INSTITUTO DE ENSINO SUPERIOR DO ESPÍRITO SANTO
FACULDADE DO ESPÍRITO SANTO - UNES
CURSO DE SISTEMAS DE INFORMAÇÃO

LUCAS ZUCOLOTO FELIPE
WELESSON PEREIRA LOPES

AVALIAÇÃO DE DESEMPENHO DE BANCOS DE DADOS UTILIZANDO PRÁTICAS DE NORMALIZAÇÃO E DESNORMALIZAÇÃO

CACHOEIRO DE ITAPEMIRIM
2013

LUCAS ZUCOLOTO FELIPE
WELESSON PEREIRA LOPES

AVALIAÇÃO DE DESEMPENHO DE BANCOS DE DADOS UTILIZANDO PRÁTICAS DE NORMALIZAÇÃO E DESNORMALIZAÇÃO

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação na Faculdade do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador:Prof. Me. Alexandre Romanelli.

CACHOEIRO DE ITAPEMIRIM
2013

LUCAS ZUCOLOTO FELIPE
WELESSON PEREIRA LOPES

AVALIAÇÃO DE DESEMPENHO DE BANCOS DE DADOS UTILIZANDO PRÁTICAS DE NORMALIZAÇÃO E DESNORMALIZAÇÃO

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação na Faculdade do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Aprovado em 07 de novembro de 2013.

COMISSÃO EXAMINADORA

_____
Prof. Me. Alexandre Romanelli
Orientador

_____
Prof. Me. Jocimar Fernandes

_____
Prof. Me. Thiago Caliman

Dedico aos meus pais, e toda minha família, do qual foram os grande responsáveis pela minha trajetória de sucesso até aqui.
Lucas.

Dedico este trabalho aos meus pais e ao meu irmão, pelo incentivo, cooperação, apoio e por compartilharem os momentos de tristezas e também de alegrias até aqui.
Welesson.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por me conceder vida e conseguir chegar até aqui. Agradeço a os meus pais Luiz Antônio Felipe e Sidinéia Zucoloto Felipe, e toda minha família pelo apoio moral e conselhos para enfrentar os desafios da vida e ao longo do curso de graduação. Ao segundo tenente do exército brasileiro Fernando Figueira Cromack e ao primeiro sargento Helder Francisco David Alves, que contribuíram com meu desenvolvimento intelecto-social ao longo do serviço militar obrigatório que fiz em paralelo ao curso de graduação. Aos meus amigos de faculdade, que sempre me ajudaram nos momentos difíceis durante a caminhada, estão sempre incentivando e dando um apoio moral. Agradeço também a todos que contribuíram direta ou indiretamente para a minha caminhada até aqui.

Lucas.

Agradeço primeiramente a Deus que permitiu que tudo isso acontecesse ao longo de minha vida. Aos meus pais, Marli Pereira e Sebastião Welesson Coelho Lopes, que me deram toda a estrutura para que me tornasse a pessoa que sou hoje, pela confiança e pelo amor que me fortalece todos os dias. O meu irmão Icaro Tadeu Pereira Lopes, por estar sempre presente, a cada dia nos tornamos mais amigos. Agradeço a todos os professores por me proporcionar o conhecimento e educação no processo de formação profissional. Agradeço a todos os meus amigos e colegas que de alguma maneira ajudaram para esta realização.

Welesson.

"Os problemas significativos com os quais nos deparamos não podem ser resolvidos no mesmo nível de pensamento em que estávamos quando eles foram criados."
Albert Einstein

FELIPE, Lucas Zucoloto; LOPES, Welesson Pereira. **Avaliação de desempenho de bancos de dados utilizando práticas de normalização e desnormalização.** 2013. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) – Faculdade do Espírito Santo, Cachoeiro de Itapemirim, 2013.

RESUMO

A informação hoje é algo de muito valor para a sociedade tendo que ser obtida de forma rápida e confiável, principalmente quando se trata de informações que podem ajudar na tomada de decisão nas empresas. Bancos de dados possibilitam o controle e a disponibilização dessas informações, por isso é indispensável uma escolha de qual SGBD a ser utilizado e também de se como será a modelagem. O modelo de um banco pode reduz custos de hardware e de manutenção. Com isso o presente trabalho se propõe a demonstrar as práticas de normalização de banco de dados e avaliar os desempenhos das formas normais em SGBDs diferentes. Todavia, dada a riqueza do tema e sua extensão, focalizou-se as avaliações de desempenho dos SGBDs sobre comandos executados para cada forma normal testada. Para tanto, foi realizada uma pesquisa bibliográfica em algumas obras de referência e um estudo de caso para avaliação dos SGBDs. Ao final, demonstra-se através de gráficos os resultados dos desempenhos dos SGBDs. Também fica ratificado os SGBDs e as formas normais com melhor desempenho para uma determinada quantidade de dados, podendo assim auxiliar na hora de modelagem do banco de dados aumentando o desempenho.

Palavras-chave: MySQL. PostgreSQL. Normalização. Desnormalização.

FELIPE, Lucas Zucoloto; LOPES, Welesson Pereira. **Avaliação de desempenho de bancos de dados utilizando práticas de normalização e desnormalização.** 2013. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) – Faculdade do Espírito Santo, Cachoeiro de Itapemirim, 2013.

ABSTRACT

Today, information has a great value for society, and has to be obtained quickly and reliably, mainly when it is related to decision-making in enterprises. Databases allow control and availability of that informations and, therefore, a right choice of a RDBMS to be used is indispensable, so do a correct modeling. A good modeling can reduce costs of hardware and of maintainability. Thus, this document is aimed to demonstrate the practices of database normalization and evaluate the performance of normal forms applied on two differents RDBMS. Nevertheless, given the richness of the field and its extensions, we have focused on RDBMS's performance analysis, observing the executed commands for each tested normal form. To do so, we did a review of reference works and also we did a case study to evaluate the RDBMS. At the end, we showed the RDBMS's performance results by the extensive use of charts. It was found that some normal forms and some RDBMS are more adequate to some amount of data, and this information can help to model more efficient databases.

Key-words: MySQL. PostgreSQL. Normalization. Denormalization.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE SIGLAS

1NF - Primeira Forma Normal

2NF - Segunda Forma Normal

3NF - Terceira Forma Normal

4NF - Quarta Forma Normal

5NF - Quinta Forma Normal

FNBC - Forma Normal de BOYCE/CODD

Gb - Gigabit

GB - Gigabyte

MD5 - Message-Digestalgorithm5

SGBD - Sistema de Gerenciamento de Banco de Dados

SHA - Secure Hash Algorithm

SSL - Secure Socket Layer

SQL - Structured Query Language

TB - Terabyte

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Um sistema de banco de dados é um sistema que guarda, mantém e disponibilizam informações armazenadas através de um computador, um sistema de banco de dados pode guardar todo tipo de informação, onde normalmente são salvas as informações mais importantes de uma pessoa ou empresa.

Um banco de dados aumenta a confiabilidade e disponibilidade das informações, uma pesquisa ou relatório pode ser gerado em questão de segundos, se fosse feito de forma manual com poderia levar horas ou ate dias, com o uso do banco de dados também é mais fácil manter as informações atualizadas e ainda pode ser usado por vários usuários ao mesmo tempo.

O trabalho consiste em alterar ou eliminar algumas regras de normalização na criação do banco de dados mais comuns, pois com as três formas normais o banco vai servir para todas as aplicações, e para que as regras de normalização sejam alteradas ou modificadas basta apenas algumas mudanças na modelagem lógica e física.

Observando se a desnormalização traz aumento de desempenho para o banco de dados e até que ponto ela pode aumentar o desempenho, ou seja, quando o banco de dados desnormalizado começa a ficar mais lento que o banco de dados normalizado. Com uma boa escolha na modelagem lógica e sendo feita de maneira aperfeiçoada, escolhendo o melhor SGBD e a melhor normalização, de acordo com o tamanho previsto da base de dados, mantendo o desempenho do banco de dados ao longo do tempo sem a necessidade de aumentar o hardware do servidor.

1.1 Justificativa

Muitos programadores constroem o banco de dados de forma padrão, pois não tem tempo ou não tem paciência para construí-lo de formas diferentes e mais eficientes, acontece que o banco construído de forma padrão muitas vezes tem regras excessivas o que faz com que o desempenho do banco de dados em geral caia

como tempo e exigem mais recursos de hardware aumentando os custos para manter o servidor funcionando.

A pesquisa consiste em alterar ou eliminar algumas regras na criação do banco de dados mais comuns, pois com todas as regras o banco vai servir para todas as aplicações, eliminando regras desnecessárias podemos aumentar e manter o desempenho do banco de dados ao longo do tempo sem a necessidade de aumentar o hardware do servidor, e para isso bastava apenas algumas mudanças nas configurações quando o bando de dados for criado.

1.2 Problemas

Antes da normalização são encontrados alguns problemas no projeto do banco de dados acabam trazendo falta de integridade, redundância e inconsistência dos dados. Com a normalização esses problemas são resolvidos, porém acarreta em outro problema que é o de desempenho do banco de dados, causando lentidão em pesquisa de objetos nesses bancos de dados com poucoou muito volumes de dados.Ao utilizar várias tabelas em consultasque necessitam de alto desempenho, faz as consultas se tornarem mais complexas, provocado pelo excesso de normalizações no sistema, o que causa problemas de lentidão.

1.3 Hipóteses

Quando o banco de dados for usado em atividades para ler arquivos previamente registrados, como em um DataMart gerando relatórios ou fazendo algum tipo de pesquisa, o banco de dados desnormalizado tende a ser mais rápido que o banco de dados normalizado. Quando o banco de dados for usado em atividades para escrita ou exclusão de arquivos, ou seja, para escrever, alterar ou excluir arquivos, o banco de dados normalizado tende a ser mais rápido que o banco de dados desnormalizado.

A normalização de um banco de dados pode trazer os benefícios da eliminação de redundância e melhor clareza ao modelo. Mas isto eleva o custo de processamento de tarefas comuns a sistemas transacionais.

1.4 Objetivos

- O trabalho tem o objetivo de analisar o desempenho dos SGBDs com os diferentes níveis que normalização e diferentes bases de dados.
- Examinar qual é a melhor alternativa para otimizar a performance do banco de dados.
- Estudar as melhores técnicas de normalizar cada tipo banco de dados (SGBDs), para que possam atingir a melhor performance.
- Elaborar testes para que sejam aplicados nos banco de dados, verificar os resultados dos testes e demonstrar os resultados obtidos nos testes.
- Verificar e demonstrar os resultados obtidos com os testes realizados.

1.5 Estrutura

O trabalho está dividido em seis capítulos:

- O primeiro capítulo é a introdução ao tema;
- O segundo capítulo contém toda a parte teórica sobre anomalias, normalização e desnormalização de banco de dados;
- O terceiro capítulo mostra quais os SGBDs utilizados para os testes;
- O quarto capítulo demonstra a configurações para os testes, como o hardware, softwares, a criação das tabelas e quais os comandos utilizados nos testes;
- O quinto capítulo mostra os resultados obtidos em todos os testes realizados e a comparação entre os SGBDs;
- O sexto capítulo faz uma conclusão do trabalho.

2 NORMALIZAÇÃO

A normalização de banco de dados tem como objetivo evitar problemas no projeto do banco de dados avaliando e corrigindo estruturas e tabelas para diminuir a redundância de dados, manter a integridade dos dados e facilitar a manutenção dos dados. Um dos grandes problemas que devem ser evitados em um projeto de banco de dados é a mistura de assuntos em uma mesma tabela.

> O processo de normalização tem sido visto por muitas pessoas apenas como um formalismo necessário à implementação de estruturas relacionais. Esse formalismo, quando compreendido em sua finalidade, acaba por transformar-se mais em um fator complicante do que em um fator otimizador durante o projeto de banco de dados (COUGO, 1999, p. 173).

A normalização de banco de dados tem algumas regras que são divididas em um conjunto de cinco formas normais, mas as mais utilizadas são as três primeiras, pois a 4FN e a 5FN são raramente utilizadas(REZENDE, 2012).

O processo de normalização simplifica muito uma tabela de banco de dados, isso colabora para a estabilidade dos dados reduzindo a necessidade de manutenção no banco de dados.

> Uma forma normal é uma regra que deve ser obedecida por uma tabela para que esta seja considerada ‘bem projetada’. Há diversas formas normais, isto é, diversas regras, cada vez mais rígidas, para verificar tabelas relacionais (HEUSER, 2001, p. 125).

Esse processo de normalização não deve ser exagerado, de forma que sejam criadas muitas tabelas com muitos relacionamentos senão muitos recursos serão exigidos do sistema para executar uma pesquisa no banco de dados.

“Normalmente após a aplicação das formas normais, algumas tabelas são divididas em duas ou mais tabelas, o que no final gera um número maior de tabelas do que o originalmente existente” (MACHADO, 2004, p. 181).

Por exemplo, na TABELA1 - Tabela Pedido, está mostrando informações de clientes, pedido, produto. E ocorre redundância de dados em varias linhas dessa

tabela e também a mistura de assuntos, ficando muito confuso para se ler os dados da tabela.

TABELA 1 - Tabela Pedido

Num_Ped	Data	Val_total	Cod_Cli	Nome_Cli	Cod_Pro	Nome_Pro	Qtde	Val_Unit	Val_Item
00001	17/05/11	500	C1	João	P2	Pen Drive	2	50	100
00001	17/05/11	500	C1	João	P1	Ap. Som	1	400	400
00002	18/05/11	200	C4	Carlos	P4	Celular	1	200	200
00003	19/05/11	1200	C2	Maria	P3	TV	1	1200	1200
00004	21/05/11	400	C1	João	P1	Ap. Som	1	400	400
00005	22/05/11	800	C3	Paulo	P4	Celular	2	200	400
00005	22/05/11	800	C3	Paulo	P1	Ap. Som	1	400	400

Fonte: Pesquisa do autor, 2013

Se fosse necessário fazer uma atualização no nome do cliente seria preciso passar por todas as linhas dessa tabela para que se trocasse o nome desse cliente um a um quantas vezes ele aparecesse nessa tabela, sendo assim se teria uma perda de tempo para se fazer essa atualização. Do mesmo modo se for feita uma exclusão do pedido de número 00003 ocorreria uma perda de dados, as informações da cliente Maria seriam apagas junto ao pedido, pois é o único lugar do banco onde estão armazenadas essas informações. Esses fatos são considerados anomalias.

Então se deve utilizar a normalização como se fosse um formato das estruturas de dados das tabelas do banco de dados para diminuir essas anomalias, manter a integridade dos dados e fazer com que as informações apareçam de forma correta.

2.1 Anomalias de Atualização

Observando a TABELA 1 – Tabela Pedido, aparecem algumas anomalias de atualizações, que segundo Elmasri e Navathe (2011) são diferenciadas em três tipos de anomalias:

a) Anomalia de Inclusão: Acontece quando, ao inserir um dado, este dado pode gerar uma inconsistência no banco de dados. Por exemplo, na tentativa de inserir um novo cliente não seria possível, pois um cliente só pode ser inserido se ele estiver relacionado com um pedido.

b) Anomalia de Exclusão: Acontece quando, ao remover um registro, você pode gerar inconsistência no banco de dados. Por exemplo, na tentativa de excluir um cliente você iria também apagar os dados da compra desse cliente, mesmo não sendo o que você queira excluir.

c) Anomalia de modificação: Acontece quando, ao atualizar um registro, você pode gerar inconsistência no banco de dados. Por exemplo, se for preciso alterar o preço de um produto na tabela seria preciso percorrer toda a entidade para se realizar múltiplas alterações. Toda linha em que esse produto estivesse seria preciso ser feita uma alteração para que não ocorresse inconsistência no banco de dados.

2.2 Benefícios Trazidos Pela Normalização

Dentre vários benefícios trazidos pela normalização de banco de dados tem alguns que são os principais como a estabilidade do modelo lógico, a estabilidade lógica das aplicações, a estabilidade lógica dos dados, a integridade dos dados, flexibilidade, economia de espaço de armazenamento, fidelidade dos dados e do próprio banco de dados.

“Esse processo causa a simplificação dos atributos de uma tabela, colaborando significativamente para a estabilidade do modelo de dados, reduzindo-se consideravelmente as necessidades de manutenção.” (MACHADO, 2004, p.181).

A normalização também evita os problemas de anomalias que acontecem em alguns bancos de dados assim ao deletar uma determinada linha não serão apagados dados indesejados aumentando assim a confiabilidade.

2.3 Formas Normais

Um processo de normalização apresenta algumas etapas a serem seguidas. Essas etapas são denominadas formas normais, sendo no total, cinco formas normais, que serão mostradas a seguir.

2.3.1 Primeira forma normal (1FN)

Para a aplicação da primeira forma normal é preciso eliminar os elementos repetitivos podendo criar uma tabela com esses elementos, daí destaca-se mais de um atributo como chave primária da nova entidade. Em uma tabela na primeira formal normal não é permitido uso de atributos multivalorados e nem é permitida repetição de atributos de mesmo domínio (nota1, nota2, nota3) sendo que todas as linhas da tabela têm de serem diferentes umas das outras.

“Uma tabela está em 1FN se e somente se, em todo valor dessa tabela, cada linha contém exatamente um valor para cada atributo”(DATE, 2003, p. 309).

Uma tabela ainda na primeira forma normal que é o mais baixo nível de normalização ainda pode conter algumas anomalias.

Aplicando a 1FN, obtêm-se então duas estruturas de dados ou tabelas, a TABELA2 - Tabela Pedido1FNjá com as alterações ocorrida pela 1FN e uma nova tabela foi criada à partir da primeira que é a TABELA 3 - Tabela Item Pedido, que juntas representam a realidade da TABELA 1 - Tabela Pedido.

Segundo Ferrari (2007) uma tabela respeita a estrutura da primeira forma normal, quando é determinado que cada coluna contenha apenas um atributo, ou seja, uma informação, o conteúdo dessa coluna dever ser um único dado não divisível.

TABELA 2 - Tabela Pedido 1FN

Num_Ped	Data	Val_Total	Cod_Cli	Nome_Cli
00001	17/05/11	500	C1	João
00002	18/05/11	200	C4	Carlos
00003	19/05/11	1200	C2	Maria
00004	21/05/11	400	C1	João
00005	22/05/11	800	C3	Paulo

Fonte: Pesquisa do autor, 2013

Na TABELA 2 se encontra as informacões do pedido e na TABELA 3 as informações dos itens do pedido.

TABELA 3 - Tabela Item Pedido

Num_Ped	Cod_Pro	Nome_Pro	Qtde	Val_Unit	Val_Item
00001	P2	Pen Drive	2	50	100
00001	P1	Ap. Som	1	400	400
00002	P4	Celular	1	200	200
00003	P3	TV	1	1200	1200
00004	P1	Ap. Som	1	400	400
00005	P4	Celular	2	200	400
00005	P1	Ap. Som	1	400	400

Fonte: Pesquisa do autor, 2013

2.3.2 Segunda forma normal (2FN)

Para que ocorra a segunda forma normal é preciso retirar das estruturas de dados as chaves compostas, ou seja, chaves primárias formadas por mais de um campo e que cada coluna que não e chave não depende de toda essa chave primária, como por exemplo, a TABELA 3 - Tabela Item Pedido, que tenha os campos Cod_Produto e Num_Ped como chaves primárias. Então se em uma tabela se tiver apenas uma coluna como chave primária ela não terá dependências parciais e estará em 2FN.

> Uma tabela está na 2FN quando está na 1FN e não contém dependências parciais. Portanto, uma tabela 1FN encontra-se automaticamente em 2FN se sua chave primária basear-se apenas em um único atributo. Uma tabela em 2FN pode conter ainda dependências transitivas (ROB; CORONEL, 2011, p. 194).

Para que seja feita a redução das tabelas, de modo correto, para eliminaras chaves compostas é preciso que as novas tabelas sejam equivalentes à tabela original, no sentido de que com a união das tabelas geradas possa se obter a tabela original.

TABELA 4 - Tabela Pedido 2FN

Num_Ped	Data	Val_Total	Cod_Cli	Nome_Cli
00001	17/05/11	500	C1	João
00002	18/05/11	200	C4	Carlos
00003	19/05/11	1200	C2	Maria
00004	21/05/11	400	C1	João
00005	22/05/11	800	C3	Paulo

Fonte: Pesquisa do autor, 2013

TABELA 5 - Tabela Item Pedido 2FN

Num_Ped	Cod_Pro	Qtde	Val_Item
00001	P2	2	100
00001	P1	1	400
00002	P4	1	200
00003	P3	1	1200
00004	P1	1	400
00005	P4	2	400
00006	P1	1	400

Fonte: Pesquisa do autor, 2013

Foi criada uma nova tabela, como apresentado na TABELA 6, que contém os elementos que são identificados apenas pelo Cod_Produto, ou seja, são independentes do Num_Ped.

TABELA 6 - Tabela Produto

Cod_Pro	Nome_Pro	Val_Unit
P1	Ap. Som	400
P2	Pen Drive	50
P3	TV	1200
P4	Celular	200

Fonte: Pesquisa do autor, 2013

2.3.3 Terceira forma normal (3FN)

Para que uma tabela esteja na terceira forma normal é preciso que se elimine as dependências funcionais transitivas, ou seja, eliminar a transitividade entre atributos não chave ocorrendo a diminuição da redundância através da divisão da tabela em entidades com propósitos únicos.

“Podemos afirmar que uma estrutura está na terceira forma normal se ela estiver na segunda forma normal e não possuir campos dependentes de outros campos não chaves”(MACHADO, 2004, p.191).

Um exemplo disso pode observar na TABELA 4 - Tabela Pedido 2FN, a coluna Nome_Cliente depende da coluna Cod_Cliente, que não é um atributo chave.

> Independentemente de se nossa chave é composta por uma ou diversas colunas deveremos omitir essas colunas da análise. As colunas que nos interessam para a análise são somente as não formadoras da chave primária da tabela (COUGO, 1997, p.209).

Passando essa tabela para terceira forma normal essa tabela é dividida em duas, a TABELA 7 - Tabela Pedido3FN e a TABELA 8 - Tabela Cliente, que agora tem como a chave primária a coluna Cod_Cliente e o atributo dependente que é o Nome_Cliente eliminando assim a dependência funcional transitiva. Isso evita também a inconsistência de dados e economiza espaço por eliminar o armazenamento frequente desses dados e sua repetição.

> A definição de 3FN permite certas dependências funcionais que não são permitidas na FNBC. Uma dependência α→β que satisfaça apenas a terceira alternativa da definição 3FN não é permitida na FNBC, mas é permitida na 3FN (SILBERSCHATZ; KORTH; SUNDARSHAN, 2006, p. 184).

Abaixo veja como ficaram as tabelas depois da aplicação das três primeiras formas normais:

TABELA 7 - Tabela Pedido 3FN

Num_Ped	**Data**	**Valor_Total**	**Cod_Cli**
00001	17/05/11	500	C1
00002	18/05/11	200	C4
00003	19/05/11	1200	C2
00004	21/05/11	400	C1
00005	22/05/11	800	C3

Fonte: Pesquisa do autor, 2013

Na TABELA 7 se tem as informações dos pedidos.

TABELA 8 - Tabela Clientes

Cod_Cli	**Nome_Cli**
C1	João
C2	Maria
C3	Paulo
C4	Carlos

Fonte: Pesquisa do autor, 2013

A TABELA 8 foi criada para ter as informações dos clientes, que ate a 2FN estava junto à tabela dos pedidos.

TABELA 9 - Tabela Produto 3FN

Cod_Pro	**Nome_Pro**	**Val_Unit**
P1	Ap. Som	400
P2	Pen Drive	50
P3	TV	1200
P4	Celular	200

Fonte: Pesquisa do autor, 2013

A TABELA 9 traz às informações dos produtos e na TABELA 10 se encontra as informações dos itens dos pedidos.

TABELA 10 - Tabela Item Pedido 3FN

Num_Ped	Cod_Pro	Qtde	Val_Item
00001	P2	2	100
00001	P1	1	400
00002	P4	1	200
00003	P3	1	1200
00004	P1	1	400
00005	P4	2	400
00006	P1	1	400

Fonte: Pesquisa do autor, 2013

2.3.3.1 Forma normal de BOYCE/CODD (FNBC)

A forma normal de boyce/codd é um caso mais raro de ser encontrado, porque é derivada de erros na modelagem realizada na estruturação dos atributos de uma tabela.

> Uma das formas normais mais desejáveis que podemos obter é a boyce-codd normal form(FNBC). Ela elimina toda a redundância que pode se descoberta com base nas dependências funcionais, embora, [..], pode se haver outros tipos de redundância (SILBERSCHATZ; KORTH; SUNDARSHAN, 2006, p.181).

Ela foi criada como parte da terceira forma normal para suprir uma falha que ocorria nos casos onde se tem mais de uma chave candidata composta.

> Na verdade, a FNBC é uma extensão da 3FN, que não resolvia certas anomalias presentes na informação contida em uma entidade. O problema foi resolvido porque a 2FN e 3FN só tratavam dos casos de dependência parcial e transitivas de atributos fora de qualquer chave(MACHADO, 2004, p.194).

Ou seja, as outras formas normais não tratavam bem as tabelas que tivesse duas ou mais chaves candidatas e se essas chaves fossem compostas e com superposição.

2.3.4 Quarta forma normal (4FN)

A quarta forma normal é baseada na dependência multivalorada, que é provocada pela mistura dos atributos multivalorados independentes em uma única tabela. A

maioria das tabelas são normalizadas apenas até a terceira forma normal, pois são mais fáceis de entender, atualizar e de recuperar dados.

Para Rezende (2012) a quarta forma normal é apenas para a remoção de dependências multivaloradas em uma tabela para várias tabelas.

> Se tivermos dois ou mais atributos independentes multivalorados no mesmo esquema de relação, obtemos o problema de ter que repetir cada valor de um dos atributos com cada valor do outro atributo, a fim de manter o estado da relação coerente e as independências entre os atributos, envolvidos. Essa restrição é especificada por uma dependência multivalorada (ELMASRI; NAVATHE, 2011, p.357).

Mas ainda pode ocorrer problema com atributo não chave que recebe valores multivalorados para que ocorra a quarta forma normal é preciso que elimine esses atributos multivalorados não podendo ter mais de um a respeito da entidade descrita.

“Note que a definição da 4FN difere da definição da FNBC somente no uso de dependências multivaloradas em vez de dependências funcionais. Todo esquema na 4FN está na FNBC” (SILBERSCHATZ;KORTH; SUNDARSHAN, 1999, p. 233).

Essa dependência multivalorada vem da primeira forma normal que um atributo não pode ser multivalorado.

2.3.5 Quinta forma normal (5FN)

A quinta forma normal só é aplicada em casos de relações que temtrês ou mais atributos como parte da chave, onde é preciso criar uma pequena redundância para que não haja perda de dados ao decompor uma relação.

> Esta última forma normal trata do conceito de dependência de junção, quando a noção de normalização é aplicada à decomposição, devido a uma operação de projeção, e aplicada na reconstrução devido a uma junção (MACHADO; ABREU, 1996, p.171).

Ou seja, a divisão de uma relação que se tem três atributos como parte da chave, irá resultar em três relações e não em duas como nas outras formas normais, na junção

dos registros criados através do primeiro que não possuam a mesma chave primária não conseguir recuperar as informações contidas no registro original, então este registro está em 5FN.

2.4 Desnormalização

Ao se modelar um projeto de banco de dados não se pode pensar em apenas normalizar. Como no caso de um *DataWarehouse*, onde é preciso de relacionamentos entre entidades e façam consultas de forma eficiente, ou se aplicando a normalização não se consegue atingir um desempenho mínimo em um sistema.

O processo de desnormalização de banco de dados nada mais é do que o processo “contrário” da normalização.

“O objetivo é aumentar a redundância, assegurando que R está em um nível mais baixo de normalização que as RelVars R1,R2,...,Rn” (DATE, 2003, p. 339).

Mas é importante estudar bem as situações do projeto para que não se tenha um efeito contrário do esperado tanto na normalização quanto na desnormalização.

> Em outras palavras, utilize a desnormalização com cuidado e certifique-se que seja possível explicar por que tabelas não normalizadas são uma escolha melhor em certas situações do que as normalizadas correspondentes (ROB; CORONEL, 2011, p. 193).

Segundo Rezende (2012), uma tabela pode estar “normalizada em excesso” fazendo com que se tenham junções inadequadas entre muitas tabelas para encontrar uma única informação. Esse é um tipo de situação que se pode utilizar a desnormalização, quando a normalização foi feita de forma excessiva, pois em relação ao desempenho essa prática não se torna viável. Existem também várias outras situações onde se pode utilizar a desnormalização como, por exemplo, se em uma consulta for necessário realizar elevados números de cálculos com duas ou mais colunas antes da resposta ou número elevado de chaves estrangeiras em uma tabela.

Um modelo de banco de dados desnormalizado tem as anomalias de atualização e eliminação que implicam na integridade dos dados, porém, tem que ser usada com cuidado e com critérios como a identificação de consultas, os caminhos de acesso, a avaliação e a aplicação das configurações de prioridades que podem favorecer a performance do sistema melhorando o desempenho das consultas.

3 SGBDs UTILIZADOS

Para os testes de desempenhos foram escolhidos dois SGBDs, o MySQL e o PostgreSQL, pois além de serem uns dos SGBDs mais populares do mundo segundo Solid IT (acesso em: 20 out. 2013), também foram escolhidos por questões de ausência de custos.

3.1 PostgreSQL

Alguns alunos da universidade americana *Berkeley*, localizada no estado da Califónia, no ano de 1986 desenvolveram um projeto que era um modelo de armazenamento de dados que foi da onde surgiu o PostgreSQL. O projeto também obteve a ajuda de alguns órgãos que favoreceram o crescimento do projeto que em 1989 já havia uma versão estável e pronta para o uso. No ano de 1991 a empresa *Illustra Information Tecnologies* comprou o projeto que até então era denominado Postgres e se uniu a Informix, que hoje pertence a IBM. No ano de 1994 surgiu o Postgres95 com a introdução da linguagem SQL e em 1996 o programa passou a se chamar PostgreSQL (MILANI, 2008).

MILANI (2008) descreve o PostgreSQL como um SGBD relacional completo, em todas as funções obedece a linguagem SQL e além de ser gratuito, que não gera custo para o desenvolvedor, ele tem capacidade de gerenciamento de várias conexões com o banco de dados ao mesmo tempo com a utilização de recurso *multithread*. O PostgreSQL fornece também a replicação de dados entre servidores a fim de manter a integridade dos dados. Suporta criptografia SHA1 e MD5 e trabalha com SSL para garantir a segurança dos dados trafegados. E possui uma capacidade de armazenamento que pode ter um número ilimitado de bancos de dados, o tamanho de uma tabela pode chegar a 32 TB podendo ter de 250 a 1600 colunas dependendo do tipo de dado, o tamanho de uma linha pode chegar a 1,6 TB e o tamanho máximo de um campo pode ser de 1 GB.

Ou seja, o PostgreSQL é um sistema de gerenciamento de banco de dados relacional de alto desempenho, possui arquitetura paralela, que executa as funções simultaneamente para diversos usuários e tira proveito de sistemas com múltiplos

processadores. É um banco de dados robusto e usado por sistemas corporativos dos mais diversos portes.

3.2 MySQL

O MySQL surgiu na década de 90, foi criado para suprir as necessidades que Allan Larsson, David Axmark e Michael MontyWidenius, tinham em suas rotinas com aplicações e tabelas. No inicio tentaram utilizar o mSQL, mas não adiantou, então escreveram em C e C++ uma outra aplicação que deu origem ao MySQL (MILANI, 2007).

“Com o ótimo resultado gerado por essa nova API, o MySQL começou a ser difundido e seus criadores fundaram a empresa responsável por sua manutenção, que é a MySQL AB.” (MILANI, 2007, p. 23).

Hoje o MySQL é o mais popular banco de dados na comunidade *open source*. É utilizado por milhares de servidores e sistemas pelo mundo, além de possuir licença dupla (sendo uma delas de software livre) é o maior concorrente de bancos com código fechado.

O MySQL é um gerenciador de banco de dados (SGBD) relacional, que no princípio foi programado para pequenas e medias aplicações, mas possui características de um banco de dados de grande porte. Assim como o PostgreSQL, o MySQL tem características como a portabilidade, segurança, *multithreads* e a linguagem SQL.

> Além de armazenar os dados, a ferramenta provê todas as características de multiacesso a estes, entre outras funcionalidades de um SGBD, como, por exemplo, gerenciamento de acesso, integridade dos dados e relacional, concorrência, transações, entre outros (MILANI, 2007, p. 26).

A capacidade de armazenamento do MySQL pode chegar até 65536 TB dependendo do tipo da tabela.

4 CONFIGURAÇÃO DO AMBIENTE DE TESTE

Todas as informações que foram utilizadas para os testes estão destacadas em tópicos para uma maior interação e em alguns pontos há referências para tópicos anteriores.

4.1 Equipamentos de Hardware, Sistema Operacional e Softwares Utilizados

Inicialmente, vale salientar que não houve utilização da máquina para outros fins durante os processos para que não houvesse qualquer perca de desempenho durante a execução dos comandos. Para a criação dos testes realizados foi utilizado como fim acadêmico uma máquina com as seguintes configurações:

- Configuração de Hardware:
  - Plataforma: Intel Calpella;
  - Placa Mãe: Compal NBLB2;
  - Chipset: Intel PM55;
  - Processador: Intel Core i5 520M;
  - Memória RAM: 4096 MB (2 x 2048 DDR3-SDRAM PC3-8500 Kingston);
  - Hard Disk: TOSHIBA MK5065GSX 500GB, Taxa de Transferência: 3Gb/segundo.

- Sistema operacional:
  - O sistema operacional escolhido para os testes foi o Windows 7 Ultimate Professional Media Center 6.01.7600 (64-bit).

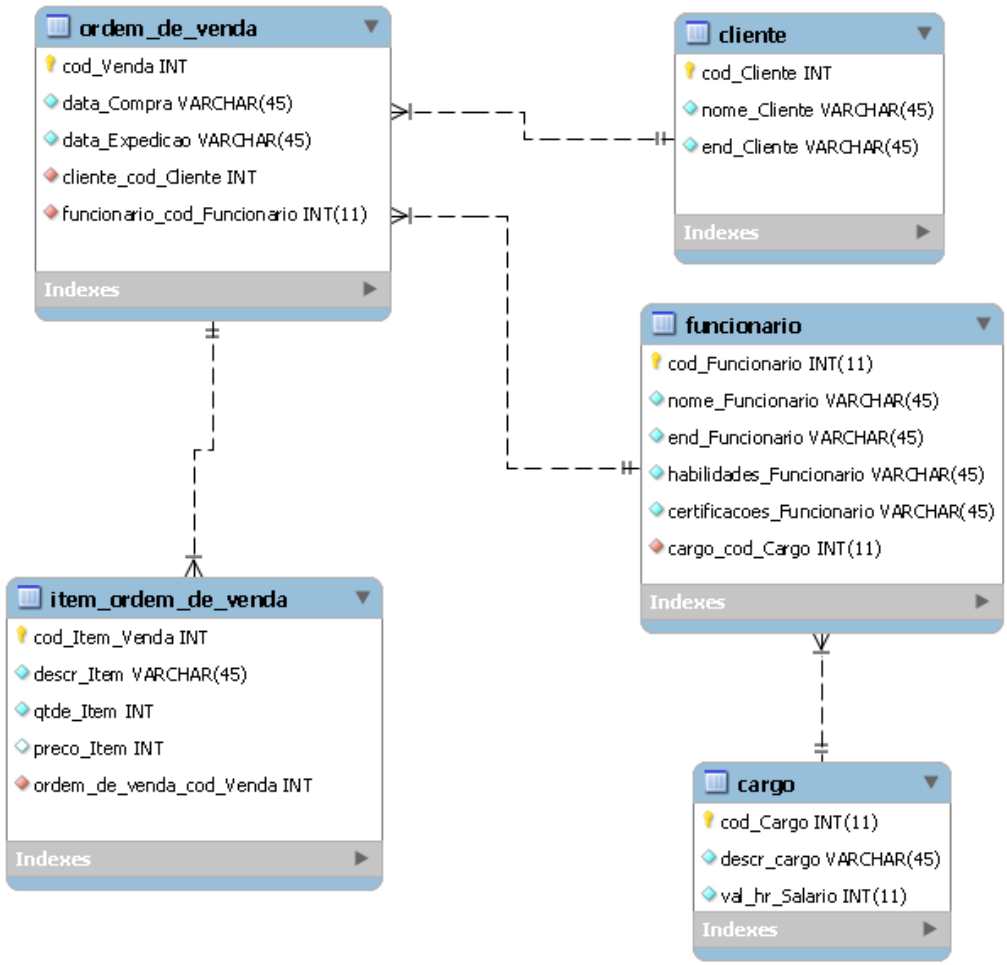
- Sistemas de gerenciamento de banco de dados (SGBD):
  - Por questões de popularidade e ausência de custos foi utilizado o MySQL (versão 5.6.13), com a ferramenta MySQL Workbench (versão 5.2.47 CE);
  - Por questões de popularidade e ausência de custos foi utilizado o PostgreSQL (versão 9.2.4 Windows(64-bit), com a ferramenta pgAdminlll (Versão 1.16.1).

4.2 Criação das Tabelas

Foram criados três tipos de banco de dados diferentes para cada SGBD. Um estará obedecendo à terceira forma normal, outro estará na segunda forma normal e o outro na primeira forma normal. Cada tipo de banco de dados será criado com dois tamanhos diferentes em cada SGBD (um com cem mil registros de vendas e outra com um milhão de registros de vendas).

Todos os bancos de dados têm as mesmas informações que serão apenas representadas de maneira diferente em cada forma normal. Os modelos das tabelas estão representados na FIGURA 1, FIGURA 2 e FIGURA 3.

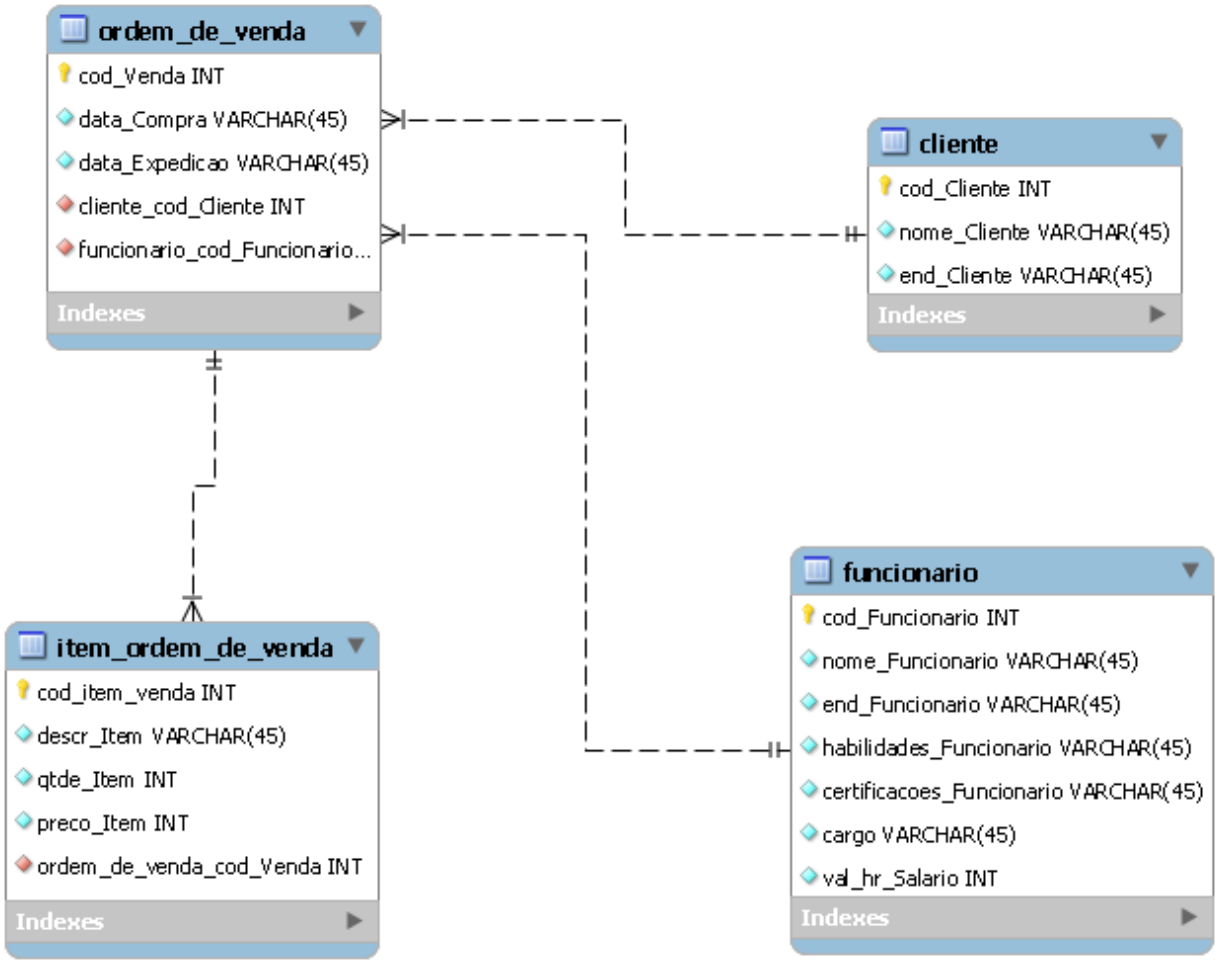
FIGURA 1 – Modelo das tabelas na 3FN



Fonte: Pesquisa do autor , 2013

Na FIGURA 1 está representado o banco de dados na terceira forma normal, onde se encontra cinco tabelas (ordem_de_venda, item_ordem_de_venda, cliente, funcionario, cargo).

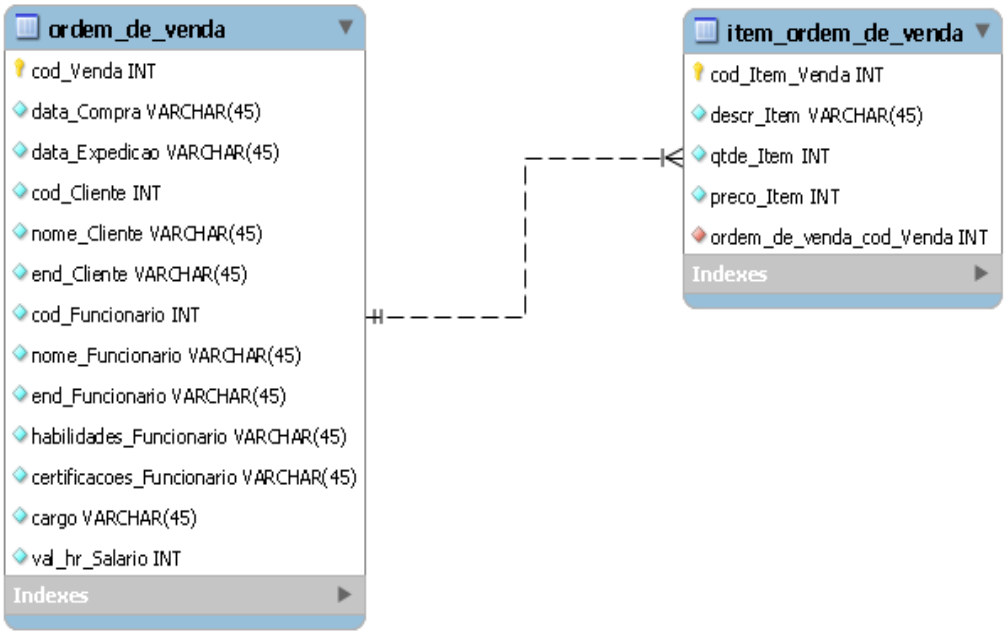
FIGURA 2 – Modelo das tabelas na 2FN



Fonte: Pesquisa do autor, 2013

Na FIGURA 2 tem-se o banco de dados na segunda forma normal, onde os campos descr_cargo e val_hr_Salario da tabela cargo passam a estar na tabela funcionário que ao invés de ter apenas o código do cargo já traz essas informações, sendo assim uma dependência funcional transitiva. Pode ser encontrado mais detalhes sobre dependência funcional no tópico 2.3.3 deste trabalho.

FIGURA 3 – Modelo das tabelas na 1FN



Fonte: Pesquisa do autor, 2013

E por último é representado na FIGURA 3 o modelo do banco na primeira forma normal, que tem apenas duas tabelas. Uma tabela traz todas as informações que se tinha antes com as tabelas cliente, funcionário e inclusive os seus respectivos códigos que assim se tornam dependências parciais. Também pode ser encontrados detalhes sobre dependência parcial no tópico 3.3.2 deste trabalho.

4.3 Comandos Utilizados para os Testes

Como serão utilizados dois tipos SGBD diferentes, cada tipo de SGBD terá três tipos de banco de dados diferentes e cada tipo de banco de dados será criado com dois tamanhos diferentes. Foram criados vários *scripts* para a inserção dos registros.

Primeiro foi feito o um *script* para gerar dados aleatórios para preencher as tabelas a fim de fazer com que todos os banco de dados recebessem os mesmos dados, apenas representados de formas diferentes. Cada um em sua forma normal e com algumas alterações, apenas de sintaxe, de um SGBD para outro. No *script* gerado, cada ordem de venda tem dois itens de venda associados, cada funcionário e cada cliente estão associados à mesma quantidade de ordens de vendas uns dos outros.

Cada cargo está relacionado a dois funcionários, sendo essas as configurações de aleatoriedade dos *scripts*.

Para cada modelo de banco de dados com seu tamanho definido os testes serão repetidos três vezes, para que se possa ter uma variação dos tempos que será demonstrada em forma de gráfico.

Tanto para o SGBD MySQL e para o PostgreSQL, com 100.000 vendas ou com 1.000.000 de vendas serão executados o comando INSERT (tópico 4.3.1), após será o comando SELECT, que é subdividido em três, mais detalhes no tópico 4.3.2. E por último ser o comando UPDATE(tópico 4.3.3). Esses processos serão repetidos para cada ocorrência e serão representados como Teste 1, Teste 2 e Teste 3.

4.3.1 Inserção de dados

Para os testes de 100.000 registros de ordens de vendas na terceira forma normal o *script* cria 10 registros de funcionários, 5 registros de cargos, 40 de clientes, 100.000 registros de ordens de vendas e mais 200.000 registros de itens de venda, totalizando 300.055 registros.

Nos testes de 100.000registros de ordens de vendas na segunda forma normal o *script* cria 10 registros de funcionários, 40 de clientes, 100.000 registros de ordens de vendas e mais 200.000 registros de itens de venda, totalizando 300.050 registros. Lembrando que esta etapa os dados dos cargos estão juntos com seus respectivos funcionários.

Já para os testes 100.000registros de ordens de vendas na primeira forma normal o *script* cria 100.000 registros de ordens de vendas e mais 200.000 registros de itens de venda, totalizando 300.000 registros. Sendo que todos outros dados dos clientes e funcionários estão inclusas nos registros de ordem de venda.

Para dos testes de 1.000.000 de vendas segue a mesma lógica que as de 100.000registros de ordens de vendas, alterando apenas a quantidade de ordens de

venda que passa a ser de 1.000.000 e as dos itens de vendas que passa para 2.000.000.

Segundo Price (2008), a instrução INSERT é usada para inserir dados nas tabelas dos bancos de dados, passando as referências de em qual tabelas será inserido os registros e quais os dados a serem inseridos.

A estrutura do comando INSERT utilizada foi:

```
INSERT INTO <nome tabela>(<colunas>) VALUES
(<registro 1>),
⋮
(<registro n>);
```

Nessa estrutura de comando INSERT, “<nome tabela>” corresponde a nome da tabela que será inserido os registros. No lugar de “<colunas>” são passadas quais as colunas que iram receber os dados e após VALUES são passados os dados que serão inseridos no banco. Onde “n” é o último registro do comando.

Para esse teste é previsto que quanto menor for o nível de normalização menor será o desempenho para a conclusão do comando.

4.3.2 Seleção de dados

Após há inserção dos dados no banco, o comando a ser executado é o SELECT que será subdividido em três comandos denominados primeira, segunda e terceira seleção. Esses comandos foram realizados para que simulem operações realizadas no cotidiano em bancos de dados de vendas.

Segundo BATTISTI (2005), a instrução SELECT é utilizada para especificar quais os campos de quais tabelas farão parte da consulta, quais os critérios de pesquisa que serão utilizados, qual a ordem de classificação.

Para as seleções dos dados é esperado que a 1FN tenha um melhor desempenho para a conclusão do comando, sendo também é esperado que quando maior o volume de dados maior será a diferença entre a 1FN e às outras formas normais.

4.3.2.1 Primeira seleção de dados

O primeiro SELECT fará com que sejam selecionadas todas as vendas do banco, trazendo as seguintes informações: o código da venda, a data da venda, a data da expedição, o código do cliente, o nome do cliente, o código e o nome do funcionário.

Para cada forma normal tiveram que ser feitas algumas alterações na estrutura do SELECT, para que trouxesse os mesmos dados. Uma vez que o cliente e o funcionário eram tabelas separadas da tabela de ordem de venda na 3FN e na 2FN, mas na 1FN as informações do cliente e do funcionário estão na tabela de ordem de venda.

O comando do primeiro SELECT para a 1FN utilizada foi:

```
SELECT
`ordem_de_venda`.`cod_Venda`,
`ordem_de_venda`.`data_Compra`,
`ordem_de_venda`.`data_Expedicao`,
`ordem_de_venda`.`cod_Cliente`,
` ordem_de_venda`.`nome_Cliente`,
`ordem_de_venda`.`cod_Funcionario`,
` ordem_de_venda`.`nome_Funcionario`
FROM 1fn.ordem_de_venda;
```

O comando faz a seleção das informações no MySQL, para o PostgreSQL foram feitas apenas adaptações de sintaxe.

Para obter os mesmos dados da 1FN selecionados na 2FNena3FN será necessário realizar algumas alterações no mesmo comando da 1FN, como por exemplo, o código do cliente de ser selecionado da tabela cliente e não da tabela ordem de venda como na 1FN e acrescentar dois comandos:

```
INNER JOIN 2fn.cliente ON 2fn.ordem_de_venda.cliente_cod_Cliente = 2fn.cliente.cod_Cliente
INNER JOIN 2fn.funcionario ON 2fn.ordem_de_venda.funcionario_cod_Funcionario =
2fn.funcionario.cod_Funcionario;
```

Essas duas linhas de comandos INNER JOIN foram utilizadas para criar duas junções de tabelas, para que fossem listados os campos ditos acima para o primeiro SELECT, na 3FN e 2FN.

4.3.2.2 Segunda seleção de dados

Para o segundo SELECT serão selecionadas todas as vendas do banco feitas por um funcionário que tenha o cargo igual a (VARCHAR), onde (VARCHAR) representa um texto qualquer, trazendo as seguintes informações: o código da venda, a data da venda, a data da expedição, o código do cliente, o nome do cliente, o códigodo funcionário, nome do funcionário e a descrição do cargo.

O comando do segundo SELECT para a 1FN utilizada foi:

```
SELECT
`ordem_de_venda`.`cod_Venda`,
`ordem_de_venda`.`data_Compra`,
`ordem_de_venda`.`data_Expedicao`,
`ordem_de_venda`.`cod_Cliente`,
` ordem_de_venda`.`nome_Cliente`,
`ordem_de_venda`.`cod_Funcionario`,
` ordem_de_venda`.`nome_Funcionario`,
` ordem_de_venda`.`cargo`
FROM 1fn.ordem_de_venda
WHERE cargo = 'Vendedor';
```

Todos os dados a serem listados na 1FN estão em apenas uma tabela. Já para obter os mesmos dados da 1FN selecionados na 2FN, além de algumas alterações no comando da 1FN, deve ser acrescentado entre o FROM e o WHERE, dois comandos:

```
INNER JOIN 2fn.cliente ON 2fn.ordem_de_venda.cliente_cod_Cliente = 2fn.cliente.cod_Cliente
INNER JOIN 2fn.funcionario ON 2fn.ordem_de_venda.funcionario_cod_Funcionario =
2fn.funcionario.cod_Funcionario
```

Esse dois comandos INNER JOIN foram utilizados para criar duas junções de tabelas, para seleção das informações de cliente, funcionário sendo que as informações de cargo estão na tabela funcionário. Mas cargo é uma tabela

separada de funcionário na 3FN, então deve ser feita mais uma alteração na estrutura acima. Deve ser acrescentado antes de WHERE, o seguinte comando:

```
INNER JOIN 3fn.cargo ON 3fn.funcionario.cargo_cod_Cargo = 3fn.cargo.cod_Cargo
```

Assim 3FN selecionará os mesmos dados que as outras formas normais.

4.3.2.3 Terceira seleção de dados

Já para o terceiro e último o SELECT serão selecionados todos os itens vendidos por um funcionário que tenha o nome igual a (VARCHAR), onde (VARCHAR) representa um texto qualquer, trazendo as seguintes informações: o código da Venda, o código do item, nome do item, Quantidade comprada, o preço unitário e o nome do funcionário.

A estrutura do comando do primeiro SELECT para a 1FN utilizada foi:

```
SELECT
`item_ordem_de_venda`.`ordem_de_venda_cod_Venda`,
`item_ordem_de_venda`.`descr_Item`,
`item_ordem_de_venda`.`qtde_Item`,
`item_ordem_de_venda`.`preco_Item`,
` ordem_de_venda`.`nome_Funcionario`
FROM 1fn.item_ordem_de_venda
INNER JOIN 1fn.ordem_de_venda ON 1fn.item_ordem_de_venda.ordem_de_venda_cod_Venda =
1fn.ordem_de_venda.cod_Venda
WHERE 1fn.ordem_de_venda.nome_funcionario = (VARCHAR);
```

O comando faz a seleção das informações citadas acima para o primeiro SELECT na 1FN.

Para que a 2FN e a 3FN tenham os mesmos dados que na 1FN é necessário,além de algumas alterações no comando da 1FN, acrescentar mais um INNER JOIN, antes do WHERE:

```
INNER JOIN 3fn.funcionario ON 3fn.ordem_de_venda.funcionario_cod_Funcionario =
3fn.funcionario.cod_Funcionario
```

Essa linha de comando INNER JOIN fará a junçãocom a tabela funcionário, já que na 2FN e 3FN os dados dos funcionários estão em uma tabela separada.

4.3.3 Alteração dos dados

O comando UPDATE será utilizado para fazer à alteração do cargo e habilidades de um funcionário que tenha o código igual a (INT), onde (INT) representa um número inteiro qualquer e (valor) será o novo valor que a coluna vai receber. Aestruturado comando UPDATE utilizada foi:

UPDATE <tabela 1> SET <coluna>=(valor) WHERE <coluna>=(INT)

Segundo Price (2008), o comando UPDATE é a instrução usada para os dados das tabelas, com um comando se pode alterar uma ou várias linhas das tabelas.

A expectativa para esse teste é que a 1FN terá um pior desempenho, devido a quantidade de dados que terá que ser alterada, sendo que nas outras formas normais para que seja feita a mesma alterações que na 1FN a quantidade de dados à ser alterada deve ser menor.

4.4 Coletas dos Dados

Todos os testes foram realizados cinco vezes para os mesmos SGBDs em cada forma normal. Foram descartados o maior e o menor tempo dos testes, ficando apenas com três resultados para cada tipo de teste, o Teste 1, Teste 2 e Teste 3 são os mesmos testes, para o mesmo comando, no mesmo SGBD,que serão demonstrados em gráficos.

5 RESULTADOS OBTIDOS

Nesse capítulo serão apresentados os resultados obtidos dos testes nos dois SGBDs, realizando os comparativos de desempenho de cada comando utilizado para cada forma normal em cada SGBD. Ao final será realizado um comparativo entre os resultados dos dois SGBDs.

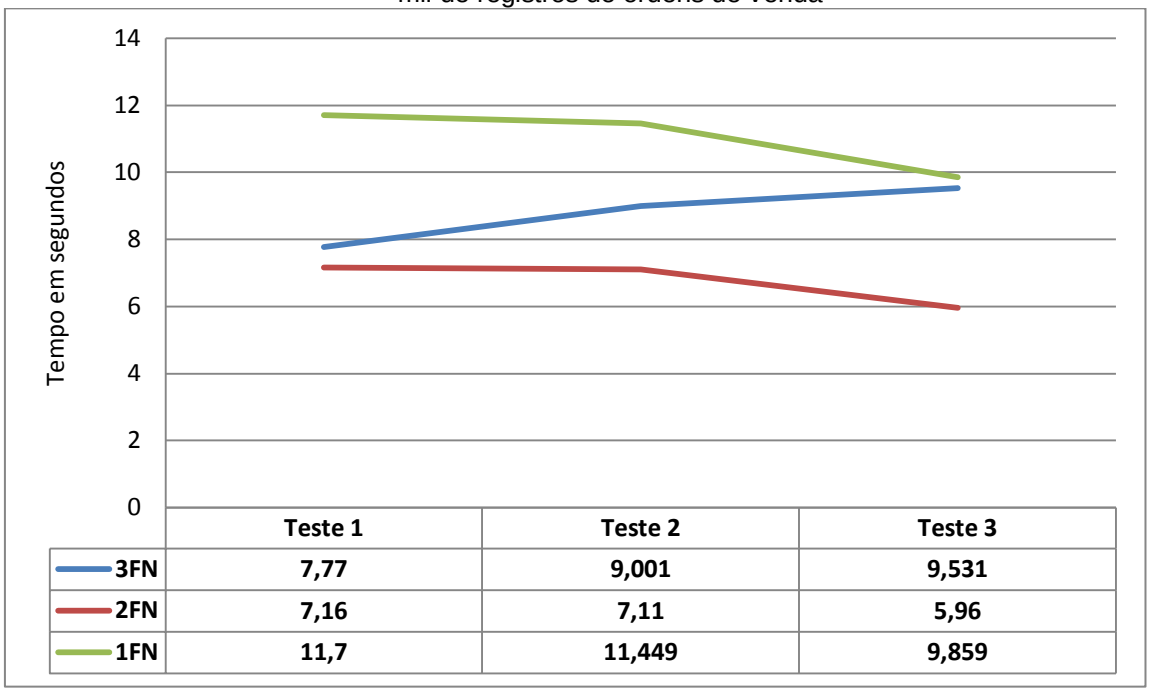
5.1 Resultados no MySQL

Nos tópicos 5.1.1 e 5.1.2, serão apresentados os comparativos de desempenho de cada comando utilizado para cada forma normal, utilizadas no MySQL, separados pelas quantidades de registros de ordens de vendas que foram de 100.000 e 1.000.000.

5.1.1 Resultados dos testes com 100.000 ordens de vendas

Para cada comando utilizado foi criado um gráfico que terá o desempenho de cada forma normal, com tempo em segundos, para a visualização dos resultados no MySQL com 100.000 ordens de vendas.

GRÁFICO 1 - Comparação entre as tabelas do MySQL na 3FN, 2FN e 1FN para a inserção de cem mil de registros de ordens de venda



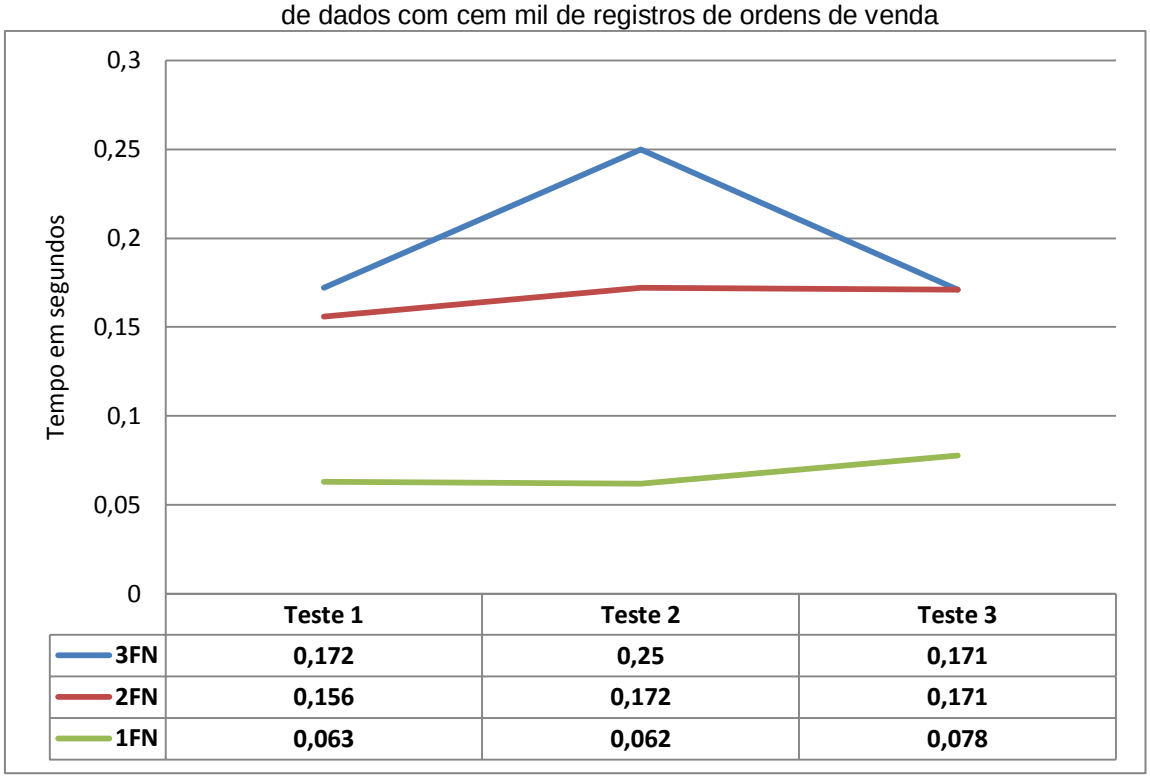
	Teste 1	Teste 2	Teste 3
3FN	7,77	9,001	9,531
2FN	7,16	7,11	5,96
1FN	11,7	11,449	9,859

Fonte: Pesquisa do autor, 2013

No GRÁFICO 1, a diferença no desempenho das inserções dos registros às tabelas de 100.000 registros de vendas entre a 1FN e a 2FN foi a que apresentou maior diferença, a 1FN chegando a ser 38,71% mais lenta que a 2FN e 20,31% mais lenta que a 3FN. Isso ocorreu, pois a 1FN, mesmo precisando de menoscomandos INSERT para inserir os mesmos dados que as outras formas normais em cada comando se tinham mais colunas a serem preenchidas.

Um exemplo é que nas outras formas normais, para se inserir um cliente é preciso um comando a mais, porém na ordem de venda será apenas registrado o código do cliente, que é bem mais rápido do que inserir todos os dados do cliente em cada ordem de venda, já que se trata de uma grande quantidade de dados, que é o que ocorre na 1FN. É o que ocorre ao contrário na 3FN onde é criado um INSERT a mais para cada cargo a ser registrado, cada funcionário recebe apenas o código do cargo, como se trata de poucos dados acaba se tornando mais lento ocasionando uma diferença de 23%, onde a 2FN foi mais rápida que a 3FN.

GRÁFICO 2 - Comparação entre as tabelas do MySQL na 3FN, 2FN e 1FN para a primeira seleção de dados com cem mil de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	0,172	0,25	0,171
2FN	0,156	0,172	0,171
1FN	0,063	0,062	0,078

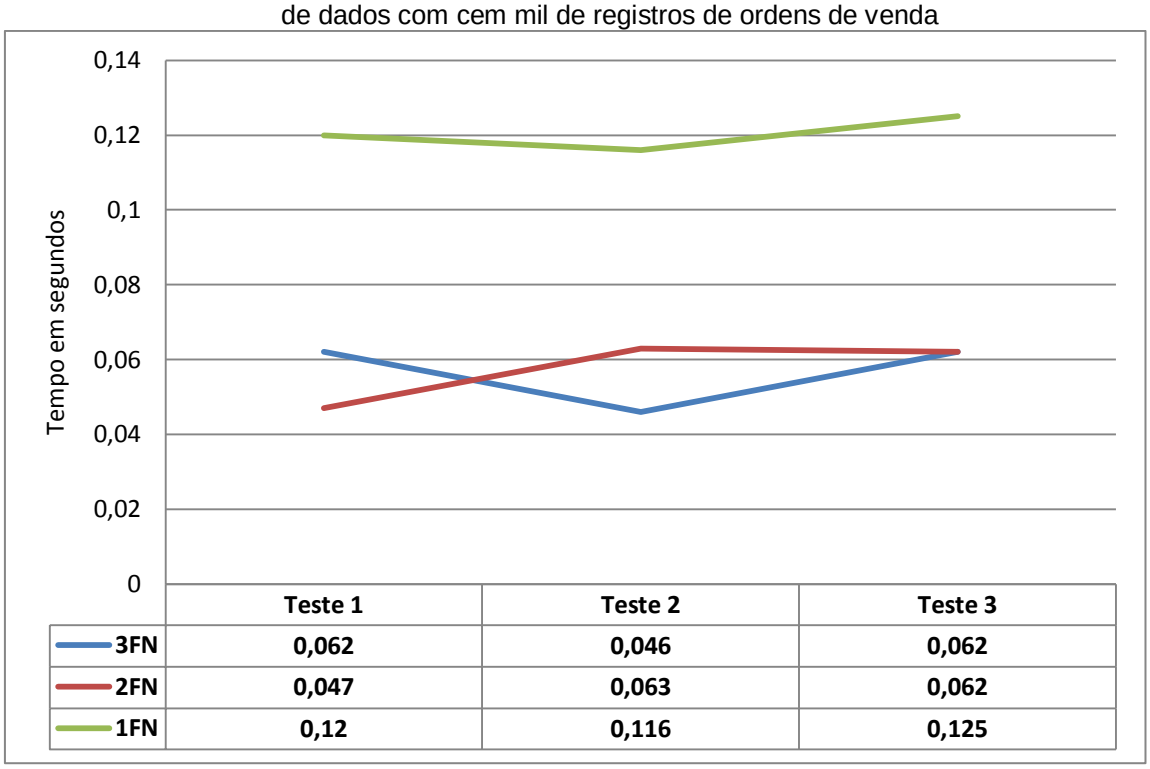
Fonte: Pesquisa do autor, 2013

O GRÁFICO 2 traz os resultados do teste da primeira seleção de dados. Esse teste foi realizado para demonstrar o desempenho de cada banco para uma seleção de todos os registros de venda, que nesse caso retornaria 100.000 linhas, tópico 4.3.2.1 desse mesmo trabalho.

Para esse tipo de testes realizados tiveram pontos onde a 3FN e a 2FNforam muito parecidas, mas em média a 3FN foi 15,85% mais lenta que a 2FN.

Já a 1FN teve um resultado muito superior que as outras formas normais retornando os mesmos dados. A 2FN e a 3FN foram equivalentes, porém foram em média 62% mais lentas que a 1FN. Isso ocorreu devido que alguns dados necessários para a seleção desejada dos registros na 2FN e na 3FN se encontravam em outras tabelas, para isso era necessário a utilização de JOIN, que são junções entre as tabelas,mas para seleção dos mesmos dados na 1FN não era necessário nenhuma utilização de JOIN, pois todos os dados a serem selecionados estavam em uma tabela apenas o que faz uma grande diferença desempenho.

GRÁFICO 3 - Comparação entre as tabelas do MySQL na 3FN, 2FN e 1FN para a segunda seleção de dados com cem mil de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	0,062	0,046	0,062
2FN	0,047	0,063	0,062
1FN	0,12	0,116	0,125

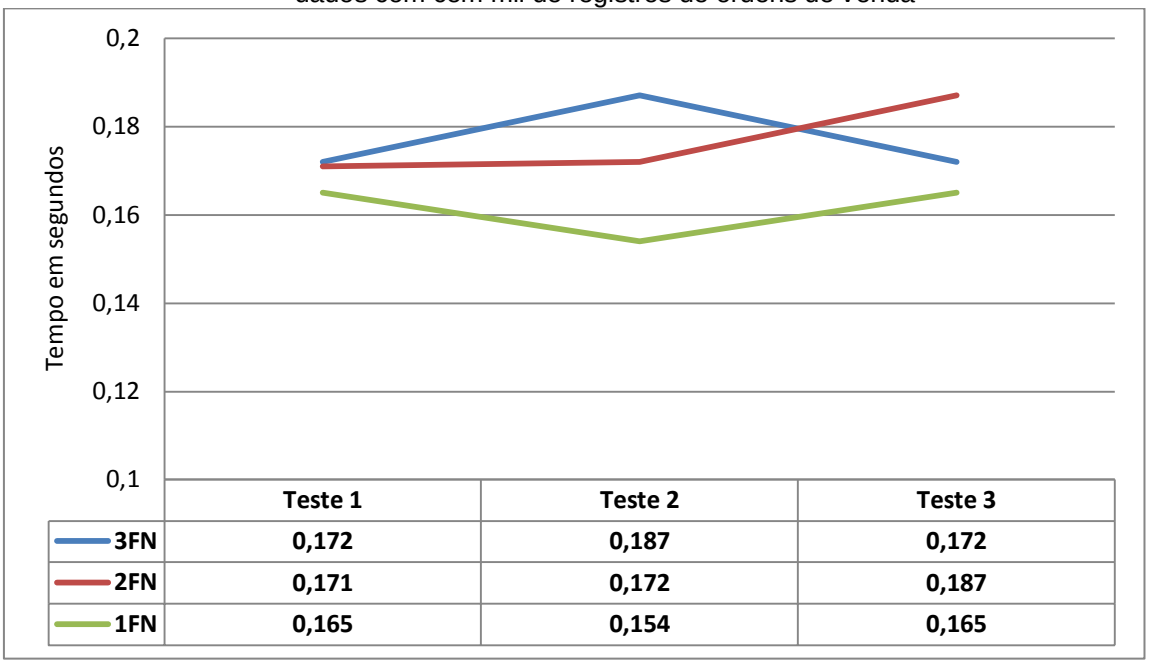
Fonte: Pesquisa do autor, 2013

Esse segundo teste foi realizado para demonstrar o desempenho de cada banco para uma seleção de todos os registros de venda, porém com um tipo de filtro, tópico 4.3.2.2 desse mesmo trabalho, que retornava 20.000 linhas. Os resultados estão demonstrados através do GRÁFICO 3.

Ao contrário do primeiro tipo se seleção nesse segundo tipo de teste a 1FN foi a que teve pior resultado sendo 46,67% em média mais lenta que as outras duas formas normais. Sendo que a 2FN e a 3FN obtiveram praticamente o mesmo desempenho para esse tipo de teste.

Esse resultado pode ser explicado devido à utilização da cláusula WHERE, que filtra quais os registros que serão listados, que para esse caso o filtro é que o cargo, que é uma dependência funcional transitiva na 1FN, seja igual um valor texto (VARCHAR) passado. Para a 2FN e 3FN essa informação está vinculada a o código do funcionário fazendo com que a comparação seja feita por um valor inteiro (INT) em todos os registros de vendas. E bem mais rápido passar por 100.000 registros comparando um valor inteiro de um campo chave, mesmo utilizando JOIN como na 2FN e a 3FN, do que um texto em um campo de dependência transitiva, que é o que ocorre na 1FN fazendo com que se tenha uma perca de desempenho.

GRÁFICO 4 - Comparação entre as tabelas do MySQL na 3FN, 2FN e 1FN para a terceira seleção de dados com cem mil de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	0,172	0,187	0,172
2FN	0,171	0,172	0,187
1FN	0,165	0,154	0,165

Fonte: Pesquisa do autor, 2013

O último teste de seleção de dados foi realizado para demonstrar o desempenho de cada banco para uma seleção de todos os registros de venda com outro tipo filtro, tópico 4.3.2.3 desse mesmo trabalho, que retornava 20.000 linhas. Os resultados estão demonstrados através do GRÁFICO 4.

Assim como no segundo teste, a 2FN e a 3FN também obtiveram praticamente o mesmo desempenho no terceiro teste, só que desta vez a 1FN foi a que teve melhor resultado sendo 9,93% em média mais rápida que as outras duas formas normais como demonstrado no GRÁFICO 4.

Também com a utilização da cláusula WHERE, a diferença de desempenho na 1FN aconteceu porque a comparação não foi feita de um campo de dependência funcional transitiva, mas sim de um campo que era comum para todas as formas normais.

GRÁFICO 5 - Comparação entre as tabelas do MySQL na 3FN, 2FN e 1FN para a alteração de dados com cem mil de registros de ordens de venda

	Teste 1	Teste 2	Teste 3
3FN	0,031	0,043	0,015
2FN	0	0	0,028
1FN	0,62	0,421	0,421

Fonte: Pesquisa do autor, 2013

Esse teste foi realizado para demonstrar o desempenho de cada banco para uma alteração de alguma informação dos registros de vendas, tópico 4.3.3 desse mesmo trabalho. Os resultados estão demonstrados através do GRÁFICO 5.

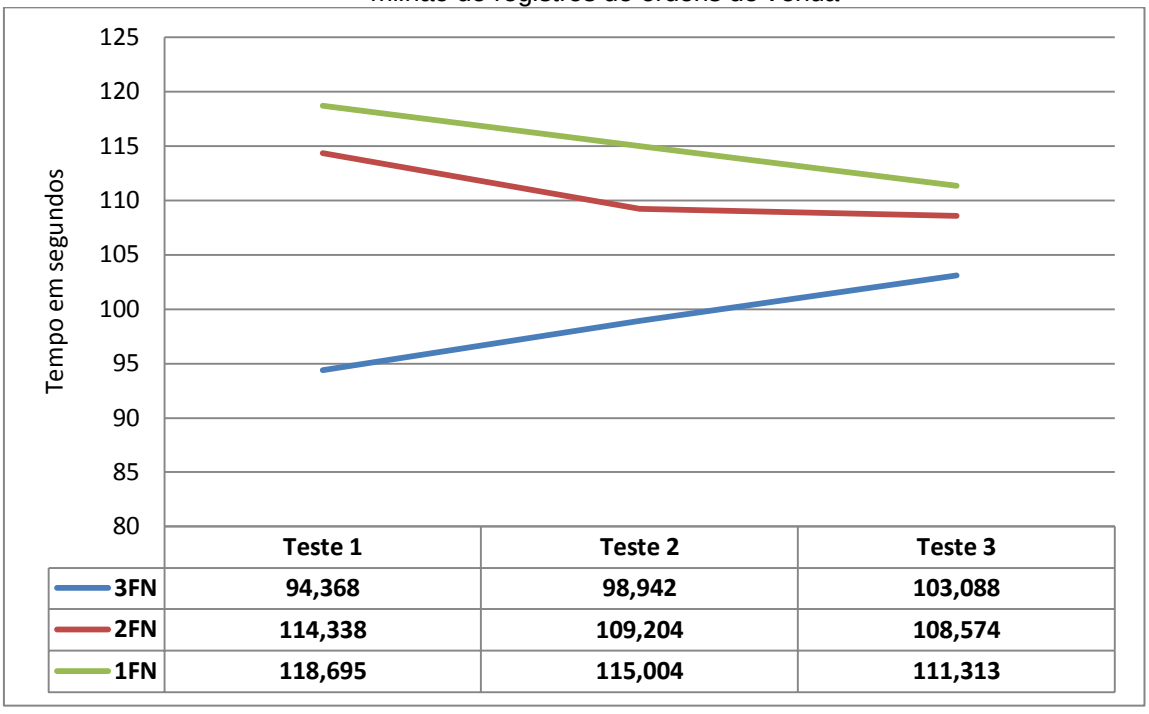
A 2FN foi 222,32% mais rápida que a 3FN. A 1FN teve um resultado muito mais lento, foi cerca de 16 vezes mais lenta que a 3FN e mais de 50 vezes mais lenta que a 2FN.

Para a alteração dos mesmos dados na 3FN foram alteradas em duas tabelas 15 linhas totais. Para a 2FN foram alteradas em uma tabela 10 linhas totais. E na 1FN foi em uma tabela, mas foram precisos alterar 10.000 linhas. O que nitidamente levou a esse resultado.

5.1.2 Resultados dos testes com 1.000.000 ordens de vendas

Para cada comando utilizado foi criado um gráfico que terá o desempenho de cada forma normal, com tempo em segundos, para a visualização dos resultados no MySQL com 1.000.000 ordens de vendas.

GRÁFICO 6 - Comparação entre as tabelas do MySQL na 3FN, 2FN e 1FN para a inserção de um milhão de registros de ordens de venda

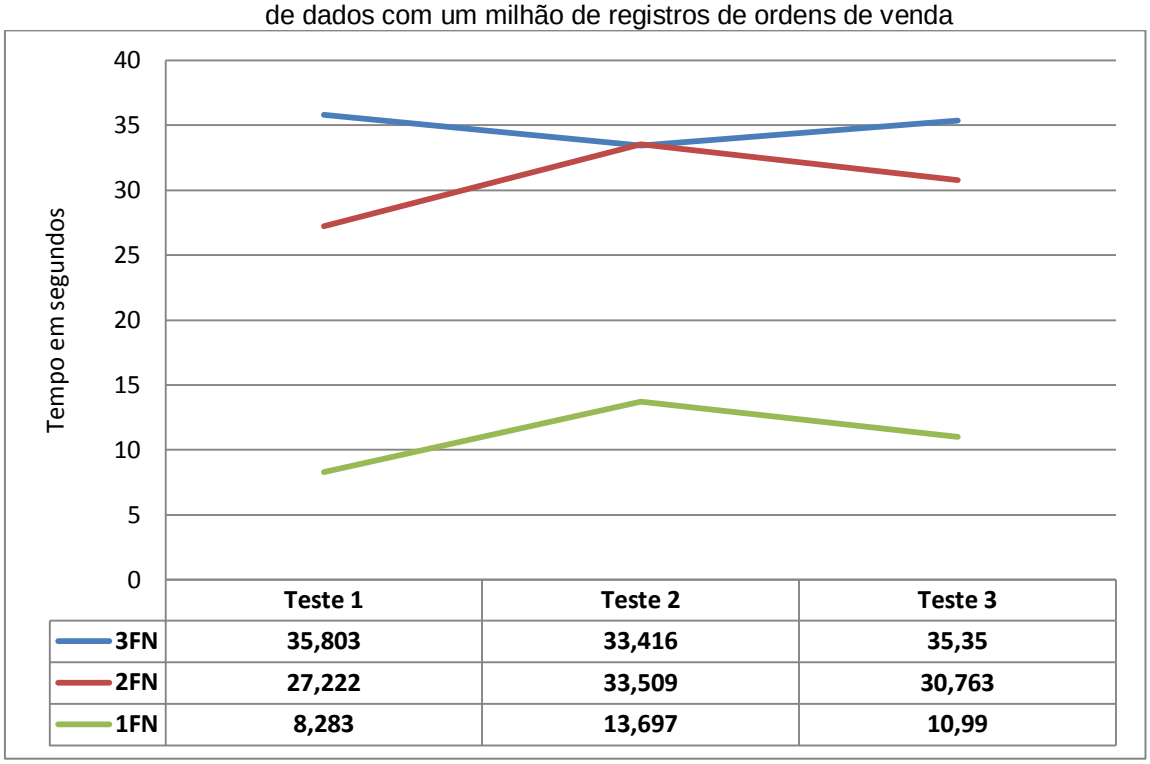


	Teste 1	Teste 2	Teste 3
3FN	94,368	98,942	103,088
2FN	114,338	109,204	108,574
1FN	118,695	115,004	111,313

Fonte: Pesquisa do autor, 2013

Para a tabela de 1.000.000 de registros de vendas a diferença no desempenho das inserções dos registros a 1FN foi a que apresentou maior desempenho, representado pelo GRÁFICO 6. A 1FN foi 16,42% mais rápida que a 3FN e 12,04% mais rápida que a 2FN e a 3FN foi 3,78% mais lenta que a 2FN. Isso ocorreu, pois com um grande volume de dados, a 1FN precisou de menos comandos INSERT para inserir os mesmos dados que as outras formas normais e em cada comando se tinham mais colunas a serem preenchidas, mesmos assim ela teve um desempenho bem melhor que as outras formas normais.

GRÁFICO 7 - Comparação entre as tabelas do MySQL na 3FN, 2FN e 1FN para a primeira seleção de dados com um milhão de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	35,803	33,416	35,35
2FN	27,222	33,509	30,763
1FN	8,283	13,697	10,99

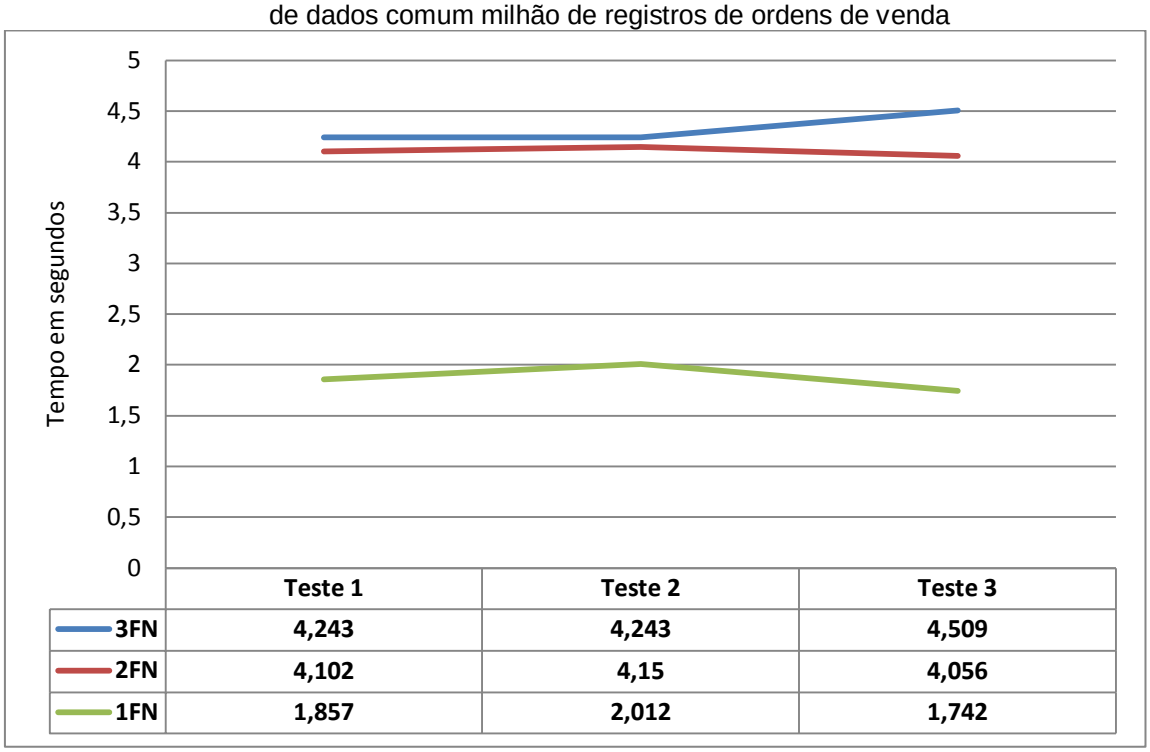
Fonte: Pesquisa do autor, 2013

Como realizado anteriormente para o banco com 100.000 registros de venda também foi realizado esse teste para demonstrar o desempenho de cada banco para uma seleção de todos os registros de venda do banco com 1.000.000 de registros de venda, que nesse caso retornaria 1.000.000 linhas, tópico 4.3.2.1 desse mesmo trabalho. Os resultados estão demonstrados através do GRÁFICO 7.

Para esse tipo de teste realizado, assim como no banco com menos dados, tiveram pontos onde a 3FN e a 2FN foram muito parecidas, mas em média a 2FN foi 14,28% mais rápida que a 3FN, resultado muito parecido também com o resultado do banco com menos registros.

O resultado obtido pela 1FN foi muito superior às outras formas normais retornando os mesmos dados, assim como no teste com menos registros. A 1FNfoi 177,50% mais rápida que a 2FNe 217,16% mais rápida que a 3FN.

GRÁFICO 8 - Comparação entre as tabelas do MySQL na 3FN, 2FN e 1FN para a segunda seleção de dados comum milhão de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	4,243	4,243	4,509
2FN	4,102	4,15	4,056
1FN	1,857	2,012	1,742

Fonte: Pesquisa do autor, 2013

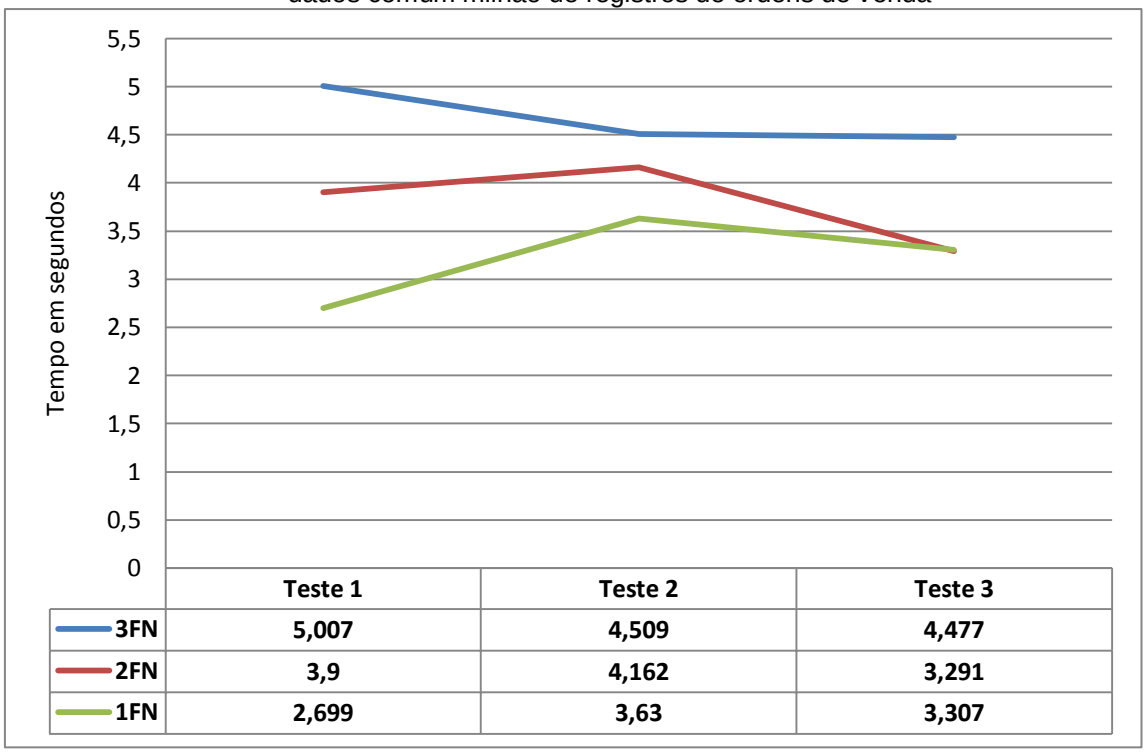
Também foi realizado o segundo para demonstrar o desempenho de cada banco para uma seleção de todos os registros de venda do banco com 1.000.000 de registros de vendas, com o mesmo tipo de filtro da tabela com menos registros, tópico 4.3.2.2 desse mesmo trabalho, que retorna 200.000 linhas. Os resultados estão demonstrados através do GRÁFICO 8.

Ao contrário do teste com menos registro, nesse teste a 1FN foi a que teve melhor resultado sendo 119% em média mais rápida que as outras duas formas normais.

Sendo que a 2FN e a 3FN obtiveram resultados muito parecidos de desempenho para esse tipo de teste.

Esse resultado pode ser explicado devido à utilização da cláusula WHERE, que filtra quais os registros que serão listados, que para esse caso o filtro é que o cargo, que é uma dependência funcional transitiva na 1FN, seja igual um valor texto (VARCHAR) passado. Para a 2FN e 3FN essa informação está vinculada a o código do funcionário fazendo com que a comparação seja feita por um valor inteiro (INT) em todos os registros de vendas. É bem mais lento passar por 1.000.000 registros comparando um valor inteiro de um campo chave, utilizando JOIN como na 2FN e a 3FN, do que um texto em um campo de dependência transitiva, que é o que ocorre na 1FN fazendo com que se tenha um ganho de desempenho.

GRÁFICO 9 - Comparação entre as tabelas do MySQL na 3FN, 2FN e 1FN para a terceira seleção de dados comum milhão de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	5,007	4,509	4,477
2FN	3,9	4,162	3,291
1FN	2,699	3,63	3,307

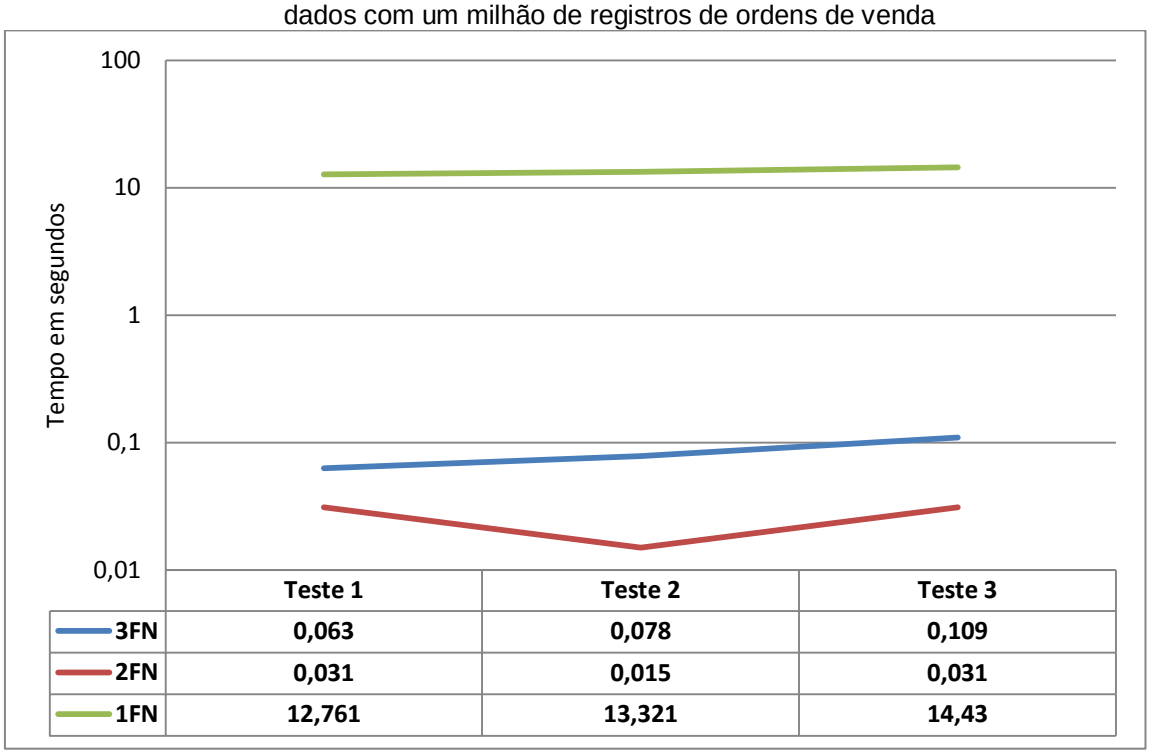
Fonte: Pesquisa do autor, 2013

O último teste de seleção de dados foi realizado também para a tabela com 1.000.000 de registros de vendas, tópico 4.3.2.3 desse mesmo trabalho, que retorna uma seleção com 200.000 linhas, demonstrado no GRÁFICO 9.

Dessa vez a 1FN foi mais rápida chegando à média de 17,80% do que 2FN e até 45,20% do que a 3FN. Mas no teste 3 a 1FN foi até um pouco mais lenta que a 2FN sendo 0,016 segundos mais lenta.

Também com a utilização da cláusula WHERE, a diferença de desempenho na 1FN aconteceu porque a comparação não foi feita de um campo de dependência funcional transitiva, mas sim de um campo que era comum para todas as formas normais. Prejudicando ainda mais o desempenho da 3FN com um banco com volume maior de dados.

GRÁFICO 10 - Comparação entre as tabelas do MySQL na 3FN, 2FN e 1FN para a alteração de dados com um milhão de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	0,063	0,078	0,109
2FN	0,031	0,015	0,031
1FN	12,761	13,321	14,43

Fonte: Pesquisa do autor, 2013

Esse teste foi realizado para demonstrar o desempenho de cada banco para uma alteração de alguma informação dos registros de vendas, tópico 4.3.3 desse mesmo trabalho. Os resultados estão demonstrados através do GRÁFICO 10.

O desempenho entre a 2FN e a 3FN foi bem parecido com o do banco com menos registros, a 2FN foi 232% mais rápida que a 3FN. Já 1FN teve um resultado ainda pior devido ao tamanho do banco, foi cerca de 165 vezes mais lenta que a 3FN e mais de 500 vezes mais lenta que a 2FN.

Para a alteração dos mesmos dados na 3FN foram alteradas em duas tabelas 15 linhas totais. Para a 2FN foram alteradas em uma tabela 10 linhas totais. E na 1FN foi em uma tabela, mas foram precisos alterar 100.000 linhas. O que provocou a piora do desempenho da 1FN.

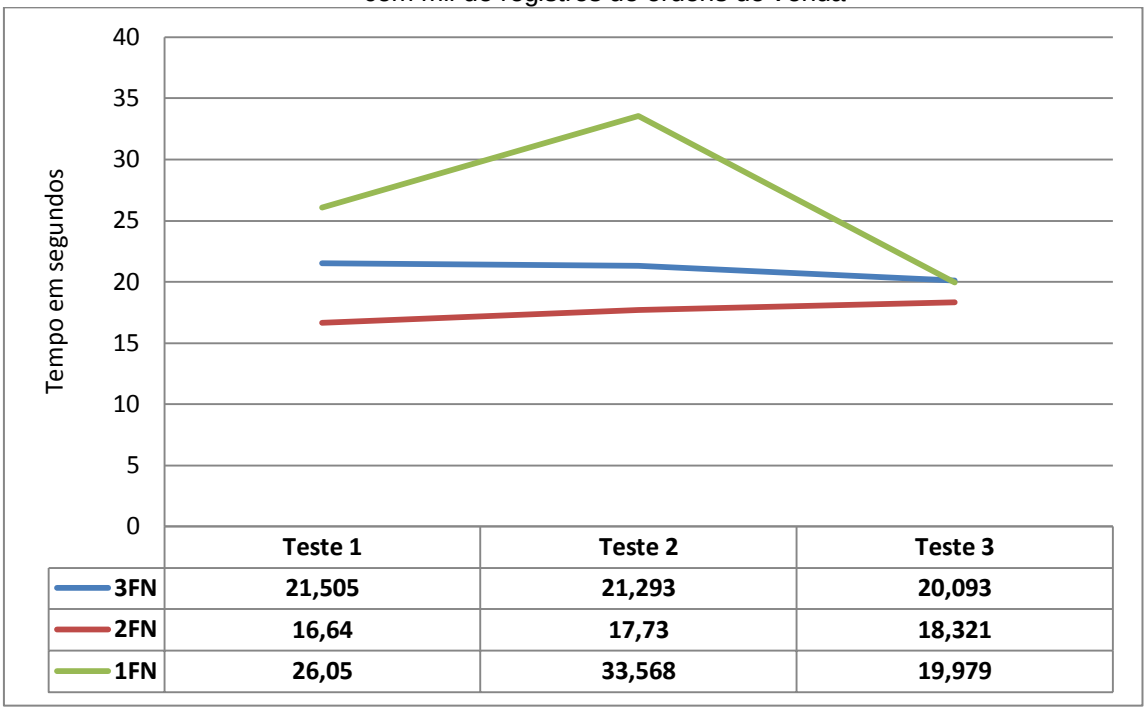
5.2 Resultados Obtidos no PostgreSQL

Nos tópicos 5.2.1 e 5.2.2, serão apresentado os comparativos de desempenho de cada comando utilizado para cada forma normal, utilizadas no PostgreSQL, separados pelas quantidades de registros de ordens de vendas que foram de 100.000 e 1.000.000.

5.2.1 Resultados dos testes com 100.000 ordens de vendas

Para cada comando utilizado foi criado um gráfico que terá o desempenho de cada forma normal, com tempo em segundos, para a visualização dos resultados no PostgreSQL com 100.000 ordens de vendas.

GRÁFICO 11 - Comparação entre as tabelas do PostgreSQL na 3FN, 2FN e 1FN para a inserção de cem mil de registros de ordens de venda



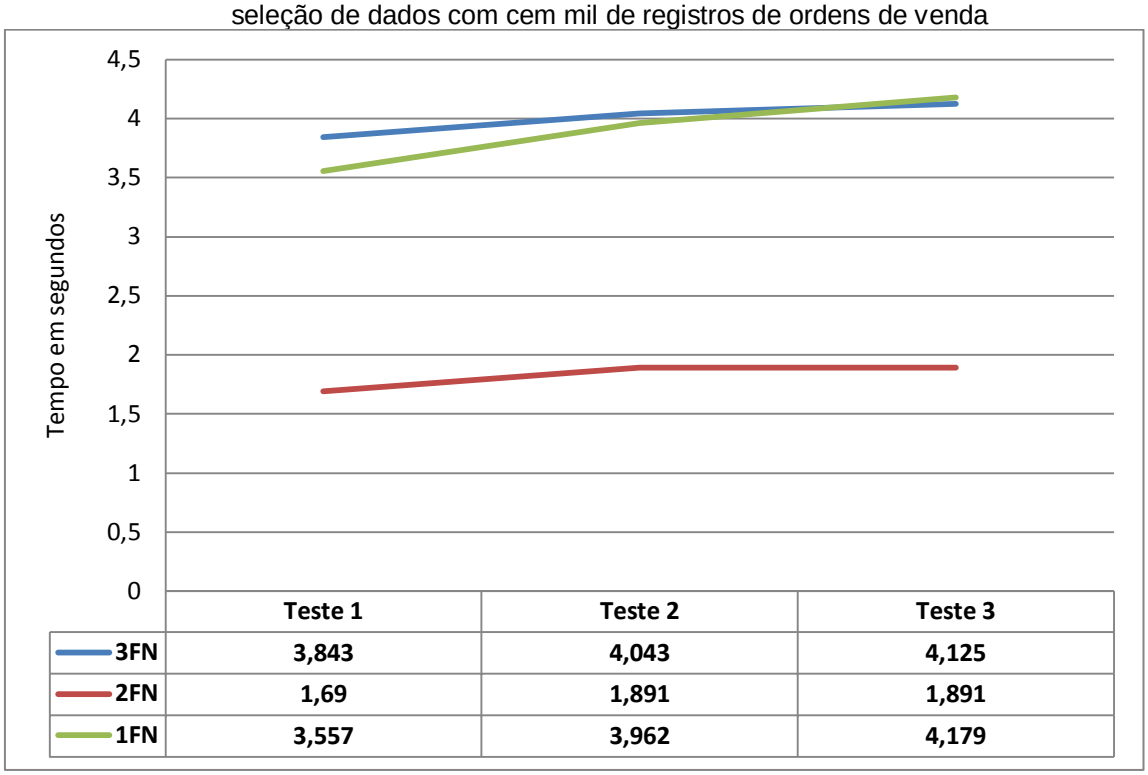
	Teste 1	Teste 2	Teste 3
3FN	21,505	21,293	20,093
2FN	16,64	17,73	18,321
1FN	26,05	33,568	19,979

Fonte: Pesquisa do autor, 2013

No GRÁFICO 11, não muito diferente dos resultados de desempenho das inserções dos registros das tabelas de 100.000 registros de vendas do MySQL, no PostgreSQL a maior diferença apresentada foi entre a 1FN e a 2FN, a 1FN chegando a ser 33,80% mais lenta que a 2FN e 20,99% mais lenta que a 3FN decorrente do mesmo motivo que no MySQL. Já na a diferença entre a 2FN e a 3FN foi de 19,35%, onde a 2FN foi a mais rápida.

Para a inserção dos registros no PostgreSQL o tempo gasto para a conclusão dos comandos foi muito superior ao tempo do MySQL, mas os resultados dos desempenhos entre as formas normais foi bem parecido entre os dois SGBDs onde a 2FN foi a que teve o melhor resultado, seguido pela 3FN e com o pior tempo a 1FN.

GRÁFICO 12 - Comparação entre as tabelas do PostgreSQL na 3FN, 2FN e 1FN para a primeira seleção de dados com cem mil de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	3,843	4,043	4,125
2FN	1,69	1,891	1,891
1FN	3,557	3,962	4,179

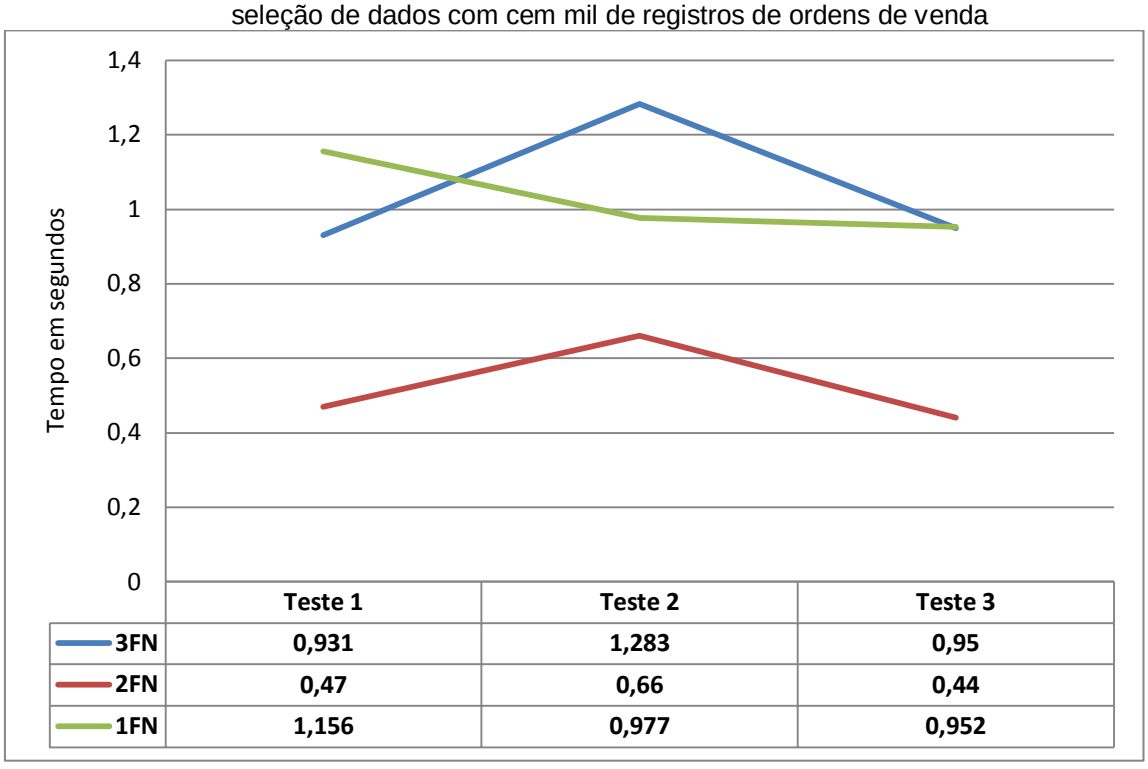
Fonte: Pesquisa do autor, 2013

Esse teste foi realizado para demonstrar o desempenho de cada banco para uma seleção de todos os registros de venda, que nesse caso retornaria 100.000 linhas, tópico 4.3.2.1desse mesmo trabalho. Os resultados estão demonstrados através do GRÁFICO 12.

Ao contrário do mesmo teste realizado no MySQL, esse teste realizado no PostgreSQL a 2FN foi a mais rápida chegando a ser 116,66% mais rápida que as outras formas normais, que para esse tipo de testes realizados no PostgreSQL tiveram pontos onde a 3FN e a 1FN foram muito próximos.

O resultado no PostgreSQL teve a inversão de posição entre a 1FN e a 2FN onde no MySQL tivemos a 1FN tendo um resultado muito superior às outras formas normais. No PostgreSQL o tempo das pesquisas foi muito maior do que no MySQL, mas a 2FN foi a que teve um resultado muito superior às outras formas normais.

GRÁFICO 13 - Comparação entre as tabelas do PostgreSQL na 3FN, 2FN e 1FN para a segunda seleção de dados com cem mil de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	0,931	1,283	0,95
2FN	0,47	0,66	0,44
1FN	1,156	0,977	0,952

Fonte: Pesquisa do autor, 2013

O segundo teste foi realizado para demonstrar o desempenho de cada banco para uma seleção de todos os registros de venda, porém com um tipo de filtro, tópico 4.3.2.2 desse mesmo trabalho, que retornava 20.000 linhas. Os resultados estão demonstrados através do GRÁFICO 13.

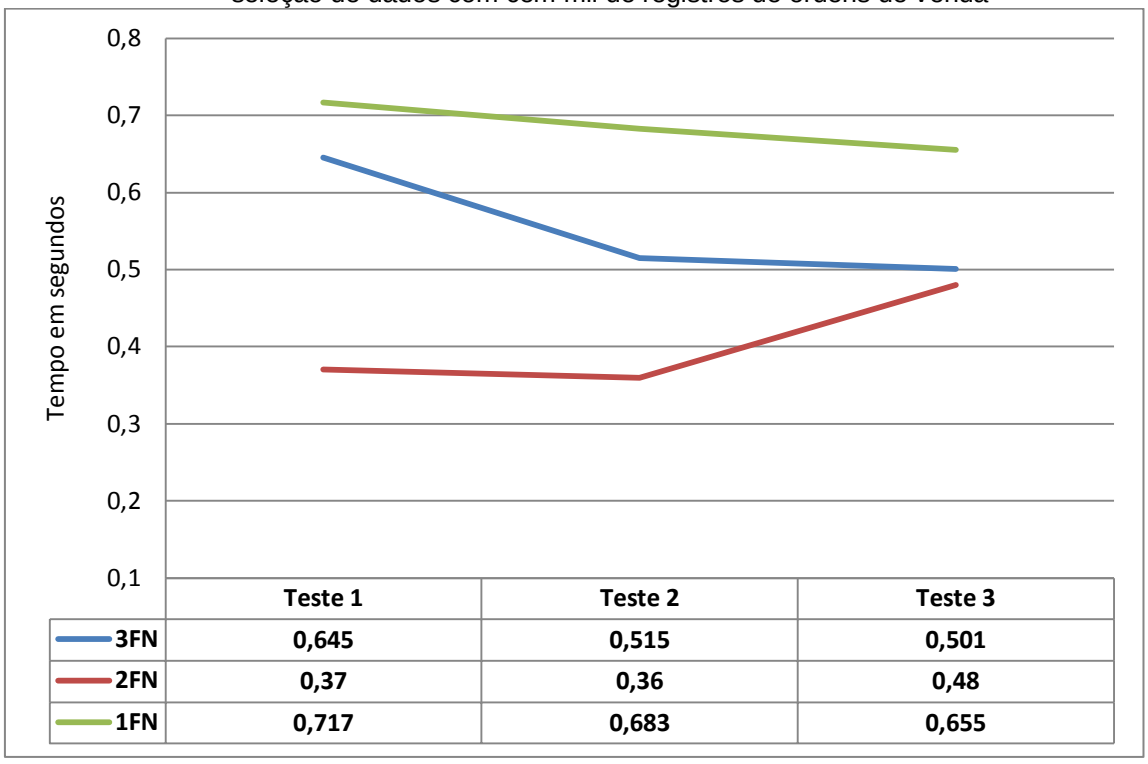
Nesse segundo tipo de teste a 2FN também teve o melhor resultado sendo 99,50% em média mais rápida que as outras duas formas normais. Sendo que a 1FN e a

3FN obtiveram praticamente o mesmo desempenho para esse tipo de teste no PostgreSQL.

Nesse teste realizado os resultados dos desempenhos no PostgreSQL também teve o tempo das pesquisas maior que o tempo do MySQL.

Tanto no MySQL, quanto no PostgreSQL, a 2FN teve um desempenho melhor, porém no MySQL a 3FN teve um desempenho tão bom quanto o da 2FN o que não ocorreu no PostgreSQL, onde a 3FN teve o pior resultado que foi bem parecido com o da 1FN, que foi mais lento nos dois SGBDs.

GRÁFICO 14 - Comparação entre as tabelas do PostgreSQL na 3FN, 2FN e 1FN para a terceira seleção de dados com cem mil de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	0,645	0,515	0,501
2FN	0,37	0,36	0,48
1FN	0,717	0,683	0,655

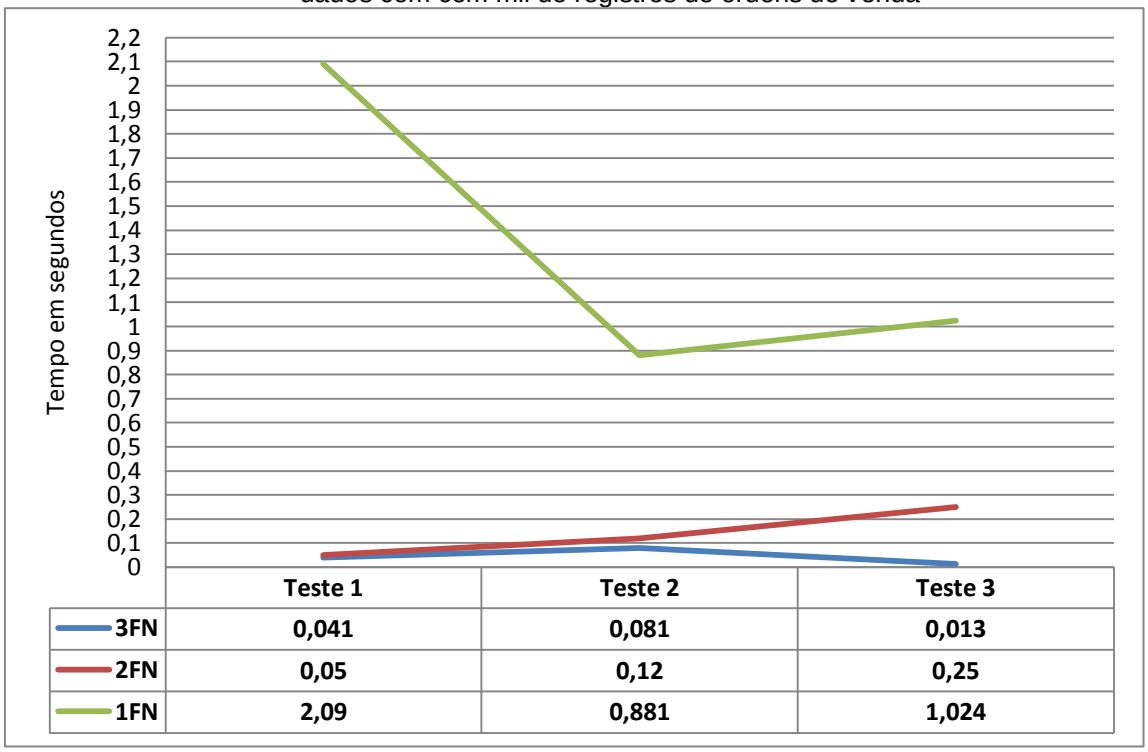
Fonte: Pesquisa do autor, 2013

A última seleção de dados testada foi realizada para demonstrar o desempenho de cada banco para uma seleção de todos os registros de venda com outro tipo filtro, tópico 4.3.2.3 desse mesmo trabalho, que retornava 20.000 linhas. Os resultados estão demonstrados através do GRÁFICO 14.

Assim como todos os testes de seleção de dados no PostgreSQL, a 2FN obteve o melhor resultado sendo 37,22% mais rápida que a 3FN e 69,97% mais rápida que a 1FN, que foi a que obteve o pior resultado e foi 19,27% mais lenda que a 3FN.

Os tempos de execução dos comandos no PostgreSQL para esse e para todos os testes realizados foram superiores ao do MySQL. A diferença entre os resultados do PostgreSQL e do MySQL para esse teste está na 1FN de foi a mais rápida no MySQL e a mais lenta no PostgreSQL que teve a 2FN como mais rápida.

GRÁFICO 15 - Comparação entre as tabelas do PostgreSQL na 3FN, 2FN e 1FN para a alteração de dados com cem mil de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	0,041	0,081	0,013
2FN	0,05	0,12	0,25
1FN	2,09	0,881	1,024

Fonte: Pesquisa do autor, 2013

Esse teste, tópico 4.3.3 deste trabalho, foi realizado para demonstrar o desempenho de cada banco para uma alteração de alguma informação dos registros de vendas. Os resultados estão demonstrados através do GRÁFICO 15.

A 2FN e a 3FN obtiveram praticamente os mesmos resultados e foram cerca de 15 vezes mais rápidas do que a 1FN.

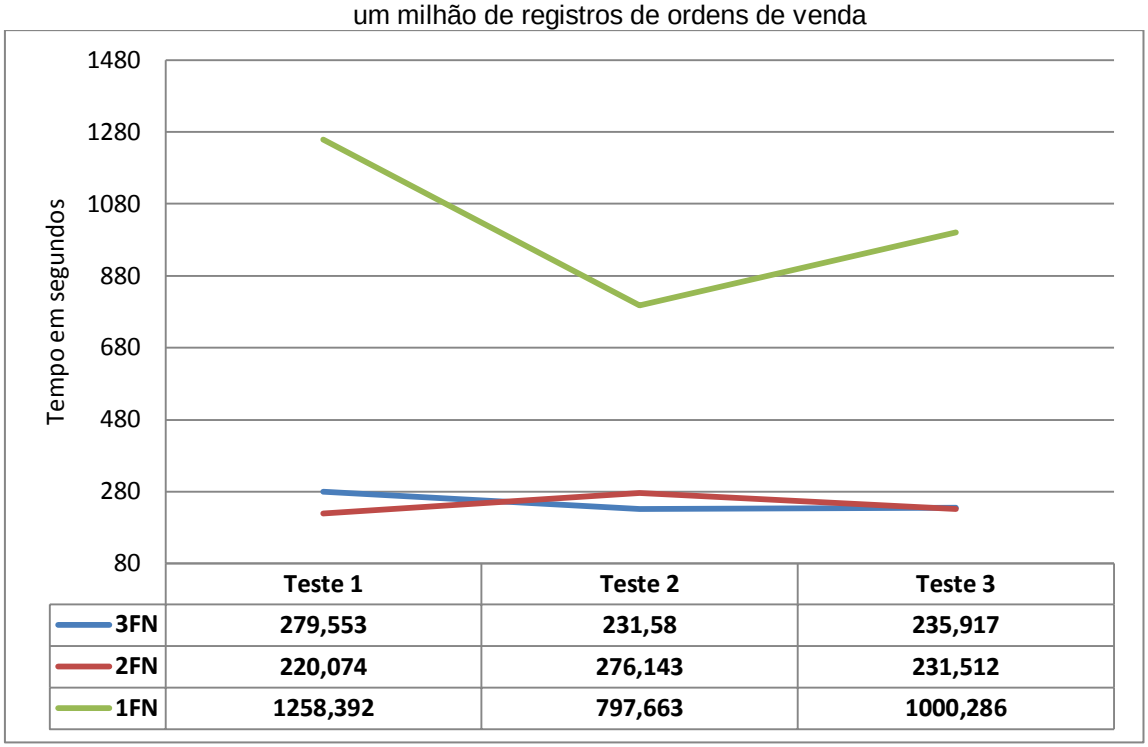
Assim como o teste no MySQL, para a alteração dos mesmos dados no PosgtreSQL na 3FN foram alteradas em duas tabelas 15 linhas totais, 10 linhas totais um uma tabela na 2FN e na 1FN foi em uma tabela nas foram preciso alterar 10.000 linhas. O que nitidamente levou a esse resultado.

5.2.2 Resultados dos testes com 1.000.000 ordens de vendas

Para cada comando utilizado foi criado um gráfico que terá o desempenho de cada forma normal, com tempo em segundos, para a visualização dos resultados no PostgreSQL com 1.000.000 ordens de vendas.

Em todos os resultados obtidos dos testes no PostgreSQL com 1.000.000 de registros de vendas, assim como ocorreu com os resultados dos testes no PostgreSQL com 100.000 registros de vendas, os tempos de execução dos comandos foi muito superior aos tempos do MySQL, mas o resultado de desempenho entre as formas normais foram parecidos havendo algumas alterações que serão mostradas abaixo com o comparativo dos gráficos.

GRÁFICO 16 - Comparação entre as tabelas do PostgreSQLna 3FN, 2FN e 1FN para a inserção de um milhão de registros de ordens de venda

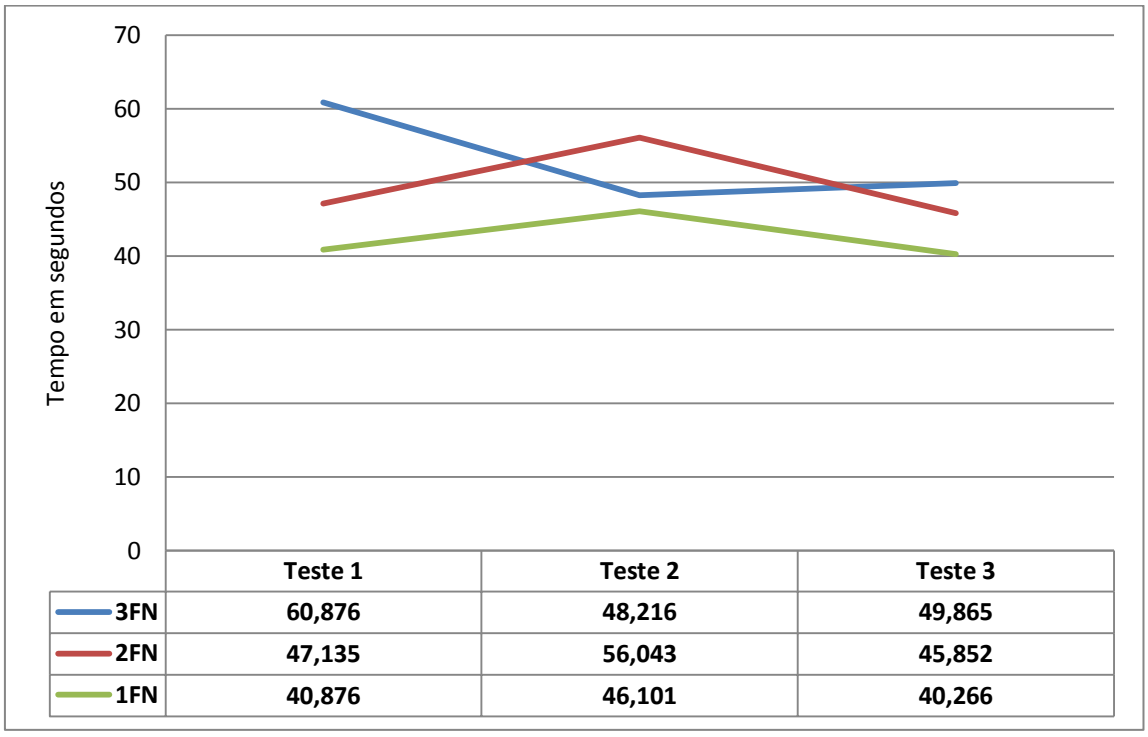


	Teste 1	Teste 2	Teste 3
3FN	279,553	231,58	235,917
2FN	220,074	276,143	231,512
1FN	1258,392	797,663	1000,286

Fonte: Pesquisa do autor, 2013

Os resultados estão demonstrados através do GRÁFICO 16, assim como em todos os testes de inserção de registros e também na inserção de 1.000.000 de registros de vendas no PostgreSQL a 1FN foi a que apresentou o pior desempenho, ela foi 76,19% mais lenta que a 2FN e 75,55% mais lenta que a 3FN. Nesse os mesmos resultados da 2FN e da 3FN foram praticamente os mesmos, o que não havia ocorrido em nenhuma outra situação de inserção de registro em nenhum dos dois SGBDs testados.

GRÁFICO 17 - Comparação entre as tabelas do PostgreSQL na 3FN, 2FN e 1FN para a primeira seleção de dados com um milhão de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	60,876	48,216	49,865
2FN	47,135	56,043	45,852
1FN	40,876	46,101	40,266

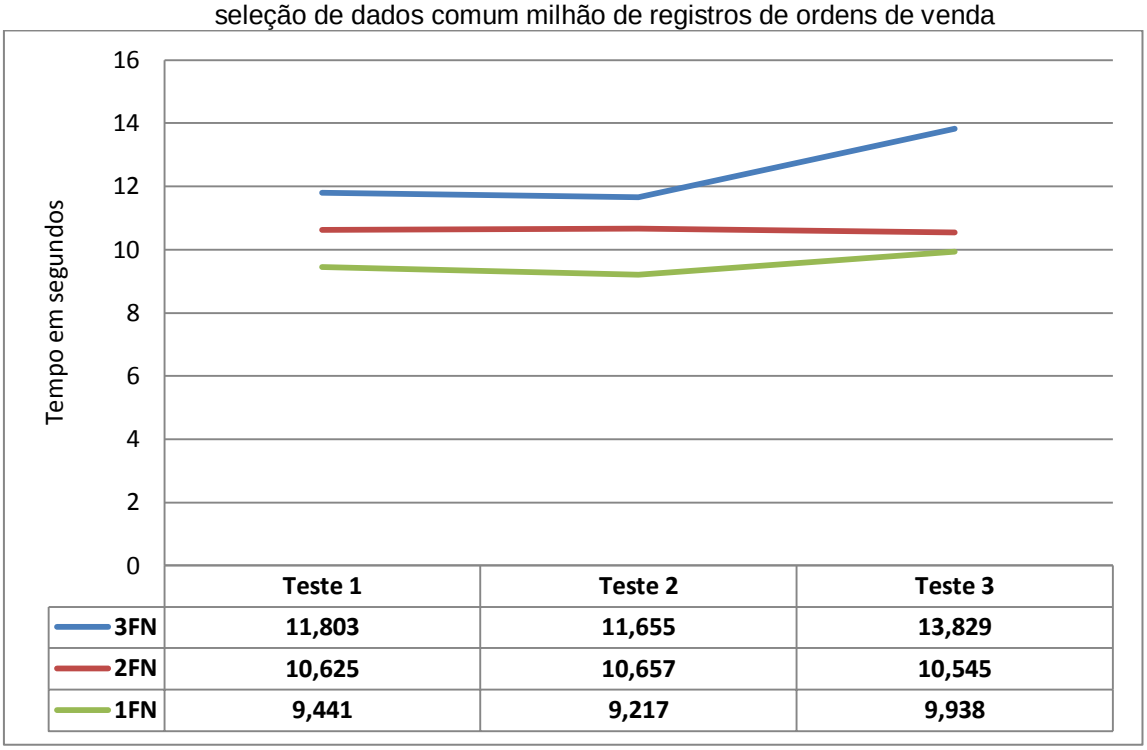
Fonte: Pesquisa do autor, 2013

Esse teste foi realizado para demonstrar o desempenho de cada banco para uma seleção de todos os registros de venda, que nesse caso retornaria 1.000.000 linhas, tópico 4.3.2.1desse mesmo trabalho. Os resultados estão demonstrados através do GRÁFICO 17.

Esse resultado foi mais parecido com o resultado do MySQL com 1.000.000 de registros onde em ambos a 1FN teve o melhor desempenho só que no MySQL a diferença foi um pouco maior, do que com o resultados no mesmo PostgreSQL com 100.000 registros onde a 2FN foi a que teve melhor desempenho.

Os resultados dos desempenhos foram muito parecidos só que a 1FN foi em média 17,12% mais rápida que a 2FN e 24,92% mais rápida que a 3FN, que foi 6,4% mais lenta que a 2FN.

GRÁFICO 18 - Comparação entre as tabelas do PostgreSQL na 3FN, 2FN e 1FN para a segunda seleção de dados comum milhão de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	11,803	11,655	13,829
2FN	10,625	10,657	10,545
1FN	9,441	9,217	9,938

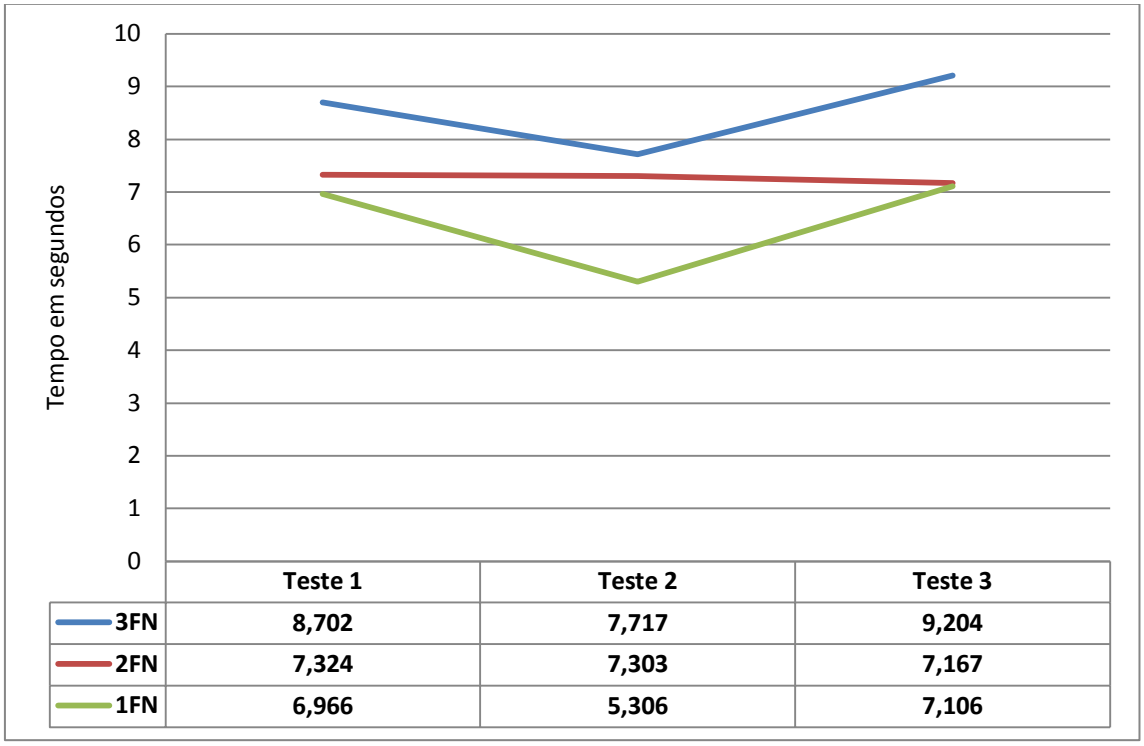
Fonte: Pesquisa do autor, 2013

Esse segundo teste foi realizado para demonstrar o desempenho de cada banco para uma seleção de todos os registros de venda, porém com um tipo de filtro, tópico 4.3.2.2 desse mesmo trabalho, que retornava 200.000 linhas. Os resultados estão demonstrados através do GRÁFICO 18.

Esse resultado também foi mais parecido com o resultado do MySQL com 1.000.000 de registros onde em ambos a 1FN teve o melhor desempenho, porém no MySQL a 1FN teve uma diferença de desempenho maior em relação as outras formas normas do que no PostgreSQL, que também apresentou uma diferença maior que desempenho na 3FN tendo um desempenho mais elevado que a 2FN, já no MySQL o resultado das duas foram bem parecidos.

No PostgreSQL com 100.000 registros a 2FN que teve o melhor desempenho, o que não aconteceu com o PostgreSQL com 1.000.000 de registros do que com o resultados no mesmo PostgreSQL com 100.000 registros onde a 1FN foi a que teve melhor desempenho sendo 11,30% mais rápida que a 2FN e 30,39% mais rápida do que a 3FN, que foi 14,64% mais lenta que a 2FN.

GRÁFICO 19 - Comparação entre as tabelas do PostgreSQL na 3FN, 2FN e 1FN para a terceira seleção de dados comum milhão de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	8,702	7,717	9,204
2FN	7,324	7,303	7,167
1FN	6,966	5,306	7,106

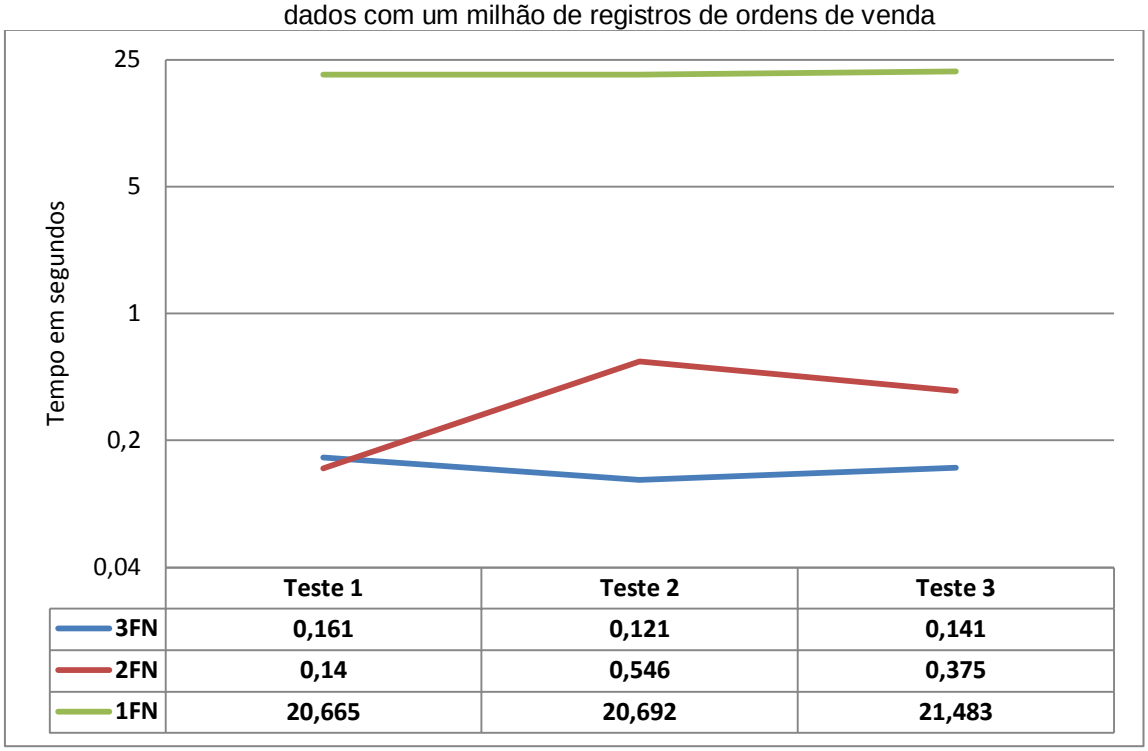
Fonte: Pesquisa do autor, 2013

O último teste de seleção de dados foi realizado também para a tabela com 1.000.000 de registros de vendas no PostgreSQL, tópico 4.3.2.3 desse mesmo trabalho, que retorna uma seleção com 200.000 linhas. Os resultados estão demonstrados através do GRÁFICO 19.

No GRÁFICO 19, assim como todos os testes de seleção de dados no PostgreSQL e no MySQL com 1.000.000 de registros, nesse terceiro teste no PostgreSQL a 1FN foi a forma normal que obteve o melhor resultado, diferentemente dos resultados no PostgreSQL com 100.000 registros onde a 2FN e a forma normal com melhor resultado de desempenho, onde muitas vezes a 1FN era a que tinha o pior desempenho.

Nesse caso a 1FN foi 12,28% mais rápida que a 2FN e 32,02% mais rápida que a 3FN, que foi a que obteve o pior resultado sendo 14,95% mais lenda que a 2FN.

GRÁFICO 20 - Comparação entre as tabelas do PostgreSQL na 3FN, 2FN e 1FN para a alteração de dados com um milhão de registros de ordens de venda



	Teste 1	Teste 2	Teste 3
3FN	0,161	0,121	0,141
2FN	0,14	0,546	0,375
1FN	20,665	20,692	21,483

Fonte: Pesquisa do autor, 2013

Esse teste foi realizado para demonstrar o desempenho de cada banco para uma alteração de alguma informação dos registros de vendas, tópico 4.3.3 desse mesmo trabalho. Os resultados estão demonstrados através do GRÁFICO 20.

Para a alteração dos mesmos dados na 3FN foram alteradas em duas tabelas 15 linhas totais. Para a 2FN foram alteradas em uma tabela 10 linhas totais. E na 1FNfoi em uma tabela nas foram precisos alterar 100.000 linhas.

A 3FN foi 150% em média mais rápida que a 2FN. Igualmente em todos os resultados de desempenho para esse teste em todos os SGBDs e todas as quantidades de registros a 1FN teve o pior resultado. Nesse a 3FN foi 148 vezes mais rápida que a 1FN e a 2FN foi 59 vezes mais rápida que a 1FN.

5.3 Resultados Geral MySQL x PostgreSQL

O QUADRO 1 mostra os melhores resultados obtidos, em cada teste para os SGBDs MySQL e PostgreSQL, podem ser observados também a diferença de tempo entre os SGBDs.

QUADRO 1 – Melhores resultados obtidos em cada teste com uma breve comparação dos SGBDs MySQL e PostgreSQL

Operação SQL	Tamanho BD	Normalização	Média dos Tempos (s)	
			MySQL	PostgreSQL
Insert	100000	1FN	11,002	20,963
		2FN	**6,743***	**17,563***
		3FN	8,767	26,532
	1000000	1FN	115,004	1018,780
		2FN	110,705	**242,576***
		3FN	**98,779***	249,016
Select (1)	100000	1FN	**0,067***	3,899
		2FN	0,166	**1,824***
		3FN	0,197	4,003
	1000000	1FN	**10,990***	**42,414***
		2FN	30,498	49,676
		3FN	34,856	52,985
Select (2)	100000	1FN	0,120	1,028
		2FN	0,057	**0,523***
		3FN	**0,056***	1,054
	1000000	1FN	**1,870***	**9,532***
		2FN	4,102	10,609
		3FN	4,331	12,429
Select (3)	100000	1FN	**0,161***	0,685
		2FN	0,177	**0,403***
		3FN	0,177	0,553
	1000000	1FN	**3,212***	**6,459***
		2FN	3,784	7,265
		3FN	4,664	8,541
Update	100000	1FN	0,487	1,331
		2FN	**0,009***	0,140
		3FN	0,029	**0,045***
	1000000	1FN	13,504	20,946
		2FN	**0,025***	0,353
		3FN	0,083	**0,141***

Fonte: Pesquisa do autor, 2013

No QUADRO 1, os tempos demonstrados são uma média entre os três tempos dos testes. Os tempos mais baixos entre os bancos de dados estão com o fundo cinza e os tempos que estão com o símbolo * são os tempos em que a formal normal obteve o melhor desempenho para cada SGBD.

Para 100.000 registros, na inserção de dados, o MySQL e o PostgreSQL foram mais rápidos com a segunda forma normal. Na primeira seleção de dados, o MySQL foi mais rápido com a primeira forma normal, já o PostgreSQL foi mais rápido com a segunda forma normal. Na segunda seleção de dados, o MySQL foi mais rápido com a segunda e terceira formas normais, com o mesmo resultado, já o PostgreSQL continuou sendo mais rápido com a segunda forma normal. Na terceira seleção de dados, o MySQL foi mais rápido com a primeira forma normal, já o PostgreSQL continuou sendo mais rápido com a segunda forma normal. Na alteração de dados, o MySQL foi mais rápido com a segunda forma normal, já o PostgreSQL foi mais rápido com a terceira forma normal.

Para 100.000.000 registros, na inserção de dados, o MySQL foi mais rápido com a primeira forma normal, já o PostgreSQLfoi mais rápido com a segunda e terceira formas normais, com o mesmo resultado. Na primeira seleção de dados, o MySQL e o PostgreSQL foram mais rápidos com a primeira forma normal. Na segunda seleção de dados, o MySQL e o PostgreSQL continuaram sendo mais rápidos com a primeira forma normal. Na terceira seleção de dados, o MySQL e o PostgreSQL continuaram sendo mais rápidos com a primeira forma normal. Na alteração de dados, o MySQL foi mais rápido com a segunda forma normal, já o PostgreSQL foi mais rápido com a terceira forma normal.

De modo geral, com 100.000 registros a segunda forma normal foi a mais rápida, já com 100.000.000 a primeira forma normal foi mais rápida, a terceira forma normal conseguiu ser mais rápida somente nas alterações do PostgreSQL e o mesmo desempenho de outras formas normais em algumas situações.

6 CONCLUSÃO

Ao desnormalizar o banco de dados, em algumas situações o MySQL e o PostgreSQL, tiveram o mesmo resultado com uma determinada forma normal, o banco de dados manteve o desempenho em algumas situações com o banco todo normalizado, mas em muitas situações houve um ganho de desempenho ao desnormalizar o banco de dados.

O banco de dados todo normalizado foi mais rápido em atividades de escrita ou alteração como previsto, já em atividades de pesquisa como seleção de dados, o banco foi mais rápido em um certo nível de desnomalização, porém, o banco de dados com a menor normalização não garantiu o melhor resultado, em algumas situações o banco de dados com o maior nível de normalização adotado obteve um resultado igual ao nível de desnormalização que se mostrou mais rápido.

O MySQL e o PostgreSQL tem sintaxes diferentes, o que fez com que os *scripts* ficassem diferentes aumentando o tempo de desenvolvimento, além disso, os SGBDs tem uma limitação no tamanho do *script*, assim alguns *scripts* de inserção tiveram que ser divididos em varias partes, há uma limitação também no tamanho do comando INSERT, onde cada INSERT teve um total de 50.000 itens para que o SGBD aceitasse o comando.

De modo geral a desnormalização do banco de dados trouxe ganho de performance nos dois SGBDs, onde a segunda forma normal é a que obteve o melhor custo beneficio sendo mais rápida na maioria dos testes.

A partir do estudo realizado alguns trabalhos futuros podem ser realizados, como testes para identificar anomalias na forma normal mais rápida, para saber em que situação a forma normal não poderá ser utilizada, geração de relatórios, teste de stress com múltiplos usuários, para saber se o banco de dados mantém a mesma performance.

7 REFERÊNCIAS

BATTISTI, Júlio. **SQL Server 2005administração e desenvolvimento:** Curso Completo. Rio de Janeiro: Axcel Books do Brasil, 2005. 1016p.

COUGO, Paulo Sergio. **Modelagem conceitual e projeto de banco de dados.** 3. ed. Rio deJaneiro: Campus, 1997. 284p.

DATE, C. J. **Introdução a sistemas de banco de dados.** Tradução de Daniel Vieira**.** 8. ed. Rio deJaneiro: Elsevier, 2003. 865p.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistema de banco de dados.** Tradução de DanielVieira. 6. ed.São Paulo: Pearson Addison Wesley, 2011. 788p.

FERRARI, Fabrício Augusto. **Crie um banco de dados em MySQL.** São Paulo: DigeratiBooks,2007. 123p.

HEUSER, Carlos Alberto. **Projeto de banco de dados.**4.ed. Porto Alegre:Sagra, 2001. 206p.

MACHADO, Felipe N. Rodrigues; ABREU, Mauricio. **Projeto de banco de dados uma visão prática.** 5. ed. São Paulo: Érica, 1996. 298p.

MACHADO, Felipe N. Rodrigues. **Banco de dados projeto e implementação.** São Paulo: Érica,2004. 398p.

MILANI, André. **MySQL:** Guia do Programador. São Paulo: Novatec, 2007. 400p.

MILANI, André. **PostgreSQL:** Guia do Programador. São Paulo: Novatec, 2008. 392p.

PRICE, Jason. **Oracle Database 11g SQL:** Domine SQL e PL/SQL no banco de dados Oracle. Porto Alegre: Bookman, 2008. 684p.

ROB, Peter; CORONEL, Carlos. **Sistema debanco de dados:** projeto, implementação e administração. Tradução AllTasks. 8.ed. São Paulo: Cengage Learning, 2011. 711p.

REZENDE, Ricardo. Normalização e Desnormalização de Dados – Parte 1: Conhecendo as formas normais. **SQLMagazine: Desnormalizar para Otimizar,** Rio de Janeiro: DevMedia, v.9, n. 99, p. 14-19, abr.2012.

REZENDE, Ricardo. Normalização e Desnormalização de Dados – Parte 2. **SQLMagazine: SQL Server,** Rio de Janeiro: DevMedia, v.9, n.100, p. 6-13, mai.2012.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUNDARSHAN, S. **Sistema de banco de dados.**3. ed. São Paulo: Makron Books,1999. 778p.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUNDARSHAN, S. **Sistema de banco de dados.**Tradução de Daniel Vieira. 5. ed. Rio de Janeiro: Elsevier, 2006. 781p.

SOLID IT. New DB-Engines Ranking shows the popularity of database management systems.**BD-Engines** (Blog). Vienna: Matthias Gelbmann, Paul Andlinger, 3 out. 2012.Disponívelem: http://db-engines.com/en/blog_post/1. Acesso em: 20 out. 2013.