RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

TUNING E PERFORMANCE NO SQL SERVER: COMO MELHORAR O SEU AMBIENTE E SEUS PROCESSOS

Luiz Fernando Silva Serafim
Graduando em Sistemas de Informação – Uni-FACEF
luizfernando.sserafim@gmail.com

Davi Chagas Garcia
Graduando em Sistemas de Informação – Uni-FACEF
davichagasgarcia@gmail.com

Prof. Geraldo Henrique Neto
Mestre em Ciências com ênfase em Informática Médica – Uni-FACEF
geraldo@facef.br

Resumo

Esse projeto surgiu após um momento de estudos e análises entre os autores, relacionada à performance do banco de dados no ambiente de trabalho de ambos, e o impacto que *queries* mal formuladas causam durante o dia-a-dia. A performance do SQL Server é um dos fatores mais importantes e críticos para a eficiência de sistemas de gerenciamento de banco de dados, especialmente em ambientes corporativos que lidam com grandes volumes de dados. Este artigo tem como objetivo explorar práticas de *tuning* e otimização que podem ser aplicadas para melhorar a performance de consultas e processos no SQL Server. Serão discutidas abordagens como indexação eficiente, reescrita de *scripts* e o uso de ferramentas de monitoramento de desempenho do próprio SGBD. Além disso, será abordada a importância de rotinas de manutenção regulares, como por exemplo a atualização de estatísticas e verificação de integridade do banco (*CheckDB*). Por meio dessas técnicas, espera-se reduzir o tempo de execução das consultas, minimizar o uso de recursos do ambiente e garantir uma operação mais fluida do sistema como um todo, especialmente em cenários de alta demanda e escalabilidade crescente, otimizando tanto tempo de processamento, quanto eficiência operacional.

Palavras-chave: SQL Server. Tuning. Performance. Otimização de Consultas. Banco de Dados.

Abstract

This project emerged after a period of study and analysis among the authors, related to database performance in both of their work environments and the impact that poorly formulated queries have on day-to-day operations. SQL Server performance is one of the most important and critical factors for the efficiency of database management systems, especially in corporate environments that handle large volumes of data insertion and flow. This article aims to explore tuning and optimization practices that can be applied to improve the performance of queries and processes in SQL Server. Approaches such as efficient indexing, script rewriting, and the use of performance

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

monitoring tools from the DBMS itself will be discussed. In addition, the importance of regular maintenance routines, such as updating statistics and performing database integrity checks (CheckDB), will be addressed. Through these techniques, we aim to reduce query execution time, minimize resource usage in the environment, and ensure smoother system operation as a whole, especially in high-demand scenarios and growing scalability, optimizing both time and operational efficiency.

Keywords: *SQL Server. Tuning, Performance. Query Optimization. Database.*

1. INTRODUÇÃO

Este artigo tem como objetivo explorar práticas de *tuning* e otimização que podem ser aplicadas para melhorar a performance de consultas e processos no SQL Server, uma das plataformas mais utilizadas no mercado. A pesquisa explora não apenas a arquitetura e funcionalidades desse SGBD (Sistemas Gerenciadores de Banco de Dados), mas também se aprofunda nas práticas de otimização (*tuning*), um componente essencial para garantir o desempenho eficiente dos sistemas. O *tuning* de SQL, que abrange desde a estruturação de consultas eficientes até a configuração adequada de *hardware* e *software*, será discutido em detalhes, destacando seu papel fundamental na redução de gargalos e na maximização da eficiência do banco de dados.

Além disso, serão abordadas as melhores práticas para a manutenção contínua do ambiente de banco de dados, com foco em técnicas como ajuste de alocação de memória, otimização dos caminhos de acesso aos dados, gerenciamento de *I/O (Input/Output)* e balanceamento de carga. Esses aspectos são cruciais para garantir que o sistema seja capaz de lidar com grandes volumes de dados e demandas crescentes sem comprometer o tempo de resposta e a integridade das informações.

Com a implementação eficaz dessas práticas, é possível obter ganhos substanciais em termos de desempenho, incluindo uma redução significativa nos tempos de processamento, maior confiabilidade das aplicações e manutenção da integridade dos dados, fatores que impactam diretamente a operação e a tomada de decisões estratégicas das organizações. Ao final, este estudo visa auxiliar profissionais de TI e administradores de banco de dados na identificação de pontos críticos de melhoria, fornecendo orientações práticas que contribuam para um gerenciamento mais eficaz de um ambiente de dados mais robusto e confiável (Delaney, 2020).

2. REFERENCIAL TEÓRICO

No referencial teórico, exploraremos os principais conceitos e estruturas relacionados aos Sistemas de Gerenciamento de Banco de Dados (SGBDs), que desempenham um papel central na administração e manipulação de grandes volumes de dados. Abordaremos as diferentes modelagens de dados, destacando os modelos relacionais e não relacionais. Discutiremos também as funcionalidades fundamentais dos SGBDs, incluindo controle de concorrência, segurança e mecanismos de recuperação. Por fim, apresentaremos a importância da performance dos bancos de dados e como ajustes em áreas como tuning podem impactar diretamente o desempenho das aplicações e, consequentemente, a eficiência dos processos

organizacionais. O foco será dado ao SQL Server, uma ferramenta amplamente utilizada que oferece diversas edições adaptadas às necessidades dos usuários.

2.1 O QUE É UM SGBD?

Um Sistema de Gerenciamento de Banco de Dados (SGBD) é uma ferramenta essencial na administração e manipulação de grandes volumes de dados em diversos sistemas computacionais. Segundo Silberchatz, Korth e Sudarshan (2011), o SGBD oferece uma interface que facilita a criação, manutenção e consulta de bancos de dados, possibilitando o gerenciamento eficaz das informações. Ele é projetado para garantir que os dados sejam acessados de maneira eficiente e segura, ao mesmo tempo em que controla o acesso simultâneo de diversos usuários, mantendo a integridade e a consistência dos dados.

De acordo com Elmasri e Navathe (2016), o SGBD também é responsável por gerenciar o armazenamento físico dos dados, garantindo que as informações sejam organizadas de forma a otimizar o desempenho das operações de leitura e escrita. Além disso, o SGBD implementa políticas de segurança para proteger os dados de acessos não autorizados e falhas.

Esses sistemas são fundamentais para empresas e organizações que precisam gerenciar grandes quantidades de dados de maneira estruturada, segura e eficiente. Exemplos de SGBDs incluem plataformas populares como Oracle, MySQL, PostgreSQL, e Microsoft SQL Server.

2.2 MODELO DE DADOS

Os modelos de dados são fundamentais para definir como os dados são organizados, armazenados e manipulados em um banco de dados. Segundo Silberchatz, Korth e Sudarsshan (2011), os modelos de dados fornecem uma abstração da estrutura dos dados, permitindo que os desenvolvedores criem sistemas que possam gerenciar eficientemente grandes volumes de informações. O modelo relacional é o mais amplamente utilizado, organizando os dados em tabelas compostas por linhas e colunas, com chaves que estabelecem as relações entre diferentes conjuntos de dados.

Elmasri e Navathe (2016) complementam ao afirmar que, além do modelo relacional, outros modelos de dados, como o modelo hierárquico e o modelo de rede, oferecem abordagens alternativas para representar as relações entre os dados. Enquanto o modelo relacional é baseado em uma estrutura tabular, o modelo hierárquico organiza os dados em uma estrutura de árvore, com registros conectados de forma hierárquica. Já o modelo orientado a objetos introduz conceitos da programação orientada a objetos, permitindo o armazenamento de dados mais complexos em estruturas que refletem objetos e suas interações.

Esses diferentes modelos de dados são utilizados dependendo das necessidades específicas de cada aplicação, garantindo flexibilidade e eficiência no gerenciamento de informações.

2.3 SUBCONJUNTOS DA SQL (*Structured Query Language*)

Os subconjuntos da SQL são componentes essenciais para lidar com diferentes aspectos da manipulação e controle de dados em um banco de dados. Segundo Silberchatz, Korth e Sudarshan (2011), a SQL pode ser dividido em subconjuntos distintos, cada um com um propósito específico dentro da administração e manipulação de dados. Esses subconjuntos são fundamentais para permitir que os desenvolvedores interajam eficientemente com os dados e garantam a integridade do sistema.

O principal subconjunto é o DDL (*Data Definition Language*), que é responsável pela definição da estrutura do banco de dados, incluindo a criação, alteração e exclusão de tabelas e índices. De acordo com ELMASRI e NAVATHE (2016), a DDL permite que os administradores definam os esquemas de banco de dados, estabelecendo a forma como os dados serão armazenados e organizados.

Outro subconjunto importante é a DML (*Data Manipulation Language*), que é usado para consultar, inserir, atualizar e excluir dados em um banco de dados. A DML inclui comandos como SELECT, INSERT, UPDATE e DELETE, e é frequentemente utilizado para interações cotidianas com os dados.

Além disso, a DCL (*Data Control Language*) e a TCL (*Transaction Control Language*) complementam esses subconjuntos, garantindo o controle de permissões de acesso aos dados e o gerenciamento de transações, respectivamente. ELMASRI e NAVATHE (2016) destacam que a DCL inclui comandos como GRANT e REVOKE, usados para definir direitos de acesso aos usuários, enquanto a TCL inclui comandos como *COMMIT* e *ROLLBACK*, essenciais para manter a consistência em transações de banco de dados.

Esses subconjuntos do SQL fornecem as ferramentas necessárias para que desenvolvedores e administradores possam controlar, definir e manipular os dados de maneira eficiente e segura em um ambiente de banco de dados.

2.4 TÉCNICAS DE OTIMIZAÇÃO NO SQL SERVER

As técnicas de otimização no SQL Server são essenciais para melhorar o desempenho das consultas e garantir a eficiência do sistema como um todo. Segundo Silberschatz, Korth e Sudarshan (2011), a otimização de consultas é um processo que busca encontrar o plano de execução mais eficiente para uma consulta SQL, considerando fatores como tempo de resposta e utilização de recursos.

Uma das principais técnicas de otimização é o uso de índices. Os índices permitem que o SQL Server acesse os dados de maneira mais rápida, reduzindo o tempo necessário para localizar e recuperar informações. Elmasri e Navathe (2016) destacam que a escolha do tipo adequado de índice, como índices únicos, compostos ou em árvore B, pode ter um impacto significativo no desempenho das operações de leitura. Prakash e Pachore (2016) também enfatizam a importância de monitorar a utilização dos índices, uma vez que índices desnecessários podem prejudicar a performance de gravações. Além disso, Mussen (2015) menciona que a desfragmentação regular de índices é crucial para manter um desempenho ideal.

Outra técnica importante é a reestruturação de consultas. Consultas mal formuladas podem levar a planos de execução ineficientes. É aconselhável evitar

subconsultas desnecessárias e utilizar junções adequadas. De acordo com Krause e Zimmermann (2013), a simplificação de consultas complexas e a utilização de INNER JOIN ao invés de OUTER JOIN, quando possível, pode melhorar significativamente o desempenho. Harrison (2015) reforça essa ideia, sugerindo que a análise e a reformulação de consultas podem ajudar a minimizar o tempo de execução. Gursu (2019) também destaca que a utilização de CTEs (*Common Table Expressions*) pode facilitar a leitura das consultas e melhorar a manutenção do código.

A análise e a atualização regular das estatísticas do banco de dados são cruciais para a otimização. O SQL Server utiliza essas estatísticas para estimar a cardinalidade e o custo das operações, influenciando diretamente a escolha do plano de execução. Van Zandt e Stokes (2014) enfatizam que manter as estatísticas atualizadas ajuda a garantir que o otimizador do SQL Server tenha informações precisas para tomar decisões informadas sobre como executar as consultas. Malik (2017) também destaca que a coleta de estatísticas deve ser parte da rotina de manutenção do banco de dados. Além disso, Shaw (2018) aponta que o uso de *Automatic Statistics Update* deve ser monitorado e, em alguns casos, ajustado para melhor desempenho.

A configuração adequada do servidor e a alocação de recursos também desempenham um papel vital na otimização do SQL Server. A alocação correta de memória, a configuração de opções de paralelismo e a escolha do *hardware* apropriado são aspectos que não devem ser negligenciados. Goldberg (2015) sugere que uma análise cuidadosa da configuração do servidor pode resultar em melhorias significativas no desempenho geral do sistema. Krause (2017) também discute a importância de monitorar o desempenho do servidor e ajustar as configurações conforme necessário. Klinsmann (2020) menciona que o ajuste das configurações de *Max Degree of Parallelism* (*MAXDOP*) pode impactar significativamente o desempenho de consultas em ambientes com múltiplos processadores.

Finalmente, a implementação de particionamento de tabelas é outra técnica avançada que pode ser utilizada para otimização, especialmente em sistemas que lidam com grandes volumes de dados. Das e Clower (2015) indicam que o particionamento pode melhorar a eficiência das consultas, permitindo que o SQL Server acesse apenas as partes relevantes da tabela. Wang (2021) também observa que o particionamento pode facilitar a manutenção, como a exclusão de dados antigos.

Essas técnicas de otimização, quando aplicadas de maneira adequada, podem resultar em ganhos substanciais de desempenho no SQL Server, assegurando um ambiente de banco de dados mais responsivo e eficiente.

3.METODOLOGIA

Neste tópico serão apresentadas as metodologias que utilizamos em ambientes de teste para compor nosso artigo.

3.1 ESCOLHA DO SGBD

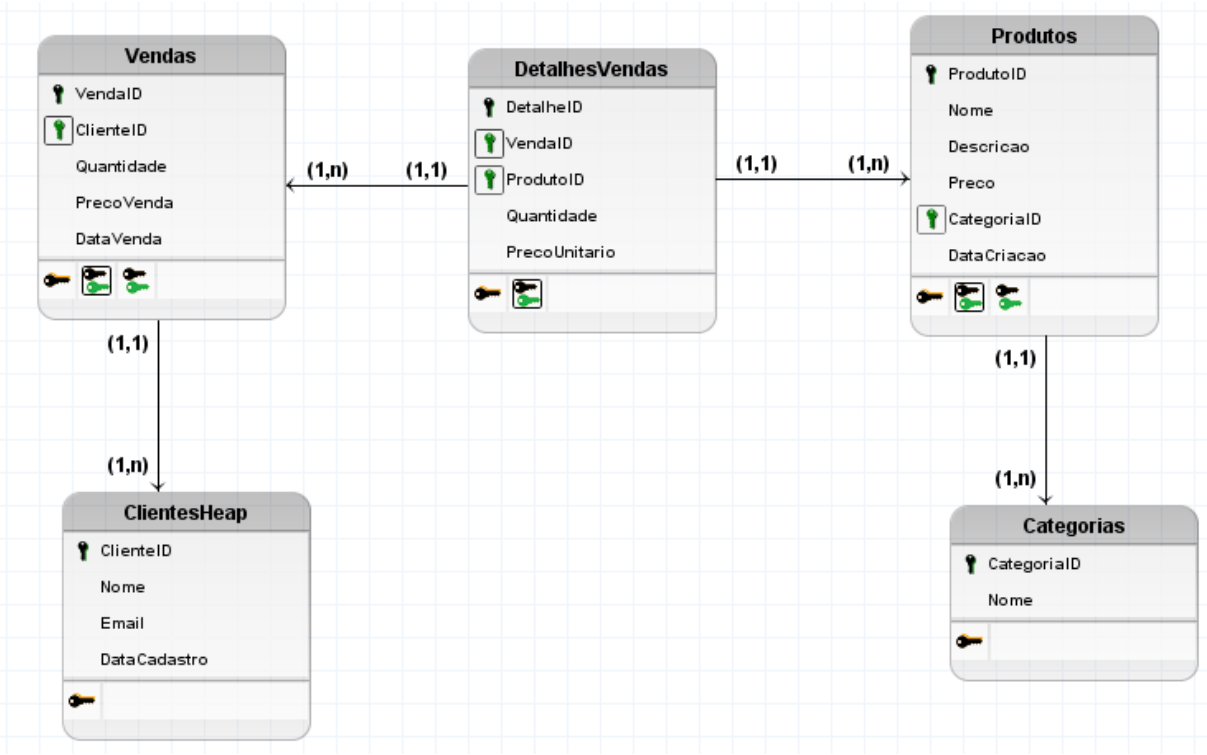
RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

O SQL Server oferece um suporte robusto para transações e recuperação de desastres, além de ferramentas avançadas de análise de dados e integração com serviços de *Business Intelligence*. Sua performance é destacada em ambientes OLTP, beneficiando-se de um eficiente mecanismo de controle de concorrência. Ferramentas integradas como *SQL Server Profiler, Query Store, Database Engine Tuning Advisor* e *SQL Server Management Studio* (SSMS) facilitam significativamente o *tuning* e a administração.

3.2 CARGA DO BANCO DE DADOS

De acordo com a *Figura 1* a seguir é exibido um diagrama que representa o modelo lógico do banco de dados utilizado no ambiente de teste. O objetivo é mostrar como o ambiente está configurado com tabelas *clusterizadas* e *indexadas*, destacando a organização dos dados e suas colunas, sendo essencial este processo para criação dos *scripts* de inserção.

Figura 1 - Modelo Lógico Ambiente Clustered



Fonte: Os autores.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

Por meio do *Script* criado para inserir dados no ambiente *clusterizado*, foi realizado um processo de inserção em loop, onde dados fictícios são repetidamente inseridos nas tabelas. Ilustrando como o ambiente foi alimentado para fins de teste de performance, utilizando um exemplo de inserção na tabela "*DetalhesVendas*" passando os valores dos campos *VendaID, ProdutoID, Quantidade, PrecoUnitario* conforme podemos verificar na *Figura 2.*

Figura 2 - Inserção Ambiente Clustered

```
-- Inserir dados em DetalhesVendas
DECLARE @i INT = 0
SET @i = 0;
WHILE @i < 1000000  -- 1.000.000 de detalhes de vendas
BEGIN
    INSERT INTO DetalhesVendas (VendaID, ProdutoID, Quantidade, PrecoUnitario)
    VALUES (
        ABS(CHECKSUM(NEWID())) % 1000000 + 1,
        ABS(CHECKSUM(NEWID())) % 100000 + 1,
        ABS(CHECKSUM(NEWID())) % 10 + 1,
        RAND(CHECKSUM(NEWID())) * 1000
    );
    SET @i = @i + 1;
END;
GO
```

Fonte: Os autores.

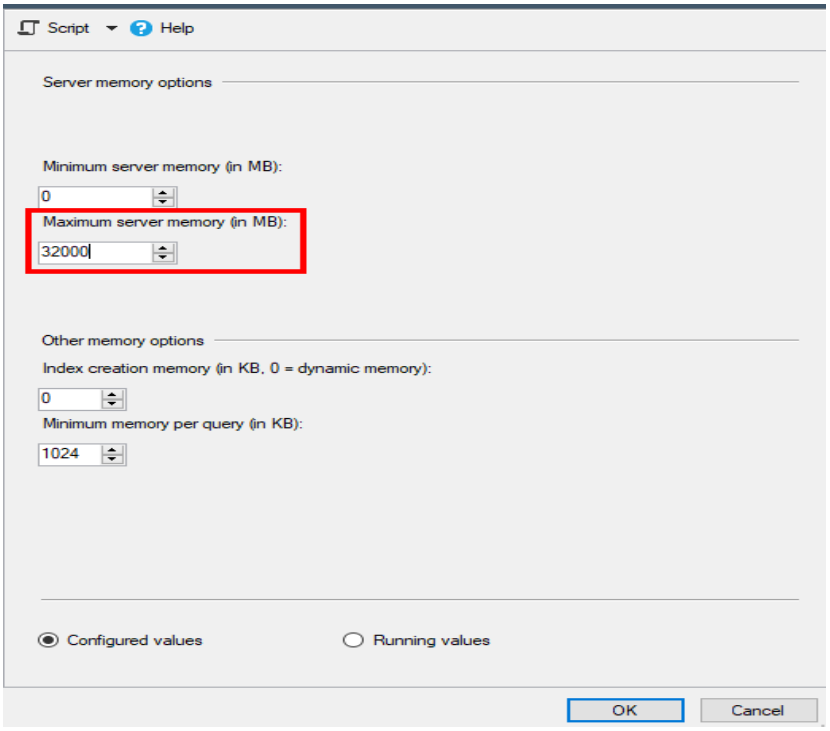
3.3 PROCEDIMENTOS DE TUNING E ANÁLISE DE PERFORMANCE

Após a instalação do SQL Server, é fundamental realizar ajustes em algumas configurações para otimizar o desempenho e garantir a estabilidade do ambiente. Uma das primeiras verificações importantes envolve a configuração da memória máxima. O SQL Server pode, por padrão, alocar toda a memória disponível do servidor, o que pode comprometer o desempenho de outros processos. A recomendação é reservar entre 2 a 4 GB de RAM para o sistema operacional e deixar o restante disponível para o SQL Server, garantindo que o sistema continue estável mesmo sob carga alta. Essa configuração evita que o servidor sofra com falta de memória para outras operações e melhora a eficiência geral (Microsoft a, 2023).

Realizamos a captura da tela, *Figura 3*, que mostra a configuração da memória máxima que foi alocada para o SQL Server na instância utilizada, como podemos observar o valor era igual a 32 GB do ambiente.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

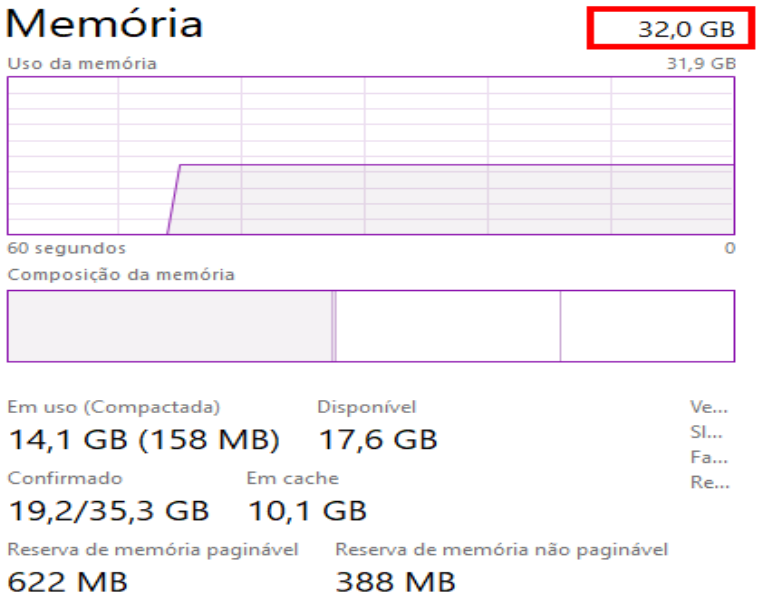
Figura 3 - Máximo de memória configurada na instância



Fonte: Os autores.

Na *Figura 4* é possível observar a configuração de memória do hardware no servidor onde o SQL Server está rodando, nela contém o total de memória física fornecida, com isso conseguimos avaliar se os limites de uso que foram configurados para o SQL estão ajustados de forma que não utilize 100% disponível no ambiente, nesta imagem devemos focar apenas na quantidade de memória demarcada, as demais informações podem ser desconsideradas no momento.

Figura 4 - Configuração de memória do Hardware



Fonte: Os autores.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

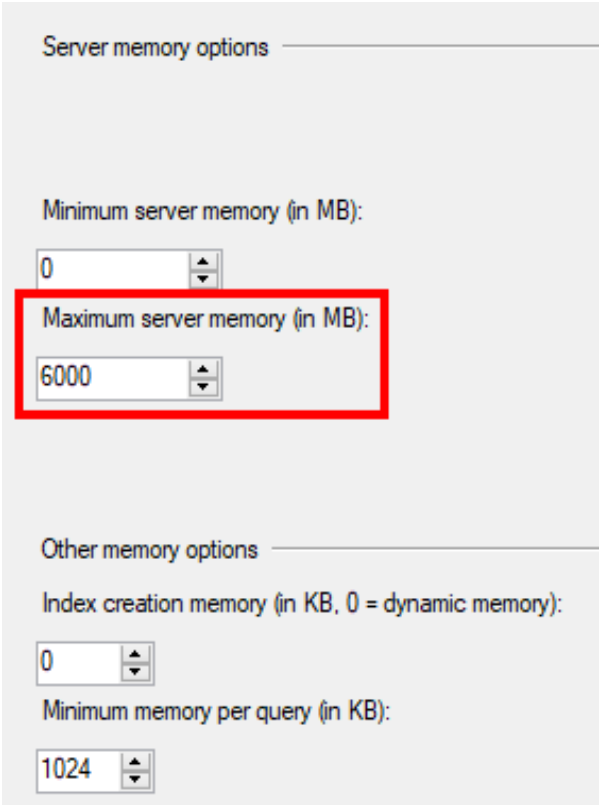
Nota-se que a memória atual configurada na instância está no limite comparada a memória do *hardware,* o que deve ser ajustado. Caso seja mantido ira gerar um consumo de 100%, com isso ocorrerá travamento e lentidão, para este ambiente será ajustado o *max memory* para 6GB de acordo com a *Figura 5*. Após a execução do *Script* de configuração a memória alocada passou a ser 6 GB como podemos observar na *Figura 6*, apresenta o *max memory* atual da instância.

Figura 5 - Configurando a memória para 6GB

```
EXEC sys.sp_configure N'show advanced options', N'1'  RECONFIGURE WITH OVERRIDE
GO
EXEC sys.sp_configure N'max server memory (MB)', N'6000'
GO
RECONFIGURE WITH OVERRIDE
GO
EXEC sys.sp_configure N'show advanced options', N'0'  RECONFIGURE WITH OVERRIDE
GO
```

Fonte: Os autores.

Figura 6 - Tela de configuração da memória



Fonte: Os autores.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

Com isso o *MAXDOP* é considerado um aspecto crítico, que controla quantos processadores podem ser utilizados em consultas paralelas. Na *Figura 7* é apresentado a quantidade de CPU que contemos no ambiente e qual a sua configuração, embora o paralelismo possa aumentar a performance de consultas complexas, ele também pode gerar contenção de CPU. Por isso, a Microsoft sugere que, para servidores com até 8 CPUs, o valor do MAXDOP seja definido como 4. Servidores com mais CPUs podem requerer ajustes adicionais com base no perfil da carga de trabalho (Microsoft a, 2023) como podemos notar no script foi ajustado para 3 CPUs conforme a *Figura 8.* A *Figura 9* se trata de uma captura de tela relacionada à configuração de *MAXDOP*, oferece uma visualização adicional sobre como o ajuste foi implementado no SQL Server conforme.

Figura 7 – CPU do ambiente

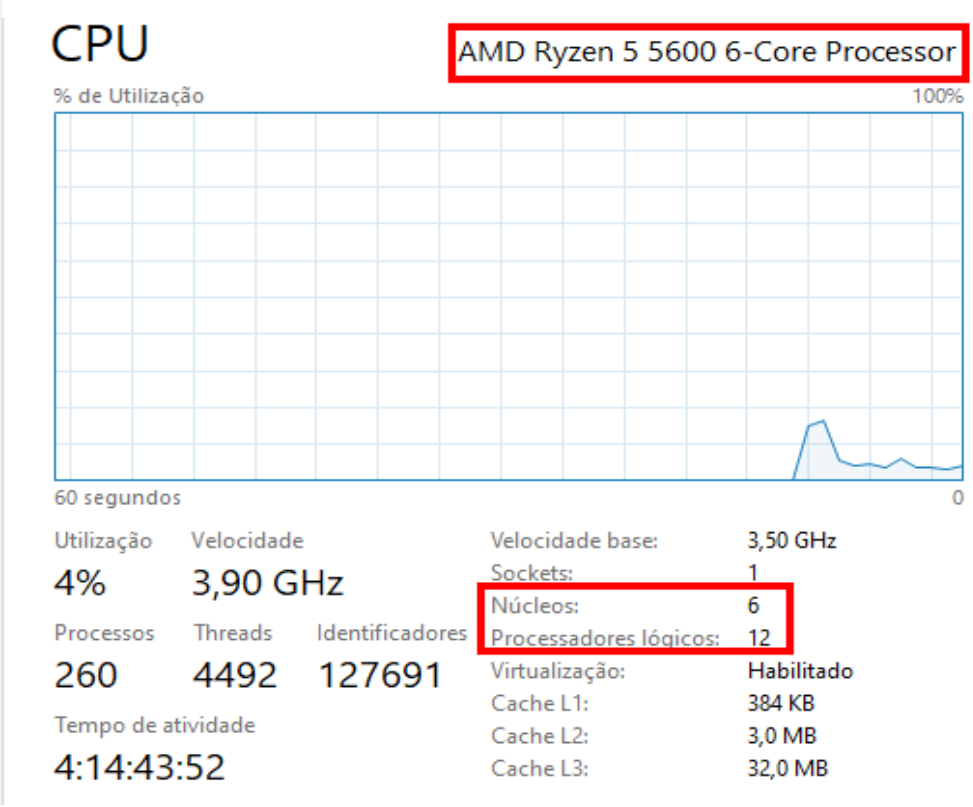


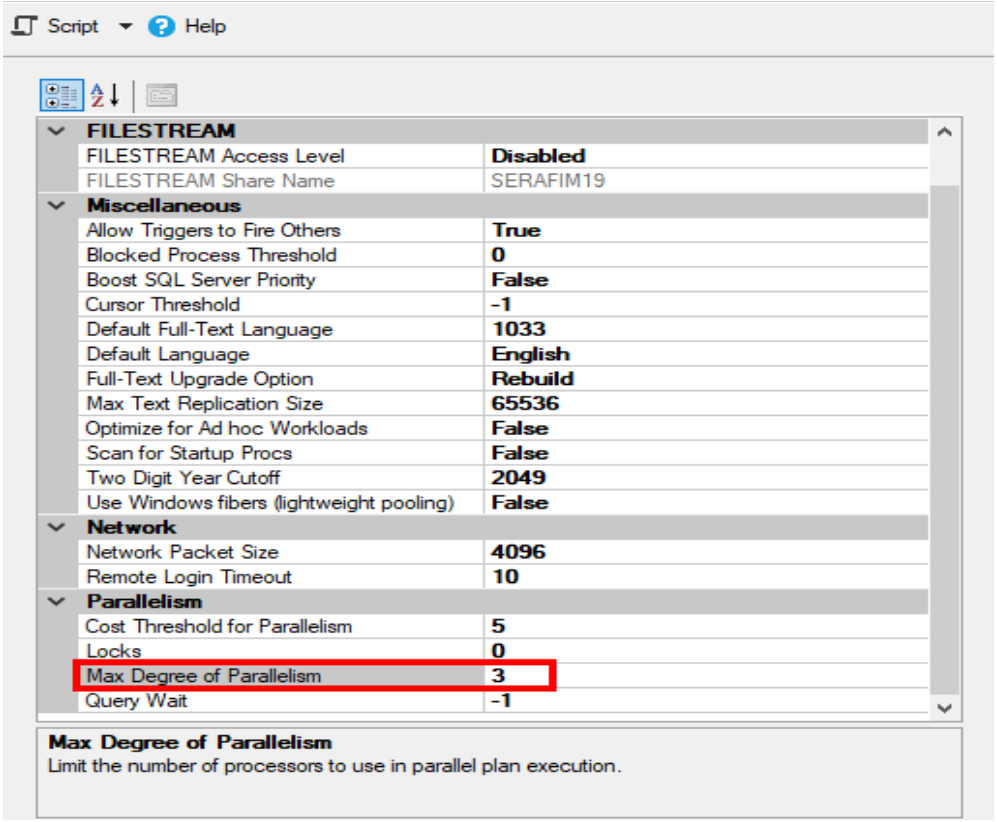
Figura 8 - Configurando MaxDop

```
EXEC sys.sp_configure N'show advanced options', N'1'  RECONFIGURE WITH OVERRIDE
GO
EXEC sys.sp_configure N'max degree of parallelism', N'3'
GO
RECONFIGURE WITH OVERRIDE
GO
EXEC sys.sp_configure N'show advanced options', N'0'  RECONFIGURE WITH OVERRIDE
GO
```

Fonte: Os autores.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

Figura 9 - Configuração de MaxDop



Fonte: Os autores.

Sendo assim a configuração do *Cost Threshold for Parallelism* é igualmente importante. Esse parâmetro define o custo mínimo de uma consulta para que ela seja executada em paralelo. O valor padrão é 5, o que pode levar a consultas relativamente simples serem executadas em paralelo, potencialmente desperdiçando recursos. Ajustar esse valor para cima em ambientes com cargas de trabalho mais complexas ajuda a evitar esse desperdício, reservando o paralelismo para consultas mais intensas e que realmente demandam mais processamento (Microsoft b, 2023) conforme podemos observar no script da *Figura 10*. Na *Figura 11* contém informações similares à imagem anterior, com detalhes sobre o valor final definido para o *Cost Threshold for Parallelism*, fornecendo uma visão clara do processo de ajuste.

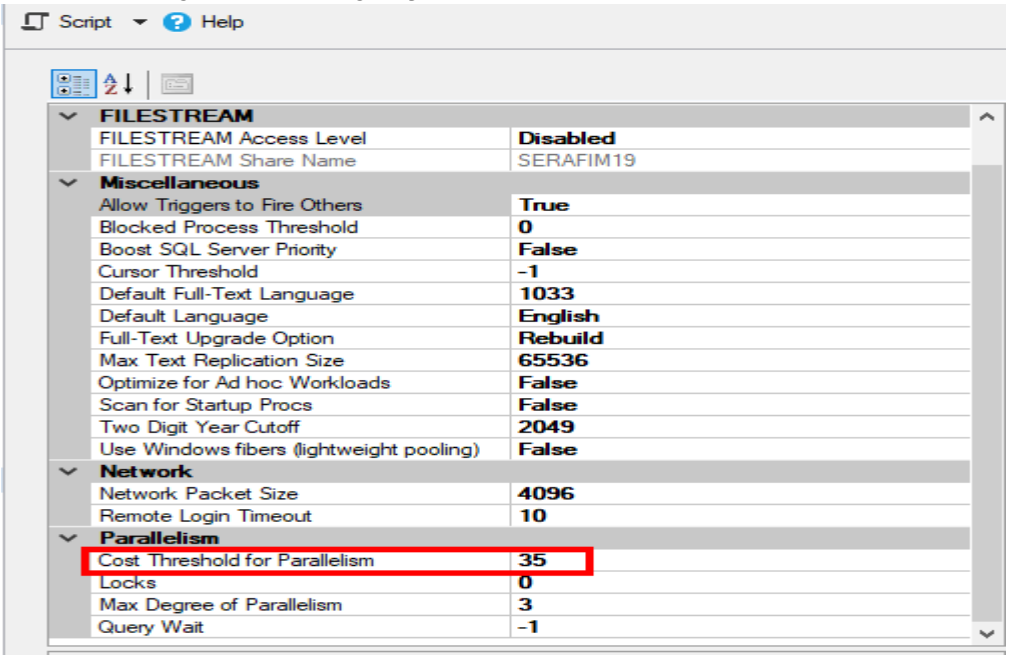
Figura 10 - Configurando Cost Threshold for Parallelism

```
EXEC sys.sp_configure N'show advanced options', N'1'  RECONFIGURE WITH OVERRIDE
GO
EXEC sys.sp_configure N'cost threshold for parallelism', N'35'
GO
RECONFIGURE WITH OVERRIDE
GO
EXEC sys.sp_configure N'show advanced options', N'0'  RECONFIGURE WITH OVERRIDE
GO
```

Fonte: Os autores.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

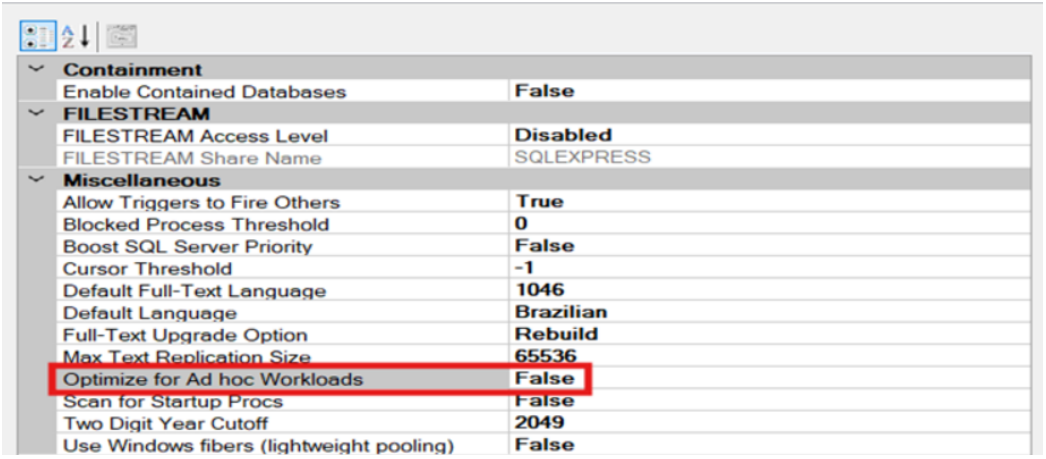
Figura 11 - Configuração de Cost Threshold for Parallelism



Fonte: Os autores.

Um ponto a ser considerado são as consultas *Ad Hoc*, que são executadas esporadicamente. O SQL Server armazena o plano de execução dessas consultas na memória, o que pode resultar em desperdício de recursos (Microsoft c, 2024) como podemos notar por padrão após a instalação da instância tal recurso vem desabilitado de acordo com a *Figura 12*. Para mitigar tal impacto, é recomendável habilitar a configuração *Optimize for Ad Hoc Workloads*, que armazena uma versão compacta do plano de execução até que a consulta seja reutilizada, economizando memória e otimizando o uso de recursos (Microsoft c, 2024). Conforme feito na *Figura 13*.

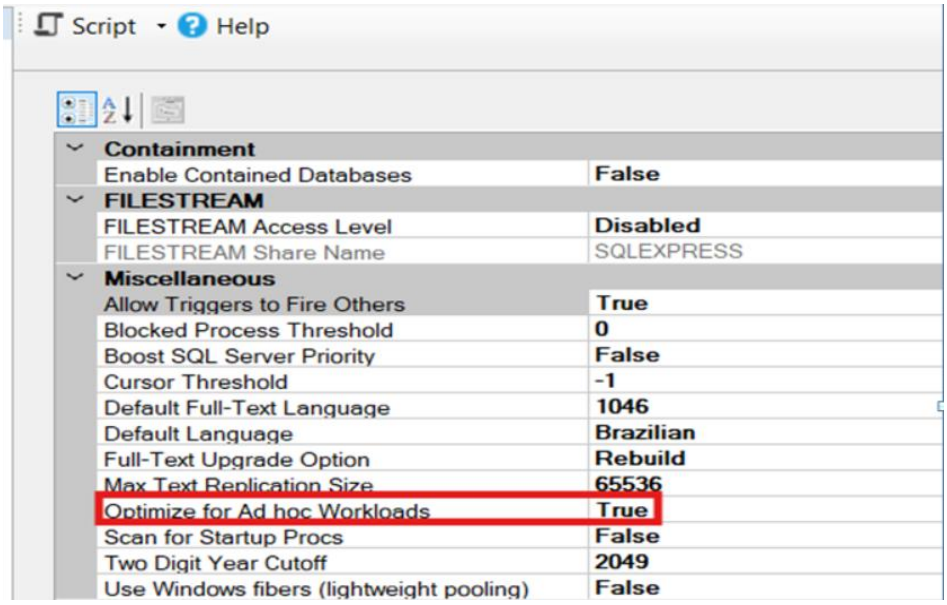
Figura 12 - Antes de configurar o Ad Hoc



Fonte: Os autores.

Figura 13 - Pós configurar Ad Hoc

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024



Fonte: Os autores.

Finalmente, é essencial otimizar o *TempDB*, que armazena objetos temporários e realiza operações internas como ordenações e agrupamentos. Uma configuração inadequada desse banco pode gerar gargalos de desempenho. Recomenda-se aumentar o número de arquivos de dados do *TempDB* para igualar o número de CPUs lógicas disponíveis, com um limite de até 8 arquivos. Além disso, o uso de discos rápidos, como SSDs (Solid State Drives), é recomendado para melhorar a resposta em operações de I/O intensivo (Microsoft a, 2023) abaixo segue o script de configuração dos arquivos na *Figura 14*. Para obter dados de performance dentro do SGBD SQL Server foi usado para capturar e analisar eventos o SQL Profiler, o que permitiu a detecção de problemas de desempenho como consumo de CPU, o número de *Reads* realizados e a duração da execução (*Duration*) conforme podemos observar a *Figura 15*.

Figura 14 - Ajustando arquivos de TempDB

```
USE [master]
GO
ALTER DATABASE [tempdb] MODIFY FILE ( NAME = N'tempdev', FILEGROWTH = 1048576KB )
GO
ALTER DATABASE [tempdb] MODIFY FILE ( NAME = N'templog', FILEGROWTH = 524288KB )
GO
```

Fonte: Os autores.

Figura 15 – Análise de Desempenho

TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Duration
select * from Clientes	Microsoft SQ...	luiz_	SERAFI...				
select * from Clientes	Microsoft SQ...	luiz_	SERAFI...	78	2434	0	1000
SET DEADLOCK_PRIORITY -10	SQLServerCEIP	SQLTELE...	NT SER...				

Fonte: Os Autores.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

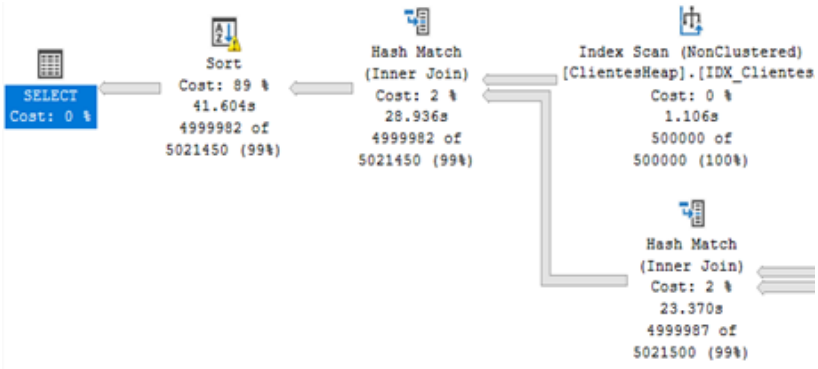
4. MÉTRICAS DE AVALIAÇÃO

A avaliação do desempenho de sistemas de gerenciamento de banco de dados (SGBD) é fundamental e envolve diversas métricas, como latência, taxa de transferência e tempo de resposta. A taxa de transferência quantifica a quantidade de dados processados em um intervalo de tempo específico, enquanto o tempo de resposta indica a rapidez com que o sistema responde a uma consulta. A latência, por sua vez, faz refere-se ao atraso na comunicação entre os componentes do sistema, impactando a experiência do usuário e a eficiência operacional (Nunes, 2008).

A escalabilidade é outra métrica crucial que avalia o desempenho do sistema sob carga e sua capacidade de expansão, tanto horizontal quanto vertical. A escalabilidade vertical implica no aumento dos recursos de uma única máquina, como CPU e memória, enquanto a escalabilidade horizontal se refere à adição de mais máquinas ao sistema. A capacidade de um *SGBD* de se expandir de maneira eficiente é vital para atender às crescentes demandas das aplicações (Reis et al., 2012).

Uma ferramenta importante para o *tuning* de consultas é o plano de execução gerado pelo *Management Studio*, que fornece uma visão detalhada dos custos associados a cada operação na *query*. Essa análise permite a identificação de pontos onde índices podem ser criados ou onde reescritas de consultas podem ser aplicadas para melhorar o desempenho e obter resultados de forma mais eficiente como podemos observar na *Figura 16* (Kumar, 2023).

Figura 16: Exemplo de plano de execução



Fonte: Os autores.

O cenário ideal para um plano de execução envolve a utilização do *Seek* em conjunto com o *Lookup*, pois isso indica que a consulta está bem otimizada e apresenta um bom desempenho. Abaixo, apresento uma explicação da Microsoft sobre o assunto:

1. *Seek:* Trata-se de uma operação de busca otimizada no SQL Server, que ocorre quando um índice é utilizado para localizar diretamente as linhas que atendem a uma condição de filtro em uma consulta. Esse processo é altamente eficiente, pois utiliza um índice, como um índice de árvore B, para navegar rapidamente até o local exato onde os dados estão armazenados. Por exemplo, uma consulta que contém uma cláusula *WHERE* em uma coluna com um índice apropriado utilizará o *seek* para acessar os dados de forma ágil.
2. *Lookup:* Esta operação ocorre quando o SQL Server precisa buscar dados adicionais que não estão presentes no índice utilizado. Normalmente, isso acontece em índices não clusterizados, onde o índice contém apenas uma

chave de referência (como uma chave primária), e o SQL Server deve acessar as colunas restantes na tabela ou no índice clusterizado. Existem dois tipos de *lookup*: *RID Lookup*, que se aplica a tabelas *heap* (sem índice clusterizado), e *Key Lookup*, que é utilizado em tabelas com índice clusterizado.

3. *Scan:* Essa operação é menos eficiente do que o *seek*, pois o SQL Server lê todas as linhas da tabela ou do índice para encontrar correspondências com os critérios da consulta. O *scan* pode ocorrer quando não há um índice apropriado disponível ou quando a consulta envolve um grande número de linhas, fazendo com que, em alguns casos, essa seja a opção mais viável. Um *Table Scan* acontece quando a tabela inteira é lida, enquanto um *Index Scan* percorre todo o índice em busca das linhas correspondentes (Microsoft c, 2024).

5. RESULTADOS

Foi realizado a criação de um ambiente *clusterizado* e realizado procedimentos de *tuning* visando uma melhor performance, para isso foi utilizado técnicas de reescritas, criação de índices e CTE's para auxiliar na execução.

5.1 AMBIENTE CLUSTERIZADO E INDEXADO

De inicio foi realizado o *insert* nas tabelas e a atualização das estatísticas das mesmas, a ativação do *Ad Hoc* ajuste no custo para paralelismo, ajuste do *Maxdop* ,configuração do *TempDB* para que possa crescer de forma ilimitada e realizado o *checkDB* da base buscando qualquer tipo de corrupção.

Consulta *"Relatório Gastos"* executada no ambiente *clusterizado* antes da aplicação de qualquer técnica de *tuning*. No script apresentado na *Figura 17* é descrito as informações do cliente, produto, quantidade, preço e total gasto no ano.

Figura 17: Relatório Gastos pré Tuning Clusterizado

```
SELECT
    v.VendaID,
    c.Nome AS NomeCliente,
    p.Nome AS NomeProduto,
    v.Quantidade,
    v.PrecoVenda,
    v.DataVenda,
    (SELECT SUM(v2.Quantidade)
     FROM Vendas v2
     WHERE v2.ClienteID = v.ClienteID AND v2.ProdutoID = v.ProdutoID) AS
QuantidadeTotalComprada,
    (SELECT SUM(v3.PrecoVenda)
     FROM Vendas v3
     WHERE v3.ClienteID = v.ClienteID AND YEAR(v3.DataVenda) = YEAR(v.DataVenda))
AS TotalGastoAno
FROM
    Vendas v
JOIN
    Clientes c ON v.ClienteID = c.ClienteID
JOIN
    Produtos p ON v.ProdutoID = p.ProdutoID
ORDER BY
    v.DataVenda DESC;
```

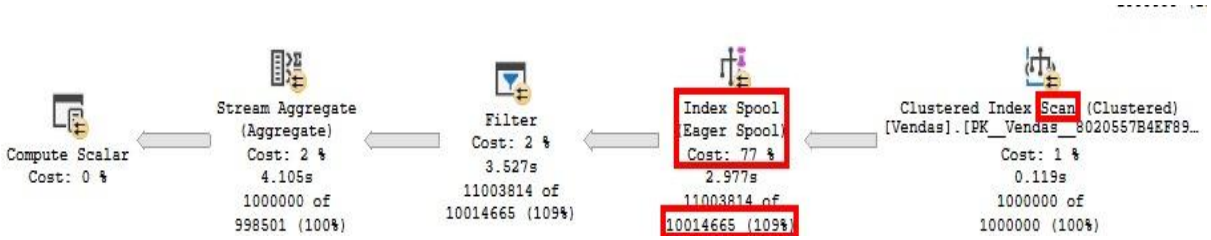
Fonte: Os autores.

Plano de execução da consulta anterior, demonstrando como o SQL Server executa a consulta antes do *tuning*. Inclui detalhes sobre as operações realizadas e os custos associados, com isso foi utilizado o *Index Spool* e realizado um *Scan* no

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

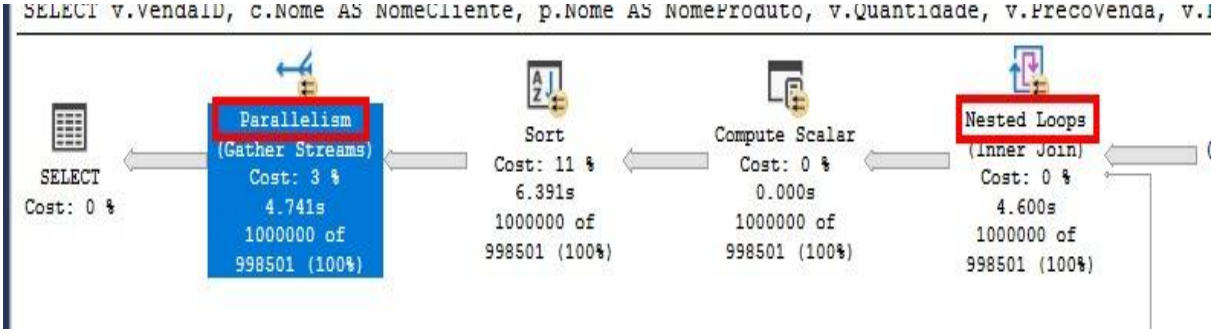
índice *clustered* conforme a *Figura 18*. Parte complementar da figura anterior, temos na *Figura 19* exibição do plano de execução sendo utilizado um *looping* e paralelismo.

Figura 18: Plano de execução Relatório Gastos pré Tuning



Fonte: Os autores.

Figura 19: Continuação do plano de execução acima



Fonte: Os autores.

Na *Figura 20 é e*xibe o perfil de consumo de recursos da primeira consulta *pré-Tuning*, com foco no uso de CPU, leituras realizadas e duração. Oferece uma análise mais detalhada do impacto da consulta no sistema. Consulta executada no ambiente *clusterizado* antes do *tuning*. A *Figura 21* ilustra o script "*Relatório Venda"* nela retorna o ID do Produto, nome, mês, quantidade vendida e quantidade vendida mês anterior.

Figura 20: Profile do consumo da consulta Relatório Gastos pré Tuning

TextData	CPU	Reads	Writes	Duration
SELECT v.VendaID, c.Nome ...	10389	6093941	6920	10322

Fonte: Os autores.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

Figura 21: *Relatório Venda pré Tuning Clusterizado*

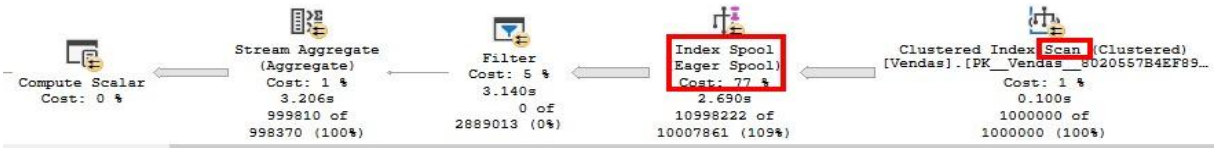
```
SET STATISTICS TIME ON;
SET STATISTICS IO ON;

SELECT
    p.ProdutoID,
    p.Nome AS NomeProduto,
    DATEPART(MONTH, v1.DataVenda) AS MesVenda,
    SUM(v1.Quantidade) AS QuantidadeVendida,
    (SELECT SUM(v2.Quantidade)
     FROM Vendas v2
     WHERE v2.ProdutoID = v1.ProdutoID
     AND DATEPART(MONTH, v2.DataVenda) = DATEPART(MONTH, v1.DataVenda) - 1
     AND DATEPART(YEAR, v2.DataVenda) = DATEPART(YEAR, v1.DataVenda)) AS
QuantidadeVendidaMesAnterior
FROM
    Vendas v1
JOIN
    Produtos p ON v1.ProdutoID = p.ProdutoID
GROUP BY
    p.ProdutoID, p.Nome, DATEPART(MONTH, v1.DataVenda),v1.ProdutoID,v1.DataVenda
ORDER BY
    p.ProdutoID, MesVenda;
```

Fonte: Os autores.

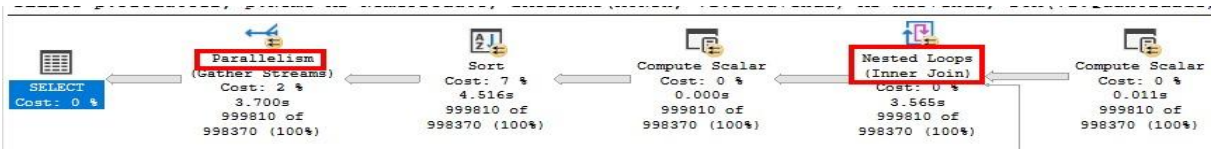
Plano de execução detalhado da imagem acima, mostrando as operações e custos envolvidos antes do ajuste de performance, foi utilizado o *Index Spool* e realizado um *Scan* no índice *clustered* conforme a *Figura 22 A*. Parte complementar do plano de execução, continuando a exibição completa das operações realizadas sendo utilizado um *looping* e paralelismo de acordo com a *Figura 22 B*. Na Figura 23 é exibido o perfil de consumo de recursos, com detalhes sobre CPU, leituras e duração.

Figura 22 A: Plano de execução Relatório Venda pré Tuning



Fonte: Os autores.

Figura 22 B: Continuação do plano de execução acima



Fonte: Os autores.

Figura 23: Profile do consumo Relatório Venda

TextData	CPU	Reads	Writes	Duration
SELECT p.ProdutoID, p.Nom...	8186	6048797	5750	8109

Fonte: Os autores.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

Para melhoria de performance foi realizado a criação dos índices e atualizado as estatísticas, essa medida faz com que as consultas busquem realizar leituras de forma mais rápida e performáticas. Como por exemplo a decisão de realizar um *scan* ou *seek* conforme a Figura 24. Já a Figura 25 é exibido o processo de criação de índices *NonClustered* na tabela "*Vendas*", mostrando como a criação de índices auxilia na otimização da performance das consultas. Na *Figura 26* ilustra a criação de um índice *NonClustered* específico nos campos *"ProductID"* e *"DataVenda"*, destacando como esses índices otimizam as consultas relacionadas.

Figura 24: Atualização de estatísticas

```
USE [TestePerformancePesadoDB]

UPDATE STATISTICS [dbo].[Categorias]
UPDATE STATISTICS [dbo].[Clientes]
UPDATE STATISTICS [dbo].[DetalhesVendas]
UPDATE STATISTICS [dbo].[Produtos]
UPDATE STATISTICS [dbo].[Vendas]
```

Fonte: Os autores.

Figura 25: Criação de índices NonClustered na tabela Vendas

```
CREATE NONCLUSTERED INDEX IDX_Vendas_ClienteID_ProdutoID ON Vendas (ClienteID,
ProdutoID);
CREATE NONCLUSTERED INDEX IDX_Vendas_ClienteID_Ano ON Vendas (ClienteID)
```

Fonte: Os autores.

Figura 26: *Criação de índices NonClustered na tabela Vendas nos campos ProductID e DataVenda*

```
CREATE NONCLUSTERED INDEX IDX_Vendas_ProdutoID_MesVenda ON Vendas (ProdutoID,
DataVenda);
```

Fonte: Os autores.

Para melhor performance foi realizado na *Figura 27 A* e na *Figura 27 B*, tentativas de reescrever as *queries* visando o mesmo resultado com menores custos, isso faz com que o que seria consumido durante a execução possa ser realocado a outra consulta ou outro sistema do ambiente.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

Figura 27 A: Consulta Relatório Gastos pós Tuning Clusterizado

```
SELECT
    v.VendaID,
    c.Nome AS NomeCliente,
    p.Nome AS NomeProduto,
    v.Quantidade,
    v.PrecoVenda,
    v.DataVenda,
    qc.QuantidadeTotalComprada,
    tga.TotalGastoAno
FROM
    Vendas v
JOIN
    Clientes c ON v.ClienteID = c.ClienteID
JOIN
    Produtos p ON v.ProdutoID = p.ProdutoID
LEFT JOIN
    (
        SELECT
            ClienteID,
```

Fonte: Os autores.

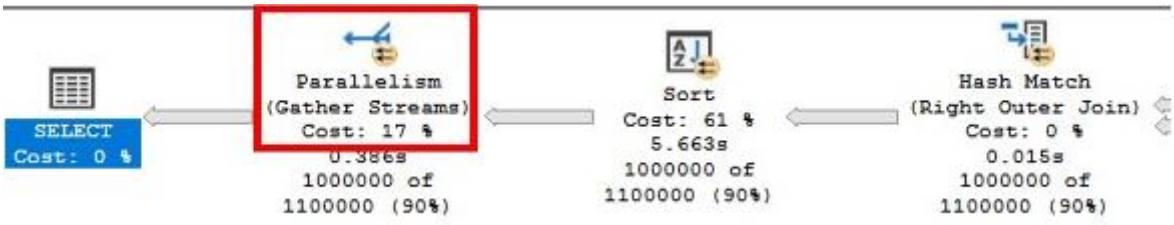
Figura 27 B: Continuação Relatório Gastos pós Tuning Clusterizado

```
LEFT JOIN
    (
        SELECT
            ClienteID,
            ProdutoID,
            SUM(Quantidade) AS QuantidadeTotalComprada
        FROM
            Vendas
        GROUP BY
            ClienteID, ProdutoID
    ) AS qc ON v.ClienteID = qc.ClienteID AND v.ProdutoID = qc.ProdutoID
LEFT JOIN
    (
        SELECT
            ClienteID,
            YEAR(DataVenda) AS AnoVenda,
            SUM(PrecoVenda) AS TotalGastoAno
        FROM
            Vendas
        GROUP BY
            ClienteID, YEAR(DataVenda)
```

Fonte: Os autores.

Plano de execução *"Relatório Gastos" após o tuning,* destacando como as otimizações melhoraram a eficiência do plano de execução, foi observado que o custo melhorou exponencialmente mantendo ainda o uso do paralelismo, porém não realizando o *looping* e *Index Spool* conforme a *Figura 28.* Na *Figura 29* é exibido o perfil de consumo de recursos da primeira consulta *pós-tuning,* mostrando uma redução significativa no uso de CPU, leituras e duração.

Figura 28: Plano de execução Relatório Gastos pós Tuning Clusterizado



Fonte: Os autores.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

Figura 29: Consumo consulta 1 pós Tuning Clusterizado

TextData	CPU	Reads	Writes	Duration
SELECT v.VendaID, c.Nome ...	1860	18866	0	5996

Fonte: Os autores.

Conforme realizado anteriormente utilizamos a mesma metodologia, reescrevemos a *query,* porém utilizamos uma *CTE* para calcular a quantidade vendida no mês anterior de forma eficiente como podemos observar na *Figura 30*.

Figura 30: Relatório Vendas pós Tuning Clusterizado

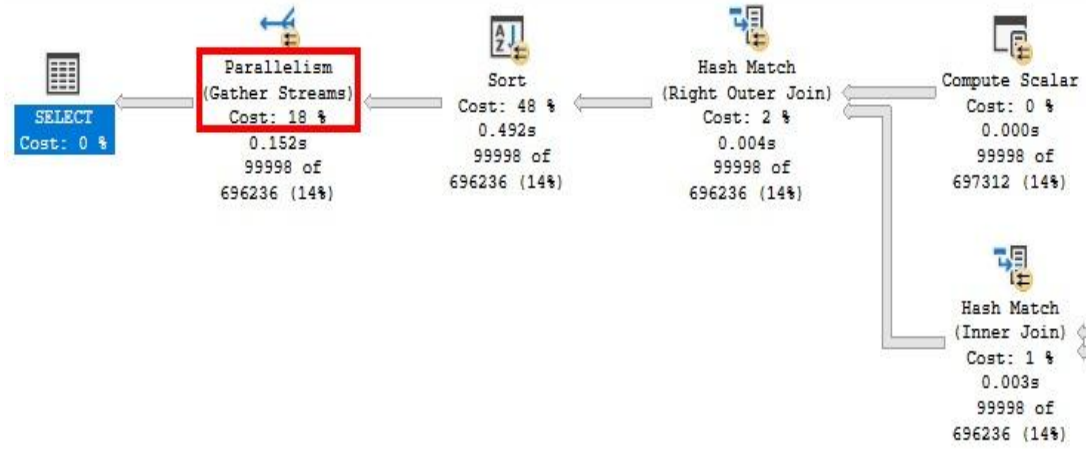
```
WITH CTE_VendasMesAnterior AS (
    SELECT
        ProdutoID,
        DATEPART(MONTH, DataVenda) AS MesVenda,
        DATEPART(YEAR, DataVenda) AS AnoVenda,
        SUM(Quantidade) AS QuantidadeVendida
    FROM
        Vendas
    GROUP BY
        ProdutoID, DATEPART(MONTH, DataVenda), DATEPART(YEAR, DataVenda)
)
SELECT
    p.ProdutoID,
    p.Nome AS NomeProduto,
    v1.MesVenda,
    v1.QuantidadeVendida,
    v2.QuantidadeVendida AS QuantidadeVendidaMesAnterior
FROM
    CTE_VendasMesAnterior v1
LEFT JOIN
    CTE_VendasMesAnterior v2 ON v1.ProdutoID = v2.ProdutoID
    AND v1.MesVenda = v2.MesVenda + 1
    AND v1.AnoVenda = v2.AnoVenda
JOIN
    Produtos p ON v1.ProdutoID = p.ProdutoID
ORDER BY
    p.ProdutoID, v1.MesVenda;
```

Fonte: Os autores.

Plano de execução *"Relatório Vendas"* após o *tuning*, destaca-se o *paralelismo* que se manteve mesmo após reescrita da *query*, porém o maior custo em *Index Spool e looping* não ocorreu, o uso dos índices se manteve realizando um *Scan* pois é necessário trazer toda informação da tabela conforme a *Figura 31*. Na *Figura 32* é exibido o perfil de consumo de recursos do *"Relatório Vendas"* pós-tuning, mostrando as melhorias em termos de CPU, leituras e duração.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

Figura 31: Plano de execução Relatório Vendas pós Tuning Clusterizado



Fonte: Os autores.

Figura 32: Consumo Relatório Vendas pós Tuning Clusterizado

TextData		CPU	Reads	Writes	Duration
WITH CTE_VendasMesAnterior AS (	...	609	12151	0	635

Fonte: Os autores.

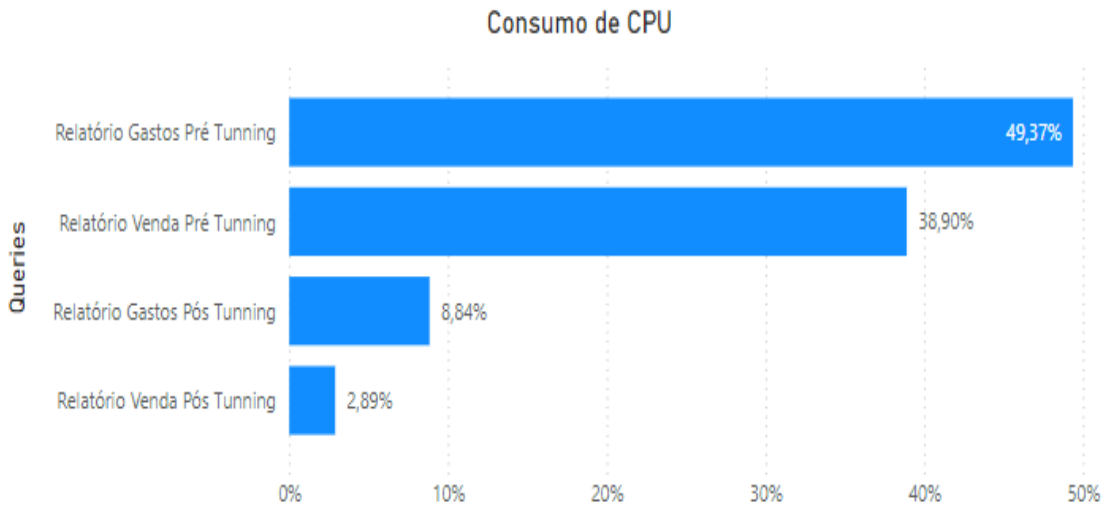
5.2 COMPARAÇÃO DE PRÉ E PÓS TUNING CLUSTERED

O tópico a seguir apresenta uma análise detalhada dos resultados obtidos após a aplicação de um processo de otimização de desempenho conhecido como "*Tuning Clustered*". Este processo foi realizado para melhorar a eficiência de consultas específicas em um banco de dados, reduzindo o consumo de recursos e o tempo de execução. As figuras ilustram os efeitos do *tuning* em diferentes métricas, como consumo de CPU, quantidade de leitura, quantidade de escrita e tempo de duração das consultas. Em cada gráfico, é possível observar a comparação entre os valores anteriores e posteriores ao processo de otimização, destacando a redução significativa no uso de recursos e a melhoria no desempenho das consultas. Esses resultados evidenciam o impacto positivo do *tuning* na eficiência do sistema e na economia de recursos computacionais.

A *Figura 33* demonstra um gráfico de barras horizontal que mostra o consumo de CPU em diferentes relatórios, antes e depois de um processo de "*Tunning*" (otimização). As barras são classificadas com os valores percentuais de consumo de CPU, indicando uma redução significativa após a otimização. De acordo com a *Figura 34* é possível observar um gráfico de barras horizontal que apresenta a Quantidade de Leitura em dois relatórios, antes e depois da otimização ("*Tunning*"). Cada barra representa o número de leituras realizadas para cada relatório, e os valores exatos são indicados ao final de cada barra.

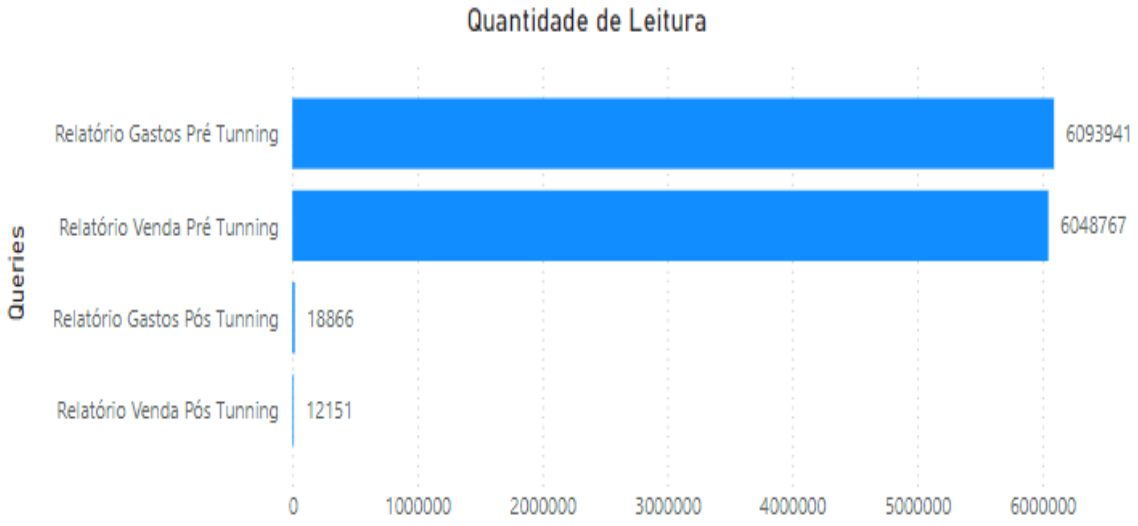
RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

Figura 33: Consumo de CPU das Queries



Fonte: Os autores.

Figura 34: Quantidade de Leitura das Queries

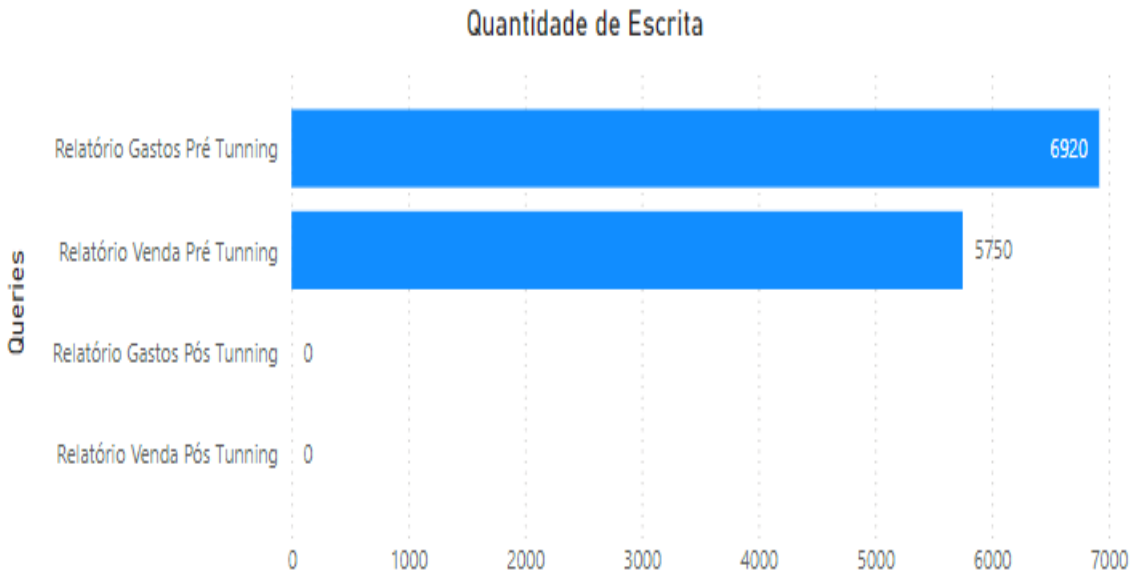


Fonte: Os autores.

Na *Figura 35* é demonstrado um gráfico de barras horizontais intitulado "Quantidade de Escrita". Ele apresenta quatro categorias na vertical, com valores representando a quantidade de escrita para cada uma delas. Já a *Figura 36* é um gráfico de barras horizontais intitulado "Tempo de Duração (ms)". Ela exibe quatro categorias de consultas na vertical, representando o tempo de duração em milissegundos (ms) para cada uma *query*.

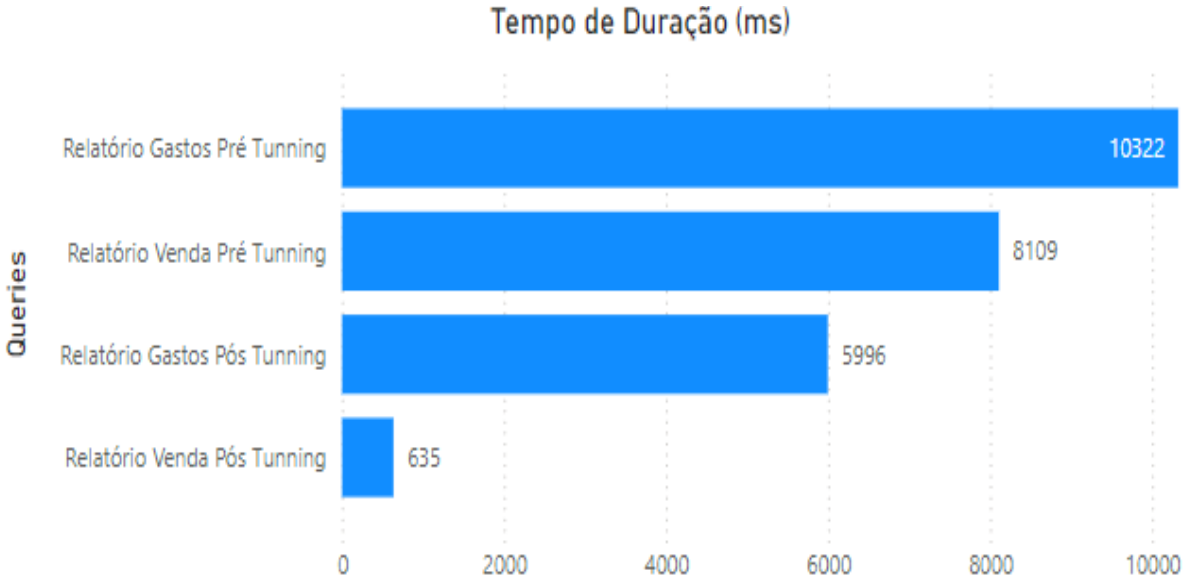
RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 2 2024

Figura 35: Quantidade de Escrita das Queries



Fonte: Os autores.

Figura 36: Tempo de Duração das Queries



Fonte: Os autores.

6. CONCLUSÃO

Este estudo explora a aplicação de técnicas de tuning e otimização no SQL Server, com o objetivo de aprimorar o desempenho de bancos de dados em ambientes corporativos de alta demanda, onde a velocidade de processamento e a eficiência no uso de recursos são fatores críticos. A pesquisa aborda uma série de estratégias que, implementadas em conjunto, oferecem uma base para melhorar a performance do sistema, reduzir tempos de resposta e otimizar o uso de recursos, garantindo um ambiente de dados escalável e resiliente, capaz de suportar o crescimento organizacional.

Um dos primeiros passos do estudo envolveu a análise e o ajuste de configurações de memória e de parâmetros de execução do SQL Server, especialmente o limite máximo de memória e o parâmetro MAXDOP (*Maximum Degree of Parallelism),* que controla o número de CPUs dedicadas a operações de consulta em paralelo. Esses ajustes foram essenciais para evitar sobrecargas, pois o SQL Server, por padrão, tende a alocar toda a memória disponível, o que pode prejudicar o desempenho do sistema em momentos de alta demanda. Ao reservar uma parcela específica de memória para o sistema operacional e limitar a alocação para o SQL Server, o ambiente ganhou estabilidade e eficiência, com um uso equilibrado de recursos, permitindo que o banco de dados atendesse a múltiplas consultas sem comprometer outras operações no servidor.

Outro aspecto crítico abordado foi a criação e manutenção de índices. A utilização de índices adequados permite que o SQL Server localize os dados rapidamente, sem precisar ler a tabela inteira, o que reduz drasticamente o tempo de execução de consultas, especialmente em tabelas com um grande volume de registros. Foram implementados índices *NonClustered*, além da atualização frequente das estatísticas do banco de dados, um componente essencial para que o otimizador de consultas tenha dados atualizados e possa escolher o plano de execução mais eficiente. A criação de índices específicos em colunas-chave foi uma das medidas mais impactantes para melhorar a performance das consultas. Essa prática permitiu que as buscas fossem realizadas de forma precisa e rápida, minimizando a quantidade de leituras necessárias.

Além disso, o estudo explorou a reestruturação de queries complexas, um fator decisivo para otimizar o uso de CPU e memória. Queries complexas podem gerar grandes custos de execução se mal formuladas, acarretando operações desnecessárias e aumentando o tempo de resposta. Assim, foram aplicadas reescritas de queries para reduzir loops, evitar subconsultas desnecessárias e simplificar operações de junção (*joins*), focando em estratégias que maximizem a eficiência sem sacrificar a precisão dos resultados. Essa reestruturação liberou recursos e reduziu o consumo de CPU, permitindo que o sistema atendesse a uma maior quantidade de requisições de maneira mais rápida e eficiente.

A aplicação das práticas de *tuning* também incluiu a configuração do paralelismo de execução, ajustando parâmetros como o *Cost Threshold for Parallelism*, que determina o custo mínimo necessário para que uma consulta seja executada em paralelo. Em sistemas com grande carga de trabalho, aumentar esse valor ajuda a reservar operações paralelas apenas para consultas mais complexas, evitando que o paralelismo seja aplicado em consultas simples, o que poderia resultar em uso ineficiente de recursos. Esse ajuste, junto com a configuração do

MAXDOP, permitiu que o SQL Server utilizasse os recursos de CPU de forma mais inteligente, garantindo que operações paralelas fossem aplicadas apenas onde realmente se justificassem, otimizando o desempenho de consultas mais custosas.

Para garantir uma manutenção constante e um desempenho ideal, o estudo também abordou a importância de práticas de rotina como a configuração do *TempDB* e o uso do recurso *Optimize for Ad Hoc Workloads*, que evita o desperdício de memória em consultas que são executadas esporadicamente. Configurações apropriadas do *TempDB*, banco temporário responsável por armazenar objetos temporários e realizar operações internas do *SQL Server*, também foram fundamentais, com ajustes no número de arquivos de dados e uso de discos rápidos, como *SSDs*, para otimizar operações de *I/O* intensivo.

Os resultados das práticas de *tuning* foram mensurados por meio de comparações detalhadas entre o consumo de recursos antes e após a aplicação das otimizações. O impacto dessas práticas foi significativo, com reduções no tempo de execução, menor uso de CPU e menos leituras realizadas. Essas melhorias refletiram diretamente na eficiência do ambiente, tornando o sistema mais rápido e responsivo, capaz de atender às exigências de dados em tempo real sem comprometer o desempenho de outros processos. Em consultas otimizadas, o SQL Server mostrou-se capaz de manter tempos de resposta ágeis, essencial para suportar operações em ambientes empresariais, onde atrasos podem afetar diretamente a tomada de decisões e a produtividade.

Em conclusão, o estudo evidencia que a aplicação integrada de técnicas de *tuning*, incluindo ajustes de memória, configuração de parâmetros de paralelismo, criação e atualização de índices e reestruturação de consultas, oferece um impacto positivo e mensurável na performance do SQL Server. A prática contínua dessas técnicas não apenas proporciona um ambiente de banco de dados mais eficiente e escalável, como também fortalece a operação de negócios, melhorando a qualidade e a velocidade das respostas em processos de análise e decisão. Para empresas que lidam com grandes volumes de dados e necessitam de respostas em tempo real, essas estratégias de otimização tornam-se fundamentais, garantindo que o sistema possa crescer junto com a organização e responder às necessidades crescentes do ambiente corporativo.

REFERÊNCIAS

ARAUJO, Juarez. Tuning em Banco de Dados: conheça tudo sobre o assunto [Internet]. Blog DBACorp, 2020. Disponível em: https://blog.dbacorp.com.br/2020/07/30/tuning-em-banco-de-dados/. Acesso em: 11 set. 2024.

DAS, S., & CLOWER, J. (2015). *SQL Server 2014 Partitioning Strategies for Performance*. 1ª Edição. Springer

DELANEY, Kalen, et al. *SQL Server 2019 Internals*. Microsoft Press, 2020.

ELMASRI, Ramez, and Shamkant B. Navathe. *Fundamentals of Database Systems*. Pearson, 2016 a.

ELMASRI, R., & NAVATHE, S. B. (2016 b). *Sistemas de Banco de Dados*. 7ª Edição. Pearson.

GARCIA-MOLINA, Hector, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Pearson, 2008.

GURSU, M. (2019). *SQL Server Query Performance Tuning*. 1ª Edição. Packt Publishing.

HARRISON, D. (2015). *SQL Server 2014 Query Performance Tuning*. 1ª Edição. Apress.

KLISMANN, T. (2020). *SQL Server Performance Tuning*. 1ª Edição. Packt Publishing.

KRAUSE, J., & ZIMMERMANN, S. (2013). *SQL Server 2012 T-SQL Fundamentals*. 2ª Edição. Pearson.

KUMAR, A. (2023). *Database Performance Tuning*. Tech Publications.

MALIK, M. (2017). *SQL Server Statistics*. 1ª Edição. Packt Publishing.
GOLDBERG, R. (2015). *SQL Server Performance Tuning*. 1ª Edição. O'Reilly Media.
KRAUSE, J. (2017). *SQL Server Performance Tuning and Optimization*. 1ª Edição. Apress.

MICROSOFT. Edições e recursos com suporte do SQL Server 2022 - SQL Server [Internet]. Learn.microsoft.com, 2023 a. [citado em 9 abr. 2024]. Disponível em: https://learn.microsoft.com/pt-br/sql/sql-server/editions-and-components-of-sql-server-2022?view=sql-server-ver16.

MICROSOFT. O que é o SQL Server? - SQL Server [Internet]. Learn.microsoft.com, 2023 b. Disponível em: https://learn.microsoft.com/pt-br/sql/sql-server/what-is-sqlserver?view=sql-server-ver16. Acesso em: 11 set. 2024.

MICROSOFT. Query Performance Tuning Techniques [Internet]. 2024 c. Disponível em: https://docs.microsoft.com/en-us/sql/performance-tuning. Acesso em: 11 set. 2024.

MICROSOFT. Tables without Clustered Indexes (Heaps) [Internet]. 2024 d. Disponível em: https://docs.microsoft.com/en-us/sql/relational-databases/sql-server. Acesso em: 11 set. 2024.

MUSSEN, R. (2015). *SQL Server Indexing and Tuning*. 1ª Edição. O'Reilly Media.

NUNES, C. P. (2008). DBMS-Analyzer: Um Framework para Análise Holística de Desempenho de SGBDs. Universidade Federal de Campina Grande1

PRAKASH, R., & PACHORE, K. (2016). *Performance Evaluation of SQL Server using Different Types of Indexes*. International Journal of Computer Applications.

REIS, A. C., Santos, E. M. D., & Arruda, M. R. de A. (2012). Modelos de Avaliação de Desempenho de Sistemas de Saúde: Diferenças e Similaridades. Escola Nacional de Saúde Pública, Fiocruz

SHAW, R. (2018). *SQL Server Maintenance Solutions*. 1ª Edição. Apress.

SILBERSCHATZ, Abraham, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, 2010.

SILBERSCHATZ, A., KORTH, H. F., & SUDARSHAN, S. (2011). *Sistemas de Banco de Dados*. 6ª Edição. McGraw-Hill. Pearson.

VAN ZANDT, C., & STOKES, A. (2014). *SQL Server 2012 High Availability*. 1ª Edição. Apress.

WANG, X. (2021). *SQL Server Partitioning and Performance Optimization*. 1ª Edição. Apress.