OTIMIZAÇÃO DE CONSULTAS EM UM BANCO DE DADOS ORACLE *(Tuning)*

TEIXEIRA, Evandro

Orientador: TRINDADE, Bruno G.

RESUMO: A utilização de sistemas de informação vem aumentando dia após dia nas mais variadas atividades. O emprego de um sistema que possibilite gerir com precisão as atividades de uma organização, facilitando e também possibilitando um planejamento da mesma dando suporte à alta diretoria para uma tomada de decisão mais precisa, têm se tornado quase que prioridade dentro de uma empresa, uma vez que temos diversas dessas ferramentas a nossa disposição. Todavia, vemos um crescente volume de dados sendo produzido, o que torna o desempenho desses sistemas já não muito satisfatório, ou seja, o sistema continua eficaz (cumpre com o objetivo proposto inicialmente), porém não é eficiente (cumpre com o objetivo inicial, mas não com o custo de tempo desejado). O referido trabalho aborda o estudo e aplicação de técnicas de tuning visando a otimização de consultas SQL, também chamado de sintonização. A motivação desta pesquisa deu-se pela pertinente necessidade de melhora no desempenho do banco de dados do Grupo Sudati Ltda., que de acordo com relatos constantes dos usuários, entendeu-se que haviam gargalos em determinadas rotinas do sistema e que deveriam receber uma atenção especial.

PALAVRAS-CHAVE: Otimização de Banco de Dados. Banco de Dados. Consultas SQL.

INTRODUÇÃO

O emprego de sistemas de informação vem aumentando dia após dia nas mais variadas atividades. O uso de um sistema que possibilite gerir com precisão as atividades de uma organização, que facilite e automatize a transmissão de informações ao governo sejam elas informações fiscais, tributárias, previdenciárias, entre outras e que também possibilite um planejamento da organização dando suporte à alta diretoria para uma tomada de decisão mais precisa, têm se tornado quase que prioridade dentro de uma empresa, uma vez que temos diversas dessas ferramentas a nossa disposição. Em contrapartida, vemos um grande volume de dados sendo produzido, o que torna o desempenho desses sistemas já não muito satisfatório, ou seja, o sistema continua eficaz (cumpre com o objetivo proposto inicialmente), porém não é eficiente (cumpre com o objetivo inicial, mas não com o custo de tempo desejado).

De acordo com LEITE A.K.L, et al., na etapa de desenvolvimento do Banco de Dados pode não possível avaliar o seu desempenho (embora hajam

controvérsias), porém, após o banco estar num ambiente de produção, é possível identificar as falhas que ainda não haviam sido consideradas e aplicar as técnicas para otimização.

Conforme Couto, E. (2006), o objetivo da otimização é exibir a informação requisitada pelo usuário com o menor custo e tempo. Conforme LEITE A.K.L, et al., nesta fase de aperfeiçoamento, pode se aplicar o *Tuning*, que irá otimizar o acesso aos dados, fazendo com que o tempo de resposta fornecido pela aplicação seja cada vez menor, tornando o sistema mais eficiente num todo.

Para se ter um melhor entendimento do que é o tuning, sua tradução literal nada mais é que: afinação, sintonização, refinamento, ajuste.

Segundo Navathe & Elmasri (2005), um projeto e implementação de Banco de Dados se dá nos seguintes passos:

1. Levantamento e análise de requisitos.
2. Projeto conceitual do banco de dados.
3. Escolha de um SGBD.
4- Mapeamento do modelo de dados (também chamado projeto lógico de banco de dados).
5. Projeto físico do banco de dados.
6. Implementação e sintonização (*tuning*) do sistema de banco de dados.

No referente trabalho, será abordado apenas a sintonização (*tuning*).

Conforme Navathe & Elmasri (2005), nesta fase, o banco de dados e os programas de aplicação são implementados, testados e eventualmente utilizados em produção. São testadas também várias transações e aplicações individualmente e, posteriormente em conjunto. Geralmente, esta fase revela necessidades de alterações no projeto físico, na indexação, na reorganização e na alocação de dados, atividade esta que é denominada de sintonização (afinação, *tuning)* do banco de dados.

Sintonizar é uma atividade contínua, parte da manutenção do sistema que perdura durante todo o ciclo de vida de um banco de dados, contanto que o banco de dados e as aplicações continuem evoluindo ou à medida que forem surgindo problemas de desempenho.

A motivação desta pesquisa deu-se a partir da necessidade de melhora no desempenho do banco de dados do Grupo Sudati Ltda., que conforme relatos de usuários percebeu-se que havia lentidão em determinadas rotinas. Motivações em

paralelo deram-se também pela afinidade para com a área de banco de dados, e, também pelo fato do objeto da pesquisa ser pertinente, visto que é um assunto que desperta um grande interesse entre profissionais da área de BD e também estar relacionado a um dos tópicos para submissão de artigos do SBBD (Simpósio Brasileiro de Banco de Dados) sendo ele, a avaliação de Desempenho e Benchmarking de Bancos de Dados.

1 FUNDAMENTAÇÃO TEÓRICA

1.1 SISTEMA GERENCIADOR DE BANCO DE DADOS

Conforme Silberschatz *et. al* (2006), um SGBD (Sistema Gerenciador de Banco de Dados) é uma coleção de dados inter-relacionados juntamente com conjunto de programas para acesso desses dados. Essa coleção é comumente chamada de Banco de Dados e contém informações pertinentes à uma determinada organização. O principal objetivo de um SGBD é possibilitar a recuperação da informação do Banco de Dados que seja tanto conveniente quanto eficiente.

Silberschatz (2006) *et. al* diz ainda que os SGBD's são projetados para trabalhar com grandes blocos de informação. Gerenciar estes dados, demanda definir estruturas para armazenamento e mecanismos para manipulação dessas informações. Além disso, é crucial a garantia da segurança das informações ali armazenadas, embora ocorram falhas na aplicação ou ataques visando o roubo das mesmas.

Uma outra análise sobre conceitos de Banco de Dados e SGBD do ponto de vista de Navathe & Elmasri (2005), é que um banco de dados é uma coleção de dados relacionados. Os dados são informações que podem ser gravadas e que possuem um significado implícito.

Essas informações são uma coleção de dados com um significado implícito, conseqüentemente, um banco de dados. Os mesmos autores, afirmam ainda que um banco de dados pode ser de qualquer tamanho e de complexidade variável.

Sob a perspectiva de Navathe & Elmasri (2005), um sistema gerenciador de banco de dados (SGBD) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. O SGBD é, portanto, um sistema de software de

propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações. A definição de um banco de dados implica especificar os tipos de dados, as estruturas e as restrições para os dados a serem armazenados em um banco de dados.

1.1.1 LINGUAGENS DE BANCO DE DADOS (DDL, DML, DQL)

Um banco de dados necessita de linguagens tanto para sua definição, quanto para sua manipulação, tendo em vista que alguns autores consideram a consulta um outro tipo de linguagem.

Segundo Navathe & Elmasri (2005), a linguagem DDL *(Data Definition Language) é* usada pelo Database Administrator (DBA) e pelos projetistas do banco de dados para definir ambos os esquemas. O SGBD terá um compilador DDL cuja função é processar os comandos DDL a fim de identificar os construtores e para armazenar a descrição do esquema no catálogo do SGBD. Os autores ainda sugerem que quando os esquemas do banco de dados estiverem compilados e o banco de dados populado com os dados, os usuários devem ter alguns meios para manipular esse banco. As manipulações típicas são a recuperação, inserção, remoção e modificação dos dados. O SGBD fornece uma série de operações, ou uma linguagem chamada DML *(Data Manipulation Language).*

Segundo Salish, A. (2010), os comandos DQL são usados para exibir registros de tabelas para o usuário/DBA. Os dados atuais na forma de tabelas são armazenados no disco rígido do Oracle Database Server e os comandos DQL permitem a visualização das informações / registros armazenados no banco de dados no monitor por meio do comando SELECT.

1.2 TABLESPACES

De acordo com Legatti (2011)*, o* Oracle armazena dados logicamente em tablespaces e fisicamente em arquivos de dados (*datafiles*). Apesar dos arquivos de dados e os tablespaces se encontrarem demasiadamente "inter-relacionados", os mesmos dispõem de diferenças importantes conforme o autor:

- Um banco de dados Oracle constitui-se em uma ou mais unidades de armazenamento lógicas chamada de tablespaces, que armazenam coletivamente todos os dados do banco de dados.
- Cada tablespace em um banco de dados Oracle consiste em um ou mais arquivos denominados arquivos de dados (*datafiles*), que são estruturas físicas compatíveis com o sistema operacional no qual o Oracle é executado.
- Os dados de um banco de dados são armazenados coletivamente nos arquivos de dados que constituem cada tablespace do banco de dados.

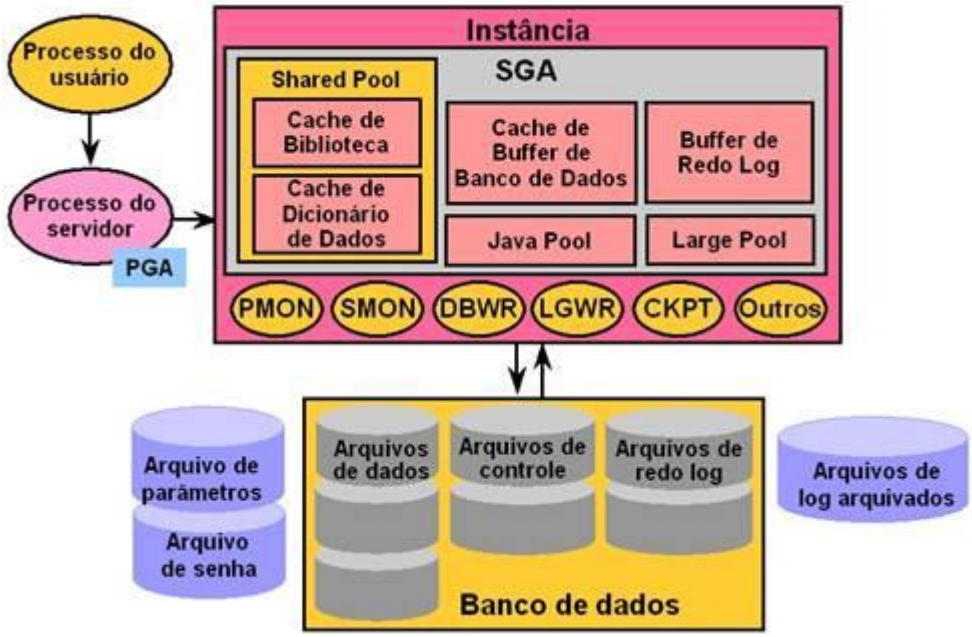

Figura 1. Arquitetura Banco de Dados Oracle. (Rezende, R. 2007).

O Oracle agrupa esses conjuntos de arquivos de dados sob a proteção de um objeto de banco de dados chamado tablespace. Antes mesmo de inserir dados em um banco de dados Oracle, primeiro é necessário a criação de um tablespace e posteriormente uma tabela dentro desse tablespace que conterá os dados, segundo Legatti (2011). Pode-se observar também, que na criação de um banco de dados utilizando o **DBCA**, o Oracle por padrão sempre cria um tablespace de dados chamado USERS. Ao criar uma tabela, se faz necessário realizar a inclusão de todas as informações pertinentes ao tipo de dados que se deseja manter.

1.3 ÁLGEBRA RELACIONAL

Conforme Navathe & Elmasri (2005), um modelo de dados abarca um conjunto de operações para manipulação do banco de dados, além claro, dos conceitos de modelo de dados para a definição das restrições e estrutura mesmo. O conjunto básico de operações para o modelo relacional é a **álgebra relacional.**

Os autores reforçam ainda, que estas operações possibilitam à um usuário especificar as solicitações básicas de recuperação. O resultado dessa recuperação consistirá numa nova relação, que pode ter sido formada a partir de uma ou mais relações. As operações de álgebra originam, assim, novas relações, que podem ser manipuladas adiante utilizando as operações de mesma álgebra.

A álgebra relacional é de suma importância, pois provê um fundamento formal para operações do modelo relacional, servindo de base para implementar e **otimizar** as consultas em sistemas de gerenciadores de banco de dados relacional (SGBDRs). Por fim, alguns de seus conceitos são incorporados na linguagem de consulta-padrão SQL para os SGBDRs.

1.3.1 OPERAÇÕES RELACIONAIS UNÁRIAS: SELEÇÃO (SELECT)

Segundo os autores Navathe & Elmasri (2005), a operação *SELECT* é utilizada para recuperar um subconjunto de tuplas de uma relação que satisfaça uma condição de seleção. Uma operação pode ser considerada *SELECT* quando o filtro mantém apenas as tuplas que satisfaçam uma condição de qualificação. Ainda conforme os autores Navathe & Elmasri (2005), a operação *SELECT* pode também ser visualizada como um particionamento horizontal da relação em dois conjuntos de tuplas — aquelas tuplas que satisfazem a condição e são selecionadas, e as tuplas que não satisfazem a condição e são descartadas.

1.3.2 OPERAÇÕES UNIÃO (UNION), INTERSEÇÃO (INTERSECTION) E SUBTRAÇÃO (MINUS)

Conforme Navathe & Elmasri (2005), inúmeros conjuntos de operações teóricas são utilizados para união dos elementos de dois conjuntos de diferentes maneiras, incluindo UNIÃO, INTERSEÇÃO e DIFERENÇA DE CONJUNTO (também chamada de SUBTRAÇÃO). Os autores ainda sugerem que as operações citadas anteriormente são operações binárias, ou seja, cada uma delas é aplicada à dois conjuntos (de tuplas). Quando estas operações são convertidas à um banco de dados relacional, as duas relações nas quais qualquer uma dessas três operações for aplicada devem ter o mesmo tipo de tuplas**;** essa condição comumente tem sido chamada de compatibilidade de união.

1.3.2 OPERAÇÕES RELACIONAIS BINÁRIAS: (JUNÇÃO JOIN) E DIVISÃO

Conforme Navathe & Elmasri (2005), a operação JUNÇÃO, indicada por *n*, é utilizada para combinar as tuplas relacionadas em duas relações dentro de uma única tupla. Essa operação é de grande importância para qualquer banco de dados relacional com mais de uma relação, porque possibilita o processamento dos relacionamentos entre as relações.

Ainda conforme os autores, na JUNÇÃO, apenas as combinações de tuplas que satisfizerem a condição de junção aparecerão no resultado, enquanto que no produto cartesiano todas as combinações de tuplas serão compreendidas no resultado.

1.4 DEPURAÇÃO (Debug)

Segundo Macêdo (2012), um depurador é uma ferramenta para testar outros programas e fazer sua depuração, com o objetivo de encontrar anormalidades e falhas do programa. O autor ainda cita que para determinados tipos de problema existem ferramentas de análise do código fonte, que buscam por erros específicos no código, o que depende da linguagem de programação em uso.

1.5 TUNING DE BANCO DE DADOS

Conforme Ikematu (2009), a tradução literal de ‘tuning’ é uma sintonia ou ajuste de algo com o objetivo de uma melhora no seu desempenho. Um SGBD é um

produto de software sofisticado e flexível, de forma que permita vários ajustes. Ajustes estes que podem vir a afetar a performance do banco de dados de forma que obtenha um bom desempenho ou não. Ainda segundo Ikematu (2009), por ‘tuning’ da base de dados, podemos entender como uma customização do sistema feita sob medida para que a performance atenda melhor as suas necessidades.

1.5.1 TIPOS DE TUNING

O tuning pode ser realizado em diversos pontos do seu ambiente, como por exemplo uma melhora na infraestrutura de rede, uma melhora nos equipamentos de hardware. Ou, como estaremos abordando no referido trabalho, otimização de consultas SQL, entre diversas outras.

Zorzi, M.T. (2015) diz que podem ser aplicadas técnicas de tuning por exemplo no Sistema Operacional, a fim de eliminar gargalos e uma melhor utilização dos componentes de hardware. No Banco de Dados poderíamos dividir entre ajustes na arquitetura do banco de dados, otimizações de instruções e objetos SQL, bem como ajustes no projeto do Banco de Dados.

1.5.2 UMA VISÃO GERAL DA SINTONIZAÇÃO DE BANCO DE DADOS EM SISTEMAS RELACIONAIS

Após o projeto e implementação de um Banco de Dados, ou seja, a partir do uso real das aplicações, das transações, das consultas e das visões são revelados fatores e áreas de problemas que podem não ter sido identificados durante o projeto físico inicial, afirmam Navathe & Elmasri (2005). A utilização dos recursos, bem como o processamento interno do SGBD (otimização de consultas por exemplo) podem ser monitorados à fim de identificar gargalos, tais como a disputa pelos mesmos dados ou dispositivos. Com isso, os volumes de atividades e os tamanhos dos dados podem ser mais bem estimados. Portanto, é necessário monitorar e revisar o projeto físico do banco de dados constantemente. Os objetivos da sintonização são os seguintes:

• Fazer com que as aplicações sejam executadas mais rapidamente.
• Diminuir o tempo de resposta de consultas/transações.
• Melhorar o desempenho geral das transações.

1.5.3 SINTONIZAÇÃO DE ÍNDICES

De acordo com Navathe & Elmasri (2005), a escolha inicial de índices pode precisar de uma revisão pelos seguintes motivos:

• As consultas podem ter um custo de tempo alto para ser executadas por conta da ausência de um índice.

• Certos índices podem não ser utilizados.

• Certos índices podem estar ocasionando algum tipo de sobrecarga porque são baseados em um atributo que sofre sucessivas alterações.

Os autores ainda citam que, a maioria dos SGBDs possuem um comando ou um meio de rastreamento *(trace facility)* que pode ser usado pelo DBA para solicitar que o sistema mostre como uma consulta foi executada, quais operações foram realizadas e em qual ordem e quais estruturas de acesso secundário foram utilizadas. Através da análise desses planos de execução, é possível diagnosticar as causas dos problemas citados anteriormente. Alguns índices podem ser excluídos e alguns novos índices podem ser incluídos com base na análise de sintonização.

Segundo Navathe & Elmasri (2005, p.388):

> O objetivo da sintonização é avaliar dinamicamente os requisitos, os quais às vezes variam sazonalmente ou durante diferentes períodos do mês ou da semana, e reorganizar os índices para proporcionar melhor desempenho geral.

Os autores afirmam ainda que, a exclusão e a criação de novos índices são uma sobrecarga que pode ser justificada em consequência das melhorias de desempenho. A atualização de uma tabela geralmente é suspensa enquanto um índice estiver sendo excluído ou criado; deve-se contabilizar essa perda de serviço. Além da exclusão ou da criação de índices e a mudança a partir de um índice que não é *clustering* para um índice *clustering* e vice-versa, a **reconstrução** do índice pode melhorar o desempenho. A grande maioria dos SGBDRs usam árvores-B em seus índices. Caso haja muitas exclusões na chave do índice, páginas do índice poderão ter espaço desperdiçado, o que pode ser recuperado ao longo da operação de reconstrução. Em contrapartida, inclusões em excesso podem ocasionar *overflow* em um índice *clustering* afetando seu desempenho. A reconstrução de um índice *clustering* corresponde a reorganizar a tabela ordenada segundo aquela chave.

1.5.4 SINTONIZAÇÃO DE CONSULTAS

Percebe-se como o desempenho de consultas é dependente da seleção apropriada de índices e como eles precisam ser sintonizados mediante análise das consultas, realizadas através do plano de execução da consulta. Navathe & Elmasri (2005) afirmam que há principalmente duas indicações que sugerem que a sintonização da consulta pode ser necessária:

1. Uma consulta resulta em muitos acessos a disco (por exemplo, uma consulta de correspondência exata que percorre uma tabela inteira);
2. Um plano de consulta mostra que os índices relevantes não estão sendo utilizados.

Segundo Navathe & Elmasri (2005, p.389), existem algumas situações típicas que indicam a necessidade da sintonização de consultas:

> 1. Muitos otimizadores de consulta não usam índices na presença de expressões aritméticas (tais como SALÁRIO/365 > 10,50), de comparações numéricas de atributos de diferentes tamanhos e níveis de precisão (tais como QTDDE_A = QTDDE_B, onde QTDDE_A é do tipo INTEGER e QTDDE_B é do tipo SMALLINTEGER), comparações COm NULL (tais COmO DATANASC is NULL) , e comparações com *substrings* (tais como UNOME LIKE "%MANN").
>
> 2. Freqüentemente os índices não são usados em consultas aninhadas usando IN; por exemplo, a consulta **SELECT** SSN **FROM** EMPREGADO
> **WHERE** NRD **IN (SELECT** NUMEROD **FROM** DEPARTAMENTO **WHERE** GERSSN = '333445555');
> pode não usar o índice para NRD em EMPREGADO, enquanto o uso de NRD = NUMEROD na cláusula WHERE com uma consulta em um único bloco pode gerar o índice a ser usado.
>
> 3. Algumas cláusulas DISTINCT podem ser redundantes e podem ser evitadas sem a modificação do resultado. Um DISTINCT freqüentemente causa uma operação de ordenação e deve ser evitado sempre que possível.
>
> 4. O uso desnecessário de tabelas de resultado temporário pode ser evitado por meio da aglutinação de múltiplas consultas em uma única consulta, *a menos* que a relação temporária seja necessária para algum processamento intermediário.

5. Em algumas situações que envolvem o uso de consultas correlatas, os temporários são úteis. Considere a consulta:

SELECT SSN
FROM EMPREGADO E
WHERE SALÁRIO = **SELECT MAX** (SALÁRIO)
FROM EMPREGADO **AS** G
WHERE G.NRD = E.NRD;

Ela possui o perigo potencial de pesquisar toda a tabela EMPREGADO G interna para *cada* tupla da tabela EMPREGADO E externa.

Para torná-la mais eficiente, ela pode ser quebrada em duas consultas, na qual a primeira apenas calcula o máximo salário em cada departamento conforme segue:

SELECT MAX (SALÁRIO) **AS** MAI0RSALARI0, NRD **INTO** TEMP **FROM** EMPREGADO **GROUP BY** NRD;
SELECT SSN
FROM EMPREGADO, TEMP
WHERE SALÁRIO = MAIORSALARIO **AND** EMPREGADO.NRD = TEMP.NRD;

6. Se múltiplas opções de condições de junção são possíveis, escolha uma que use um índice *clustering* e evite as qu contenham comparações de cadeias de caracteres. Por exemplo, supondo que o atributo NOME é uma chave candidat em EMPREGADO e ALUNO, é melhor usar EMPREGADO.SSN = ALUNO.SSN como uma condição de junção em vez de EMPREGADO.NOME : ALUNO.NOME se SSN possuir um índice *clustering* em uma ou ambas as tabelas.

7. Uma idiossincrasia dos otimizadores de consulta é que a ordem das tabelas na cláusula FR0M pode afetar o processa mento de junções. Se esse for o caso, pode-se precisar alterar essa ordem de forma que a menor das duas relações sej varrida e a maior relação seja usada com um índice adequado.

8. Alguns otimizadores de consulta têm um desempenho pior em consultas aninhadas em comparação com suas consultas não aninhadas correspondentes. Há quatro tipos de consultas aninhadas:

• Subconsultas não correlatas com agregações na consulta interna.
• Subconsultas não correlatas sem agregações.
• Subconsultas correlatas com agregações na consulta interna.
• Subconsultas correlatas sem agregações.

Dos quatro tipos acima, o primeiro geralmente não apresenta problemas, uma vez que a maioria dos otimizadores d consulta avalia a consulta interna uma vez. Entretanto, para uma consulta do segundo tipo, tal como o exemplo n< item 2 acima, a maioria dos otimizadores

pode não utilizar um índice para NRD em EMPREGADO. OS mesmos otimizadore podem fazê-lo se a consulta for escrita como uma consulta não aninhada. A transformação de subconsultas correia tas pode envolver o estabelecimento de tabelas temporárias.

9. Finalmente, muitas aplicações são baseadas em visões que definem os dados de interesse para aquelas aplicações. Às vezes, essas visões se tornam excessivas, porque uma consulta pode ser aplicada diretamente sobre uma tabela base em vez de ser realizada sobre uma visão que é definida por uma junção.

2 METODOLOGIA

Para o projeto, inicialmente será realizada uma pesquisa bibliográfica a fim de aprimorar as principais técnicas de *tuning* de banco de dados, bem como suas melhores práticas e métodos.

Após isso, serão identificadas as principais rotinas com lentidão e possíveis gargalos através de uma pesquisa com o usuário por meio de formulário.

Identificadas as rotinas, as mesmas serão executadas com a ferramenta de debug do ERP a fim de extrair os comandos *sql* e verificar seu tempo de execução. Com isso, serão aplicadas técnicas de *tuning* nas queries para comparar a melhora no tempo de execução.

O ERP que será utilizado será o TOTVS Logix, bem como sua ferramenta de BI correspondente, fornecida pela empresa desenvolvedora de software TOTVS. O Sistema Gerenciador de Banco de Dados utilizado será o Oracle *11g Enterprise Edition Release* 11.2.0.1.0.

Para a administração e manipulação do Banco de Dados, bem como a aplicação das técnicas de *tuning*, será utilizada a ferramenta PL/SQL *Developer* versão 8.0.0.1480.

Para demonstrar os resultados obtidos, será utilizada a ferramenta CamStudio, gravando a execução dos comandos sql exibindo o tempo de execução, pois assim evitamos qualquer tipo de desconfiança com relação aos dados apresentados (como manipulação de imagem alterando o tempo de execução por exemplo) e aplicamos as técnicas no próprio ambiente de produção, sem qualquer

tipo de interferência interna e externa coletando informações mais precisas com relação à melhora no desempenho.

3 ANÁLISE DOS RESULTADOS

3.1 CONFIGURAÇÃO DO AMBIENTE

O servidor de banco de dados utilizado consiste numa arquitetura x86_64, possui um processador Intel ® Xeon ® CPU E5-2620 v3 com 2.40 GHz e memória RAM com um total de 62Gb e conta com o sistema operacional Linux. Devido a grande quantidade de partições, segue informações pertinentes ao armazenamento do servidor de banco de dados conforme figura abaixo:

```
Disk /dev/sda: 805.3 GB, 805306368000 bytes, 1572864000 sectors
/dev/sda1   *        2048     1026047      512000   83  Linux
/dev/sda2         1026048  1572863999   785918976   8e  Linux LVM
Disk /dev/sdb: 322.1 GB, 322122547200 bytes, 629145600 sectors
/dev/sdb1   *        2048     1026047      512000   83  Linux
/dev/sdb2         1026048   629145599   314059776   8e  Linux LVM
Disk /dev/mapper/ol-root: 53.7 GB, 53687091200 bytes, 104857600 sectors
Disk /dev/mapper/ol-swap: 25.4 GB, 25367150592 bytes, 49545216 sectors
Disk /dev/mapper/ol-home: 725.7 GB, 725656535040 bytes, 1417297920 sectors
Disk /dev/mapper/ol00-swap: 25.4 GB, 25367150592 bytes, 49545216 sectors
Disk /dev/mapper/ol00-home: 242.5 GB, 242472714240 bytes, 473579520 sectors
Disk /dev/mapper/ol00-root: 53.7 GB, 53687091200 bytes, 104857600 sectors
```

Figura 2. Informações referente ao armazenamento e disco do servidor de Banco de Dados. FONTE – Autor.

Já o servidor de aplicação (onde rodam todos os sistemas e BI), possui as especificações conforme figura abaixo:

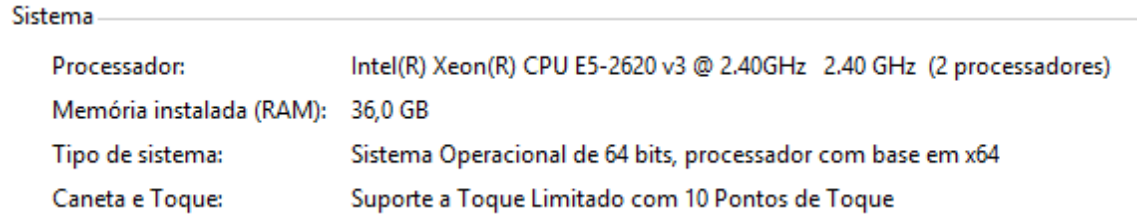

Figura 3. Informações referentes ao Sistema Operacional do servidor de Aplicação. FONTE – Autor.

O servidor de aplicação possui 2 partições, sendo elas: Disco Local (C:) com 599Gb, sendo 69.8Gb livres e o Disco Local (D:) com 499Gb, sendo 477Gb livres.

3.2 QUESTIONÁRIO (A FIM DE IDENTIFICAR OS PONTOS DE GARGALO)

O questionário abaixo, obteve resposta de 29 usuários de diversos setores da empresa, levando em consideração que alguns usuários de setores chave não responderam ao questionário por motivos desconhecidos.

O questionário foi realizado em cima da plataforma Google Forms no período de 30/07/2018 à 10/08/2018, segue abaixo o resultado:

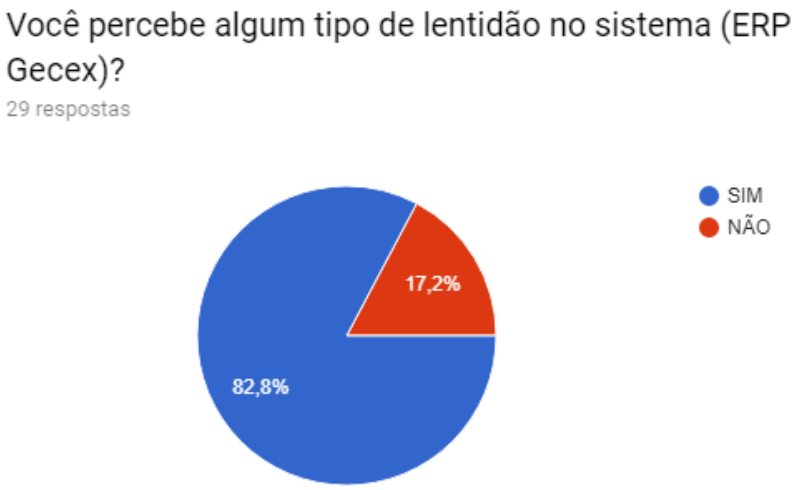

Figura 4. Totalização referente à percepção de lentidão. FONTE – Autor.

Em seguida, foi realizada uma pergunta à fim de identificar qual sistema, bem como quais rotinas existia possível lentidão, conforme figura abaixo:

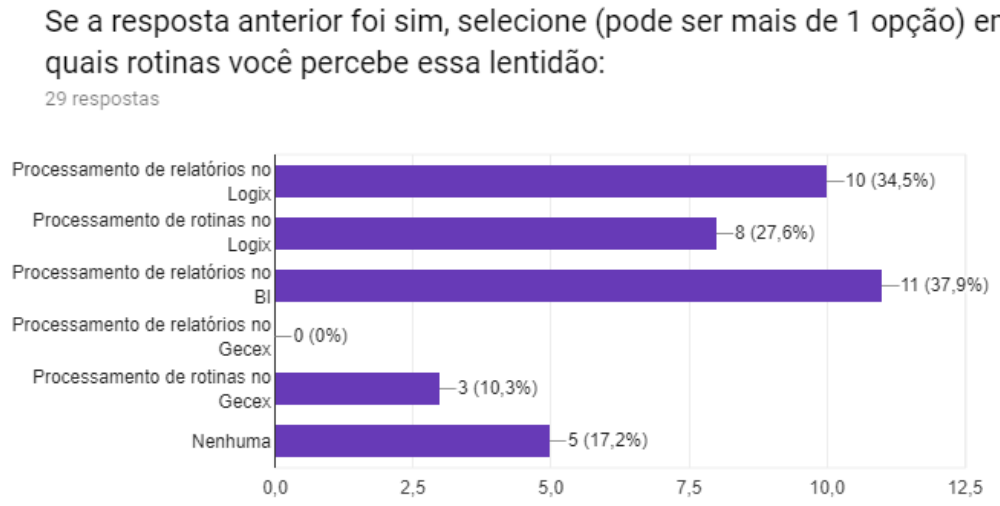

Figura 5. Totalização de Sistemas/Rotinas. FONTE – Autor.

Na sequencia o usuário deveria informar seu setor, à fim de identificar os módulos com maior congestionamento de dados:

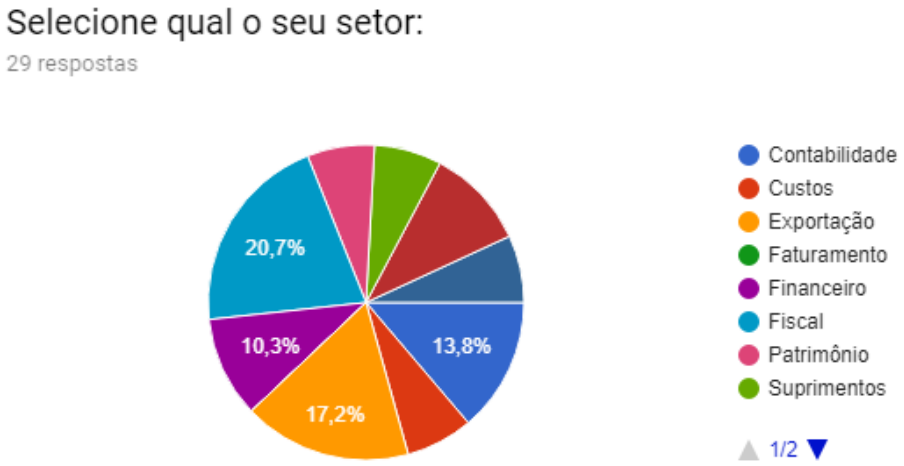

Figura 6. Totalização por setor. FONTE – Autor.

Todavia, as informações obtidas com o questionário acima foram insuficientes, visto que a percepção da lentidão é subjetiva e pode não ser precisa. Contudo, foi elaborado um novo questionário composto de 9 perguntas, à fim de obter informações pertinentes à qual sistema com maior utilização pelo usuário, setor em que atua, se o mesmo atua na matriz ou filiais, qual o dispositivo utilizado para acesso ao sistema, tipo de rede utilizada, percepção de lentidão e se a mesma ocorre em horário e período do mês em específico.

Como citado acima, alguns usuários não responderam ao primeiro questionário e com isso, buscou-se saber o motivo desta não participação. Conversando com estes usuários, constatou-se que alguns não responderam devido à bloqueios no firewall das filiais, portanto os mesmos não obtiveram acesso ao link do formulário do google. Pensando nesta situação, foi elaborado um novo questionário no editor de texto word (salvo em 2 formatos, .doc e .txt para usuários que não dispunham de uma ferramenta da suíte office ou libreoffice) e enviado por email para os mesmos 33 usuários do questionário anterior no período de 04/09/2018 à 06/09/2018.

Devido ser um período do mês de muita demanda de trabalho (fechamento fiscal, contábil) e com o feriado de 07 de setembro nesta mesma semana, foram

obtidas 17 respostas neste período, e, embora fugindo do cronograma, será aceito o formulário preenchido até 14/09/2018 por ser de suma importância a opinião dos usuários, e logo após efetuada a tabulação dos dados.

Contudo, caso não seja obtido o *feedback* esperado, serão adotadas de 1 a 2 consultas SQL de cada módulo do sistema/BI para análise.

4. CENÁRIO ATUAL

Analisando diversos arquivos de logs das principais rotinas do sistema, constatou-se que são utilizados comandos sql específicos para cada função e/ou retorno de informação, com isso, comandos de *include, update, delete* e até mesmo *select* não levem tanto tempo para serem executados pois não são muitos complexos e em diversos casos, com relação aos comandos *select* são “quebradas” em diversas consultas.

Com isso, decidiu-se apenas focar nas análises do BI, sendo que ao rodar os comandos do ERP extraídos através de *debug,* percebeu-se que o seu tempo de execução era satisfatório, pois comandos complexos eram subdividos em vários outros comandos.

Baseando-se no questionário, identificou-se 3 usuários distintos que têm algum tipo de lentidão no processar os relatórios. Com essa informação, foi identificado junto aos usuários 8 relatórios do BI que foram analisados.

4.1 ANÁLISE 1: CONHECIMENTOS DE FRETE

Um dos relatórios analisados foi obtido junto ao setor de tributos, que hoje é utilizado para conferência de valores de impostos de pis e cofins. Este relatório, como é utilizado para conferência, geralmente é utilizado com filtros de conhecimentos de frete do mês corrente, fazendo com que o seu processamento ao fim do mês retorne um número maior de registros.

Esse relatório, como está montado hoje utiliza-se de uma tabela mestre onde constam informações pertinentes ao conhecimento de frete, bem como tabelas relacionadas que contem informações de notas fiscais de entrada, de saída e relacionamentos entre conhecimentos de frete e notas fiscais.

No caso dos conhecimentos de frete, ao emitir uma nota fiscal de saída ou entrada, é emitido este documento a fim de acompanhar a carga até o seu destino. No caso da Sudati, também é dado entrada em conhecimentos de frete que não

possuem relacionamentos com notas fiscais, fazendo com que não existam informações na tabela SUP_FRETE_X_NF_ENTRADA.

Analisando a *query* do atual relatório, a mesma foi montada com 3 *selects* distintos, sendo eles: fretes de entrada, fretes de saída e os fretes relacionados.

Na querie atual, foi identificado que haviam campos em duplicidade que no momento do *tuning* foram eliminados, e problemas no relacionamento entre as tabelas, fazendo com que a leitura de todos os registros para aplicar os filtros passados no parâmetro *where* fossem um número muito maior.

Um exemplo dessa falha no relacionamento se dá no uso incorreto de um left join, uma vez que a maioria dos relacionamentos obriga a se ter um relacionamento entre 2 tabelas diferentes, ou seja, o que contenha em tabela A, deverá conter na tabela B, sendo que o uso do left join retornaria tudo o que contesse em A, e não apenas o que resultasse da junção entre a A e B.

Para resolver esse problema, foi utilizado o uso apenas do join, que retorna apenas a junção entre a A e B, retornando um número bem menor de registros.

Ex: Toda nota fiscal deverá ter um item, logo o relacionamento entre nf x item deverá ser verdadeiro, pois não existiria uma nota fiscal sem item.

Outros problemas encontrados na *query* atual, foram que argumentos passados no parâmetro where, faziam com que a consulta fizesse uma varredura em registros que não seriam utilizados no relatório, como por exemplo:

“Para a conferência, são utilizados apenas conhecimentos e notas fiscais com situação normal, ou seja, não há necessidade de fazer a leitura de documentos cancelados”. Nesse caso, a validação da situação da nota fiscal e conhecimento de frete, foi retirado da cláusula *where* e foi inserido nos parâmetros de relacionamento no prórprio *join*, fazendo com que por exemplo, num total de 100 registros apenas 10 tivessem situação normal, apenas 10 registros estão sendo lidos agora, melhorando considerávelmente o tempo de execução com um numero grande de registro ao final do mês.

Outra situação que se apresentou nesse relatório, foi a duplicação de conhecimentos de frete retornados, pois no *select* onde se faz a leitura dos fretes de entrada, não se distingue dos que possuem relacionamento com a nota fiscal de entrada. Nesse sentido, fez-se uso do *not exists* para validar os conhecimentos lançados a vulso, dos que possuem relacionamento.

Porém, quando foi efetuada essa validação, verificou-se junto ao pessoal do setor de tributos que alguns conhecimentos de frete não constavam no relatório. Analisando esta situação, foi identificado que alguns conhecimentos de frete não estavam sendo relacionados com a sua respectiva nota fiscal no momento de seu lançamento, embora estes possuissem uma nota fiscal “relacionada” perante a receita federal. Com isso, a partir de então foi atentado para esta situação e corrigidas estas falhas.

Percebeu-se também que na *query* atual se fazia uso do de*code* e *case.* No trabalho foi adotado o uso apenas do *case* para manter um certo padrão, porém não se notou alteração nos tempos de execução da *query.*

Ao final, com a aplicação das técnicas de *tuning* e a reescrita das *queries*, obteve-se um número menor de registros retornados e de colunas retornadas no relatório enxugando as informações e facilitando a sua leitura no momento da conferência. Consequentemente teve um número menor de registros lidos e realizadas as aplicações dos filtros, logo seu tempo de execução foi menor do que com a *querie* atual.

Segue abaixo antes e depois do *tuning*:

Atualmente:

```
FROM FRETE_SUP A
   LEFT JOIN NF_SUP B ON (B.COD_EMPRESA    = A.COD_EMPRESA
                AND   B.COD_TRANSPOR   = A.COD_TRANSPOR
                AND   B.NUM_CONHEC     = A.NUM_CONHEC
                AND   B.SER_CONHEC     = A.SER_CONHEC
                AND   B.SSR_CONHEC     = A.SSR_CONHEC  )

   RIGHT JOIN AVISO_REC  C ON (C.COD_EMPRESA    = B.COD_EMPRESA
                    AND   C.NUM_AVISO_REC  = B.NUM_AVISO_REC )

   LEFT JOIN FORNECEDOR E ON (E.COD_FORNECEDOR = B.COD_TRANSPOR)

   LEFT JOIN SUP_PAR_AR F ON (F.EMPRESA          = C.COD_EMPRESA
                     AND F.AVISO_RECEBTO     = C.NUM_AVISO_REC
                     AND F.SEQ_AVISO_RECEBTO = C.NUM_SEQ
```

```
                    AND F.PARAMETRO        IN ('COD_CST_COFINS_FRT') )

WHERE A.TIP_FRETE      = 'C'
```

Tempo de execução: 10,218 segundos

Após o tuning:

```
FROM FRETE_SUP A
   JOIN NF_SUP B ON (B.COD_EMPRESA    = A.COD_EMPRESA
                AND   B.COD_TRANSPOR   = A.COD_TRANSPOR
                AND   B.NUM_CONHEC     = A.NUM_CONHEC
                AND   B.SER_CONHEC     = A.SER_CONHEC
                AND   B.SSR_CONHEC     = A.SSR_CONHEC  )

   JOIN AVISO_REC  C ON (C.COD_EMPRESA    = B.COD_EMPRESA
                  AND   C.NUM_AVISO_REC  = B.NUM_AVISO_REC )

   JOIN FORNECEDOR E ON (E.COD_FORNECEDOR = B.COD_TRANSPOR)

   JOIN SUP_PAR_AR F ON (F.EMPRESA          = C.COD_EMPRESA
                    AND F.AVISO_RECEBTO     = C.NUM_AVISO_REC
                    AND F.SEQ_AVISO_RECEBTO = C.NUM_SEQ
                    AND F.PARAMETRO         IN ('COD_CST_COFINS_FRT') )
WHERE A.TIP_FRETE      = 'C'
AND NOT EXISTS (
        SELECT * FROM SUP_FRETE_X_NF_ENTRADA REL
        WHERE A.COD_EMPRESA = REL.COD_EMPRESA
        AND A.NUM_CONHE=REL.NUM_CONHEC
        AND A.SER_CONHEC=REL.SER_CONHEC
        AND A.COD_TRANSPOR=REL.COD_TRANSPOR
)
```

OBSERVAÇÃO: As queries completas encontram-se no apêndice.

Tempo de execução: 4,219 segundos

4.2 ANÁLISE 2: FATURAMENTO

Outro relatório analisado foi o de informações pertinentes ao faturamento da empresa, obtido juntamente ao setor de faturamento da filial de Otacílio Costa (MDF).

Este relatório é utilizado para exibir o faturamento da empresa em um determinado período, de um determinado cliente, de uma determinada família de itens, entre outros.

Como podemos perceber é um relatório bastante volátil, fazendo com que certas execuções (principalmente um período de tempo superior à um trimestre) sejam impossíveis de serem executadas pela ferramenta de BI, ocasionando a queda do serviço.

Este relatório é composto de 4 consultas diferentes (notas de venda, devolução, notas com entrega futura e notas de complemento de valor).

Assim como o relatório anterior, este também apresenta falhas nos relacionamentos entre as tabelas, quebras de campos indevidos, utilização de parâmetros na cláusula *where* ocasionando uma leitura de registros desnecessários.

Então para resolução do problema desta *query,* foram unificados campos de data (que atualmente estavam dispostos de 3 colunas):

Atualmente:

```
YEAR(A.DAT_HOR_EMISSAO) ANO,
MONTH(A.DAT_HOR_EMISSAO) MES,
TRUNC(A.DAT_HOR_EMISSAO) DAT_HOR_EMISSAO,
```

Após o tuning:

```
TRUNC(A.DAT_HOR_EMISSAO)   DAT_HOR_EMISSAO,
```

Também foi revisto o relacionamento entre as tabelas, onde se estava fazendo uso de *left join*, visto que para se obter a informação do faturamento temos 2 elementos base: pedido e nota fiscal. Com isso, percebe-se a obrigatoridade da junção apenas de pedidos que possuam nota fiscal relacionada e, esta nota fiscal

não esteja cancelada. Nesse mesmo contexto, percebeu-se também a utilização de parâmetros na cláusula *where* (situação da nota fiscal e estatística da natureza de operação) que foram adicionados como filtros na própria junção das tabelas, evitando assim a leitura de registros de notas canceladas (para validação da situação) bem como a estatística da natureza de operação.
Segue abaixo antes e depois do *tuning*:

Atualmente:

```
FROM FAT_NF_MESTRE A

  LEFT JOIN FAT_NF_ITEM B
    ON A.EMPRESA = B.EMPRESA
   AND A.TRANS_NOTA_FISCAL = B.TRANS_NOTA_FISCAL

 LEFT JOIN NAT_OPERACAO C
    ON A.NATUREZA_OPERACAO = C.COD_NAT_OPER

WHERE
 A.SIT_NOTA_FISCAL = 'N'
 AND C.IES_ESTATISTICA <> 'N'
```

Tempo de execução: 53,187 segundos

Após o *tuning*:

```
  FROM FAT_NF_MESTRE A

  JOIN FAT_NF_ITEM B
    ON A.EMPRESA = B.EMPRESA
   AND A.TRANS_NOTA_FISCAL = B.TRANS_NOTA_FISCAL
   AND A.SIT_NOTA_FISCAL='N'

  JOIN NAT_OPERACAO C
    ON A.NATUREZA_OPERACAO = C.COD_NAT_OPER
   AND C.IES_ESTATISTICA <> 'N'
   AND A.SIT_NOTA_FISCAL='N'
```

Tempo de execução: 20,828 segundos

Com a retirada da situação da nota fiscal da cláusula *where,* efetuamos a listagem apenas das notas com situação normal dentro deste SQL que como no exemplo anterior, reduzimos a quantidade de registros retornados e lidos.

Também foi adotado a utilização do *UNION ALL,* que segundo alguns autores têm um desempenho melhor pois faz a união sem realizar qualquer tipo de identificação de registros duplicados, visto que foi utilizado o comando *distinct* em todas as consultas, bem como os filtros à fim de eliminar a possibilidade de duplicações.

OBSERVAÇÃO: As queries completas encontram-se no apêndice.

4.3 ANÁLISE 3: NOTAS DE ENTRADA

O relatório de notas de entrada analisado, geralmente é utilizado para conferência nos fechamentos dos setores de tributos e fiscal, e consiste basicamente em informações pertinentes a notas fiscais de entrada, especificamente englobando a parte de impostos onde poderíamos citar ICMS, IPI, PIS E COFINS.

Atualmente, este relatório consiste nos mesmos problemas dos relatórios anteriores com o uso do left join, que neste caso em específico, quando se dá a entrada da nota fiscal, a mesma deve possuir seus respectivos itens e ao final é gerado um documento chamado “aviso de recebimento”, portanto notas válidas para este relatório, bem como sua conferência, devem ser notas válidas (e com isso devem possuir itens e aviso de recebimento).

Outro problema encontrado, foi a utilização de uma segunda *view* que realiza a conversão do formato de data, o que foi julgado desnecessário, uma vez que o campo de data de emissão e data de entrada da nota fiscal atendem os requisitos de formato utilizado pelo usuário.

Para a resolução do problema desta *query,* foram retiradas as junções *left join* em tabelas que o relacionamento é obrigatório, e por final foi retirado a junção com a *view* de formato de data.

Segue abaixo antes e depois do *tuning*:

Atualmente:

```
FROM NF_SUP A
  LEFT JOIN AVISO_REC B
    ON (B.COD_EMPRESA = A.COD_EMPRESA AND B.NUM_AVISO_REC =
A.NUM_AVISO_REC)

  LEFT JOIN ITEM_SUP IT ON IT.COD_EMPRESA = B.COD_EMPRESA
                AND IT.COD_ITEM    = B.COD_ITEM

  JOIN EIS_DIMENSAO_DAT DT
   ON (TRUNC(A.DAT_ENTRADA_NF) = DT.DAT_REFER)
```

Tempo de execução: 13,515 segundos

Após o tuning:

```
FROM NF_SUP A
  JOIN AVISO_REC B
    ON (B.COD_EMPRESA = A.COD_EMPRESA AND B.NUM_AVISO_REC =
A.NUM_AVISO_REC)

  JOIN ITEM_SUP IT ON IT.COD_EMPRESA = B.COD_EMPRESA -->>
                AND IT.COD_ITEM    = B.COD_ITEM

  JOIN FORNECEDOR C  -->>
   ON (C.COD_FORNECEDOR = A.COD_FORNECEDOR)
```

Tempo de execução: 8 segundos

OBSERVAÇÃO: As queries completas encontram-se no apêndice.

4.4 ANÁLISE 4: MOVIMENTAÇÃO ESTOQUE

O relatório de movimentação de estoque analisado consiste em informações pertinentes ao setor de custos e manufatura, e geralmente é utilizado para conferência de saídas e produção de itens da empresa.

Vendo este relatório em um primeiro momento, ele apresenta as mesmas falhas dos relatórios anteriores no quesito relacionamento de tabelas. Porém, analisando mais a fundo, percebe-se que há alguns relacionamentos existentes na *view* que não estão sendo utilizados, sendo assim foram eliminados.

Outra situação, é que não estava sendo validado a situação atual do item, sendo que na *view* atual, eram lidos itens cancelados e inativos, fazendo com que houvesse uma sobrecarga e uma utilização de recursos maior.

Então como solução, foi incluída uma validação na própria junção da tabela de itens, alteradas as junções *left join* para apenas *join* e eliminados 2 relacionamentos de tabelas que não estavam sendo utilizadas.

Segue abaixo antes e depois do *tuning*:

Atualmente:

```
FROM ESTOQUE_TRANS A
  LEFT JOIN ITEM B
    ON A.COD_EMPRESA = B.COD_EMPRESA
   AND A.COD_ITEM = B.COD_ITEM

LEFT JOIN EMPRESA C
    ON A.COD_EMPRESA = C.COD_EMPRESA

  LEFT JOIN ESTOQUE_OPERAC D
    ON A.COD_EMPRESA = D.COD_EMPRESA
   AND A.COD_OPERACAO = D.COD_OPERACAO

  LEFT JOIN LOCAL LOCAL_ORIG
    ON A.COD_LOCAL_EST_ORIG = LOCAL_ORIG.COD_LOCAL
   AND A.COD_EMPRESA = LOCAL_ORIG.COD_EMPRESA

  LEFT JOIN LOCAL LOCAL_DEST
    ON A.COD_LOCAL_EST_DEST = LOCAL_DEST.COD_LOCAL
   AND A.COD_EMPRESA = LOCAL_DEST.COD_EMPRESA

  LEFT JOIN ESTOQUE_OBS OBS
    ON A.COD_EMPRESA = OBS.COD_EMPRESA
```

```
   AND A.NUM_TRANSAC = OBS.NUM_TRANSAC
  LEFT JOIN FAMILIA FAM
    ON B.COD_FAMILIA = FAM.COD_FAMILIA
   AND B.COD_EMPRESA = FAM.COD_EMPRESA
```

Tempo de execução: 1,687 segundos

Após o tuning:

```
FROM ESTOQUE_TRANS A
  JOIN ITEM B
    ON A.COD_EMPRESA = B.COD_EMPRESA
   AND A.COD_ITEM = B.COD_ITEM
   AND B.IES_SITUACAO='A'

  JOIN ESTOQUE_OPERAC D
    ON A.COD_EMPRESA = D.COD_EMPRESA
   AND A.COD_OPERACAO = D.COD_OPERACAO

  JOIN ESTOQUE_OBS OBS
    ON A.COD_EMPRESA = OBS.COD_EMPRESA
   AND A.NUM_TRANSAC = OBS.NUM_TRANSAC

  JOIN FAMILIA FAM
    ON B.COD_FAMILIA = FAM.COD_FAMILIA
   AND B.COD_EMPRESA = FAM.COD_EMPRESA
```

Tempo de execução: 0,5 segundos

OBSERVAÇÃO: As queries completas encontram-se no apêndice.

4.5 DEMAIS RELATÓRIOS

Foram apresentados acima 4 dos 8 relatórios onde foram aplicadas as técnicas de *tuning*, visto que destes outros 4 relatórios alguns deles não obtiveram

melhoras nos resultados pois após avaliadas as *views* envolvidas, as mesmas estavam escritas corretamente e sua lentidão se dá pelo número muito grande de registros retornados (sendo o caso de 3 relatórios do setor contábil).

Os relatórios do setor contábil, como mencionado acima, são relatórios muito específicos que requerem diversos tipos de cálculo em algumas colunas e por vezes, retornam um número muito grande de registros devido ao alto número de lançamentos contábeis que são efetuados, e, como a empresa possui diversas filiais este número fica cada vez maior conforme o decorrer do mês.

Um outro relatório que é utilizado tanto pelo setor de tributos, bem como pelo setor de faturamento é pertinente às notas fiscais de saída. Como foram aplicadas as técnicas de *tuning* já na *view* de faturamento, não se fez necessário a aplicação na parte de notas de saída pois ambos os relatórios utilizam a mesma *view.* O motivo por ser dois relatórios diferentes se dá por questão de restrições de acesso,sendo que o relatório utilizado pelo setor de faturamento é utilizado na filial MDF, e o utilizado pelo tributos faz parte da matriz Compensados.

5 CONSIDERAÇÕES FINAIS

O cuidado com o desempenho de um Banco de Dados trata-se de um tema atual e de extrema relevância, porém, pouco se é explorado dentro desta área, visto que a maioria do material bibliográfico para pesquisa é escasso na lingua portuguesa, fazendo com que muitas vezes o administrador de Banco de Dados busque outras soluções paliativas para resolver determinados problemas. Sabe-se que é indispensável o profissional da área de tecnologia da informação ter conhecimento do básico da lingua inglesa, porém alguns profissionais não dispõem de tempo/oportunidade para tal, o que não será julgado no presente trabalho.

Em virtude do grande volume de dados que são gerados com os sistemas de informação atualmente, perante à necessidade de um tempo de resposta cada vez mais rápido faz-se necessário a elaboração de um código limpo que retorne as informações com um custo de tempo considerável.

Pode-se observar através deste estudo referente ao *tuning* em *queries* de Banco de Dados Oracle, que a aplicação destas técnicas é de suma importância

desde a sua concepção e até mesmo a sua manutenção posteriormente, visando a melhora no seu tempo de execução, desde que haja um monitoramente prévio ou qualquer outra forma de avaliar e diagnosticar a lentidão, visto que para se obter a informação através do usuário notou-se um grande problema para o trabalho, pois dos 2 questionários enviados o *feedback* foi relativamente pequeno.

Para que se obtenha um retorno de informações em um tempo de execução considerável, faz-se necessário atentar para os relacionamentos corretos entre as tabelas elaborando um plano de consulta, a fim de evitar que hajam duplicidade de campos, ou que sejam quebrados campos indevidamente.

Também se faz necessário atentar para parâmetros utilizados nas cláusulas *where,* que fazem com que sejam lidos um número muito maior de registros ocasionando lentidão na exibição do relatório.

No decorrer do trabalho, buscou-se adotar as melhores práticas de escrita SQL em um Banco de Dados Oracle, das quais algumas queries obtiveram melhora significativa no seu tempo de execução, já outras, não obtiveram melhora significativa pois trabalham com um número muito grande de registros e ao serem analisadas já estavam escritas corretamente.

Contudo, não se descarta como trabalhos futuros uma análise e possível *tuning* de sistema operacional ou até mesmo de infraestrutura, visto que é disposto de 2 servidores principais (aplicação e Banco de Dados) onde são executados diversos serviços e aplicações pertinentes a realização de tarefas da empresa (em muitos casos uma execução concorrente) fazendo com que tenha um consumo grande de seus recursos.

REFERÊNCIAS

CHAN, Immanuel.; ASHDOWN, Lance. **Oracle Database – Performance Tuning Guide 11g – Release 2 (11.2).** Copyright ORACLE, 2014.

COUTO, Eder. **Artigo: Aumentando a Performance da Aplicação Através da Otimização de SQL**. Disponível em:<https://imasters.com.br/artigo/4055/bancodedados/aumentando_a_performance_da_aplicacao_atraves_da_otimizacao_de_sql/>. Acesso em: 02/04/2018.

ELMASRI, Ramez.; NAVATHE, Shamkant B.**Sistemas de Banco de Dados. 4.** Ed. São Paulo: Editora Pearson Education do Brasil Ltda., 2005.

IKEMATU, Ricardo Shoiti. **Realizando Tuning na Base de Aplicações.** Disponível em: <http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=1592>. Acesso em: 03/04/2018.

LEGATTI, Eduardo. **Introdução ao conceito de Tablespaces.** Disponível em: <https://www.oracle.com/technetwork/pt/articles/database-performance/introducao-conceito-de-tablespaces-495850-ptb.html> Acesso em: 10/09/2018.

LEGATTI, Eduardo. **Qual é mesmo o tamanho de uma tabela no Oracle?** Disponível em: <http://www.oracle.com/technetwork/pt/articles/database-performance/tamanho-de-uma-tabela-no-oracle-495868-ptb.html >. Acesso em: 06/04/2018.

LEITE, Alison K. L. et al. **Tuning em queries em Banco de Dados.** Curso de Tecnologia em Banco de Dados - Faculdade de Tecnologia de Bauru (FATEC)

MACÊDO, Diego. **Depuradores (Debuggers).** Disponível em: <http://www.diegomacedo.com.br/depuradores-debbugers/ >. Acesso em: 05/04/2018.

REZENDE, Ricardo. **Introdução ao Oracle 9i – Parte II.** Entendendo o Oracle Server. Disponível em: <https://www.devmedia.com.br/artigo-sql-magazine-13-introducao-ao-oracle-9i-parte-ii/5655>. Acesso em: 10/09/2018.

SALISH, Asnani, **Oracle Database 11g – Hands on SQL and PL/SQL**, New Delhi: editora PHI Learning Private Limited, 2010.

SILBERSCHATZ, Abraham.; KORTH, Henry F.; SUDARSHAN. S. **Sistema de Banco de Dados. 5.** Ed. São Paulo: Editora Elsevier Ltda., 2006.

ZORZI, Marcelo Tomiello. **Tuning De Sistema Operacional Em Banco De Dados Oracle.** UNIVERSIDADE DE CAXIAS DO SUL CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA, 2015.