SÃO LUCAS EDUCACIONAL DE JI-PARANÁ

BRUNO DE BRITO BISPO

TUNING EM BANCO DE DADOS POSTGRESQL PARA SERVIDORES DE BAIXA PERFORMANCE: UMA ABORDAGEM DE USO DA EMPRESA POSTO LINHA 94

Ji-paraná
2021

SÃO LUCAS EDUCACIONAL DE JI-PARANÁ

BRUNO DE BRITO BISPO

TUNING EM BANCO DE DADOS POSTGRESQL PARA SERVIDORES DE BAIXA PERFORMANCE: UMA ABORDAGEM DE USO DA EMPRESA POSTO LINHA 94

Projeto de Pesquisa apresentado à disciplina de Trabalho de Conclusão de Curso, do Sétimo Período do Curso de Sistemas de Informação, do Grupo Educacional São Lucas, como parte dos requisitos para obtenção do título de Bacharel, sob a orientação do Prof. José Rodolfo Milazzotto Olivas

Ji-paraná
2021

SÃO LUCAS EDUCACIONAL DE JI-PARANÁ

Dados Internacionais de Catalogação na Publicação - CIP

B622t Bispo, Bruno de Brito.

Tuning em banco de dados PostgreSQL para servidores de baixa performance: uma abordagem de uso da empresa Posto Linha 94. / Bruno de Brito Bispo. – Ji-Paraná, 2021.
42 p.; il.

Trabalho de Conclusão de Curso (Curso de Sistemas de Informação) – Centro Universitário São Lucas Ji-Paraná, 2021.

Orientador: Prof. Esp. José Rodolfo Milazzotto Olivas

1. Tuning. 2. Refinamento. 3. Gargalos no Postgres. 4. Banco de dados. I. Olivas, José Rodolfo Milazzotto. II. Título.

CDU 004.43

Ficha Catalográfica Elaborada pelo Bibliotecário Giordani Nunes da Silva CRB 11/1125

SÃO LUCAS EDUCACIONAL DE JI-PARANÁ

BRUNO DE BRITO BISPO

TUNING EM BANCO DE DADOS POSTGRESQL PARA SERVIDORES DE BAIXA PERFORMANCE: UMA ABORDAGEM DE USO DA EMPRESA POSTO LINHA 94

Projeto de Pesquisa apresentado à disciplina de Trabalho de Conclusão de Curso, do Sétimo Período do Curso de Sistemas de Informação, do Grupo Educacional São Lucas, como parte dos requisitos para obtenção do título de Bacharel, sob a orientação do Prof. José Rodolfo Milazzotto Olivas.

Ji-paraná, dezembro de 2021.
Avaliação/Nota:

BANCA EXAMINADORA:

_____
Titulação e Nome
Centro Universitário São Lucas Ji-paraná

_____
Titulação e Nome
Centro Universitário São Lucas Ji-paraná

_____
Titulação e Nome
Centro Universitário São Lucas Ji-paraná

RESUMO

Com o crescente volume de dados que as empresas devem armazenar, tem-se cada vez mais necessidade de se usar um sistema de banco de dados para controlar isso. Porém nem sempre as empresas investem em um servidor adequado às suas necessidades, visto que é lá que se é instalado o banco de dados. O objetivo deste estudo de caso é melhorar o desempenho de um servidor de banco de dados com baixa performance, tendo em vista as empresas que não dão tamanha importância a este. Dessa forma, é papel do Analista de banco de dados fazer um bom trabalho de Tuning para minimizar ao máximo a falta de um bom hardware. Para ter resultado, é preciso conhecer as ferramentas necessárias, como ajustes no sistema operacional e no banco de dados. Abordaremos aqui passo a passo como ajustar um sistema operacional Windows e traremos refinamentos dentro de um banco de dados PostgreSQL, fazendo comparativos das respostas antes e depois de cada alteração.

Palavras-chave: Tuning. Refinamento. Gargalos no Postgres.

ABSTRACT

With the growing volume of data that companies must store, there is an increasing need to use a database system to keep track of this. But companies do not always invest in a server that is suitable for their needs, since that is where the database is installed. The goal of this case study is to improve the performance of a database server with low performance, since companies do not give such importance to it. Thus, it is the Database Analyst's role to do a good Tuning job to minimize the lack of good hardware as much as possible. To get results, it is necessary to know the necessary tools, such as adjustments to the operating system and the database. We will approach here step by step how to tune a Windows operating system and bring refinements within a PostgreSQL database, making comparisons of responses before and after each change.

Keywords: Tuning. Refinement. Postgres bottlenecks.

LISTA DE TABELAS

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
IBM	International Business Machines Corporation
WORD	Microsoft Word
XML	Extensible Markup Language
E-MAIL	Eletronic Mail
GHZ	Giga-hertz
GUI	Graphical User Interface
SRS	Senhores
IT	Information Technology
GPL	General Public License
CPU	Unidade Central de Processamento
POSTGRES	PostgreSQL
PL/pgSQL	Linguagem Procedural estendida do Postgres
PSQL	Terminal front-end do Postgres
QUERY	Consulta
ART	Artigo
FUNCTIONS	Funções

SUMÁRIO

1 INTRODUÇÃO

Viver hoje é muito mais difícil do que há 20 anos atrás. Estamos em um mundo onde a informação corre veloz e as empresas demandam um armazenamento de um grande volume de informação para sobreviverem. Mas, infelizmente algumas empresas não estão preparadas para essa realidade, investindo em equipamentos ruins que não suportam seus sistemas. Com isso em mente, faremos um estudo de caso, levando em consideração quais os problemas que uma empresa que usa um servidor com pouca potência de hardware enfrenta nos sistemas e iremos abordar como um analista pode detectar problemas que influenciam diretamente no desempenho de um banco de dados, assim, extraindo o máximo através de ajustes. Abordaremos também como fazer pequenas mudanças no sistema operacional do servidor para melhorar a eficiência do banco de dados, refinaremos algumas consultas às tabelas mais pesadas para minimizar o tempo de espera para o usuário final e iremos gerar relatórios para comparar antes e depois, enfatizando assim, a importância desse tipo de trabalho para o melhor aproveitamento das aplicações.

1.1 PROBLEMATIZAÇÃO

Algumas pequenas e médias empresas possuem sistemas com banco de dados robustos que exigem boa performance de hardware. Porém, nem todos os proprietários desse tipo de comércio estão dispostos a adquirir um bom servidor para gerir esse sistema. Dessa forma, é comum vermos bancos de dados serem instalados em estações de trabalho, com sistema operacional de uso pessoal e muitas vezes, nem são servidores dedicados. Assim, como aperfeiçoar o desempenho do banco de dados neste contexto?

1.2 HIPÓTESES

Apesar de quase sempre o desempenho do próprio hardware do computador definir o quão ágil serão as aplicações, sempre podemos melhorá-las usando as ferramentas que temos disponíveis. É possível testarmos em um servidor de baixo desempenho, limitá-lo apenas para uso do sistema e aplicarmos boas práticas de banco de dados para extrairmos o máximo de performance, melhorando assim o tempo de resposta da nossa aplicação. Para esse refinamento que é feito para melhorar o banco de dados, damos o nome de Tuning, ou sintonia de banco de dados.

1.3 OBJETIVO GERAL

Pesquisar e comparar algumas soluções práticas para obter melhoria significativa de performance no banco de dados, reduzindo o tempo de resposta para determinadas operações.

1.4 OBJETIVOS ESPECÍFICOS

Demonstrar procedimentos de identificação de problemas de performance do banco de dados;

Fazer otimizações de consultas SQL e mostrar boas práticas ao elaborá-las;

Gerar relatórios comparando o tempo de consulta antes e depois, especificando softwares que contribuíram para tal resultado.

1.6 JUSTIFICATIVA

Este estudo irá avaliar a qualidade dos ajustes de sintonia no banco de dados. Essa prática é importante no projeto do banco de dados e faz com que melhore todo o sistema, deixando-o mais fluido e melhorando a experiência dos usuários das aplicações.

A motivação vem do trabalho com sistemas para pequenas e médias empresas e o presente estudo pode beneficiar toda a comunidade de analistas de sistemas. Assim, gostaria de inspirar os colegas de profissão a aperfeiçoar os mais diversos sistemas através desta contribuição.

Como justificativa do sistema gerenciador de banco de dados, utilizou-se o PostgreSQL devido a sua ampla comunidade de usuários que o alimenta, o tornando uma excelente opção a ser estudada.

2 REFERENCIAL TEÓRICO

2.1 DADO E INFORMAÇÃO

Os dados são termos isolados, que são usados para que se forme uma informação. Podemos representar dados como as letras, que formam as palavras, e as palavras são as informações que transmitimos, e não ao contrário. Da mesma forma, no mundo corporativo as empresas precisam fazer uma coleta de dados de seus clientes, fornecedores, de seu financeiro, e etc. Tudo isso fica registrado em computadores que são chamados de servidores, aonde ficam responsáveis por gerir os dados e as aplicações das empresas utilizando sistemas específicas para essa finalidade.

> A indústria de produção de informação tem se desenvolvido à margem das revoluções e do crescimento industrial, absorvendo, assim, as suas características marcantes. A geração de estoques de informação adotou para si os preceitos da produtividade e da técnica como o seu mercado de trabalho. A crescente produção de informação precisa ser reunida e armazenada de forma eficiente, obedecendo critérios de produtividade na estocagem, ou seja, o maior número de estruturas informacionais deve ser colocado em menor espaço possível dentro de limites da eficácia e custo. (Barreto, 1994, p.3)

Concluímos então que, desde o passado, pesquisadores estudam a informação e a importância do armazenamento dela para uso corporativo e para uso pessoal. E há pessoas que se preocupam com a forma como ela deve ser gerida e armazenada.

2.2 HISTÓRIA E CONCEITO DE BANCO DE DADOS

No passado, antes de existirem sistemas de computador, as empresas armazenavam os dados em arquivos físicos. Quaisquer que fossem os dados, seja de cliente, funcionários, fornecedores, caixas, financeiro, contas a pagar ou receber da empresa. Para isso, era necessário ter um volume de funcionários para armazenar e organizá-los. Quanto maior a empresa, mais dados a serem organizados e, consequentemente, mais funcionários. Esse processo acabava se tornando custoso para as empresas, aumentando os gastos com funcionários, com papeis e arquivos, além de gastar tempo para procurá-los. Assim então foram surgindo buscas de se melhorar esse processo.

Igualmente a muitas tecnologias na computação industrial, os fundamentos de bancos de dados relacionais surgiram na empresa IBM, nas décadas de 1960 e 1970, através de pesquisas de

funções de automação de escritório. Foi durante um período da história na qual empresas descobriram que estava muito custoso empregar um número grande de pessoas para fazer trabalhos como armazenar e indexar (organizar) arquivos. Por este motivo, valia a pena os esforços e investimentos em pesquisar um meio mais barato e ter uma solução mecânica eficiente. (SANCHES, 2005, p.1)

Nesse cenário, em meados dos anos 1970, grandes empresas começaram a investir em pesquisas com foco em obter uma maneira de abaixar os custos com esse processo, visto que não haviam muitas tecnologias na época e os dados começavam a ser guardados digitalmente, porém de forma simples, salvos em um arquivo com uma sequência única de informações. O que era ruim, pois era difícil manutenção e difícil crescimento. Logo, começava a se pensar em uma maneira de baixo e custo e alta escalabilidade.

> Em 1970 um pesquisador da IBM - Ted Codd - publicou o primeiro artigo sobre bancos de dados relacionais. Este artigo trata sobre o uso de cálculo e álgebra relacional para permitir que usuários não técnicos armazenassem e recuperassem grande quantidade de informações. Codd visionava um sistema onde o usuário seria capaz de acessar as informações através de comandos em inglês, onde as informações estariam armazenadas em tabelas. (SANCHES, 2005, p.1)

Então, a partir desse artigo, começava a surgir surgirem os bancos de dados e a usar a linguagem estruturada, a qual se tornou posteriormente a linguagem padrão. Apesar disso, não foi a IBM quem inventou o primeiro sistema de banco de dados. Assim, podemos definir um banco de dados como nada mais do que um conjunto de dados relacionados entre si, disponíveis para serem acessados a qualquer momento por qualquer pessoa autorizada a acessá-lo.

> Os sistemas de banco de dados são projetados para gerenciar grandes grupos de informações. O gerenciamento de dados envolve a definição de estruturas para armazenamento de informação e o fornecimento de mecanismos para manipulá-las. Além disso, o sistema de banco de dados precisa fornecer segurança das informações armazenadas, caso o sistema dê problema, ou contra tentativas de acesso não-autorizado. Se os dados devem ser divididos entre diversos usuários, o sistema precisa evitar possíveis resultados anômalos. (SANCHES, 2005, p.1)

Logo, um banco de dados não somente armazena as informações, mas estrutura em níveis de segurança para protegê-las contra usuários que não devem acessá-las. Podemos fazer uma analogia a como no passado, aonde precisava-se que os dados fossem protegidos em salas onde somente pessoas autorizadas entravam. Assim, no presente, as empresas investindo em um banco de dados, além de evitarem despesas

com funcionários e pastas para armazenamento, se ganha em segurança, volume rapidez e escalabilidade.

2.3 BANCO DE DADOS RELACIONAL

No modelo relacional, os dados são armazenados em um conjunto de tabelas bidimensionais, e esses dados podem ser alterados por uma linguagem específica, que é interpretada pelo sistema e que podem ser alterados em grande volume ou não. No modelo relacional podemos imaginar nosso banco de dados como uma grade, onde as linhas são as tuplas, as colunas são os atributos e as tabelas são as relações entre as entidades.

> Informalmente, uma relação é uma tabela na qual cada linha expressa em uma coleção de dados relacionados, cujos valores podem ser interpretados como um fato que descreve uma entidade ou um relacionamento. As linhas de uma relação, ou tabela, são chamadas de tuplas. O cabeçalho das colunas representa atributos e o conjunto de valores que podem aparecer em cada coluna é chamado de domínio. (SANTANCHÈ; ROCHA, 2012 p.5)

2.4 SISTEMA GERENCIADOR DE BANCO DE DADOS

Os sistemas gerenciadores de banco de dados vieram para facilitar o gerenciamento e armazenamento de bancos de dados de forma rápida e eficaz através de comandos e interface intuitiva. Existem diversos no mercado, como MySQL, Oracle, Postgres, MariaDB, SQL Server, entre outros.

> [...] um banco de dados é uma estrutura computacional compartilhada que armazena um conjunto de dados do usuário final. Para manipularmos esses dados, usamos o Sistema Gerenciador de Banco de Dados (SGBD) que é um conjunto de programas que gerenciam a estrutura do banco de dados e controlam o acesso aos dados armazenados, além de garantir a disponibilidade dos dados de forma eficaz. (ROB; CORONEL, 2011, p.6 apud THUMS, 2018, p14)

As vantagens de se usar um SGBD são imensas. Podemos manipular milhares de bytes de forma simples e precisa, com segurança e agilidade. Existem diversos tipos de SGBDs no mercado, gratuitos e pagos. Abordaremos a seguir um SGBD pago e um gratuito e iremos comparar as diferenças para que possamos entender porquê neste projeto utilizaremos o PostgreSQL.

Logo, vemos que, assim como o existem ferramentas que processam textos e nos dão uma plataforma completa para seus usuários gerenciá-los, o banco de dados armazena os dados dos usuários e também te permite gerenciá-los com facilidade.

Apesar de que os textos criados em tais ferramentas não dependerem exclusivamente delas, da mesma forma os bancos de dados não dependem exclusivamente de um sistema gerenciador de banco de dados. Existem outras soluções que nos trazem a finalidade de gerenciar informações, como serialização, arquivos binários e xml. Porém, muitas vezes a quantidade de dados são limitados e os recursos de compartilhamento com outros usuários em tempo real não existem. Então com um SGBD temos uma ferramenta que nos permite criar, acessar, proteger, editar, excluir, importar e exportar dados.

2.5 MYSQL

O MySQL é um sistema gerenciador de banco de dados criado em 1994 construído no modelo cliente-servidor, é multiplataforma, construído na linguagem C, que utiliza a linguagem SQL como interface. Originalmente foi projetado como uma solução para gerenciar grandes bancos de dados com rapidez e agilidade, tendo seu grande foco em ambiente web. Atualmente o MySQL possui a licença GPL, sendo necessário adquirir uma licença para projetos que não utilizam dessa licença.

> [...] O MySQL é um servidor e gerenciador de banco de dados relacional, de licença dupla (sendo uma delas de software livre), projetado inicialmente para trabalhar com aplicações de pequeno e médio portes, mas hoje atendendo a aplicações de grande porte e com mais vantagens do que seus concorrentes. Possui todas as características que um banco de dados de grande porte precisa, sendo reconhecido por algumas entidades como o banco de dados open source com maior capacidade para concorrer com programas similares de código fechado[...] (Milani, 2006a, p.22)

O MySQL além de um banco de dados, também possui um módulo gerenciador de banco de dados, com diversas funcionalidades, como controle de concorrência e modelagem de dados. De fato, é um gerenciador de banco de dados com todas as funções de um excelente SGBD. Porém, é preciso avaliar o uso e a finalidade da aplicação, pois é permitido modificar seu código fonte caso seja necessário, mas se o usuário pretende utilizá-lo para fins comerciais e, ou distribuí-lo junto à sua aplicação, deve comprar a licença de uso.

2.6 POSTGRESQL

O Postgres, como é comumente chamado, é um sistema gerenciador de banco de dados multiplataforma, escrito na linguagem C, descendente de código aberto com uma vasta comunidade que o atualiza periodicamente.

> Para tanto, usa-se as funcionalidades de triggers, visões, procedures, chaves estrangeiras, integridades transacionais, data types e agregações.
> O PostgreSQL suporta cargas de trabalho consideráveis e consegue processar grandes volumes de informações. Esse sistema executa consultas SQL para retornar informações e mantém vários módulos para otimizar a performance das aplicações internas. (Multiedro, 2019)

Apesar de ser uma ferramenta de código aberto, ele possui as mais avançadas ferramentas para gerenciamento de bancos de dados. Conta com recursos complexos, como: controle de concorrência multi-versão, linguagem procedural em várias linguagens para procedimentos armazenados, estrutura para guardar dados georreferenciados, etc. Além de tudo, pessoas interessadas em ampliar e implementar melhorias no software ou fazer correções pode enviar sugestões e a própria equipe que desenvolve pode aceitar a mudança e lançar em uma nova atualização para todos os usuários. Tendo isso em vista, e levando em consideração que não teríamos gastos pecuniários, foi decidido então usá-lo como nosso banco de dados neste projeto.

2.7 INTRODUÇÃO AO SQL TUNING

Na realidade em que vivemos, o volume de dados dos sistemas das organizações estão se expandindo exponencialmente, devido ao grande volume de informação coletada dos usuários, dados como: preferências, lugares que visitou, e-mail de contato e até mesmo vídeos que assistiu. E isso se transforma em terabytes de informação que serão controlados por SGBDs. Logo, a disputa por esses dados pode acarretar lentidão e atraso no tempo de resposta das consultas.

> [...] Tuning diz respeito ao ajuste do SGBD para melhor utilização dos recursos, provendo um uso eficaz e eficiente do SGBD. A fase de Tuning de um BD é um processo de refinamento que envolve modificações em vários aspectos, abordando desde mudanças nos conceitos aprendidos nos Diagramas Entidade-Relacionamento até a troca de hardware, passando pela configuração dos softwares que executam nesse sistema. (Souza; Campos; Dias; Alves; Vieira; Carelli; Sá, 2009a, p.20 apud Baptista, 2008, p. 15)

Em muitos casos os bancos de dados crescem gigantescamente, mas a configuração dos servidores permanece a mesma. É um conceito que, infelizmente, em alguns casos é preciso se trabalhar e amadurecer nas empresas. Nesse contexto, surgiu a necessidade de melhorar o desempenho do banco de dados através de refinamentos e ajustes, melhorando o tempo de resposta das operações.

2.8 PROBLEMAS DE DESEMPENHO NO SISTEMA OPERACIONAL

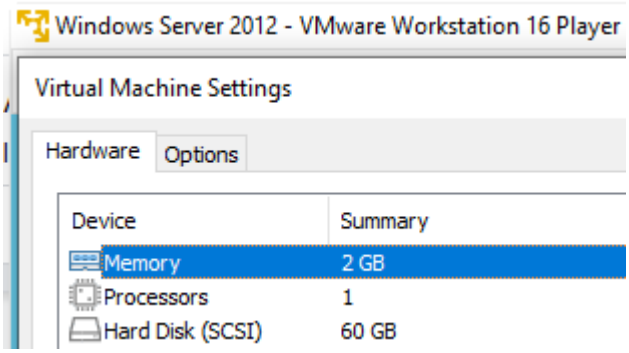
Em nosso trabalho, iremos focar nos servidores com baixa performance, logo teremos que estudar a melhor maneira de aumentar o desempenho do nosso banco através de ajustes, seja no sistema operacional, nos arquivos do postgres ou através de ajustes manuais dentro do banco de dados. É necessário levar em consideração que diferentes tipos de sistemas operacionais podem acarretar em desempenhos diferentes. E por estarmos focados em comércios, utilizaremos o sistema operacional Microsoft Windows, por ser o sistema operacional utilizado pela maior parte dos usuários e também pela maior facilidade didática.

> Em debate com profissionais com mais de 10 anos de experiência em gerenciamento de banco de dados, os Srs. Lúcio Gomes Peixoto Júnior e Erick de Souza Carvalho, constata-se que 45% das ações de Tuning são destinadas a configuração do Sistema Operacional, 35% destinadas a configuração do SGBD e 20% destinadas ao refinamento das consultas. (Souza; Campos; Dias; Alves; Vieira; Carelli; Sá, 2009b, p.20 apud Peixoto Júnior e Carvalho, 2008)

Como vemos, uma grande parte do desempenho do banco de dados se dá com ajustes no sistema operacional. Um ajuste recomendado pela Microsoft é a remoção da GUI dos sistemas operacionais Windows Server, deixando apenas uma interface de linha de comando. Assim, reduz-se drasticamente o consumo de processamento do servidor. Neste estudo, nosso servidor não é dedicado, assim, não removeremos a interface gráfica.

O nosso servidor tem a seguinte configuração:

Figura 1 – Configurações da máquina virtual



Fonte: Próprio autor

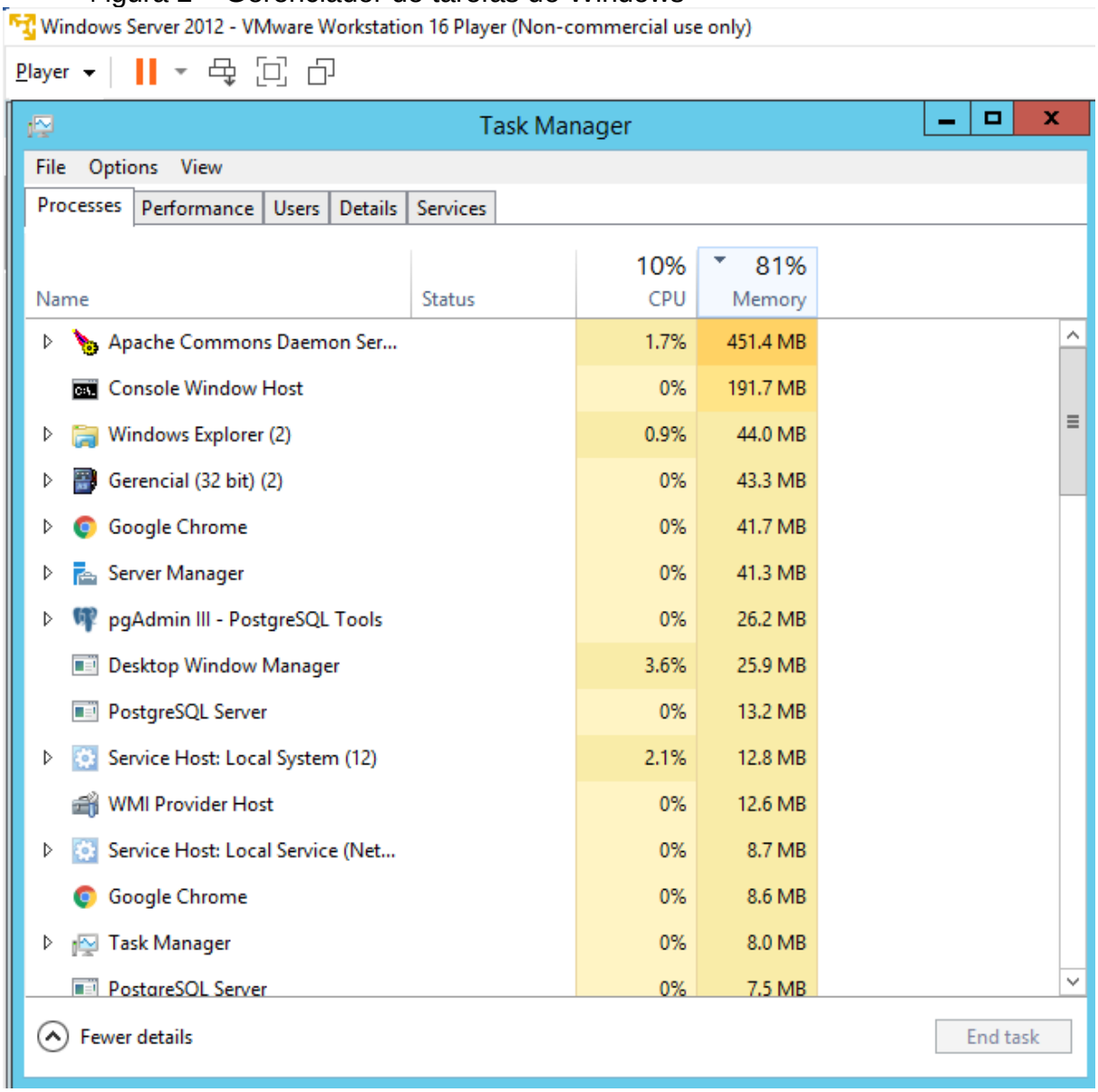
Hardware: 2 Gigabytes de Memória, 1 núcleo de processador de 2.80GHz e 60 Gigabytes de Armazenamento. **Software:** Usaremos o sistema operacional Windows Server 2012 R2 Standard dentro de uma máquina virtual criada pelo autor. Usaremos

a versão do Postgres 9.5 devido a melhor compatibilidade de nossas tabelas nesta versão do que às novas.

Para que nosso banco de dados possa fluir livremente sem gargalos, é necessário entendermos o quanto as aplicações instaladas exigem dos nossos recursos de hardware como processador, disco e memória ram e devemos monitorar, não deixando chegar na capacidade máxima. “Para a realização do processo de *Tuning*, é necessário o entendimento de como o sistema operacional gerencia os recursos disponíveis e cada subsistema poderá ser ajustado, dependendo o cenário proposto” [...] (Zorzi, 2015a, p.26) .

Uma função básica no Windows é o Gerenciador de tarefas, que pode ser aberto pressionando as teclas CTRL + SHIFT + ESC simultaneamente.

Figura 2 – Gerenciador de tarefas do Windows

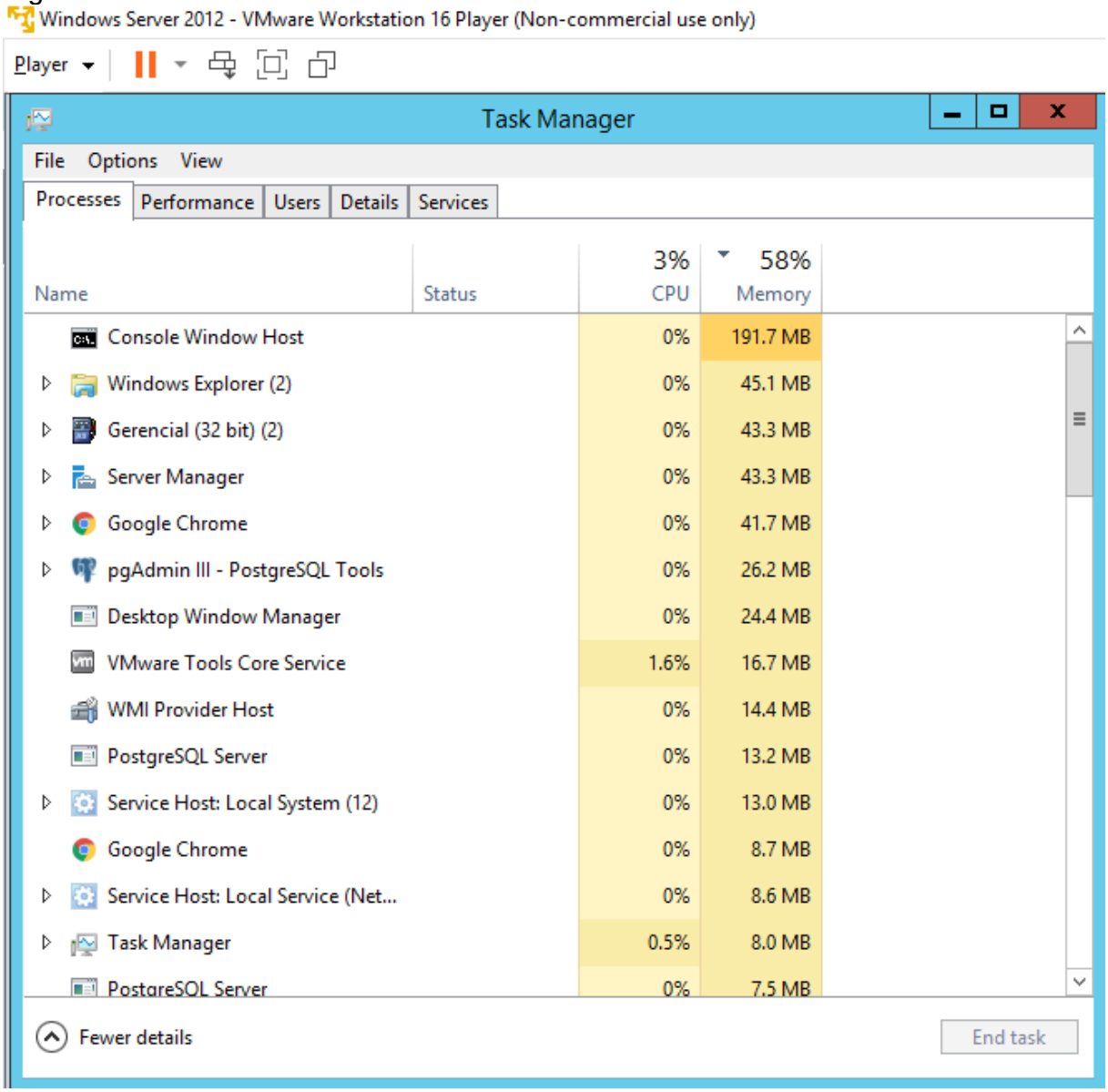


Fonte: Próprio autor

No nosso servidor de exemplo, vemos que o maior problema está na memória ram, que com poucas aplicações em aberto já está consumindo 81% de memória e em um uso cotidiano com várias aplicações abertas irá chegar nos 100%. Logo, devemos entender quais aplicações mais exigem recursos de memória e verificarmos a real necessidade delas. Em nosso caso, o maior problema é porque temos um outro servidor de banco de dados chamado “Apache” que está consumindo 451.4MB e 1.7% do nosso processador. Por serem robustos, os sistemas de banco de dados consomem muitos recursos e por esse motivo não é interessante deixarmos dois bancos de dados em um mesmo computador.

> A identificação dos gargalos de memória se dá realizando o levantamento das aplicações que rodam no servidor e através de ferramentas de monitoramento para verificar como o sistema está alocando os recursos necessários. (ZORZI, 2015b, p.21)

Figura 3 – Gerenciador de tarefas do Windows



Fonte: Próprio autor

Ao desinstalarmos o Apache, 30% de memória ram está liberada, abaixando para 58% e o uso de CPU abaixou para 3%. Assim podemos destinar mais recursos ao nosso banco de dados e outras aplicações.

Apesar de que no neste caso o problema de consumo era o servidor Apache sem necessidade, essa demonstração é apenas para evidenciar que devemos procurar observar quais são as aplicações em execução no momento, o porquê de elas estarem executando e se há, de fato, necessidade de destinarmos recursos a elas. Existem diversos tipos de aplicações que consomem alto processamento e memória, como antivírus, navegadores, players, dependências de programas e até mesmo vírus.

> A verdade é que não existe uma solução definitiva para toda e qualquer inconveniência quanto às quedas de desempenho. No entanto, você tem algum poder sobre o sistema e pode tentar se prevenir para evitar que os programas cheguem a seus limites e comecem a prejudicar o uso geral do computador. (TECMUNDO, 2017)

Logo, um bom trabalho de Tuning não envolve apenas entender sobre o banco de dados ou sobre linguagem sql. Mas, o analista precisa se atentar aos fatores externos à aplicação.

2.9 IDENTIFICAÇÃO DE PROBLEMAS NO POSTGRES

Para identificarmos problemas de desempenho podemos monitorar nosso banco a fim de encontrar as transações que mais são executadas, quanto tempo levam essas transações, como também podemos consultar quais tabelas que possuem maior tamanho. Alguns índices podem ser construídos a partir dessas informações. Para o monitoramento podemos utilizar de consultas via linha de comando, logs de transações no arquivo do postgres e também ferramentas externas.

> [...] Outras informações obtidas a partir do monitoramento das atividades do sistema de banco de dados incluem as seguintes:
> • estatísticas de armazenamento: dados a respeito da alocação de armazenamento para espaço de tabelas, espaço de índices e portas de buffer.
> • estatísticas de desempenho de entrada/saída: atividade total de leitura/escrita (paginação) do disco.
> • estatísticas de processamento de consultas: tempos de execução de consultas, tempos de otimização durante a otimização de consultas. [...] (CARNEIRO, et al, 2009, p.2)

Nessa parte de identificação, devemos encontrar a origem do problema. O postgres conta com um arquivo na sua pasta de instalação chamado “postgresql.conf” e nele é onde ficam configurados os parâmetros de todo o sistema, como alocação de memória para compartilhamento com disco, memória utilizada nas operações internas e consumo de recursos na cpu por consulta. Devemos ter ciência da nossa estrutura física para definirmos esses parâmetros.

> Ajustar de forma apropriada os recursos de memória para as estruturas do SGBD pode beneficiar o desempenho. Alocar esses recursos de memória da melhor forma possível e definir corretamente o *buffer* do banco de dados contribui na redução da paginação, que é o processo de virtualização da memória que faz a subdivisão desta memória física em pequenas partições. (ANCHIETA, 2012, p.20)

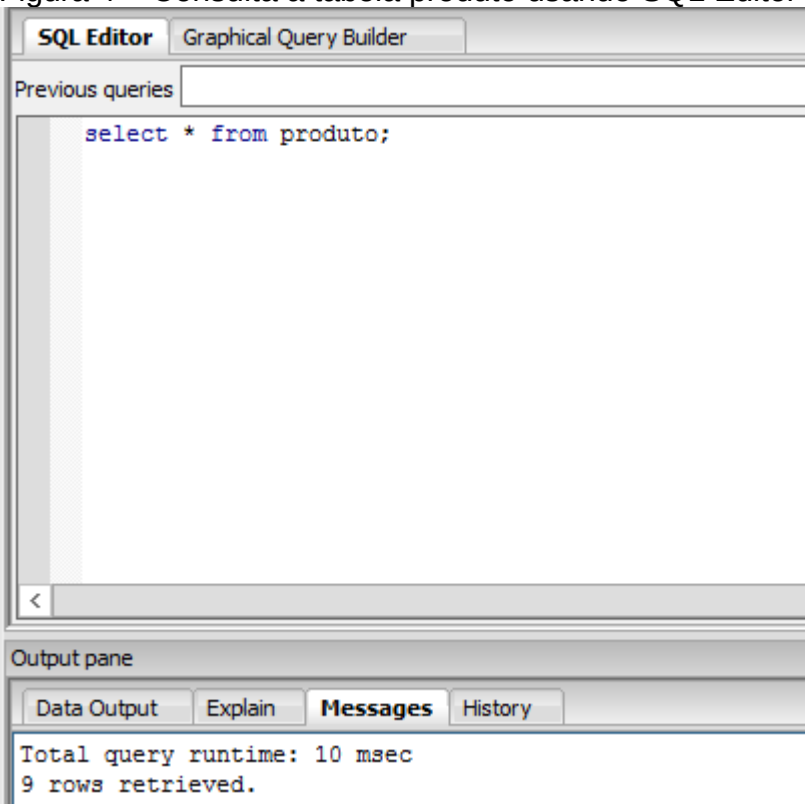
2.10 SINTONIA DE TABELAS

Para começarmos nosso refinamento no postgres, iniciaremos pelas tabelas. Deve-se procurar consultas com duração de tempo alto e que exigem muito processamento do servidor, consultas mal elaboradas, índices não utilizados e tabelas muito utilizadas ou de grande tamanho, mas sem índices criados. Para isso, iniciaremos falando sobre o comando *vacuum*.

Quando é feito um delete, o postgres assim como muitos sistemas não apaga a linha que está com esse registro, mas ele irá escrever em uma linha abaixo e aquela antiga linha ficará em branco. Isso gera inúmeras linhas inúteis que acarretam em um maior espaço em disco. Para limparmos essas linhas em branco e deixar o espaço para ser utilizado novamente, é utilizado o vacuum.

Usando nosso sgbd, faremos uma consulta à tabela produto que possui 5 linhas e veremos quanto tempo leva para processar nossa query.

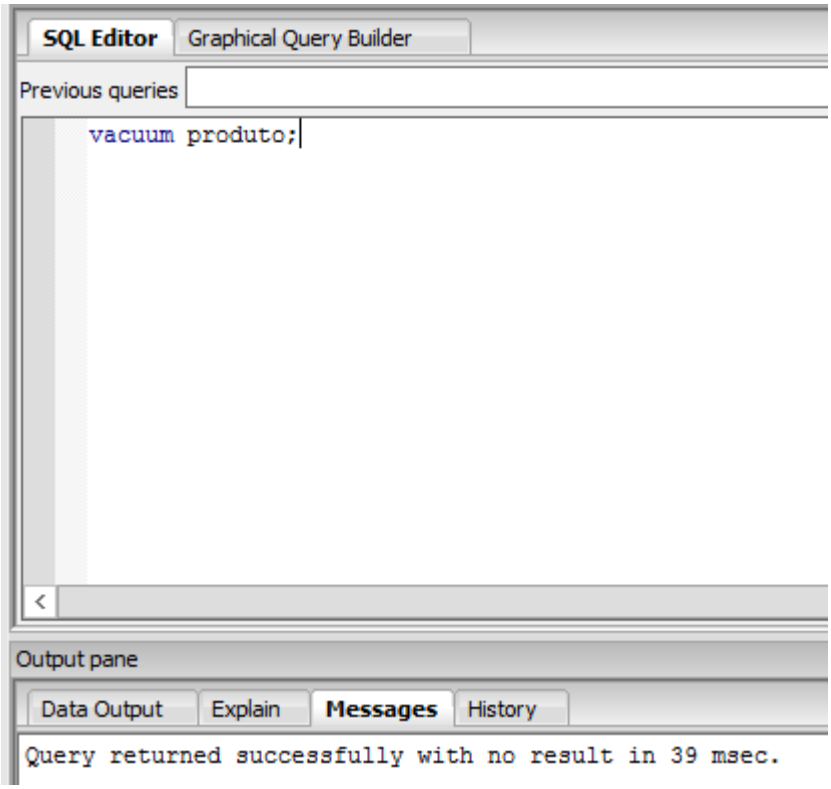
Figura 4 – Consulta à tabela produto usando SQL Editor do Postgres



Fonte: Próprio autor

A query levou 10ms para percorrer toda a tabela que haviam 9 linhas. Tendo esse resultado armazenado na foto acima, faremos agora o vacuum na mesma tabela

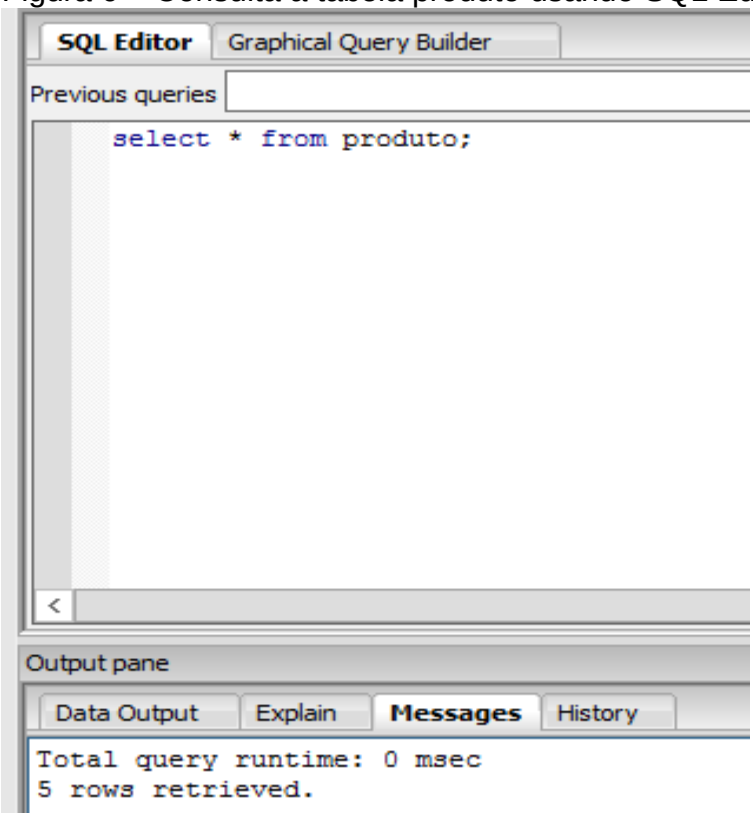
Figura 5 – Comando vacuum à tabela produto usando SQL Editor do Postgres



Fonte: Próprio autor

Nossa query levou 39 ms para executar. Após isso, faremos novamente a mesma consulta na tabela:

Figura 6 – Consulta à tabela produto usando SQL Editor do Postgres

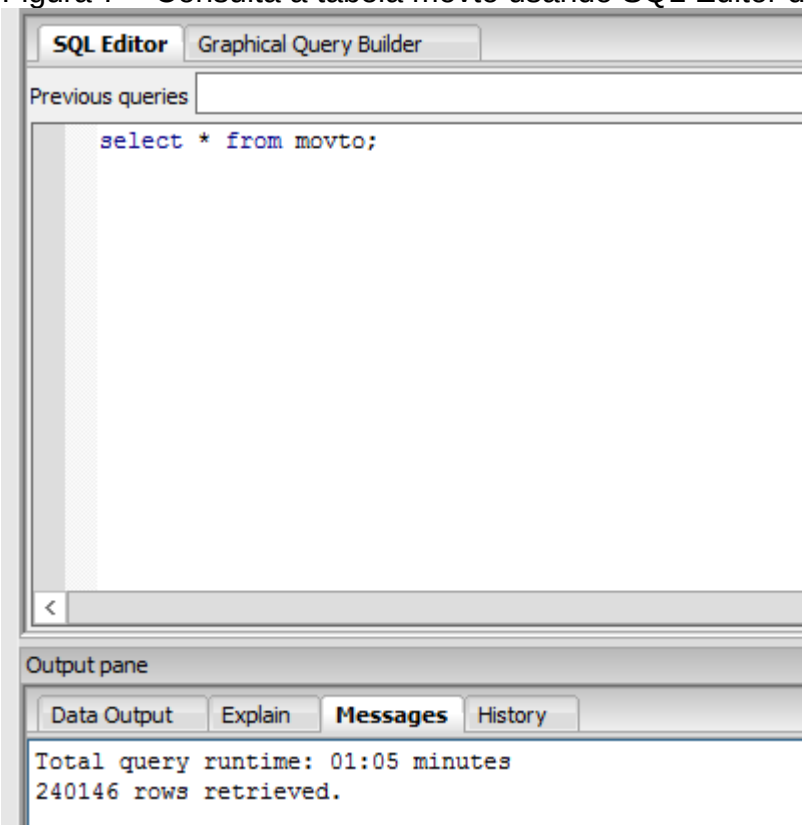


Fonte: Próprio autor

Apesar do tempo ter levado milissegundos para executar, em termos percentuais temos 100% de melhora nessa consulta.

Faremos agora um comparativo com uma tabela com milhares de linhas

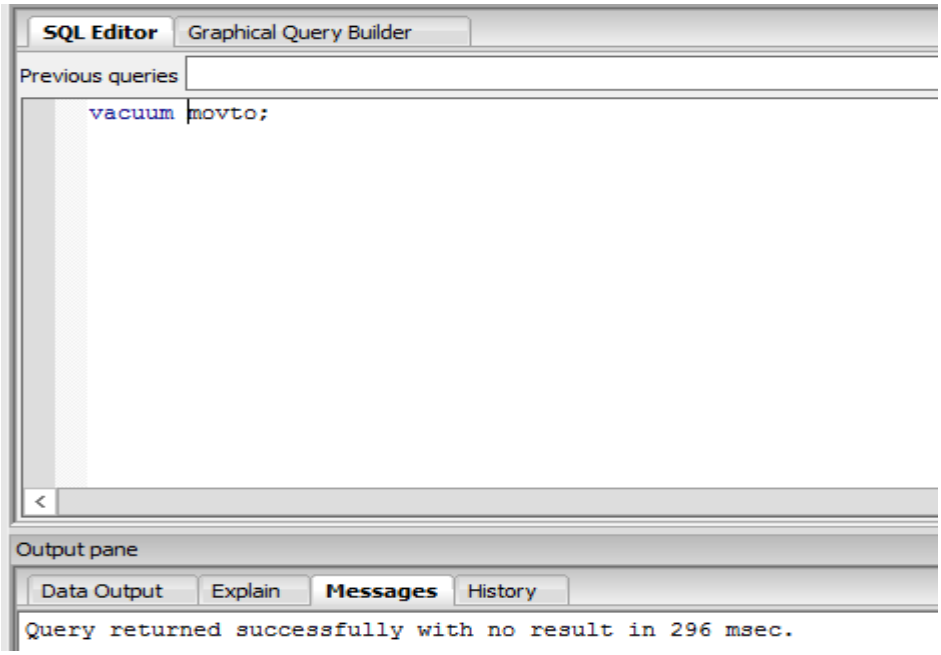
Figura 7 – Consulta à tabela movto usando SQL Editor do Postgres



Fonte: Próprio autor

Nessa tabela, temos 240 mil linhas que se levaram 01:05 minutos para percorrer ela inteira. Faremos agora o vacuum e posteriormente a comparação da mesma consulta.

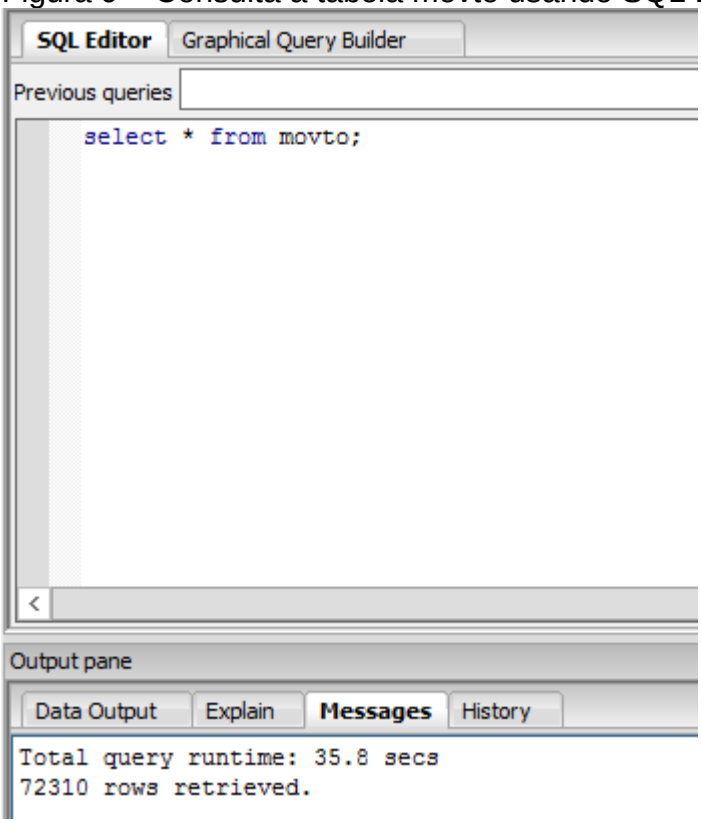
Figura 8 – Comando vacuum à tabela produto usando SQL Editor do Postgres



Fonte: Próprio autor

Levou-se 296 ms para realizar o vacuum da nossa tabela. E faremos novamente a consulta:

Figura 9 – Consulta à tabela movto usando SQL Editor do Postgres



Fonte: Próprio autor

Como podemos ver, houve uma melhora significativa de 44% no tempo de resposta da aplicação e foram diminuídas 167.836 linhas da tabela.

2.11 ÍNDICES

Os índices fazem com que as aplicações não percorram a tabela sequencialmente em busca da informação, mas que encontrem rapidamente os dados. Podemos fazer alusão a um livro, onde existe uma página de índices indicando qual página fica cada assunto. Dessa mesma forma trabalhamos no banco de dados. Para criarmos índices no banco de dados, usamos o comando “create index”.

> Os índices são uma forma comum de aprimorar o desempenho do banco de dados. Um índice permite que o servidor de banco de dados encontre e recupere linhas específicas muito mais rápido do que faria sem um índice. Mas os índices também adicionam sobrecarga ao sistema de banco de dados como um todo, portanto, devem ser usados de maneira sensata. (POSTGRESQL, 2021)

Ao gravar uma informação, o banco grava na tabela e no índice, logo é uma dupla gravação de informação. Então uma forma de melhorar a performance do nosso banco de dados é diminuindo a sobrecarga de informação, excluindo os índices não utilizados.

Figura 10 – Consulta à índices pela interface de linha de comando do Postgres

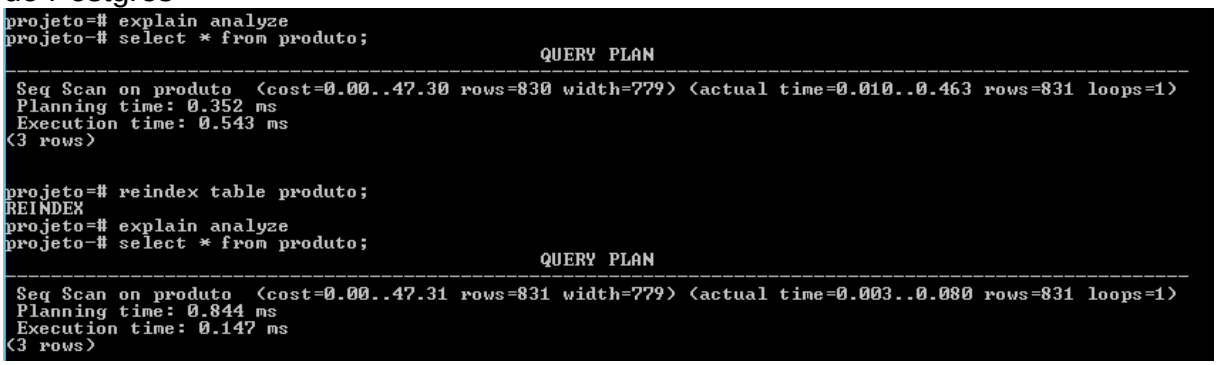
Fonte: Próprio autor

Através de uma query em nosso banco, listamos alguns índices que nunca foram utilizados e que ocupam espaço em nosso disco. Eliminando-os, além de melhorar o desempenho, ganharemos armazenamento.

Existe também um comando chamado REINDEX que elimina as tuplas e páginas mortas ou danificadas dentro de uma tabela, liberando assim espaço livre delas para o disco. A sintaxe desse comando seria: reindex table minha_tabela.

> O comando REINDEX reconstrói o índice usando os dados armazenados na tabela do índice, substituindo a cópia antiga do índice. Existem várias situações nas quais o comando REINDEX é utilizado:
>
> - O índice está danificado, e não contém mais dados válidos.
> - O índice ficou "dilatado", ou seja, contém muitas páginas vazias ou quase vazias. Esta situação pode ocorrer com índices B-tree no PostgreSQL sob certos padrões de acesso fora do comum. O comando REINDEX fornece uma maneira para reduzir o consumo de espaço do índice através da escrita de uma nova versão do índice sem as páginas mortas. (POSTRESQL, 2021)

Figura 11 – Consulta e reindex à tabela produto pela interface de linha de comando do Postgres



Fonte: Próprio autor

Como vemos na imagem acima, na nossa primeira consulta à tabela produto através do comando SELECT * FROM que lista todas as tuplas da tabela, nosso banco demorou 0.543ms para executá-lo e isso custou 0.010 ms para nosso banco. Logo após, executamos um reindex na tabela e fizemos a mesma consulta novamente. Os resultados foram que o banco levou 0.147 ms para executá-lo e isso custou 0.003 ms para o banco.

2.12 APERFEIÇOANDO QUERYS

Para começar, iremos fazer algumas consultas e iremos nos basear no custo que o nosso banco de dados leva para gerar a consulta. A partir daí, faremos melhorias, comparando o antes e depois. Para isso, utilizaremos o comando chamado analyze e o comando explain. O banco de dados possui uma tabela de estatística que ao ser realizado uma consulta, verifica se deve ser feito uma consulta sequencial ou se deve ser utilizado um índice. Para isso, utilizamos o comando analyze.

> ANALYZE coleta estatísticas sobre o conteúdo das tabelas no banco de dados e armazena os resultados no catálogo do sistema pg_statistic. Posteriormente, o planejador de consultas usa essas estatísticas para ajudar a determinar os planos de execução mais eficientes para as consultas. (POSTGRESQL, 2021)

O comando explain analisa o plano de execução da nossa query e se foi uma consulta através de um índice ou se foi uma busca sequencial. Após isso, ele nos diz quanto tempo isso custou.

> O comando *explain* é utilizado para visualizar e analisar o comportamento do otimizador sobre uma consulta SQL no SGBD. Com ele, é possível encontrar informações de como é feita a análise das tabelas, como são feitas as junções e a ordem executadas, os custos dos métodos utilizados e o plano de

> execução [...] (MAGALHÃES et al, 2015a, p.10 apud SAMPAIO; GOVEIA; MARQUES, 2011)

Então, vemos que é muito vantajoso planejarmos nossas consultas usando índices. E ao observarmos o nosso banco, podemos utilizar os comandos explain analyze para monitorar as nossas querys. “Com o uso do comando *Explain Analyze*, há estatísticas sobre qual o caminho percorrido pelo otimizador do SGBD e os custos utilizados para chegar ao resultado” (MAGALHÃES et al, 2015b, p.12).

Faremos um teste de consulta sequencial no nosso banco chamado “projeto”:

Figura 12 – Consulta à tabela movto pela interface de linha de comando do Postgres

```
projeto=# explain analyse
projeto-# select * from movto where documento='60874';
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Seq Scan on movto  (cost=0.00..8912.83 rows=1 width=163) (actual time=0.005..245.370 rows=1 loops=1)
   Filter: (documento = '60874'::text)
   Rows Removed by Filter: 240145
 Planning time: 0.186 ms
 Execution time: 245.395 ms
(5 rows)
```

Fonte: Próprio autor

Em nossa query, buscamos um documento pelo número, o qual nosso banco levou o tempo de 0.186 ms para descobrir um plano de consulta e 245.395 ms para executar. Agora criaremos um índice para essa tabela para fazermos consultas posteriormente e compararmos o tempo de execução.

Figura 13 –Criação de um índice pela interface de linha de comando do Postgres

```
projeto=# CREATE INDEX movto_motivo_ix
projeto-#    ON public.movto
projeto-#    USING btree
projeto-#    (motivo, documento COLLATE pg_catalog."default");
CREATE INDEX
projeto=# _
```

Fonte: Próprio autor.

Com nosso índice criado, vamos retornar a mesma query e visualizar o tempo de resposta.

Figura 14 – Consulta à tabela movto pela interface de linha de comando do Postgres

```
projeto=# explain analyse
projeto-# select * from movto where documento='60874';
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Index Scan using movto_motivo_ix on movto  (cost=0.42..5577.53 rows=1 width=163) (actual time=18.062..45.278 rows=1 loops=1)
   Index Cond: (documento = '60874'::text)
 Planning time: 0.169 ms
 Execution time: 45.306 ms
(4 rows)
```

Fonte: Próprio autor

Nessa query, nosso banco levou o tempo de 0.169 ms para descobrir o plano de consulta e 45.306 ms para executá-la. Assim, ficou evidente o tempo de resposta de uma boa consulta.

2.13 VISÕES

Outra forma de maximizar a performance do banco de dados é através de views ou visões. Nesta, iremos melhorar consultas específicas, de preferência as mais utilizadas. “Uma visão (também conhecida como uma relação virtual) é uma relação nomeada cujo valor em um tempo qualquer t é o resultado da avaliação de determinada expressão relacional nesse tempo t...” (Date, 2014, p.46). Sendo assim, a visão nada mais é do que o resultado de uma consulta sql e que pode ser armazenada em uma tabela virtual cujo essa consulta fica armazenada em cache, podendo ser chamada a qualquer momento, o que facilita o desempenho do servidor. Com a view fica simples visualizar consultas complexas e economiza grande quantidade de esforço de digitação, além de apresentar apenas as informações que desejamos que sejam exibidas, influenciando diretamente no processamento da máquina.

Figura 15 – Criação de uma view pelo SQL Editor do Postgres

```
CREATE OR REPLACE VIEW public.municipio_view AS
 SELECT m.codigo,
    m.nome AS municipio,
    uf.codigo AS uf_codigo,
    uf.sigla AS uf,
    uf.nome AS uf_nome,
    mi.codigo AS microregiao_codigo,
    mi.nome AS microregiao_nome,
    me.codigo AS mesoregiao_codigo,
    me.nome AS mesoregiao_nome
   FROM municipio m
     JOIN micro_regiao mi ON m.micro_regiao = mi.codigo
     JOIN meso_regiao me ON mi.meso_regiao = me.codigo
     JOIN unidade_federativa uf ON me.uf = uf.codigo;

ALTER TABLE public.municipio_view
  OWNER TO postgres;
```

Fonte: Próprio autor

Neste comando sql foi criado uma view para minimizar as consultas à tabela município, trazendo apelas as informações mais relevantes, melhorando assim a performance no momento de sua execução. Obviamente é uma consulta complexa dentro do banco de dados, porém a partir dessa criação, podemos passar a consultar somente à view, ao invés de fazer uma consulta complexa todas as vezes que for necessário extrair essa informação.

> “Atualmente, as visualizações são somente leitura: o sistema não permite a inserção, atualização ou exclusão de uma visualização. Você pode obter o efeito de uma visão atualizável criando gatilhos INSTEAD na visão, que devem converter as tentativas de inserção, etc. na visão em ações apropriadas em outras tabelas.” (POSTGRESQL, 2017)

Então observe que as visões não armazenam dados, mas apenas visualizam. Contudo, existem as visões materializadas, que guardam dados e atualizam constantemente na tabela base. Através da view, podemos também definir quem poderá acessar aquelas informações específicas, melhorando a segurança da nossa base de dados.

Figura 16 – Consulta à tabela município pela interface de linha de comando do Postgres

```
projeto=# explain analyze
projeto-# SELECT m.codigo,
projeto-#     m.nome AS municipio,
projeto-#     uf.codigo AS uf_codigo,
projeto-#     uf.sigla AS uf,
projeto-#     uf.nome AS uf_nome,
projeto-#     mi.codigo AS microregiao_codigo,
projeto-#     mi.nome AS microregiao_nome,
projeto-#     me.codigo AS mesoregiao_codigo,
projeto-#     me.nome AS mesoregiao_nome
projeto-#    FROM municipio m
projeto-#      JOIN micro_regiao mi ON m.micro_regiao = mi.codigo
projeto-#      JOIN meso_regiao me ON mi.meso_regiao = me.codigo
projeto-#      JOIN unidade_federativa uf ON me.uf = uf.codigo;
                                                                         QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=121.08..494.30 rows=23672 width=122) (actual time=5.418..18.224 rows=5570 loops=1)
   Hash Cond: (m.micro_regiao = mi.codigo)
   ->  Seq Scan on municipio m  (cost=0.00..94.70 rows=5570 width=20) (actual time=0.077..3.607 rows=5570 loops=1)
   ->  Hash  (cost=91.38..91.38 rows=2376 width=106) (actual time=5.043..5.043 rows=559 loops=1)
         Buckets: 4096  Batches: 1  Memory Usage: 83kB
         ->  Hash Join  (cost=53.85..91.38 rows=2376 width=106) (actual time=1.251..3.364 rows=559 loops=1)
               Hash Cond: (mi.meso_regiao = me.codigo)
               ->  Seq Scan on micro_regiao mi  (cost=0.00..9.59 rows=559 width=20) (actual time=0.059..0.620 rows=559 loops=1)
               ->  Hash  (cost=46.53..46.53 rows=586 width=90) (actual time=1.119..1.119 rows=138 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 18kB
                     ->  Hash Join  (cost=4.11..46.53 rows=586 width=90) (actual time=0.486..0.722 rows=138 loops=1)
                           Hash Cond: (uf.codigo = me.uf)
                           ->  Seq Scan on unidade_federativa uf  (cost=0.00..18.50 rows=850 width=68) (actual time=0.030..0.048 rows=28 loops=1)
                           ->  Hash  (cost=2.38..2.38 rows=138 width=26) (actual time=0.361..0.361 rows=138 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 17kB
                                 ->  Seq Scan on meso_regiao me  (cost=0.00..2.38 rows=138 width=26) (actual time=0.065..0.132 rows=138 loops=1)
 Planning time: 4.818 ms
 Execution time: 19.904 ms
(18 registros)
```

Fonte: Próprio autor

Fazendo uma consulta completa a uma tabela pequena do nosso banco de dados, tivemos um tempo de planejamento de 4,181 ms enquanto o tempo de execução foi de 19,904 ms.

A mesma consulta, porém, utilizando view, tivemos o seguinte resultado:

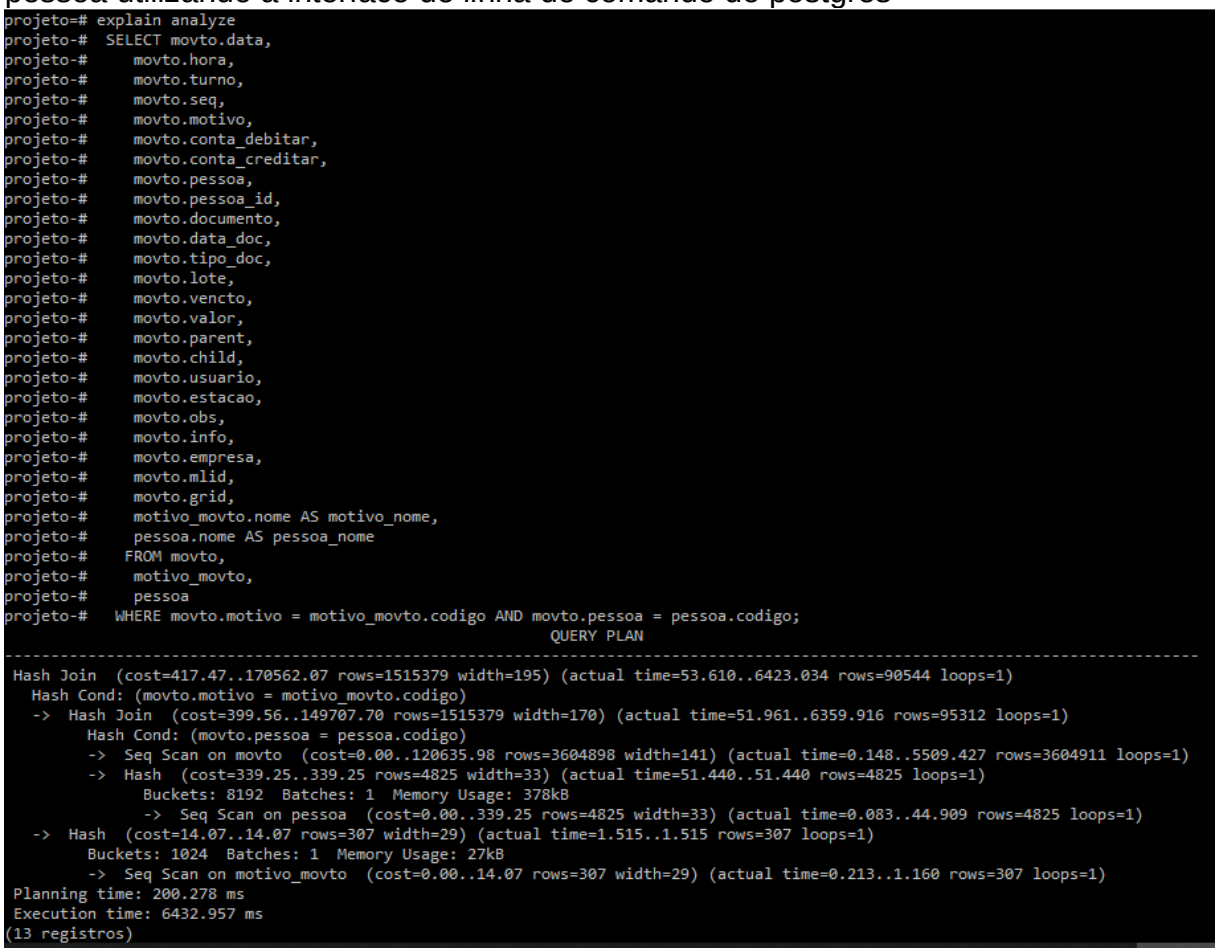
Figura 17 – Consulta à view município pela interface de linha de comando do Postgres

```
projeto=# explain analyze
projeto-# select * from municipio_view;
                                                                         QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=121.08..494.30 rows=23672 width=122) (actual time=0.683..3.139 rows=5570 loops=1)
   Hash Cond: (m.micro_regiao = mi.codigo)
   ->  Seq Scan on municipio m  (cost=0.00..94.70 rows=5570 width=20) (actual time=0.009..0.543 rows=5570 loops=1)
   ->  Hash  (cost=91.38..91.38 rows=2376 width=106) (actual time=0.643..0.643 rows=559 loops=1)
         Buckets: 4096  Batches: 1  Memory Usage: 83kB
         ->  Hash Join  (cost=53.85..91.38 rows=2376 width=106) (actual time=0.179..0.427 rows=559 loops=1)
               Hash Cond: (mi.meso_regiao = me.codigo)
               ->  Seq Scan on micro_regiao mi  (cost=0.00..9.59 rows=559 width=20) (actual time=0.005..0.059 rows=559 loops=1)
               ->  Hash  (cost=46.53..46.53 rows=586 width=90) (actual time=0.157..0.157 rows=138 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 18kB
                     ->  Hash Join  (cost=4.11..46.53 rows=586 width=90) (actual time=0.077..0.110 rows=138 loops=1)
                           Hash Cond: (uf.codigo = me.uf)
                           ->  Seq Scan on unidade_federativa uf  (cost=0.00..18.50 rows=850 width=68) (actual time=0.002..0.004 rows=28 loops=1)
                           ->  Hash  (cost=2.38..2.38 rows=138 width=26) (actual time=0.059..0.059 rows=138 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 17kB
                                 ->  Seq Scan on meso_regiao me  (cost=0.00..2.38 rows=138 width=26) (actual time=0.005..0.016 rows=138 loops=1)
 Planning time: 0.371 ms
 Execution time: 3.544 ms
(18 registros)
```

Fonte: Próprio autor

Ou seja, o tempo de planejamento diminuiu para 0,371 milissegundos e o tempo de execução diminuiu para 3,544 ms. Tendo assim, 5x mais rápido nossa consulta. Levando a mesma lógica para tabelas maiores, temos mudanças significativas dentro de performance. Vejamos abaixo:

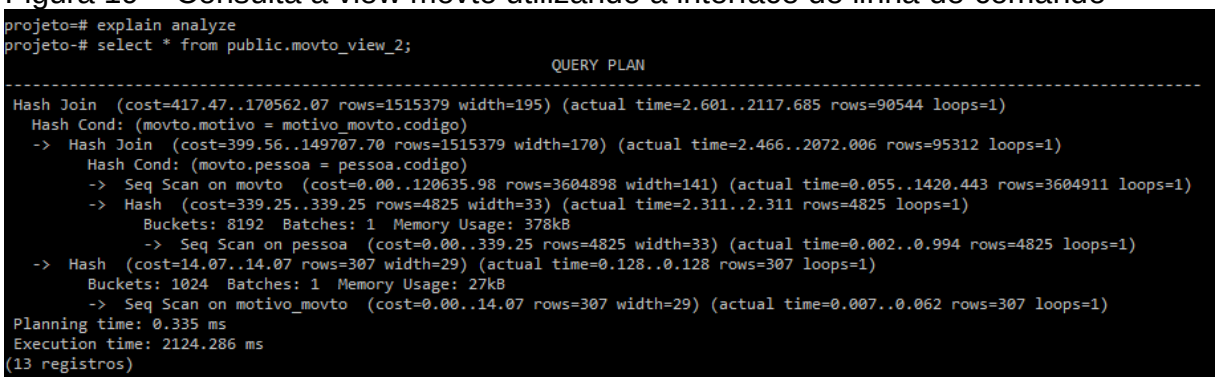
Figura 18 – Consulta à tabela movto utilizando um join à tabela motivo_movto e pessoa utilizando a interface de linha de comando do postgres



Fonte: Próprio autor

Nesta tabela, temos um pouco mais de colunas e mais lançamentos. Logo, o tempo de planejamento da consulta foi de 2 segundos, enquanto o de execução foi de 6 segundos. Quando passamos a utilizar view, tivemos o resultado a seguir:

Figura 19 – Consulta à view movto utilizando a interface de linha de comando



Fonte: Próprio autor

Tempo de planejamento:0,3 segundos; tempo de execução:2 segundos.

2.14 STORED PROCEDURES

Para trabalhar com Stored Procedures no PostgreSQL, devemos utilizar as functions ou funções. Elas são passadas para dentro do banco de dados através de algoritmos específicos, que podem ser na linguagem PL/PgSQL, C++ ou Java. Este é um recurso muito utilizado pelos desenvolvedores, pois fornece agilidade nas aplicações e na escrita de código, pois quando a regra do negócio é presente no banco de dados e não na aplicação, exige menos chamada por parte das aplicações e melhor aproveitamento de um único código para várias aplicações.

Figura 20 – Consulta à function delete_orcamento_vazio utilizando o SQL Editor do Postgres

```
-- Function: public.delete_orcamento_vazio()

-- DROP FUNCTION public.delete_orcamento_vazio();

CREATE OR REPLACE FUNCTION public.delete_orcamento_vazio()
  RETURNS boolean AS
$BODY$
DECLARE
    r record;
BEGIN

    loop
        SELECT o.grid INTO r FROM orcamento o WHERE o.status in ('A', 'P') AND
                NOT EXISTS (SELECT 1 FROM comanda_produto cp WHERE cp.orcamento = o.grid) AND
                NOT EXISTS (SELECT 1 FROM orcamento_produto op WHERE op.orcamento = o.grid);

                if found then
            RAISE NOTICE 'Deletando orcamento %...', r.grid;
            EXECUTE 'DELETE FROM orcamento WHERE grid = ' || r.grid;
        else
            exit;
        end if;
    end loop;

    RAISE NOTICE 'Finalizado!';

    return true;
END;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
ALTER FUNCTION public.delete_orcamento_vazio()
  OWNER TO postgres;
```

Fonte: Próprio autor

Acima, temos uma função que checa se existe um orçamento ativo ou não. Esta função, apesar de longa extensão de código, ela agiliza o desempenho da aplicação por já estar gravado a consulta e o resultado em sua memória, exigindo menos processamento. Entretanto, aumenta o tamanho da estrutura do banco de dados. Assim, cabe ao analista identificar as consultas que podem ser melhoradas através de funções.

2.15 BOAS PRATICAS

Para melhorar as consultas, devemos nos atentar aos operadores, as cláusulas e os comandos executados. São eles quem irão guiar o que o postgres deve fazer e como fazer. Por exemplo, quando usamos a cláusula In, o sistema percorre sequencialmente a tabela buscando aonde esteja aquele valor. Podemos eliminar isso usando outros operadores, como “=” ou “or”. Devemos observar também que os comandos update, insert e delete não somente mexem nos registros das tabelas, mas também nos índices, por isso deve ser usado de maneira eficiente.

No Postgres existe um comando interno do chamado Copy, aonde ele carrega vários dados escritos em algum arquivo do computador. Sempre que possível utilizá-lo ao invés de usar o comando insert e isso reduzirá consumo de recursos, de acordo com alguns autores.

> O PostgreSQL prove um mecanismo de carga em lote chamado COPY, que pode pegar uma entrada de um arquivo ou pipe delimitado por tabulações ou CSV. Quando o COPY pode ser usado no lugar de centenas de INSERTs, ele pode cortar o tempo de execução em até 75%. (Rodriguez, 2007, p1. apud Josh Berkus, 2006, p1.)

E por fim o uso do commit apenas no final de todas as operações, e não ao final de cada uma isoladamente. Isso pode ser ajustado automaticamente no arquivo de configurações, ora mencionado.

> O PostgreSQL pode aguardar um determinado período de tempo recebendo os COMMITs de mais de uma transação ativa, e, ao atingir determinado número de operações, gravá-las em disco de uma vez só. (Milani,2006b p.238)

3 MATERIAL E MÉTODOS

Neste tópico será apresentado a metodologia científica utilizada neste projeto de pesquisa e posteriormente, as ferramentas exploradas.

3.1 METODOLOGIA DE PESQUISA

Como metodologia, usou-se um estudo de caso. Neste tipo de metodologia, usa-se testes reais para se chegar a um fato ou opinião a respeito de um assunto. E nem sempre a realidade será a mesma para todos, mas sim uma base para que outras pessoas possam observar e compreender certos tipos de comportamentos. Os parâmetros de pesquisa são estudo de caso, pesquisa aplicada, descritiva, hipotético-dedutivo.

A pesquisa aplicada consiste em investigar e aplicar maneiras eficientes de se obter um resultado. “Enquadram-se como pesquisa aplicada os trabalhos executados com o objetivo de adquirir novos conhecimentos, com vistas ao desenvolvimento ou aprimoramento de produtos, processos e sistemas.” (BRASIL, 1993).

Por ser uma pesquisa descritiva, analisamos primeiro as características das funções e seu funcionamento, e somente então passamos para parte prática.

E por ser um método hipotético-dedutivo, primeiro criamos algumas hipóteses baseado em um problema, fizemos alguns testes e então tivemos um resultado, que é a resposta da nossa hipótese. Concluímos então nossa metodologia.

3.2 MÉTODOS

Para este projeto, foi utilizado o sistema gerenciador de banco de dados PostgreSQL na versão 9.5 para armazenar e gerenciar o banco de dados. Para gerar os relatórios de benchmark, utilizou-se o psql e a ferramenta PgAdmin, ambos do próprio Postgres.

4 RECURSOS

Descrição	Quantidade	Unidade	Valor (R$)
Computador	1	UN	6.000,00
Internet	1	UN	80,00

5 CRONOGRAMA

Tabela 1 - Cronograma

Atividade	Execução em Meses										
	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Escolha do sistema operacional											
Problemas de desempenho no SO											
Identificando problemas no postgres											
Refinando o banco de dados											
Testes de benchmark											
Escrita do trabalho											

6 RESULTADOS ESPERADOS

Portanto, as empresas da atualidade precisam investir em boas maneiras de armazenar seus dados, de forma que consigam um bom resultado sem gerar gastos desnecessários. Através disso, entram em cena os sistemas de banco de dados e com eles tem-se um gerenciamento completo de toda essa estrutura. Porém, é preciso sempre estar atento aos problemas que se pode enfrentar ao usá-los, e estar atento às melhores práticas para resolvê-los.

Com este trabalho, acredita-se que a comunidade de analistas de sistemas possa ter uma visão mais clara de como identificar problemas que afetam diretamente no desempenho das aplicações, os problemas de códigos ruins e sobre a importância de constantemente se refinar um banco de dados. É um trabalho muito importante e que as empresas precisam que seja feito, e espera-se que os resultados aqui trazidos possam conduzi-las a uma solução de problemas nas suas aplicações.

Como visto, existem diversas escolhas para sistemas de banco de dados e para essa escolha é necessário levar em consideração a finalidade do seu uso, preço e os benefícios de cada um. Independente do sistema escolhido, à medida que seu volume de informação armazenado vai crescendo e o sistema tomando proporção de tamanho, é preciso fazer ajustes manuais para não enfrentar quedas de desempenho, e para esse ajuste chamamos de Tuning. Esse trabalho pode ser feito pelo sistema gerenciador de banco de dados, no sistema operacional e também com upgrades no hardware.

No sistema operacional, observamos que ele destina recursos à medida que as aplicações exigem, e é necessário estar atento se isso não impacta no desempenho do próprio banco de dados. Os autores recomendam que isso seja observado constantemente pelo gerenciamento de recursos do sistema operacional. No teste teve-se o seguinte resultado:

PRÉ-TUNING	PÓS-TUNING	APERFEIÇOAMENTO
Memória ram: 81 (%)	Memória ram: 58 (%)	28(%)
Processador: 10 (%)	Processador: 3 (%)	70(%)

Tabela 2 – Valores de memória, cpu e aperfeiçoamento

Então eliminamos aplicação desnecessária e tivemos um bom ganho de performance, podendo destinar mais recursos para onde precisamos.

Tão logo se não for possível minimizar os problemas de perda de desempenho através do sistema operacional, devemos recorrer ao arquivo de configurações do

Postgres, que como visto, possui parâmetros para alocar memória e recursos de processamento para si. Deve-se ter conhecimento sobre o cenário vivenciado para poder alocar com caução.

Aperfeiçoando nossas tabelas, começamos eliminando registros fantasmas e liberando espaço não mais utilizado, isso através de um importante comando dentro do gerenciador do postgres. Os testes foram feitos em duas tabelas, uma pequena com unidades de linhas e outra grande, com milhares de linhas. Os resultados foram os seguintes para a tabela pequena:

PRÉ-TUNING	PÓS-TUNING	APERFEIÇOAMENTO
Resposta: 10 (ms)	Resposta: 0 (ms)	100(%)

Tabela 3 – Valores de resposta, custo e aperfeiçoamento na consulta.

Por ser uma tabela pequena, este resultado não causa tanto impacto. Porém em nossa tabela maior, temos um resultado surpreendente, como veremos a seguir:

PRÉ-TUNING	PÓS-TUNING	APERFEIÇOAMENTO
Resposta: 65 (s)	Resposta: 35.8 (s)	44(%)

Tabela 4 – Valores de resposta, custo e aperfeiçoamento na consulta.

A este resultado, podemos fazer uma alusão a um usuário que passa o dia utilizando a aplicação e a cada requisição feita, seriam 29 segundos de espera a menos por requisição, caso essa requisição percorresse toda a tabela, assim como fizemos.

Para que não se precise percorrer toda a tabela sempre que fizer uma requisição ou consulta, foi demonstrado como fazer a criação de um índice e a sua importância para a melhoria de desempenho.

PRÉ-TUNING	PÓS-TUNING	APERFEIÇOAMENTO
Resposta: 245.395 (ms)	Resposta:45.306 (ms)	81(%)
Custo: 245.370 (ms)	Custo: 45.278 (ms)	81(%)

Tabela 5 – Valores de resposta, custo e aperfeiçoamento na consulta.

Apenas criando o índice, melhoramos ainda mais as consultas à nossa tabela. Pode-se criar vários índices, mas não é interessante criar índices para cada coluna para não causar redundância da informação.

Apesar de ser uma excelente ferramenta, os índices podem atrapalhar caso sejam criados e não usados. Isso porque agrega um volume maior de informação e o banco entende que ao gravar os dados, precisa também atualizar no índice. Outra coisa importante é que podem haver índices deletados e para isso precisamos fazer uma manutenção neles. Para isso, usamos um comando no nosso gerenciador e analisamos os resultados:

PRÉ-TUNING	PÓS-TUNING	APERFEIÇOAMENTO
Resposta: 0.543 (ms)	Resposta:0.147	98(%)
Custo: 0.010 (ms)	Custo: 0.003 (ms)	98(%)

Tabela 6 – Valores de resposta, custo e aperfeiçoamento na consulta.

Espera-se que com as boas práticas aqui descritas e demonstradas, se bem aplicadas, pode-se melhorar e aperfeiçoar o desempenho dos mais diversos de bancos de dados existentes no mercado, assim como demonstrou os testes aplicados. Apesar de limitar-se a um sistema operacional neste projeto, é possível ter o mesmo resultado em outros ambientes seguindo os princípios aqui demonstrados.

7 REFERÊNCIAS BIBLIOGRÁFICAS

BARRETO, A.A. **A questão da informação**. Volume 8. Edição 4. Revista São Paulo, 1994. Disponível em: https://www.academia.edu/1750493/A_Quest%C3%A3o_da_Informa%C3%A7%C3%A3o?from=cover_page. Acesso em: 05 de junho de 2021.

SANCHES, A.R. **Fundamentos de armazenamento e manipulação de dados:** introdução e história. 2005. Disponível em: https://www.ime.usp.br/~andrers/aulas/bd2005-1/aula3. Acesso em: 05 de junho de 2021.

SANTANCHÈ, A.; ROCHA, A. **Banco de dados**: Teoria e prática. Aula 4. Modelo relacional: definições e formalização. Campinas: 2012. Disponível em: http://www.ic.unicamp.br/~rocha/teaching/2012s2/mc536/aulas/aula-04.pdf. Acesso em: 05 de junho de 2021.

MILANI, A. **MySQL**: Guia do programador. São Paulo: Novatec Editora, 2006.

MULTIEDRO. **PostgreSQL:** O que é e como ele melhora a produtividade na empresa. Disponível em: https://blog.multiedro.com.br/postgresql. Acesso em: 05 de junho de 2021.

SOUZA, A.P.S; CAMPOS, B.F; DIAS, C.G.G; ALVES, M.B; VIEIRA, C.E; CARELLI, F.C; SÁ, L.F.C. **Tuning em banco de dados.** Ed.10. Volta Redonda: 2009. Disponível em: http://revistas.unifoa.edu.br/index.php/cadernos/article/view/965. Acesso em: 05 de junho de 2021.

ZORZI, M.T. **Tuning de sistema operacional em banco de dados Oracle**. Caxias do Sul: 2015. cap. 2-3, p.21-26. Disponível em: https://repositorio.ucs.br/handle/11338/1210. Acesso em: 05 de junho de 2021.

CARNEIRO, A.P; MOREIRA, J.L; FREITAS, A.L.C. **Tuning - Técnicas de otimização de banco de dados**: Um estudo comparativo: Mysql e Postgresql. Rio grande do Sul: 2009. Cap. 2, p.2. Disponível em: http://repositorio.furg.br/handle/1/1692. Acesso em: 05 de junho de 2021.

THUMS, J.E. **Tuning em banco de dados postgresql**: um estudo de caso. Caxias do Sul: 2018. Cap 5, p.50. Disponível em: https://repositorio.ucs.br/xmlui/handle/11338/4020. Acesso em 05 de junho de 2021.

ANCHIETA, M. M. **Sintonia em banco de dados através do particionamento de tabelas**: Alegrete, 2012. Cap. 2, p.20

POSTGRESQL. **Índices**. Disponível em: https://www.postgresql.org/docs/7.1/indices.html. Acesso em: 05 de junho de 2021.

POSTGRESQL. **Analyze**. Disponível em: http://pgdocptbr.sourceforge.net/pg82/sql-analyze.html. Acesso: 05 de junho de 2021.

RODRIGUEZ, F.T. **Dicas de performance em aplicações com PostgreSQL**. Savepoint, 2007. Disponível em: https://www.savepoint.blog.br/2007/01/05/dicas-de-performance-em-aplicacoes-com-postgresql/. Acesso em:06 de junho de 2021.

MAGALHÃES, L. B; MARACCI, F. V; ZAUPA, A.P; SILVA, F.A. **Análise comparativa dos algoritmos de otimização de consultas do postgresql**. São Paulo: 2015a-b. Cap. 5, pag. 10-12. Disponível em: http://revistas.unoeste.br/index.php/ce/article/view/1028. Acesso: 05 de junho de 2021.

TECMUNDO. **Como saber o que está drenando recursos do meu PC**. Disponível em: https://www.tecmundo.com.br/software/118936-saber-drenando-recursos-meu-pc.htm. Acesso: 05/06/2021.

BRASIL, Decreto 949 (1993). Capítulo 1 – DAS DISPOSIÇÕES PRELIMINARES, Art, 4. Disponível em: http://www.planalto.gov.br/ccivil_03/decreto/antigos/d949.htm. Acesso em: 06 de junho de 2021.

POSTGRESQL: **Create view**. Disponível em: https://www.postgresql.org/docs/9.2/sql-createview.html. Acesso em: 20/11/2021.