Revista EduFatec: educação, tecnologia e gestão
V.1 N.1 janeiro-junho/2018

TUNING DE BANCO DE DADOS COM SQL SERVER

Kátia Lima de Oliveira[1]

Claudio Eduardo Paiva[2]

Resumo

Bancos de dados estão sempre presentes no dia a dia, ainda que não sejam percebidos. Com a crescente utilização de sistemas informatizados e redes sociais, o volume de dados produzidos nos últimos tempos cresceu muito. Usuários não toleram sistemas lentos e falhos. O que fazer para garantir que bancos de dados, cada vez maiores, não percam desempenho e desapontem seus usuários? O objetivo deste trabalho é apresentar o *tuning.* Um processo de otimização do desempenho de bancos de dados através da modificação das configurações do Sistema de Gerenciamento de Banco de dados (SGBD). Utilizando o SQL Server, serão testadas algumas técnicas e verificadas sua eficácia nos sistemas operacionais Windows 8.1 e Ubuntu 16.04.

Palavras-chave: Banco de dados relacional. Desempenho. Otimização.

Abstract

Databases are always present in everyday life, although they are not noticed. And with the increasing use of computerized systems and social networks, the volume of data produced in recent times has grown a lot. Users do not tolerate slow and flawed systems. What can be done to ensure that larger databases do not lose performance and disappoint its users? The objective of this research is to present the tuning. A process of optimizing database performance by modifying the Database Management System (DBMS) settings. Using SQL Server, some techniques will be tested and their effectiveness verified in Windows 8.1 and Ubuntu 16.04 operating systems.

Keywords: *Relational database. Performance. Optimization.*

1 Introdução

O mundo está cada vez mais dependente da tecnologia. Empresas preferem dispensar os antigos livros de contabilidade e os controles manuais em papel para usar sistemas informatizados que lhes ofereçam segurança, praticidade e maior produtividade. Dessa necessidade surgem diversas empresas de desenvolvimento de software construindo aplicações que atendam às necessidades dos clientes. A

1 Graduanda em Análise e Desenvolvimento de Sistemas pela Fatec Dr Thomaz Novelino – Franca/SP. Endereço eletrônico: katia.oliveira10@fatec.sp.gov.br.
[2] Especialista em Análise de Sistemas pela PUC-Campinas – Campinas/SP. Endereço eletrônico: claudiopaiva2@yahoo.com.br

crescente utilização da aplicação pode revelar alguns problemas de performance com o passar do tempo. Sistemas lentos ou que apresentam travamentos geralmente não são tolerados pelos usuários. Uma empresa que fornece software de desempenho abaixo do esperado pelos seus clientes, corre risco de perdê-los.

Nem sempre problema de lentidão no software significa desenvolvimento ruim. Souza (2009) estima que de 60% a 90% dos casos de problemas de performance tem como causa o banco de dados pois, consultas mal elaboradas e configurações mal ajustadas do SGBD e índices mal formulados podem comprometer a utilização do sistema. Uma solução para esta questão é o chamado *tuning* de banco de dados. *Tuning,* segundo Baptista (2008 *apud* Pansonato, 2012), é o ajuste do SGDB para prover um uso mais eficaz e eficiente dos recursos. Seu objetivo, portanto, é minimizar o tempo de retorno de uma consulta, tornando mais rápida a resposta da aplicação aos comandos do usuário.

Este trabalho tem como objetivo estudar o *tuning* de banco de dados relacionais, utilizando o MS SQL Server nos sistemas operacionais Ubuntu e Windows. Usando o banco de dados com dados gerados aleatoriamente, será analisado o tempo de resposta das tabelas por meio dos logs que capturam os dados necessários para a análise de desempenho do banco de dados. Partindo destes resultados e da pesquisa sobre as técnicas de otimização, serão realizados testes para comprovar a eficácia dos métodos e técnicas estudados.

2 O Sistema Gerenciador de Banco de Dados

O aperfeiçoamento do desempenho do banco de dados depende das configurações do SGBD e, para Elmasri (2011), é ele “que facilita o processo de definição, construção, manipulação e compartilhamento de banco de dados entre diversos usuários e aplicações”.

De acordo com Silberschatz (2006), o SGBD também tem a função de garantir a segurança das informações, mesmo em casos de acesso não autorizado ou falhas. Para o autor, o objetivo do SGBD é oferecer um ambiente eficiente para o armazenamento e a recuperação das informações e, para efetuar suas tarefas, está dividido em alguns subsistemas, como o gerenciador de armazenamento e o processador de consultas.

Ainda segundo Silberschatz (2006), o gerenciador de armazenamento é o responsável por armazenar, recuperar e atualizar os dados, traduzindo comandos de *Data Manipulation Language (*DML - linguagem de manipulação de dados), em comandos de baixo nível e é composto da seguinte maneira:

- **Gerenciador de autorização e integridade:** responsável pelas restrições de integridade e pela autorização de acesso aos dados;
- **Gerenciador de transação:** garante a consistência dos dados mesmo em casos de falhas, e que as transações concorrentes sejam executadas sem conflitos;
- **Gerenciador de arquivos:** controla a alocação de espaço em disco e as estruturas de dados utilizadas para armazenar as informações;
- **Gerenciador de *buffer*:** resgata dados do armazenamento de disco e seleciona quais deles colocar no cache da memória principal.

O processador de consulta é formado pelo:

- **Interpretador de DDL:** interpreta as instruções de *Data Definition Language* (DDL - linguagem de definição de dados) para definir a estrutura do banco de dados, e armazena as novas definições no dicionário de dados;
- **Compilador de DML:** traduz as instruções DML em instruções de baixo nível para serem executadas pelo mecanismo de avaliação de consultas. Também realiza uma otimização das consultas, selecionando a forma de menor custo para se realizar a consulta;
- **Mecanismo de avaliação de consulta:** executa as instruções geradas pelo compilador de DML.

2.1 Tipos de SGBD

Os Sistemas de Gerenciamento de Banco de Dados podem ser classificados em diversos tipos. São alguns deles:

Hierárquico: o primeiro SGBD comercialmente popular foi hierárquico. O *Information Management System* (IMS) da *International Business Machine* (IBM) foi bastante popular na década de 70 e ainda existem alguns em uso. Este tipo de SGBD

Revista EduFatec: educação, tecnologia e gestão
V.1 N.1 janeiro-junho/2018

é baseado em segmentos em um relacionamento pai-filho organizados em forma de árvore. Cada segmento filho pode ter apenas um pai relacionado a ele, enquanto um segmento pai pode estar relacionado a vários filhos. O nó principal da árvore do sistema é chamado de raiz e todas as buscas se iniciam a partir dele (BITTTENCOURT, 2004)

Rede: é semelhante ao modelo hierárquico, pois foi concebido como um aperfeiçoamento deste. Se organiza em forma de grafos, contendo registros proprietários e registros membros. Ao contrário do modelo hierárquico, registros membros podem possuir mais de um proprietário (DATE, 1989; TAKAI, 2005).

Relacionais: o modelo relacional é organizado em tabelas, compostas por atributos (colunas da tabela) definidos por um tipo de dado. Cada linha da tabela é chamada de registro ou tupla (TAKAI, 2005). Este é o modelo que será estudado neste trabalho.

Orientado a objetos: surgiu a partir das limitações do modelo relacional, já que este não se mostrava adequado para lidar com dados complexos como informações científicas, médicas, geográficas e multimídia. Neste paradigma, o diagrama de classes da UML (*Unified Modeling Language*) serve para modelar tanto a aplicação quanto o banco de dados. Os objetos armazenam os dados relacionados a sua respectiva classe bem como as operações que devem ser executadas sobre eles. A linguagem *Structured Query Language* (SQL) é substituída pela linguagem de programação utilizada no desenvolvimento da aplicação (BOSCARIOLI et al., 2006).

Objeto-relacional: foi criado para buscar sanar as dificuldades do modelo relacional em lidar com dados complexos. Inclui as funcionalidades da orientação a objeto mantendo a organização em tabelas e passa a armazenar tipos de dados complexos (SILBERSCHATZ, 2006; TAKAI, 2005).

Não-relacionais: surgiu para sanar problemas dos bancos de dados relacionais associados a desempenho e escalabilidade. Seu modelo de persistência de dados se baseia em desempenho, disponibilidade e escalabilidade. Não possui um *schema* de dados pré-definido, podendo ser um banco de dados de documentos, grafos ou chave-valor (TOTH, 2011).

2.3 Características de um SGBD Relacional

Para que seja caracterizado como um Sistema de Gerenciamento de Banco de Dados, o software que controla o banco de dados precisa ter as seguintes características:

Controle de redundâncias: chama-se 'redundância' o armazenamento duplicado de uma informação em tabelas diferentes. Existem dois tipos de redundância de dados: a não-controlada e a controlada (HEUSER, 2009). Por exemplo: em uma tabela 'alunos' são armazenados nome e data de nascimento. Para facilitar a consulta aos dados de matrícula, foi feita uma desnormalização, sendo adicionados à tabela 'matrícula' os campos de nome e data de nascimento do aluno, para que fossem gravados no momento da inserção. Se houver uma alteração destes dados da tabela 'alunos' e eles não forem automaticamente atualizados na tabela 'matrícula', ocorre uma redundância não-controlada e os dados se tornam inconsistentes, pois não existe controle da atualização dos dados e uma mesma informação aparece em dois lugares com valores distintos. Mas se o desenvolvedor utilizou alguma forma de atualizar a tabela 'matrícula' ao alterar-se os dados da tabela 'alunos', seja por um controle na própria aplicação ou por meio de *triggers*, ocorre uma redundância controlada e os dados do banco se mantêm consistentes (ELMASRI, 2011).

Compartilhamento dos dados e Controle de concorrência: um SGBD multiusuário permite o acesso simultâneo de vários usuários ao banco. Para que as operações sejam executadas de maneira controlada é necessário que haja um controle de concorrência, o que garante que o resultado de diversas transações simultâneas seja correto. Por exemplo: em um sistema de reserva de passagens aéreas, o SGBD deve fazer com que os usuários reservem as poltronas uma de cada vez, garantindo que cada poltrona tenha apenas um passageiro destinado a ela (ELMASRI, 2011; SILBERCHATZ, 2006).

Controle de acesso: o SGBD possui um mecanismo capaz de restringir o acesso dos usuários a determinadas funções ou dados. Por motivos de segurança, o administrador do banco de dados pode permitir que apenas alguns usuários visualizem ou modifiquem dados de determinadas tabelas (ELMASRI, 2011).

Controle de integridade: para que um banco de dados se mantenha

consistente, é necessário que o SGBD controle sua integridade, ou seja, previna a entrada de dados incorretos ou inválidos.

- Integridade de dados**:** garante a qualidade dos dados inseridos. Exemplo: se um aluno recebe como número identificador '321', não deve haver outro aluno com o mesmo número. Se uma coluna de notas é definida para receber apenas valores entre 0 e 10, o número 22 não deve ser aceito. São controladas por restrições *primary key, foreign key, unique, check* e definições *default, null*, etc.
- Integridade referencial: garante que um registro existente em uma tabela relacionada, possua um correspondente em uma tabela principal. Exemplo: um registro em uma tabela de matrícula deve ter um aluno relacionado a ela existente em uma tabela de alunos. Isto evita a existência de registros órfãos. É controlado pela restrição *foreign key* (MICROSOFT, 2012a, *online*).

Backups: um SGBD deve ser capaz de fazer *backups* e restaurações para que possa recuperar dados em caso de falhas. Se uma transação não for realizada com sucesso, o banco de dados deve ser restaurado da forma como estava antes da transação. Em casos mais graves de perda de dados, os *backups* restauram as informações perdidas (ELMASRI, 2011).

Armazenamento persistente: em um SGBD os dados são persistentes, ou seja, ao final da execução de um programa, os dados continuam gravados e disponíveis para consultas e atualizações. Uma vez que os dados foram inseridos, só podem ser excluídos por meio de um comando explícito, jamais devido ao término de uma execução de uma aplicação. Porém, nem todos os dados são persistentes. Existem dados transientes, ou seja, dados temporários resultantes de instruções SQL, resultados intermediários, etc. (DATE, 2003).

Execução de consultas e atualizações: um SGBD precisa executar consultas da forma mais eficiente possível. Para isso ele conta com os índices, que são arquivos auxiliares baseados em uma estrutura de dados de árvore ou de *hash*. Com base neles, o módulo de processamento e otimização de consulta do SGBD criará um plano de execução para a consulta. Definir quais índices criar, manter ou eliminar são tarefas de um administrador de banco de dados para otimizar as consultas (ELMASRI, 2011).

2.4 Fatores Causadores de Problemas de Desempenho em Bancos de Dados Relacionais

Dentre os elementos que compõem um sistema de banco de dados, vários fatores podem causar erros, travamentos, lentidão ou instabilidade nas aplicações. O administrador de banco de dados precisa ter em vista que todo o ambiente de execução do software influencia seu desempenho. Assim, seria um erro focar em otimização do SGBD e negligenciar as configurações do sistema operacional, da rede, do hardware, etc.

Hardwares obsoletos ou com defeitos costumam fazer parte dos fatores que podem afetar o desempenho das aplicações. Um *upgrade* neste caso teria um ótimo resultado, mas se não for possível, pode-se tentar melhorar a realocação de memória (LÓPEZ e DILL, 2012).

O sistema de arquivos utilizados pelo sistema operacional também exerce papel fundamental na determinação do desempenho do SGBD. Alguns sistemas de arquivos oferecem melhor desempenho ao banco de dados do que outros. É o caso do XFS que demonstrou em testes, ser o melhor neste quesito em ambiente Linux (LÓPEZ e DILL, 2012).

Dentre estes fatores, merece destaque a qualidade da estrutura do banco de dados. É necessário dar atenção ao seu desempenho desde a fase da modelagem, pois, a ausência ou mal uso de índices e a desnormalização das tabelas afetam a velocidade das consultas e atualizações, além de causar redundâncias e erros (MACHADO, 2014).

Segundo Fritchey e Dam (2014), depois de ter-se otimizado o hardware, o sistema operacional e as configurações do SQL Server, alguns dos “assassinos de performance” são:

Indexação insuficiente: sem uma indexação adequada, uma consulta leva mais tempo para ser executada, pois o SGBD precisa recuperar e processar muito mais dados.

Estatísticas imprecisas: para que o uso dos índices seja efetivo, é necessário manter estatísticas precisas e atualizadas sobre a quantidade de linhas recuperadas nas consultas. Estes dados são extremamente importantes para que o otimizador de consultas do SQL Server decida a melhor estratégia de otimização. Sem dados precisos, sua eficiência é prejudicada.

***Design* de consulta impróprio:** a eficácia dos índices depende da qualidade das consultas executadas. Consultas que retornam uma quantidade excessiva de linhas impedem o SGBD de usar os índices adequadamente, aumentando seu tempo de execução.

***Design* do banco de dados impróprio:** a normalização adequada do banco de dados é essencial para manter um bom desempenho. Um banco insuficientemente normalizado tende a manter dados repetidos nas suas tabelas, sendo que esta repetição aumenta o tempo necessário para execução das contas dos dados. Por outro lado, um banco excessivamente normalizado, que contenha um alto número de tabelas com uma pequena quantidade de colunas, também prejudica o desempenho. Neste caso, a quantidade e a complexidade das junções (*joins)* entre a diversas tabelas requerem um esforço maior para a recuperação dos dados.

2.5 Índices e Consultas

Em um sistema de banco de dados, um índice funciona como o sumário de um livro. Se o leitor precisa encontrar determinado assunto, pode consultar o sumário e depois abrir o livro diretamente na página desejada. No sumário os assuntos estão organizados e, por ser muito menor do que o livro, torna-se muito mais rápido encontrar um assunto nele do que vasculhar todas as páginas do livro. De maneira semelhante, um índice proporciona uma maneira mais rápida de encontrar um dado, sem precisar procurar por toda a tabela (FRITCHEY, 2014).

No SQL Server existem dois tipos de índices: o clusterizado e o não-clusterizado. Ambos adotam a estrutura de dados de árvore binária *(B-Tree)* que realiza uma organização em forma de árvore, permitindo um acesso rápido aos dados.

Índices clusterizados: um índice clusterizado determina a ordem em que os registros são armazenados na tabela. Não se trata de um índice separado da tabela, como no caso do índice não-clusterizado. O índice clusterizado é a própria tabela organizada sequencialmente. Como o índice clusterizado organiza os dados fisicamente, só é possível haver um por tabela. Quando uma tabela não possui índice clusterizado, seus registros são armazenados em uma estrutura não ordenada chamada *'heap'* (JORGENSEN et al., 2012).

Índices não-clusterizados: os índices não-clusterizados não armazenam os registros como os clusterizados fazem. Eles guardam a chave do índice não

clusterizado e para cada uma destas chaves, um ponteiro para o registro na tabela base. Este ponteiro é denominado 'localizador de linhas'. A estrutura do localizador varia caso os dados estejam em uma tabela clusterizada ou em um *heap*. Se estiverem em uma tabela clusterizada, o localizador de linha é a chave do índice clusterizado, se estiverem em um *heap*, o localizador é um ponteiro para a linha (MICROSOFT, 2017b, online).

Índices podem ser compostos por uma ou mais colunas de uma tabela. Tomando como exemplo um banco de dados de uma rede de lojas, uma tabela de vendas deve possuir o número identificador da loja e o número identificador da venda. É possível utilizar estas colunas juntas tanto em um índice clusterizado como em um não-clusterizado.

Ao se adicionar uma chave primária ou uma restrição *unique* em uma tabela, o SQL Server automaticamente associará um índice clusterizado a eles. Opcionalmente podem ser não-clusterizados (JORGENSEN, 2012; MICROSOFT, 2017b, *online*).

2.6 O que é *Tuning*

Segundo Winand (2012), os problemas de performance dos bancos de dados SQL são tão antigos quanto a própria linguagem SQL. A principal vantagem desta é conseguir separar o "o que fazer" do "como fazer". Uma instrução SQL apenas descreve "o que" é necessário fazer, sem indicar "como" esta instrução deve ser executada. Esta separação permite que desenvolvedores usem instruções SQL sem saber como ocorre o processamento destes comandos.

Para o quesito desempenho do banco de dados, entretanto, isto pode representar um problema. Sem saber como o SGBD processa as instruções, o desenvolvedor tende a focar mais no "o que" ele precisa e não no "como" a operação será executada, o que pode causar instruções mal escritas e, consequentemente, aplicações mais lentas.

Por isso, a preocupação com a eficiência do SGBD deve estar presente desde o início da elaboração do banco de dados, de forma a desenvolver uma estrutura que garanta seu bom desempenho.

Entretanto, após o banco de dados entrar em operação, seu uso pode revelar problemas que não foram considerados na etapa do projeto inicial. O monitoramento

da utilização de recursos e do processamento interno do SGBD podem revelar gargalos de desempenho.

Desta forma, *tuning* é um processo de otimização da performance do banco de dados que, segundo Fritchey (2014), consiste em identificar estes gargalos, priorizar os problemas identificados, agir sobre suas causas aplicando diferentes soluções e medindo os resultados e, então, repetir este processo continuamente. O *tuning* deve ser um processo contínuo, e não pontual, de forma que o bom desempenho da aplicação fique assegurado.

2.6.1 Tipos e Objetivos do *Tuning*

Segundo Macedo (2013), existem três tipos de otimização de banco de dados:

Otimização de consultas e objetos internos: consiste em analisar as instruções SQL a serem executadas, a estrutura das tabelas e de seus índices, além de outros objetos que possam ser otimizados.

Otimização do sistema operacional: consiste em analisar e modificar as configurações do sistema operacional utilizado como servidor.

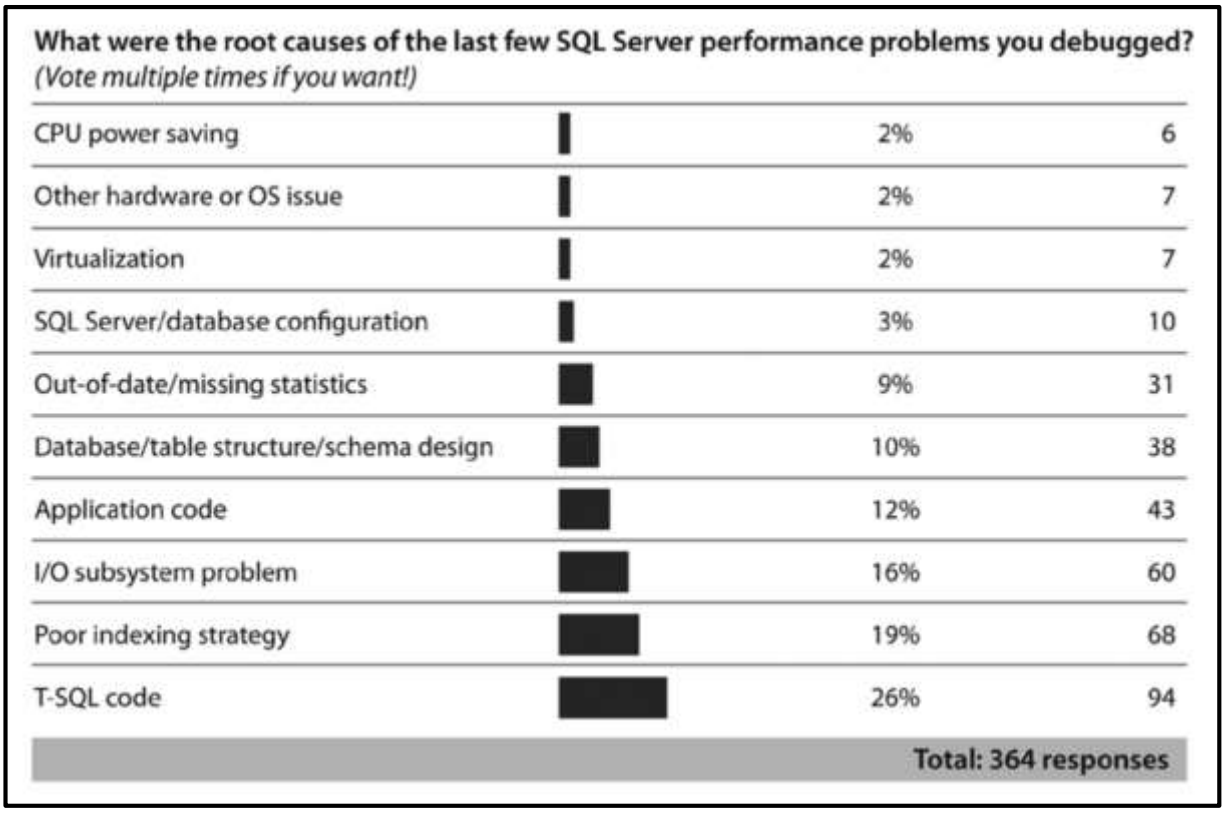
Otimização da arquitetura de memória do SGBD: é específica para cada SGBD. Consiste em analisar e modificar parâmetros de ambiente e de memória do SGBD para maximizar sua capacidade de leitura e escrita de dados.

Dados os três tipos de otimização, em qual deles focar esforços para obter melhores resultados? Segundo Fritchey (2014), ainda que uma configuração apropriada do hardware, do sistema operacional e da instância do SQL Server sejam essenciais para uma boa performance do banco de dados, estes elementos são tão padronizados que pouco tempo seria necessário para configurá-los apropriadamente. Já as estratégias de indexação e os designs de consulta são únicos para o conjunto de dados. Consequentemente, há mais para otimizar nas consultas e no acesso ao banco de dados do que no *hardware*, sistema operacional ou nas configurações do SQL Server.

Além disso, segundo Randal (*apud* FRITCHEY, 2014), as maiores causas de problemas de performance em bancos de dados são o design das consultas e a indexação, somando 45% dos problemas de desempenho encontrados pelos administradores de bancos de dados que participaram de sua pesquisa, conforme mostra a Figura 1:

Revista EduFatec: educação, tecnologia e gestão
V.1 N.1 janeiro-junho/2018

Figura 1: Gráfico com as principais causas de problemas de performance



Fonte: FRITCHEY (2014)

Percebe-se que as maiores causas de problemas de desempenho apontadas estão relacionadas a instruções SQL, índices, estrutura do banco de dados e código da aplicação, somando 67% dos casos. Concentrar esforços nas configurações do hardware traria, sem dúvida, algum ganho no desempenho, mas uma instrução SQL mal escrita poderia consumir todos os recursos de hardware disponíveis, o que tornaria este um mal investimento.

3 Técnicas e Métodos

Considerando que concentrar esforços em otimizar instruções SQL, estruturas de índices e de banco de dados pode retornar melhores resultados do que focar apenas em *hardware* ou nas configurações do SGBD, mas que sem fazer uma otimização destes o *tuning* estaria incompleto, quais técnicas deve-se aplicar nas instruções SQL e na estratégia de indexação e quais parâmetros devem ser modificados no SGBD afim de obter melhor performance?

3.1 Configurando o SQL Server

É comum que se utilize o SQL Server com as suas configurações originais. Entretanto, é possível modificar algumas de suas propriedades para obter algum ganho de desempenho. O SQL Server possui 75 opções de configuração do servidor para gerenciar e otimizar seus recursos (MICROSOFT, 2017c, *online*). Porém, nem todos impactam diretamente no desempenho das consultas. Pode-se destacar entre eles (SALES, 2013):

Fill factor: na criação de um índice, o *fill factor*, ou fator de preenchimento, especifica a porcentagem de espaço a ser preenchida com dados em cada página. O espaço restante é reservado para futuro crescimento do índice e estão localizados entre as linhas do índice, não no final. Especificar um *fill factor* 70, por exemplo, significa que 70% de cada página será preenchida com dados e 30% ficará vazia para ser usada futuramente (MICROSOFT, 2017a, *online*). Quando não há mais espaço para inserir novos dados em uma página, uma nova é criada, mantendo a ordem lógica do índice, mas não necessariamente mantendo a ordem física do índice no disco. Assim, ocorre a fragmentação de índice, algo que pode, com o passar do tempo, prejudicar as leituras ao banco de dados (FRITCHEY, 2014).

Min Memory per Query: especifica a quantidade mínima de memória a ser alocada pelo SQL Server para a execução de uma consulta. Pode-se dar a ele valores entre 512Kb e 2GB. Aumentar este valor pode trazer ganhos de performance em consultas pequenas e médias, mas pode também levar a uma maior competição por recursos de memória (MICROSOFT, 2012b, *online*).

Min Server Memory e Max Server Memory: estes parâmetros estabelecem os limites inferiores e superiores da quantidade de memória que será utilizada pelo SQL Server. A alocação de memória respeitará estes limites, desde que o limite máximo não esteja abaixo de 128 MB, o que pode impedir que o servidor inicie. Alterar estes parâmetros pode melhorar ou comprometer a eficiência do SGBD (MICROSOFT, 2017d, *online*).

Revista EduFatec: educação, tecnologia e gestão
V.1 N.1 janeiro-junho/2018

3.2 Boas Práticas na Criação de Índices

Dentre todas as estratégias possíveis de serem adotadas para dar boa performance ao banco de dados, seguir as boas práticas na criação de índices é a mais importante e indispensável. Pode-se destacar algumas delas:

- Procurar criar índices estreitos, ou seja, com a menor quantidade possível de colunas;
- Utilizar colunas com valores inteiros, nunca do tipo texto ou que contenham valores nulos;
- Não criar índices clusterizados em colunas que recebem alterações frequentemente;
- Criar primeiro os índices clusterizados, depois os não clusterizados;
- Ao planejar a criação de um índice, levar em consideração consultas que utilizarão as cláusulas *WHERE*, *HAVING* e *JOIN*. As colunas sobre as quais recaem consultas com estas cláusulas, são colunas ideais para se tornarem índices;
- Sempre inserir chave primária na tabela.

3.3 Boas Práticas na Criação de Consultas

Apesar da praticidade da linguagem SQL, que permite que apenas seja informado o que se deseja consultar, sem que se saiba como essa consulta será executada, é interessante que o desenvolvedor conheça quais cláusulas ou comandos exigem mais capacidade de processamento para que possa escrever consultas melhores:

- Evitar usar “ * “, como em SELECT * FROM ou COUNT(*). Selecionar apenas as colunas necessárias;
- Evitar usar NOT em pesquisas;
- Minimizar o uso de WHERE;
- Evitar operações aritméticas nas cláusulas WHERE;
- Assegurar que não existe conversões implícitas de tipo de dado;
- Escrever as instruções observando a correta indentação e o uso de comentários (SOUZA, 2009; FRITCHEY, 2014).

3.4 O Ambiente de Testes

Para a realização dos testes de performance será utilizado um banco de dados, com registros gerados aleatoriamente pelo seeder do Laravel 5.6, contendo 100.000 registros. Os testes serão realizados em dois ambientes, um com sistema operacional Ubuntu e outro, Windows, ambos no mesmo notebook. As configurações da máquina são: Notebook Acer Aspire, processador i5-5200U, 4GB de memória ram, HD de 1 TB e SSD de 120 GB. Ambos os sistemas operacionais estão instalados no SSD. O primeiro ambiente possui sistema operacional Ubuntu 16.04 LTS, SQL Server 2017 e DBeaver 5.1.0. O segundo possui sistema Windows 8.1, SQL Server 2016 e DBeaver 5.1.0.

Inicialmente os testes foram planejados para serem feitos com os softwares oficiais da Microsoft, no Windows o SQL Server Management, e no Ubuntu o SQL Operations Studio. Entretanto, este último executava as consultas muito lentamente. Enquanto o SQL Server Management fazia uma consulta com tempo de um segundo, o SQL Operations Studio fazia a mesma consulta em sete minutos. O mesmo não ocorreu usando o DBeaver. Uma pesquisa no GitHub revelou que outros usuários enfrentavam problema semelhante com as ferramentas da Microsoft. Por serem ferramentas ainda muito novas para o ambiente Linux, alguns problemas ainda podem ser encontrados. Por este motivo, optou-se por usar o DBeaver nos dois sistemas operacionais para padronizar os ambientes de teste.

4 Resultados e discussão

Os testes consistiram em alterar os valores dos parâmetros selecionados, executar uma instrução que selecionasse todos os registros da tabela e registrar seu tempo de execução.

4.1 Fill Factor

Sendo 0% o valor mínimo e 100% o valor máximo que este parâmetro pode assumir, os testes foram realizados adotando valores com intervalos de 20. O valor padrão que o SQL Server adota para o Fill Factor é 0%. Entretanto, os valores 0% e 100% são iguais para o SGBD (MICROSOFT, 2017a, *online*). Para alterar este valor

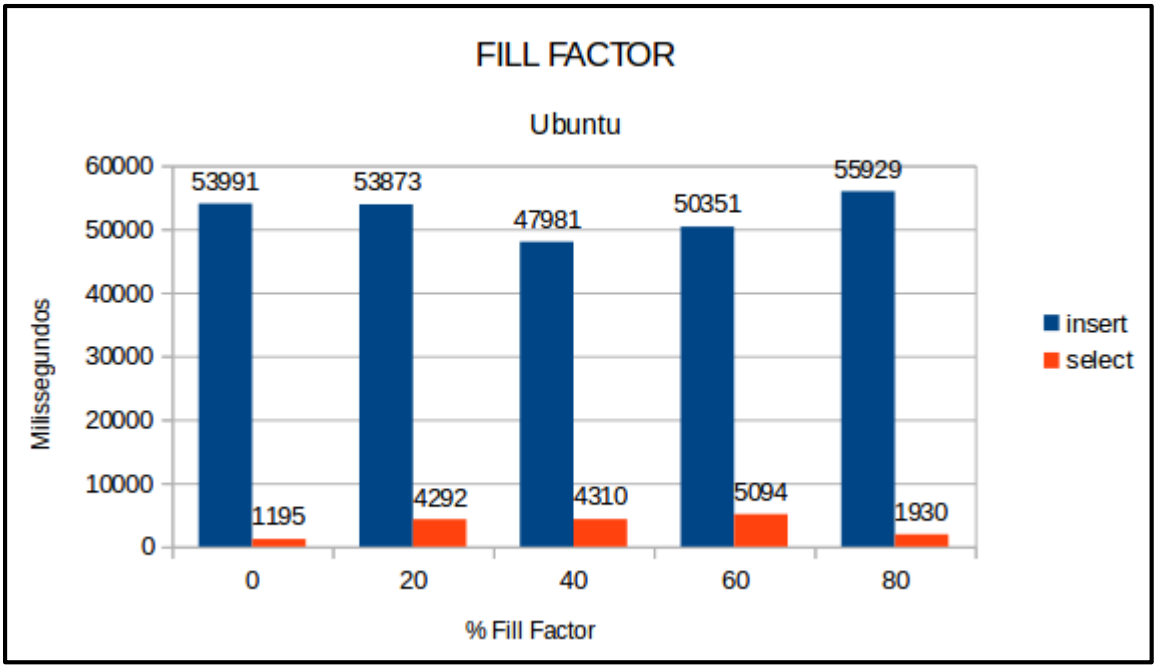
usa-se o comando apresentado na Figura 2. É necessário reiniciar o servidor para que a alteração seja efetivada.

Figura 2: Comandos SQL para alterar o parâmetro Fill Factor

```
Use tg_teste;
GO
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'fill factor', 0;
GO
RECONFIGURE;
GO
```

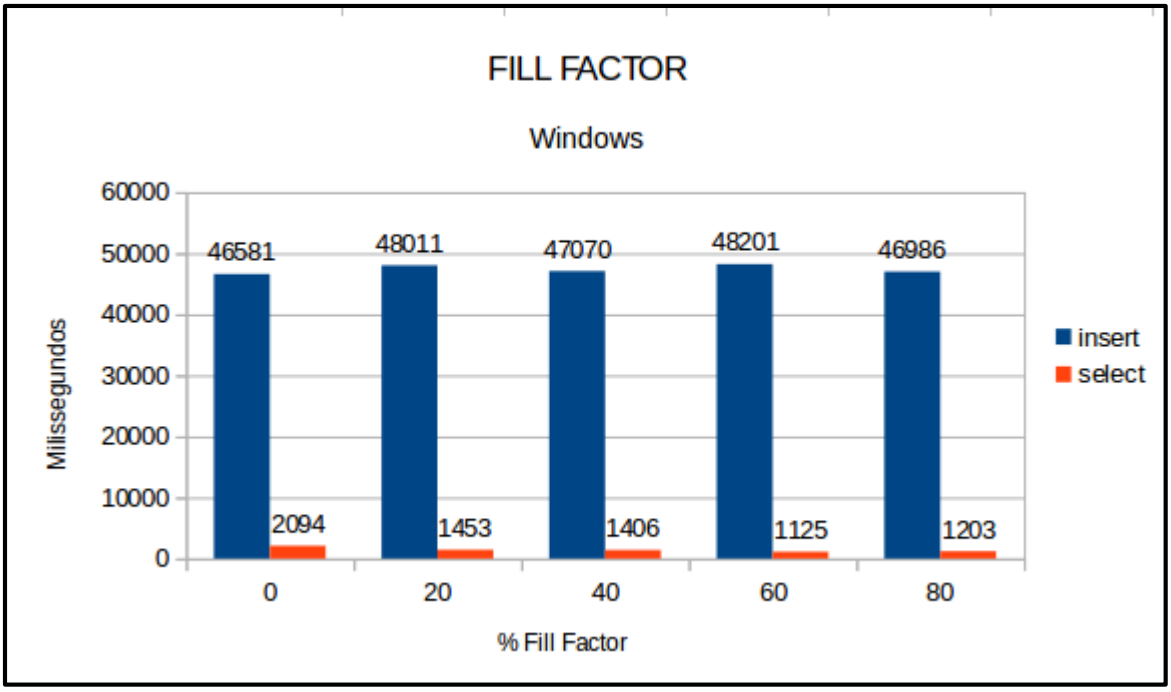
Comparando os tempos de inserção e de seleção de 100.000 registros nos dois ambientes, obteve-se os seguintes resultados, apresentados na Figura 3 e Figura 4.

Figura 3: Gráfico do resultado dos testes com o parâmetro Fill Factor no Ubuntu



Revista EduFatec: educação, tecnologia e gestão
V.1 N.1 janeiro-junho/2018

Figura 4: Gráfico do resultado dos testes com o parâmetro Fill Factor no Windows



Observa-se que, no geral, as operações de inserção e de consulta neste teste tiveram melhores resultados no ambiente Windows. A alteração das porcentagens do parâmetro Fill Factor mostrou melhor resultado para inserção de dados no ambiente Ubuntu quando adotado 40%. Para consulta, o melhor valor encontrado foi 0%. No Windows as alterações não tiveram impacto significativo. Apenas indicam resultados um pouco melhores nas operações de consulta quando adotadas porcentagens de Fill Factor acima de 40%.

4.2 Min Memory Per Query

Neste teste foram utilizados os valores mínimo (512 kb), padrão (1024 kb) e máximo (2147483647 kb) permitidos pelo SGBD para a memória mínima por instrução. Para alterar o parâmetro usa-se o comando apresentado na Figura 5.

Figura 5: Comandos SQL para alterar o parâmetro Min Memory Per Query

```
sp_configure 'min memory per query', 512;
GO
RECONFIGURE;
GO
```

Revista EduFatec: educação, tecnologia e gestão
V.1 N.1 janeiro-junho/2018

Comparando os tempos de seleção de 100.000 registros nos dois ambientes, obteve-se os seguintes resultados, apresentados na Figura 6 e na Figura 7.

Figura 6: Gráfico do resultado dos testes com o parâmetro *Min Memory per Query* no Ubuntu

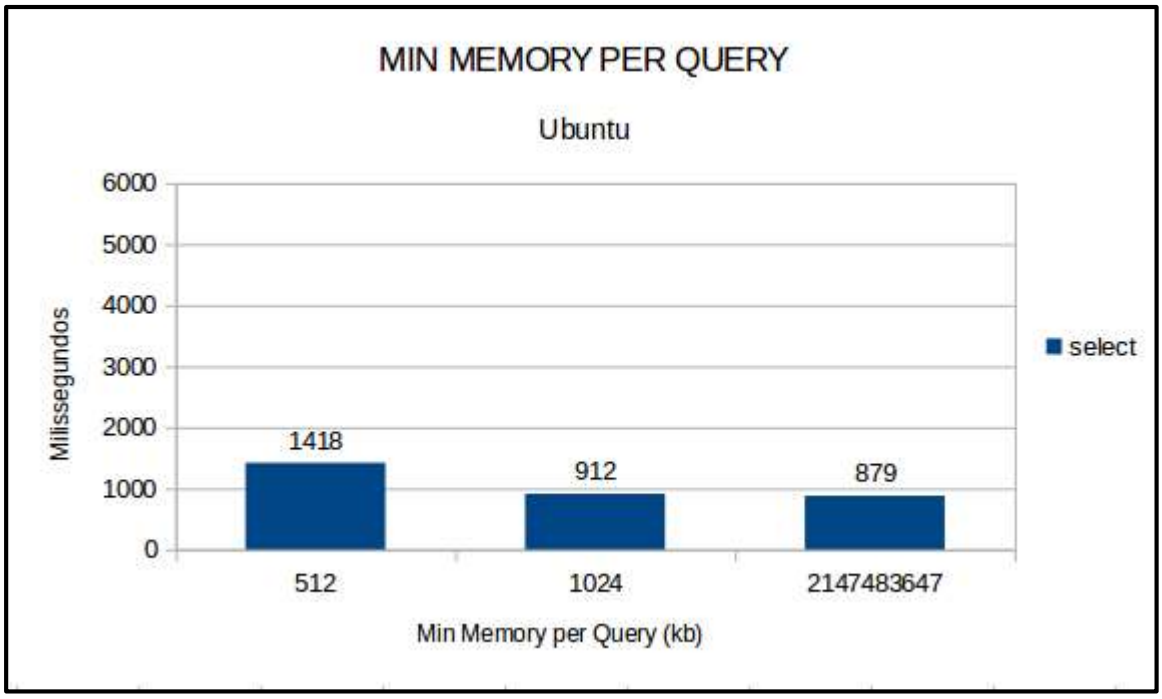
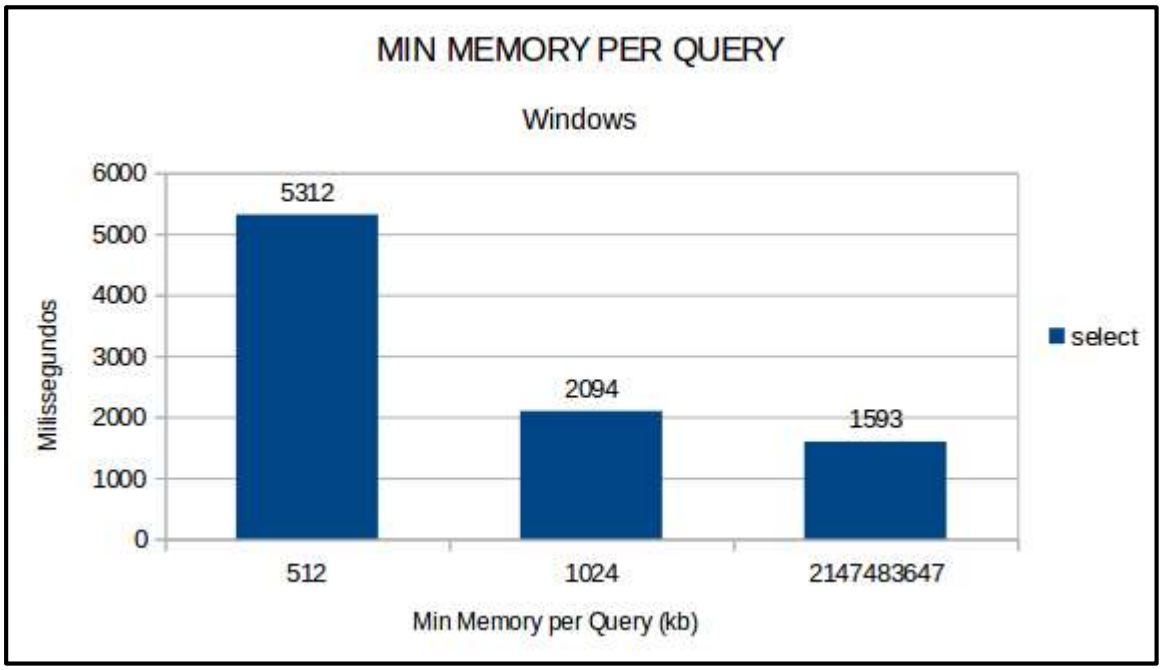


Figura 7: Gráfico do resultado dos testes com o parâmetro *Min Memory per Query* no Windows



Revista EduFatec: educação, tecnologia e gestão
V.1 N.1 janeiro-junho/2018

Percebe-se que, em geral, quanto maior o valor especificado, menor o tempo para executar a consulta. Entretanto, o ganho de desempenho obtido adotando a maior quantidade de memória possível para a consulta não foi tão significativo em relação ao valor padrão nos dois casos. Considerando que alterar este parâmetro pode levar a uma maior competição por recursos de memória, o administrador do banco de dados deve avaliar a real necessidade de se adotar valores mais altos.

4.3 Min Server Memory

Neste teste foram utilizados o valor mínimo (16 MB) e o valor máximo (2147483647 MB, valor padrão) permitidos pelo SGBD para este parâmetro. Para alterá-lo utiliza-se o comando mostrado na Figura 8.

Figura 8: Comandos SQL para alterar o parâmetro Min Server Memory

```
sp_configure 'min server memory', 16;
GO
RECONFIGURE;
GO
```

Os resultados para seleção de 100.000 registros nos dois ambientes são apresentados na Figura 9 e na Figura 10:

Figura 9: Gráfico do resultado dos testes com o parâmetro *Min Server Memory* no Ubuntu

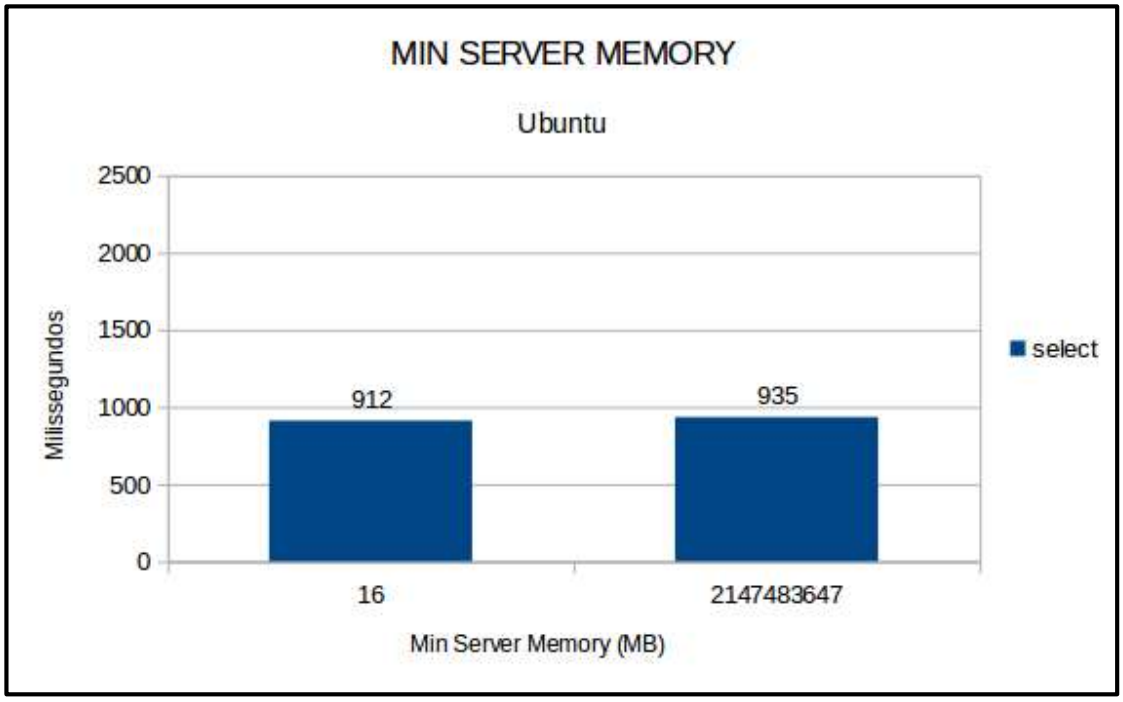
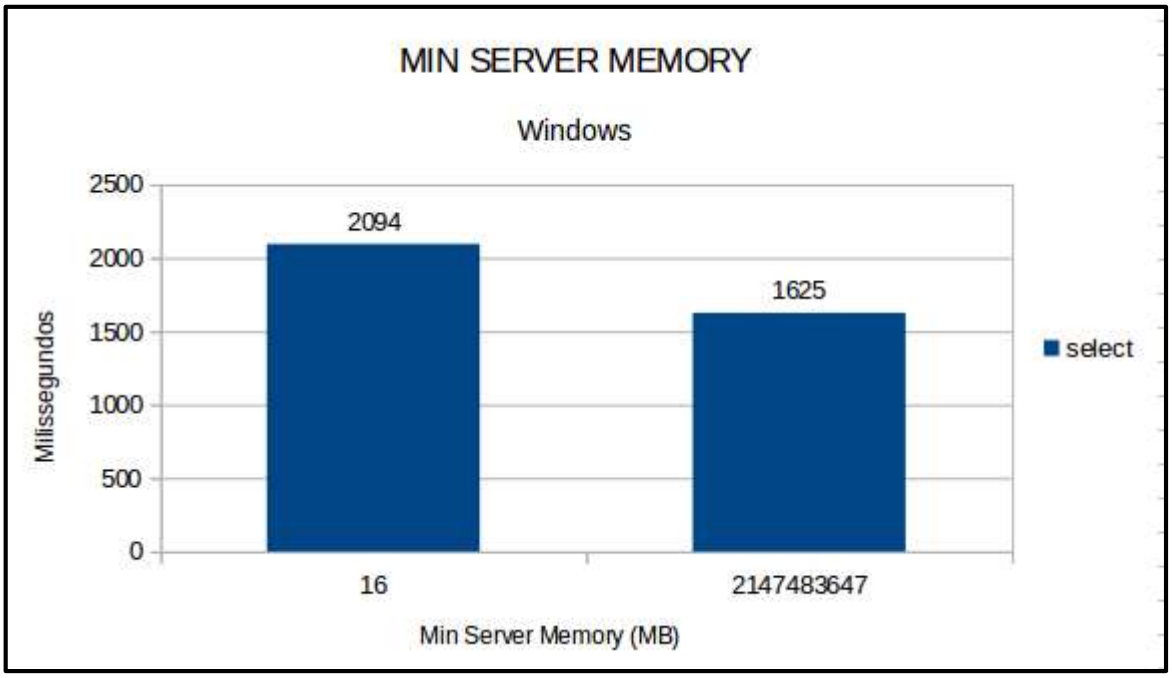


Figura 10: Gráfico do resultado dos testes com o parâmetro *Min Server Memory* no Windows



Nos dois casos é possível perceber que a alteração da memória mínima do servidor não traz benefícios consideráveis. No caso do Ubuntu, a consulta foi um pouco mais lenta utilizando o valor máximo permitido.

4.4 Max Server Memory

Segundo a documentação do SQL Server, o menor valor possível para o parâmetro *max server memory* é de 128 MB e o máximo de 2147483647 MB (valor padrão). Para alterar estes valores utiliza-se o comando apresentado na Figura 11.

Figura 11: Comandos SQL para alterar o parâmetro Max Server Memory

```
sp_configure 'max server memory', 128;
GO
RECONFIGURE;
GO
```

Entretanto, no Ubuntu não é possível ajustar a memória máxima do servidor para o seu menor valor possível. O servidor não permite a alteração. O teste foi realizado, então, com o valor 130 MB.

Revista EduFatec: educação, tecnologia e gestão
V.1 N.1 janeiro-junho/2018

No ambiente Windows a alteração foi permitida, porém o servidor não pode mais iniciar. A quantidade de memória especificada não era suficiente para que o servidor funcionasse. Os resultados são mostrados na Figura 12 e Figura 13.

Figura 12: Gráfico do resultado dos testes com o parâmetro *Max Server Memory* no Ubuntu

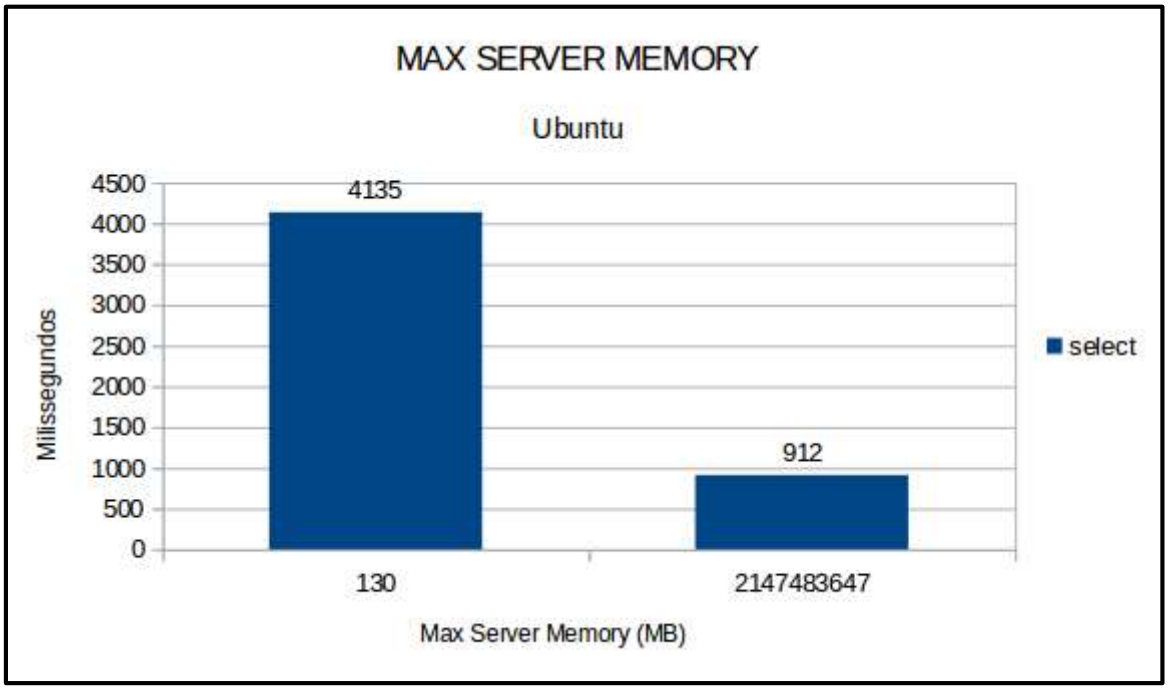
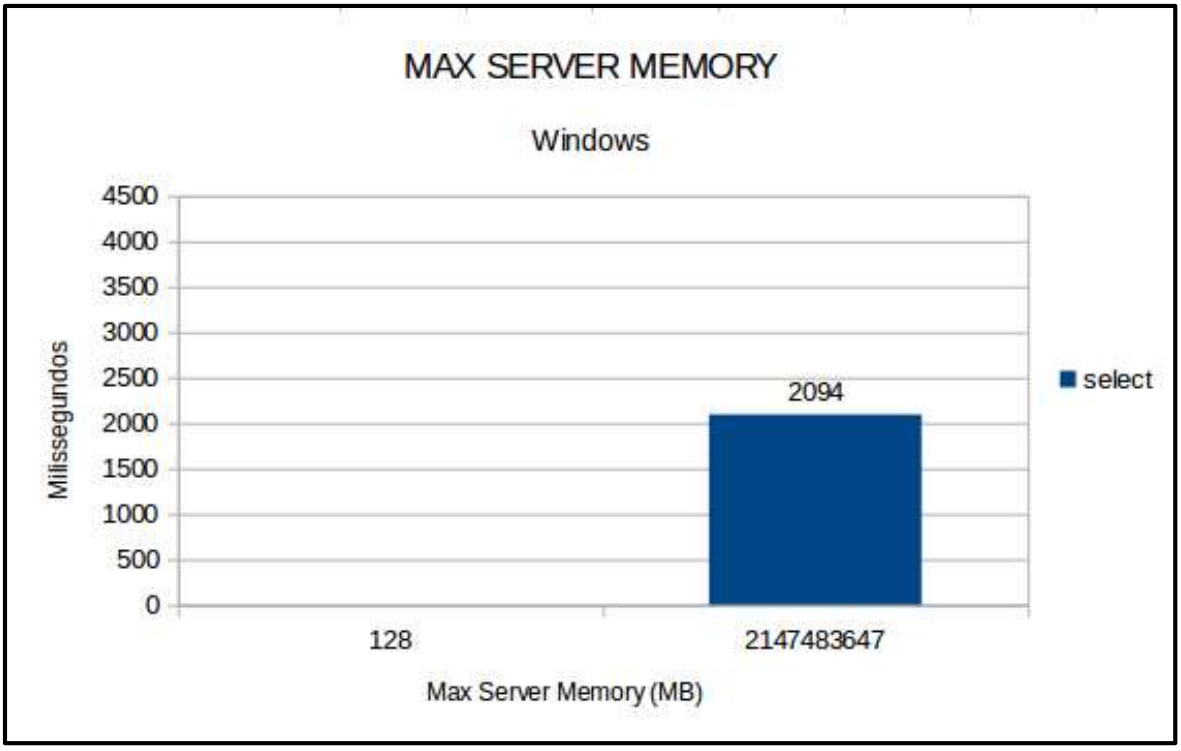


Figura 13: Gráfico do resultado dos testes com o parâmetro *Max Server Memory* no Windows



No ambiente Ubuntu foi significativa a diferença que esta alteração causou. No Windows, entretanto, não foi possível continuar o teste, pois a alteração do parâmetro não pode ser revertida. Este fato serve de alerta para o cuidado que se deve ter ao alterar parâmetros avançados do SQL Server, pois há o risco de causar danos ao servidor.

Considerações finais

O objetivo deste trabalho foi verificar através de testes a eficácia de algumas técnicas de otimização de banco de dados relacionais, além de trazer recomendações de boas práticas de criação de índices e consultas para bancos de dados.

Pode-se perceber que algumas das técnicas utilizadas trouxeram pouco resultado, enquanto outras foram mais eficientes e que estas diferenças dependem do sistema operacional utilizado. Isso se deve ao fato de que o processo de otimização de banco de dados é, em parte, um processo que exige estudo dos resultados do desempenho para diferentes configurações do ambiente e novas tentativas para aprimorar tais resultados.

O administrador de banco de dados deve conhecer bem a estrutura ou modelo de dados que precisa lidar, assim como o ambiente em que o servidor opera. Deve investigar as causas de possíveis problemas de desempenho ajustando configurações e testando a eficácia delas, até que tenha sanado ou minimizado o problema.

Agradecimentos

Agradeço à minha mãe e à minha irmã por sempre me apoiarem. Agradeço ao meu orientador, Claudio Eduardo Paiva pela paciência e pelo auxílio. Agradeço imensamente ao meu psicólogo, Gabriel Ferraz, sem seu apoio eu não teria conseguido chegar até aqui. Agradeço ao pessoal da InDB por tudo que aprendi nos últimos meses. Muito obrigada a todos!

Referências

BITTENCOURT, Rogério Gonçalves. **Aspectos Básicos de Banco de Dados**. Florianópolis, 2004. Disponível em:

<https://www.marilia.unesp.br/Home/Instituicao/Docentes/EdbertoFerneda/BD%20%20Aspectos%20Basicos.pdf> Acesso em: 22 jun. 2017.

BOSCARIOLI, Clodis; BEZERRA, Anderson; BENEDICTO, Marcos de; DELMIRO, Gilliard. **Uma reflexão sobre Banco de Dados Orientados a Objetos**. 2006. 12 f. Universidade Estadual do Oeste do Paraná, Toledo, 2006. Disponível em: <http://conged.deinfo.uepg.br/artigo4.pdf> Acesso em: 23 jul. 2017.

DATE, C. J. **Introdução a Sistemas de Banco de Dados**. Editora Campus Ltda, 1989.

DATE, C J. **Introdução a sistemas de banco de dados**. 8 ed. Rio de Janeiro: Elsevier, 2003.

DBEAVER: Free Universal Database Tool . Version 5.1.0. [S.l.: s.n.], 2018. Disponível em: <https://dbeaver.io/>. Acesso em: 01 jun. 2018.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistema de banco de dados**. 6 ed. São Paulo: Pearson Addison Wesley, 2011.

FRITCHEY, Grant; DAM, Sajal. **SQL Server Query Performance Tuning**. Apress, 2014.

HEUSER, Carlos Alberto. **Projeto de banco de dados: Volume 4 da Série Livros didáticos informática UFRGS**. Bookman Editora, 2009.

JORGENSEN, Adam et al. **Microsoft SQL Server 2012 Bible**. John Wiley & Sons, 2012.

LÓPEZ, Leandro Castoldi; DILL, Sérgio. Sintonia em Banco de Dados sob Sistemas de Arquivos livres. **Anais SULCOMP**, v. 1, 2012.

MACEDO, Bruno P. et al. Uma abordagem prática sobre otimização de Banco de Dados utilizando o gerenciamento automático de memória no Sistema Gerenciador de Banco de Dados Oracle. **Caderno de Estudos Tecnológicos**, v. 1, n. 1, 2013.

MACHADO, Carlos Augusto. Estudo sobre a otimização de desempenho em banco de dados PostgreSQL. **Repositório de Relatórios-Sistemas de Informação**, n. 2, 2014.

MICROSOFT. **Especificar fator de preenchimento para um índice**, 2017. Disponível em: <https://docs.microsoft.com/pt-br/sql/relational-databases/indexes/specify-fill-factor-for-an-index> Acesso em 26.out 2017

_____. **Índices clusterizados e não clusterizados descritos**, 2017. Disponível em: <https://docs.microsoft.com/pt-br/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described> Acesso em 12.ago 2017

_____. **Integridade de dados**, 2012. Disponível em <https://technet.microsoft.com/pt-br/library/ms184276(v=sql.105).aspx> Acesso em 01.Maio 2017

_____. **Opções de configuração do servidor (SQL Server)**, 2017. Disponível em: <https://docs.microsoft.com/pt-br/sql/database-engine/configure-windows/server-configuration-options-sql-server> Acesso em 12.out 2017

_____. **Opção min memory per query**, 2012. Disponível em: <https://technet.microsoft.com/pt-br/library/ms181047(v=sql.105).aspx>. Acesso em: 15 abr. 2018.

_____. **Opções Server Memory de configuração do servidor**, 2017. Disponível em: <https://docs.microsoft.com/pt-br/sql/database-engine/configure-windows/server-memory-server-configuration-options?view=sql-server-2017>. Acesso em: 15 abr. 2018

PANSONATO, Ricardo Leonardo; GROSSO, Pedro Roberto. TUNING DE INSTRUÇÕES SQL APLICADO EM UMA EMPRESA DE DESENVOLVIMENTO DE SOFTWARE. **Revista Network Technologies Faculdades Network–Revista da Faculdade de Sistema de Informação**, Nova Odessa, v. 6, n. 1, p. 7-18, 2012. Disponível em: <http://www.nwk.edu.br/intro/wp-content/uploads/2014/05/BSI-2012-Revista-Technologies.pdf>. Acesso em: 23 jul. 2017.

SALES, Rosicléia Bezerra; GARCIA, Thuany Cristina. **Estudo e Análise de Parâmetros de Configuração e Aplicação de Técnica de Tuning em SGBD** – São José do Rio Preto: FATEC, 2013.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistema de banco de dados**. Elsevier, 2006.

SOUZA, Ana Paula dos Santos; CAMPOS, Bruno Fidelis; DIAS, Carla Glênia Guedes; ALVES, Michel Batista; VIEIRA, Carlos Eduardo Costa; CARELLI, Flávio Campos. e Luiz Fabiano Costa de Sá. Tuning em Banco de Dados. **Cadernos UniFOA.** Volta Redonda, ano IV, n. 10, agosto. 2009. Disponível em: <http://www.unifoa.edu.br/portal_pesq/caderno/edicao/10/19.pdf>

TAKAI, Osvaldo Kotaro; ITALIANO, Isabel Cristina; FERREIRA, João Eduardo. Introdução a banco de dados. **Departamento de Ciências da Computação. Instituto de Matemática e Estatística. Universidade de São Paulo. São Paulo**, 2005.

TOTH, Renato Molina. **Abordagem NoSQL-Uma real Alternativa**. Sorocaba, São Paulo, Brasil: Abril, v. 13, 2011. Disponível em: <http://www.dcomp.sor.ufscar.br/verdi/topicosCloud/nosqlartigo.pdf> Acesso em 24 jul. 2017.

WINAND, Markus. **SQL Performance Explained:** Everything Developers Need to Know about SQL Performance. Markus Winand, 2012.