```
db.radacct_nosql.createIndex({"radacctid": 1},{unique: true})
```

Índice para coluna data acctstartdate.

```
db.radacct_nosql.createIndex({acctstartdate: 1})
```

Com a utilização dos índices no banco de dados NoSQL MongoDB os resultados obtidos nas consultas foram os seguintes:

4.4.1 Consulta com Filtro em Campo Texto

Executando consulta no campo texto username, agora com o índice criado como pode ser visto na coluna*indexName*, o tempo de execução aumentou em cerca de 700 milésimos, levando **1 segundo e 586 milésimos de segundo** para executar, fica evidente que a utilização de índice não foi efetiva para essa consulta no MongoDB, pois no item *totalKeysExamined* verificamos que mesmo com a consulta indexada todos os registros da tabela foram percorridos para retornar o resultado esperado, os dados podem ser vistos na Figura 12.

```
db.radacct_nosql.find({username:{$regex: "maria"}})
```

```
db.radacct_nosql.find({username:{$regex: "maria"}}).explain("executionStats")
1.59 se
                    }
                },
                "keyPattern" : {
                    "username" : 1.0
                },
                "indexName" : "username_1",
                "isMultiKey" : false,
                "multiKeyPaths" : {
                    "username" : []
                },
                "isUnique" : false,
                "isSparse" : false,
                "isPartial" : false,
                "indexVersion" : 2,
                "direction" : "forward",
                "indexBounds" : {
                    "username" : [
                        "[\"\", {})",
                        "[/maria/, /maria/]"
                    ]
                }
            }
        },
        "rejectedPlans" : []
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 48626,
        "executionTimeMillis" : 1586,
        "totalKeysExamined" : 1220000,
```

Figura 12 – Consulta MongoDB com Índice em Campo Texto.

4.4.2 Consulta com Filtro em Campo Numerico

A mesma consulta em campo numérico executada no MongoDB, que antes levou 700 milésimos de segundo para ser executada, agora com a criação do novo índice executou no tempo de **1 segundo e 252 milésimos de segundo**, também aumentando o tempo de consulta conforme Figura 13. Nessa consulta diferentemente da anterior fica evidente que o índice teve bom funcionamento, pois a consulta percorreu apenas os seiscentos e quarenta e oito mil e setenta e um resistros que foram apresentados em tela, não perdendo tempo de execução com os registros desnecessários, porém a consulta se saiu mais custosa processualmente acarretando em perda de desempenho.

```
db.radacct_nosql.find({radacctid:{$gte: 700000}})
```

```
db.radacct_nosql.find({radacctid:{$gte: 700000}}).explain("executionStats")
1.25 se
        "maxScansToExplodeReached" : false,
        "winningPlan" : {
            "stage" : "FETCH",
            "inputStage" : {
                "stage" : "IXSCAN",
                "keyPattern" : {
                    "radacctid" : 1.0
                },
                "indexName" : "radacctid_1",
                "isMultiKey" : false,
                "multiKeyPaths" : {
                    "radacctid" : []
                },
                "isUnique" : true,
                "isSparse" : false,
                "isPartial" : false,
                "indexVersion" : 2,
                "direction" : "forward",
                "indexBounds" : {
                    "radacctid" : [
                        "[700000.0, inf.0]"
                    ]
                }
            }
        },
        "rejectedPlans" : []
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 684071,
        "executionTimeMillis" : 1252,
        "totalKeysExamined" : 684071,
        "totalDocsExamined" : 684071,
```

Figura 13 – Consulta MongoDB com Índice em Campo Númerico.

4.4.3 Consulta com Filtro em Campo de Data

A consulta em campo data foi a única que teve melhora de *performance* após a criação do índice no MongoDB, baixando o tempo de execução em cerca de 400 milésimos de segundo, levando apenas **796 milésimos de segundo** para executar como podemos ver na Figura 14. Assim como a consulta anterior o índice teve ótimo resultado pois apenas as quinhentas mil linhas necessárias foram percorridas para apresentação dos resultados.

```
db.radacct_nosql.find({acctstartdate:{
$gte: ISODate("2018-01-01T00:00:00.000Z"),
$lt: ISODate("2018-12-31T00:00:00.000Z")}})
```

```
db.radacct_nosql.find({acctstartdate:{
$gte: ISODate("2018-01-01T00:00:00.000Z"),
$lt: ISODate("2018-12-31T00:00:00.000Z")}}).explain("executionStats")
```

0.799 se

```
            "stage" : "FETCH",
            "inputStage" : {
                "stage" : "IXSCAN",
                "keyPattern" : {
                    "acctstartdate" : 1.0
                },
                "indexName" : "acctstartdate_1",
                "isMultiKey" : false,
                "multiKeyPaths" : {
                    "acctstartdate" : []
                },
                "isUnique" : false,
                "isSparse" : false,
                "isPartial" : false,
                "indexVersion" : 2,
                "direction" : "forward",
                "indexBounds" : {
                    "acctstartdate" : [
                        "[new Date(1514764800000), new Date(1546214400000))"
                    ]
                }
            }
        },
        "rejectedPlans" : []
    },
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 500000,
        "executionTimeMillis" : 796,
        "totalKeysExamined" : 500000,
```

Figura 14 – Consulta MongoDB com Índice em Campo Data.

4.5 UTILIZANDO JSON INDEXADO NO POSTGRESQL

Para tentar melhorar ainda mais o desempenho do PostgreSQL foi feita a criação de uma nova tabela, está armazenando os dados em JSONB dentro do PostegreSQL, e foi criado também índices JSON a fim de validar novamente a velocidade das consultas.

A sintaxe de criação dessa nova tabela é a seguinte:

```
CREATE TABLE radacct_pg_jsonb (
        "content" jsonb NULL
);
```

Os índices JSON criados seguem a mesma ideologia dos índices especificos criados na seção anterior, um índice para campo texto, um para campo numérico e outro para campo data, segue abaixo a sintaxe dos indices criados:

Índice para coluna texto username.

```
CREATE INDEX trgm_idx ON radacct_pg_jsonb
USING gist ((dados->>'username') gist_trgm_ops);
```

Índice para coluna numérica radacctid.

```
CREATE INDEX idx_radacctid
ON radacct_pg_jsonb ((dados->>'radacctid'));
```

Índice para coluna data acctstartdate.

```
CREATE INDEX idx_acctstartdate
ON radacct_pg_jsonb ((dados->'acctstartdate'));
```

4.5.1 Consulta com Filtro em Campo Texto

Após a criação da nova tabela armazenando os dados em JSON e a criação dos índices para essa tabela, executando a consulta no campo texto utilizando *Bitmap Index Scan* podemos verificar que o tempo de execução foi de **573 milésimos de segundo** para apresentar os mais de quarenta e oito mil registros em tela e o tempo de planejamento foi de **0,4 milésimos de segundo**, conforme Figura 15.

```
SELECT dados->>'radacctid' AS radacctid,
       dados->>'username' AS username,
       dados->>'acctstartdate' AS acctstartdate,
       dados->>'acctstopdate' AS acctstopdate
FROM radacct_pg_jsonb where dados->>'username' like '%maria%';
```

```
explain analyse
SELECT dados->>'radacctid' AS radacctid,
       dados->>'username' AS username,
       dados->>'acctstartdate' AS acctstartdate,
       dados->>'acctstopdate' AS acctstopdate
FROM radacct_pg_jsonb where dados->>'username' like '%maria%';

QUERY PLAN
Bitmap Heap Scan on radacct_pg_jsonb (cost=111.54..7176.32 rows=1952 width=128) (actual time=234.932..542.458 rows=48626 loops=1)
  Recheck Cond: ((dados ->> 'username'::text) ~~ '%maria%'::text)
  Heap Blocks: exact=35478
  -> Bitmap Index Scan on trgm_idx (cost=0.00..111.05 rows=1952 width=0) (actual time=229.218..229.219 rows=48626 loops=1)
     Index Cond: ((dados ->> 'username'::text) ~~ '%maria%'::text)
Planning Time: 0.423 ms
Execution Time: 544.148 ms
```

Figura 15 – Consulta PostgreSQL com JSON Índexado em Campo Texto.

4.5.2 Consulta com Filtro em Campo Numerico

O índice *btree* criado para a coluna radacctid na tabela JSON do PostgreSQL tem uma limitação de registros retornados para que a consulta execute pelo índice, para esse teste

o resultado obtido para a execução dos mais de cento e onze mil registros consultados foi no tempo de **197 milésimos de segundo**, conforme Figura 16. A execução também foi feita por *Bitmap Index Scan* com tempo de planejamento de **0,3 milésimos de segundo**.

```
SELECT dados->>'radacctid' AS radacctid,
       dados->>'username' AS username,
       dados->>'acctstartdate' AS acctstartdate,
       dados->>'acctstopdate' AS acctstopdate
FROM radacct_pg_jsonb where (dados->>'radacctid') > '700000'
and (dados->>'radacctid') < '800000';
```

```
explain analyze
SELECT dados->>'radacctid' AS radacctid,
       dados->>'username' AS username,
       dados->>'acctstartdate' AS acctstartdate,
       dados->>'acctstopdate' AS acctstopdate
FROM radacct_pg_jsonb where (dados->>'radacctid') > '700000'
and (dados->>'radacctid') < '800000';

QUERY PLAN
Bitmap Heap Scan on radacct_pg_jsonb  (cost=2333.89..142198.09 rows=109801 width=128) (actual time=95.711..194.982 rows=111108 loops=1)
  Recheck Cond: (((dados ->> 'radacctid'::text) > '700000'::text) AND ((dados ->> 'radacctid'::text) < '800000'::text))
  Heap Blocks: exact=11255
  -> Bitmap Index Scan on idx_radacctid  (cost=0.00..2306.44 rows=109801 width=0) (actual time=94.207..94.207 rows=111108 loops=1)
       Index Cond: (((dados ->> 'radacctid'::text) > '700000'::text) AND ((dados ->> 'radacctid'::text) < '800000'::text))
Planning Time: 0.392 ms
Execution Time: 197.977 ms
```

Figura 16 – Consulta PostgreSQL com JSON Índexado em Campo Númerico.

4.5.3 Consulta com Filtro em Campo de Data

Com o tipo de dados data a consulta executou por *Index Scan* e se beneficiando-se do uso do índice levou o tempo de **0,6 milésimos de segundos** para efetuar o planejamento da consulta e o tempo de **172 milésimos de segundos** para executar e apresentar os quinhentos mil registros em tela, conforme a Figura 17.

```
SELECT dados->>'radacctid' AS radacctid,
       dados->>'username' AS username,
       dados->>'acctstartdate' AS acctstartdate,
       dados->>'acctstopdate' AS acctstopdate
FROM radacct_pg_jsonb
where dados->>'acctstartdate'
between '2018-01-01' and '2018-05-31';
```

```
explain analyze
SELECT dados->>'radacctid' AS radacctid,
      dados->>'username' AS username,
      dados->>'acctstartdate' AS acctstartdate,
      dados->>'acctstopdate' AS acctstopdate
FROM radacct_pg_jsonb
where dados->>'acctstartdate'
between '2018-01-01' and '2018-05-31';
```

```
QUERY PLAN
Index Scan using idx_acctstartdate on radacct_pg_jsonb (cost=0.43..123887.48 rows=105355 width=128) (actual time=0.201..168.518 rows=108174
  Index Cond: (((dados ->> 'acctstartdate'::text) >= '2018-01-01'::text) AND ((dados ->> 'acctstartdate'::text) <= '2018-05-31'::text))
Planning Time: 0.635 ms
Execution Time: 172.229 ms
```

Figura 17 – Consulta PostgreSQL com JSON Índexado em Campo Númerico.

4.6 ANÁLISE DOS RESULTADOS OBTIDOS

Os testes foram realizados inicialmente no banco de dados relacional PostgreSQL e no banco de dados NoSQL MongoDB sem a utilização de nenhum índice, a fim de analisar qual o desempenho dos bancos nesse caso de execução. Em um segundo momento foram criados os índices tanto no banco PostegreSQL quanto no MongoDB e utilizando-se desses índices novas consultas de testes foram executadas a fim de comparar novamente os resultados e o nível de melhora beneficiando-se da sua utilização. Posteriormente o teste final foi realizado no banco de dados relacional PostegreSQL usufruindo do suporte nativo JSON na criação da tabela para armazenamento dos dados e na criação de índices, com a finalidade de fazer novos testes com consultas dentro do banco de dados JSON no PostgreSQL.

Após a realização de todas as consultas, e com todos os resultados obtidos foi possível obter dados e informações a respeito da *performance*, bem como as vantagens e desvantagens do banco de dados relacional PostgreSQL e do NoSQL MongoDB.

Ao executar as consultas no PostgreSQL puro, sem nenhuma inserção de índices, percebemos que o tempo de execução variou de mais de dois segundos para a consulta sem nenhum filtro a até trezentos milésimos de segundo para a consulta no campo texto, onde a quantidade de dados retornada foi menor.

Analisando os dados em consultas equivalentes as executadas no PostgreSQL, porém no banco de dados MongoDB utilizando a sua linguagem padrão, percebemos que levando em comparação o PostgreSQL sem índices e o MongoDB também sem índices temos um equilíbrio entre os dois bancos de dados, pois o desempenho do MongoDB foi melhor em duas das quatro consultas executadas, e consequentemente inferior também em duas das consultas. Podemos levar em consideração que o MongoDB se saiu melhor nas consultas sem nenhum filtro e com

filtro em campo numérico, exatamente as duas que retornam a maior quantidade de dados, fortalecendo a ideia de que o banco NoSQL trabalha melhor com grandes quantidades de dados.

Posteriormente trabalhando nos bancos já com os índices criados nas tabelas percebemos que o resultado foi muito satisfatório no banco de dados relacional PostgreSQL, melhorando a *performance* e diminuindo o tempo de execução das consultas em cerca de metade do tempo levado sem a utilização de índices, como exemplo a consulta em campo numérico que baixou cerca de setecentos milésimos de segundo o tempo de execução.

Já no banco de dados NoSQL MongoDB a utilização de índices teve resultados negativos, pois com os índices criados na tabela a consulta buscou uma quantidade menor de registros para retornar o resultado, porém se tornou mais complexa por tratar de condições de consulta mais elaboradas, aumentando assim o tempo de execução. Em três das consultas testadas esse problema aconteceu, tendo como único êxito da utilização de índices no MongoDB a consulta com o tipo de dados data, que teve uma melhora no desempenho de cerca de quatrocentos milésimos de segundo. Conforme (CORREA, 2019) essa situação de impacto negativo no MongoDB ocorre pois o uso de índices requer muito mais recursos computacionais, afetando negativamente no tempo de consulta em casos de projetos com pequena disponibilidade de processamento, como o nosso projeto.

Com essa evidente melhora através dos índices no PostgreSQL, e com o impacto negativo da utilização de índices no MongoDB, o banco de dados relacional superou os resultados obtidos no banco de dados NoSQL em todas as consultas de teste realizadas, tendo um desempenho melhor mesmo com consultas que retornam uma grande quantidade de registros.

Mesmo com a melhor *performance* do banco de dados relacional, visando elevar ainda mais o desempenho do PostgreSQL, foram criados a tabela e os índices em JSON no banco. Manipulando os dados JSON dentro do PostgreSQL pudemos perceber uma pequena limitação quanto ao recurso computacional disponível para a execução das consultas, o que acarretou em alguns casos na lentidão da apresentação dos resultados. Quanto aos resultados obtidos podemos destacar que o tempo de execução continuou superior ao banco de dados NoSQL MongoDB, porém levando em consideração a quantidade de dados consultados sofreu uma pequena queda em relação aos resultados do PostgreSQL nativo utilizando índices, muito em decorrência das limitações citadas anteriormente.

Na Tabela 1 abaixo é possível ver o tempo de execução, em milisegundos (ms), que as consultas testadas levaram em cada um dos cenários dentro do banco de dados PostgreSQL e MongoDB.

Tabela 1 – Desempenho de Execução das Consultas.

	Sem Filtro	Campo Texto	Campo Númerico	Campo Data
PostgreSQL Sem Índices	2621ms	321ms	1222ms	426ms
MongoDB Sem Índices	538ms	867ms	738ms	1116ms
PostgreSQL Com Índices	-	254ms	469ms	287ms
MongoDB Com Índices	-	1586ms	1252ms	796ms
PostgreSQL JSON Indexado	-	544ms	197ms	172ms

Fonte: Elaborado pelo autor

É possível observar que sem nenhuma utilização de índices, em uma consulta sem filtros retornando todos os um milhão e duzentos e vinte mil registros da tabela o banco de dados NoSQL MongoDB teve um desempenho muito superior ao banco de dados relacional PostgreSQL, executando a consulta em um tempo quase cinco vezes menor. Quando observamos os demais campos consultados ainda nos bancos de dados sem índices percebemos que a consulta com mais dados retornados e com menos complexidade de filtro, que é a realizada em campo número, também teve um resultado superior no MongoDB, já consultas que tem menos dados retornados e tem filtros mais complexos, como a em compa texto e data, tiveram resultados superiores no PostgreSQL, mostrando um equilíbrio entre os bancos que apresentaram resultados superiores de acordo com suas principais características.

Quanto partimos para a comparação das consultas nos bancos de dados já com a criação dos índices vemos uma total superioridade do PostgreSQL, tendo os melhores tempos de consultas entre todos os testes, levando em consideração que o PostresSQL JSON Indexado consultou uma menor quantidade de dados para indexar as consultas e por isso teve tempo menor. Por outro lado o MongoDB apresentou alguns dos piores tempo de consulta, inferior inclusive aos tempo de execução no banco sem nenhum índice, muito em vista da grande quantidade de processamento computacional que a utilização de índices no MongoDB necessita, o que acarreta em queda de *performance*. O PostgreSQL como citado anteriormente precisou de consultas que retornassem uma menor quantidade de dados para as consultas em campo numérico e data, isso devido ao SGBD não fazer a utilização dos índices em consultas muito grandes. Porém mesmo com isso os resultados foram superiores aos executados no MongoDB e em alguns casos equivalentes aos coletados no PostgreSQL com índices.

Tendo em vista tudo isso que foi obtido e apresentado podemos perceber que o banco de dados NoSQL MongoDB, assim como destacado em teoria, teve melhores resultados em consultas que retornam maiores quantidades de dados. Percebemos que o NoSQL MongoDB é projetado para projetos com muitos dados armazenados e com recursos computacionais mais poderosos, por trabalhar com tipos de dados mais variados e ter sido desenvolvida para esse nicho de mercado, e por esse motivo apresenta desvantagens em projetos de pequeno e médio porte. Por sua vez o banco relacional PostegreSQL teve resultados superiores ao MongoDB na maioria das consultas apresentadas, beneficiando-se muito do correto uso de índices e do

suporte nativo ao JSON. Posto isto destacamos que a utilização do banco de dados relacional PostgreSQL vem se adaptando e trazendo melhorias que oferecem aos usuários um desempenho melhor, e em muitos casos superior ao NoSQL, presando sempre os conceitos de ACID e priorizando a segurança e consistência dos dados, além de manter uma linguagem padrão que simplifica muito sua manipulação, agregando a tudo isso ferramentas que oferecem um melhor desempenho.

5 CONCLUSÃO

5.1 LIMITAÇÕES

Podemos destacar que a ferramenta de administração de banco de dados DBeaver apresentou certa limitação de memória na execução das consultas na tabela em JSON. Essa limitação ocasionou em certos casos a demora da execução das consultas, o que contribuiu para o desempenho inferior do banco de dados relacional PostgreSQL utilizando JSON em relação ao banco de dados PostgreSQL nativo com a utilização de índices, que por sua vez não teve limitações em relação a recurso computacional.

Outra limitação foi quanto ao recurso computacional disponivel para a execução das consultas no MongoDB, principalmente com a utilização de índices que requer muito mais recursos de memória e disco. Como o projeto tem uma certa limitação de recursos computacionais isso foi observado no resultados obtidos dentro do banco de dados NoSQL.

5.2 CONSIDERAÇÕES FINAIS

Esse trabalho pretendeu fazer um comparativo de *performance* entre o banco de dados relacional PostgreSQL e o NoSQL MongoDB em condições equivalentes de manipulação, beneficiando-se da utilização de ferramentas de indexação disponibilizadas pelo PostgreSQL como JSON indexados, a fim de verificar qual das soluções tem maior eficiência, e analisar como o banco de dados relacional se sairia utilizando-se de ferramentas de indexação quando comparadas do banco de dados NoSQL. Para as comparações foram utilizadas consultas em tipos diferentes de dados em cenários sem e com a utilização de índices dentro dos bancos de dados.

Para se obter a melhor e mais confiável comparação de desempenho entre o banco de dados relacional PostgreSQL e o NoSQL MongoDB, definiu-se três objetivos específicos. O primeiro sendo a análise de recuperação de dados simples no PostgreSQL frente ao MongoDB. Verificou-se que houve um equilíbrio entre os bancos, sendo que entre quatro consultas realizadas duas delas obtiveram melhor resultado pelo PostgreSQL e outras duas melhor desempenho pelo MongoDB. Depois, analisando a recuperação de dados JSON comparando a eficiência em relação as buscas em ambos sistemas. Nesse caso verificou-se que o PostgreSQL por utilizar-se melhor da criação de índices teve melhores resultados, superando o desempenho das consultas realizadas no MongoDB. Por fim foi feita a utilização e medição da eficiência de índices específicos no PostgreSQL para agilizar o resultado das consultas. A análise permitiu concluir que no PostgreSQL o uso de índices teve um benefício muito superior, diminuindo o tempo de consulta em quase a metade do tempo levado sem o uso de índices. por outro lado a criação de índices no MongoDB afetou negativamente na *performance* das consultas, pois ao utilizar um índice o banco necessita de muito mais recursos computacionais, o que acareta em

perda de desempenho.

Com isso, a hipótese do trabalho de que a *performance* do banco de dados relacional, beneficiando-se da utilização de ferramentas de indexação ao trabalhar com grandes quantidades de dados, garante um melhor desempenho nas consultas, se equivalendo ao banco de dados NoSQL, se confirmou, pois de acordo com os dados coletados em consultas de teste o correto uso de índices e do suporte nativo ao JSON no PostgreSQL teve resultados superiores aos obtidos no banco de dados NoSQL MongoDB.

Sendo assim conclui-se que utilização do banco de dados relacional PostgreSQL vem se adaptando e trazendo melhorias que oferecem aos usuários um desempenho melhor, e em muitos casos superior ao NoSQL, presando sempre os conceitos de ACID e priorizando a segurança e consistência dos dados, além de manter uma linguagem padrão que simplifica muito sua manipulação, agregando a tudo isso ferramentas que oferecem um melhor desempenho. Por outro lado o NoSQL MongoDB é projetado para projetos com muitos dados armazenados e com recursos computacionais mais poderosos, por trabalhar com tipos de dados mais variados e ter sido desenvolvido para esse nicho de mercado, e por esse motivo apresenta desvantagens em projetos de pequeno e médio porte.

Em pesquisas futuras, pode-se fazer testes em diferentes bases de dados, com recursos computacionais independentes e diferentes quantidades de dados armazenados, sendo assim será possível verificar em que tipo de projeto se obtém vantagem ao utilizar o banco de dados relacional e em quais projetos o banco de dados NoSQL apresenta vantagens.

Referências

ALEXANDRE, M. **Conectando o Robo 3T a uma instância MongoDB**. 2021. Disponível em: <https://pt.linkedin.com/pulse/conectando-o-robo-3t-uma-inst%C3%A2ncia-mongodb-marcelo-alexandre>. Citado na página 18.

CORREA, V. **Impacto dos índices no MongoDB**. 2019. Disponível em: <https://medium.com/@valmircsjr/impacto-dos-%C3%ADndices-no-mongodb-9d8cb134138c>. Citado na página 38.

DBEAVER. **Ferramenta de banco de dados universal**. 2021. Disponível em: <https://dbeaver.io/>. Citado 2 vezes nas páginas 11 e 17.

ELMASRI, R.; NAVATHE, S. B. **Database systems**. [S.l.]: Pearson Education Boston, MA, 2011. v. 9. Citado na página 12.

FERNANDES, H. M. **4 ferramentas gratuitas para gerenciar MongoDB – 2020**. 2021. Disponível em: <https://marquesfernandes.com/desenvolvimento/ferramentas-gratuitas-para-gerenciar-mongodb/>. Citado 2 vezes nas páginas 11 e 18.

HALLIDAY, L. **Unleash the Power of Storing JSON in Postgres**. 2018. Citado 2 vezes nas páginas 9 e 16.

JSON. **Apresentando JSON**. 2021. Disponível em: <https://www.json.org/json-en.html>. Citado na página 16.

JUNIOR, L. F. D. **MongoDB para iniciantes em NoSQL**. 2017. Disponível em: <https://imasters.com.br/banco-de-dados/mongodb-para-iniciantes-em-nosql>. Citado na página 17.

KUNDA, D.; PHIRI, H. A comparative study of nosql and relational database. **zictjournal**, 2017. Citado na página 13.

MILANI, A. **PostgreSQL-Guia do Programador**. [S.l.]: Novatec Editora, 2008. Citado na página 16.

MONGODB. **O banco de dados para aplicativos modernos**. 2021. Disponível em: <https://www.mongodb.com/pt-br>. Citado na página 11.

MONGODB, I. Mongodb. **URL https://www. mongodb. com/. Cited on (2014)**, v. 9, 2014. Citado na página 17.

MONIRUZZAMAN, A.; HOSSAIN, S. A. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. **arXiv preprint arXiv:1307.0191**, 2013. Citado na página 12.

NAYAK, A.; PORIYA, A.; POOJARY, D. Type of nosql databases and its comparison with relational databases. **International Journal of Applied Information Systems**, v. 5, 2013. Citado na página 12.

POSTGRESQL. **PostgreSQL: o banco de dados relacional de código aberto mais avançado do mundo**. 2021. Disponível em: <https://www.postgresql.org/l>. Citado na página 11.

Pramod J., S.; Martin, F. **NoSQL Essencial: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota Essencial**. [S.l.]: Novatec Editora Ltda, 2019. ISSN 978-85-7522-XXX. Citado na página 9.

ROB, P.; CORONEL, C. Sistemas de banco de dados. **Projeto, implementação e**, 2011. Citado na página 12.

SECCO, R. R. Análise comparativa entre o banco de dados cassandra (modelo nosql) e o postgresql (modelo relacional) em duas diferentes organizações empresariais. **Colloquium Exactarum. ISSN: 2178-8332**, v. 8, 2017. Citado na página 15.

SOARES, B. E.; BOSCARIOLI, C. Modelo de banco de dados colunar: Características, aplicações e exemplos de sistemas. **Escola Regional de Banco de Dados–Sociedade Brasileira de Computação (IX ERBD–SBC), Camobiu**, 2013. Citado na página 9.

STRAUCH, C.; SITES, U.-L. S.; KRIHA, W. Nosql databases. **Lecture Notes, Stuttgart Media University**, v. 20, 2011. Citado na página 9.

WELBORN, J. **Learning Compound Index Selection in PostgreSQL with Deep Reinforcement Learning**. Tese (Doutorado) — Master's thesis, University of Cambridge, 2019. Unpublished dissertation . . . , 2019. Citado 2 vezes nas páginas 9 e 16.