UNIVERSIDADE FEEVALE

DIEGO ZANELATTO

CONVERSÃO DE UM SISTEMA DE AUTOMAÇÃO COMERCIAL DE SGBD FIREBIRD PARA POSTGRESQL

Novo Hamburgo

2013

DIEGO ZANELATTO

CONVERSÃO DE UM SISTEMA DE AUTOMAÇÃO COMERCIAL DE SGBD FIREBIRD PARA POSTGRESQL

Trabalho de Conclusão de Curso
apresentado como requisito parcial
à obtenção do grau de Bacharel em
Ciência da Computação pela
Universidade Feevale

Orientador: Juliano Varella de Carvalho

Novo Hamburgo

2013

RESUMO

Quando um software é desenvolvido e integrado ao SGBD *Firebird*, no momento em que o seu banco de dados evolui e cresce ao ponto de criar dificuldades de manutenção e desempenho, uma medida deve ser tomada para estabilizar a situação. Acompanhando estes fatos, este trabalho realizou a conversão de um banco de dados *Firebird* para o SGBD *PostgreSQL*. Para a conversão do banco de dados, foram usados softwares específicos e, com base nos resultados, foi feita uma análise comparativa entre estas ferramentas utilizadas. Juntamente com a conversão, foram escolhidos alguns módulos de um software de automação comercial onde foi verificado se o sistema em questão obteve uma melhora em seu desempenho e, também, foi analisado os problemas e dificuldades da execução de tal processo.

Palavras-Chave: *Firebird*. *PostgreSQL*. Conversão de dados. Ferramentas de Conversão de dados.

ABSTRACT

When a software is developed and integrated to the DBMS *Firebird*, in the moment that its database evolve and grows to the point of creating difficulties to the maintenance and performance, an action must be taken to stabilize the situation. With this facts, this research will make a conversion of a database in *Firebird* to a DBMS in *PostgreSQL*. For this conversion, will be used specific softwares and, with this results, will be done an analysis comparing the tools used in this process. With this conversion, will be chosen some modules of a commercial automation software to verify if the system will have an improvement in its performance and list the problems e difficulties in the execution of that process.

Key words: *Firebird*. *PostgreSQL*. Data conversion. Data conversion tools.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

TI	Tecno*log*ia da Informação
SGBD	Sistema de Gerenciamento de Banco de Dados
DBA	Database Manager
SP	Stored Procedure
GC	Garbage Collection
PA	Produtos Atendidos

SUMÁRIO

INTRODUÇÃO

Quando se pensa em SGBD (Sistemas de Gerenciamento de Banco de Dados) livres, que possuam estabilidade, anos de mercado e experiência, a preferência dos desenvolvedores, em geral, é o *PostgreSQL*. Mesmo com alguns concorrentes fortes na sua área, o *PostgreSQL* se destaca pela sua grande quantidade de funcionalidades decorrente do seu tempo de existência e desenvolvimento e pela imensa comunidade que trabalha e ajuda a evolui-lo (MILANI, 2008).

Atualmente, um dos seus concorrentes é o *Firebird*, que é considerado um dos SGDBs mais utilizados pelos desenvolvedores brasileiros. Originado do *Interbase*, o *Firebird* foi criado em julho do ano 2000, quando a *Borland* empresa desenvolvedora decidiu tornar o seu código fonte livre (CANTU, 2005). Desde então o *Firebird* vem evoluindo e atualmente se encontra na versão 2.5.2, e testes já estão sendo realizados para liberação da versão 3.0.

Existem atualmente diversos projetos robustos utilizando o *Firebird*, mesmo em sistemas de grande porte, onde o banco de dados chega próximo a casa dos *Terabytes* (CANTU, 2005). Alguns grandes projetos que podem ser citados são da *Wathermark Technologies*, que desenvolve um software de arquivamento de documentos com bases em torno dos 350GB e a empresa Bas-X que possui seu software utilizando bases de 450GB (IB-AID, 2012). Porém, apesar de existirem exemplos de sucesso, o mais comum nestas ocasiões, em que um sistema e a sua base crescem, evoluem e sofrem um aumento de complexidade, é ocorrer a migração do Banco de Dados para um SGBD mais robusto.

Neste trabalho é analisado o caso real de uma empresa que desenvolve um software de automação comercial de lojas de calçados e confecção. Esta empresa denominada de XYZ, passa atualmente por esta situação, onde o seu banco de dados chegou a um nível de maior complexidade exigindo uma tomada de decisão do que deve ser feito em relação ao seu SGBD, em um período de curto a médio prazo.

Atualmente na empresa XYZ o software desenvolvido é integrado ao SGBD *Firebird.* Este software vem sendo evoluído e atualizado desde 2003. Ele atende redes varejistas de pequeno, médio e grande porte. Redes de grande porte possuem em média mais de 70 lojas conectadas ao mesmo tempo ao banco de dados, clientes de médio porte possuem em média 20 a 60 lojas e quantidades menores de lojas são considerados como clientes pequenos. Apesar da qualidade e segurança para pequenos e médios projetos que o *Firebird*

fornece a empresa, com a chegada de grandes clientes foi visualizada a necessidade de pesquisar novas tecno*log*ias para o armazenamento de dados. Estas necessidades foram expostas a partir de uma série de eventos vivenciada pela empresa na evolução do seu software.

Uma pesquisa foi iniciada a fim de buscar uma nova tecno*log*ia de SGBD para ser avaliada a troca e uma futura migração. Para a maior parte dos clientes o fator de o SGBD ser gratuito é crucial, portanto foi necessário encontrar um novo SGBD que também seja livre assim como o *Firebird*. Trabalhos como o de Colares (2007) e (SOUZA; MATIOSKI; NEVES, 2008) demonstram comparativos entre SGBD livres e sugerem uma desvantagem do *Firebird* frente aos seus concorrentes. Através destes trabalhos e outros motivos que seguem citados, decidiu-se realizar o trabalho de migração para o SGBD *PostgreSQL*.

Um dos principais motivos para efetuar a migração foi a dificuldade de encontrar soluções aos diversos problemas enfrentados pela empresa no dia a dia relacionado ao Banco de Dados. A cada obstáculo encontrado no *Firebird* era iniciada uma busca por informações e soluções nas comunidades *online* brasileiras e estrangeiras, além de literaturas disponíveis no mercado. No entanto, todas estas fontes tem um número reduzido em comparação com os demais SGBD gratuitos e bem menor especificamente em relação ao *PostgreSQL*, que possui uma equipe e comunidade muito maiores e mais atuantes (*POSTGRESQL*, 2013a).

Um dos principais problemas encontrado pela empresa com relação ao *Firebird* foi com relação à configuração de um parâmetro específico. Em determinado momento, as bases de dados de alguns clientes estavam corrompendo sem nenhuma explicação. Foi necessária a consultoria de uma empresa que desenvolve um software de monitoramento de bancos de dados *Firdebird* para descobrir que quando o parâmetro *Forced Writes* do SGBD está desativado, o mesmo não grava as informações no momento da solicitação do cliente, e se ocorre um desligamento forçado da máquina onde está rodando o SGBD, estes dados não gravados são perdidos.

Outro ponto negativo na utilização do *Firebird* é em relação as suas atualizações. Por ter uma equipe pequena desenvolvendo suas melhorias e novas funcionalidades e dispondo de uma comunidade reduzida, as atualizações possuem suas datas de liberação muito distantes em comparação com os demais SGBD do mercado. (*FIREBIRD*, 2013a)

Como exemplo podem ser usadas as 3 últimas versões liberadas aos desenvolvedores, 2.5.2 em 06 de novembro de 2012, 2.5.1 em 04 de outubro de 2011 e a

versão 2.5.0 em outubro de 2010 (*FIREBIRD*, 2013b). Já o *PostgreSQL* trabalha de uma forma diferente, este também libera suas atualizações com longos espaços de tempo, porém entre estas versões finais são liberados ajustes e correções que são relatados e levantados pela comunidade. Enquanto o *Firebird* apresenta 3 liberações, o *PostgreSQL* no mesmo período liberou mais de 15 atualizações (*POSTGRESQL*, 2013b).

Outro obstáculo encontrado foi causado pelo crescimento da empresa XYZ. O crescimento aumentou a sua abrangência no mercado e novos clientes de grande porte foram prospectados. No entanto, em inúmeras demonstrações de projeto e avaliação de implantações, o *Firebird* acabou sendo preterido pelas equipes internas de TI dos clientes. Mesmo sendo apresentados argumentos e provas concretas do seu funcionamento estável em grandes redes, o *Firebird* fica atrás de seus concorrentes quando é necessário um maior número de opções de configurações avançadas solicitadas pelos profissionais responsáveis por TI.

Continuando a listagem dos problemas encontrados com o *Firebird* pela empresa, alguns dos módulos do seu sistema de automação comercial dependem exclusivamente do banco de dados para realizar as suas tarefas e estes apresentam alguns gargalos, deixando as suas execuções comprometidas. Módulos de grandes inserções e consultas envolvendo um grande volume de dados no banco apresentam lentidão, e segundo a hipótese deste trabalho teriam seu desempenho melhorado com a migração para o SGBD *PostgreSQL*.

No decorrer deste trabalho, foi realizada uma pesquisa bibliográfica a fim de levantar todas as diferenças e problemas que serão encontrados para realizar esta conversão. Será convertida a base de dados para *PostgreSQL* através de programas específicos nesta área. Também será montada uma estrutura apropriada para testes e com base nos resultados, comprovar que o sistema de automação comercial ganhará em desempenho.

Desta forma, este trabalho contribuirá para identificar problemas que os desenvolvedores podem encontrar no caminho ao migrar uma base de dados do sistema de gerenciamento de banco de dados *Firebird* para o *Posgresql*, assim como, também são apresentadas soluções e formas para contornar estas situações e problemas. Ao final foi possível testar os desempenhos e relatar os resultados obtidos, a fim de demonstrar um exemplo prático de melhorias em um software real a partir da realização deste processo.

Este trabalho está dividido em 4 capítulos. O primeiro capítulo apresentará um estudo realizado para levantar as características principais dos dois SGBD, e verificar as suas

diferenças na tentativa de prever as dificuldades que serão encontradas no decorrer do trabalho. No capítulo 2 será demonstrado detalhadamente a conversão dos bancos de dados. O terceiro capítulo mostrará como os módulos do sistema de automação comercial foram convertidos e no último capítulo será demonstrado os testes e seus resultados.

1 SISTEMAS DE GERENCIAMENTO DE BANCO DE DADOS ESCOLHIDOS

Em um trabalho onde será convertido um banco de dados de um sistema de gerenciamento para outro totalmente diferente, faz-se necessário o estudo e análise detalhada e comparativa de cada uma de suas propriedades, características e funções a fim de levantar as dificuldades a serem superadas no decorrer da pesquisa.

Concluindo um comparativo entre estes sistemas é possível prever os próximos passos para finalizar a conversão. Posteriormente é executado o processo de migração dos módulos do sistema de automação comercial para adaptá-lo ao novo banco de dados.

1.1 *Firebird*

No ano de 2000 a empresa Borland (atualmente denominada Embarcadero) abriu o código fonte do seu banco de dados comercial *Interbase* 6, o tornando *free* e *open source*. Porém, após um período de incertezas quanto à continuidade do projeto, acabou gerando um descontentamento na comunidade de usuários. A partir deste descontentamento um grupo de desenvolvedores resolveu criar um novo SGBD baseado no código fonte liberado do *Interbase*, que foi chamado de *Firebird*. (CANTU, 2005)

A Borland deu continuidade em seu projeto comercial *Interbase*, porém as novas versões não seriam gratuitas, causando mais desconfortos com a comunidade de usuários. Até o momento foram liberadas as versões[1] 6.5, 7, 7.5, 2007, 2009, XE3. Em compensação, o *Firebird* neste período ganhou uma legião de usuários que acabaram migrando do *Interbase*, pois estes procuravam uma ferramenta que estivesse em pleno desenvolvimento e que fosse *Free* e *Open Source* (CANTU, 2005).

O *Firebird* é um sistema gerenciador de banco de dados baseado no modelo relacional. Segundo DATE (2003) o modelo de dados relacional trata-se de um modelo criado em 1970 pelo pesquisador da IBM Edgar Frank Codd. Seu propósito é representar os dados de uma forma mais simples, através de um modelo matemático de conjunto de tabelas relacionadas. Este modelo torna o banco de dados mais flexível, tanto na forma de representar as relações dos dados, como na tarefa de modificação de sua estrutura, sem ter que reconstruir todo o banco de dados.

[1] Site oficial Embarcadero. Disponível em: < http://www.embarcadero.com/>. Acesso em 10 maio 2013.

O *Firebird* é um banco de dados cliente/servidor compatível atualmente com o padrão SQL-ANSI-92. Possui versões de instalação *Classic, SuperServer, SuperClassic*. A versão *Classic* utiliza processos separados do servidor para gerenciar cada conexão com o banco de dados, onde cada uma destas conexões possui a sua própria área de memória, porém o *cache* entre elas não é compartilhado. O *SuperServer* possui apenas uma conexão com o banco de dados e criará uma *thread* para cada conexão realizada e assim compartilhará o *Cache*. (*FIREBIRD*, 2013c)

Um dos maiores problemas do servidor *SuperServer* é que este utiliza apenas um processo e não aproveita em totalidade as novas tecno*log*ias de multi processamento dos processadores modernos, problema que a versão *Classic* não possui por usar mais de um processo dividindo-os pelos núcleos disponíveis. Na tentativa de unificar a facilidade do *cache* da versão *SuperServer* e o uso de multi processamento da versão *Classic* foi criada a versão *SuperClassic* a partir da versão 2.5.0 do *Firebird* (*FIREBIRD*, 2013d).

1.2 *PostgreSQL*

O *PostgreSQL* é um sistema de gerenciamento de banco de dados baseado no POSTGRES versão 4.2 desenvolvido pelo Departamento de Ciência da Computação da Universidade da Califórnia em Berkeley. O projeto *POSTGRES* foi liderado pelo professor Michael Stonebraker e teve seu patrocínio pela DARPA (*Defense Advanced Research Projetcts Agency*), ARO (*Army Research Office*) e NSF (*National Science Foundation*) (*POSTGRESQL*, 2013c).

Em 1994 Andrew Yu e Jolly Chen adicionaram um interpretador de SQL ao *POSTGRES* e o nomearam de *POSTGRES95*. Em seguida foi liberado na internet com seu código aberto. Muitas mudanças internas nesta versão melhoraram de 30% a 50% o desempenho em relação à versão anterior 4.2, além de resolver uma série de erros e implementar uma grande quantidade de melhorias. (*POSTGRESQL*, 2013c)

Em 1996 o projeto *POSTGRES95* foi renomeado para *PostgreSQL*. Esta nomenclatura foi escolhida para refletir o relacionamento entre o *POSTGRES* original e as versões mais recentes com capacidade SQL.

O *PostgreSQL* atualmente é um SGBD objeto relacional. Como alternativa ao modelo relacional, os fabricantes de SGBD adotaram características de orientação a objetos, criando assim os modelos de bancos de dados objeto relacionais. Estes são considerados uma

forte tendência de um novo modelo de tecno*log*ia para os bancos de dados. Nestes SGBDs objeto relacionais, classes, objetos e herança são suportados em seus esquemas e linguagens de consulta. (*POSTGRESQL*, 2013c).

Na execução do *PostgreSQL* em um servidor, existe um processo que gerencia os arquivos de banco de dados e aceita as conexões dos aplicativos clientes. Este processo de servidor de banco de dados se chama *Postmaster*. O servidor do SGBD pode tratar várias conexões simultâneas de clientes, portanto o *Postmaster* está sempre executando e aguardando novas conexões (*POSTGRESQL*, 2013d).

Com relação ao padrão SQL utilizado no *PostgreSQL*, conforme sua documentação oficial[2], é usado o padrão SQL:2008. Apesar de não implementar as partes 4 e 10 (SQL/PSM - *Persistent Stored Modules* e OLB - *Object Language Bindings* respectivamente) da definição do SQL:2008, os outros 12 grupos e suas funcionalidades estão incluídos e disponíveis.

1.3 Comparativo de funções e características

Nesta seção serão comparadas as funções e características dos dois SGBD estudados. Ao mesmo tempo que estão sendo comparadas, serão levantadas possíveis mudanças que deverão ocorrer no trabalho de conversão para adaptar o sistema de automação comercial ao novo SGBD.

As funções escolhidas para avaliação foram definidas pelo que é comum entre os dois SGBDs e o que é utilizado atualmente pelo sistema de automação comercial. *Views*, *Stored Procedures*, T*riggers*, *Generators* são funções utilizadas pelo sistema de automação que deverão ser convertidas manualmente para o novo banco de dados. Índices, tipos de dados, operadores, *backup/restore* e *garbage collection* são funções e propriedades padrões de um sistema de gerenciamento de banco de dados que devem ser analisadas para verificar a existência de divergências que deverão ser tratadas ao longo do trabalho.

1.3.1 Views

Iniciando a análise comparativa, deve-se observar o funcionamento das *Views* de ambos os bancos de dados. Uma *view* segundo Date (2003) é uma expressão nomeada da

[2] Documentação oficial *PostgreSQL*. Disponível em <http://www.*PostgreSQL*.org/docs/9.1/static/features.html>. Acesso em 12 maio 2013.

álgebra relacional. Esta expressão é definida para o SGBD sob um nome específico e simulando uma tabela, porém não contendo nenhum dado físico. A *view* é composta dinamicamente por uma consulta que é previamente escrita.

Ao estudar a definição e uso de *views* em ambos SGBDs, notou-se que os dois lidam com esta estrutura da mesma maneira, o que não ocasionará qualquer impacto durante a migração dos dados.

1.3.2 Índices

Segundo Cantu (2006) índices são estruturas que ordenam de forma crescente ou decrescente os dados, facilitando e agilizando as buscas e consultas de uma determinada informação dentro do banco de dados. Na maioria dos bancos de dados, os índices representam papel importante na otimização de consultas, isto se deve ao fato de que os índices possuem estruturas de ponteiros para os dados armazenados em colunas específicas.

Todos os índices criados no *Firebird* seguem a definição de árvore *BTree.* Estas árvores fazem com que cada tabela do banco possua sua própria IPR (*Index Page Root*). A IPR pode apontar para outras páginas de ponteiros até chegar à última página da árvore, chamada de *Leaf Page.* A *Leaf Page* contém os ponteiros que apontam diretamente para os registros da tabela. Quando um IPR é muito pequeno, a própria IPR pode ser também *Leaf Page.* (CANTU, 2006)

Em versões anteriores do *Firebird* a existência de índices com um alto nível de repetição nas chaves poderiam causar sérios problemas de lentidão nos processos de *Garbage Collection* e remoção de registros. Mas a partir da versão 2.0, com a utilização de um novo algoritmo, este problema foi resolvido.

O *PostgreSQL* disponibiliza uma maior variedade de tipos de índices para serem implementados nos bancos de dados. Além do índice *BTree* que também é utilizado no *Firebird*, estão disponíveis os índices do tipo *RTree, Hash* e *GiST*. Ao criar um índice no *PostgreSQL,* a configuração *default* escolhida pelo SGBD é a *Btree,* que é a opção adequada à maioria das situações comuns. O índice *Btree* no *PostgreSQL* pode tratar consultas de igualdade e de faixas em dados que podem ser classificados em alguma ordem. Este índice é utilizado com os seguintes operadores: “<”, “<=”, “=”, “>=”, “>”, “*Between*”, “in”. (*POSTGRESQL*, 2013e)

O tipo de índice *RTree* é utilizado em consultas de dados espaciais. É considerado um índice *RTree* a coluna indexada que está envolvida em uma comparação utilizando os seguintes operadores: "<<", "&<", "&>", ">>", "@", "~=", "&&".

O índice *Hash* é utilizado para comparações simples onde está envolvido o operador "=". Por testes realizados pela comunidade foi constatado que este índice não possui um desempenho melhor que o *BTree* e que o seu tempo de construção é muito pior. Por este motivo o uso deste tipo de índice é desencorajado atualmente. (*POSTGRESQL*, 2013e)

Por fim, o índice *GiST* (*Generalized Search Tree*) é uma árvore de busca genérica. Este algoritmo pode ser usado para construir quase todos os tipos de árvores de busca sob quase todos os tipos de dados. O *GiST*[3] pode ser usado para qualquer tipo de dado que possa ser naturalmente ordenado em uma hierarquia de conjuntos.

Um detalhe negativo da utilização dos índices no *PostgreSQL* é o operador "*is null*", citado em sua documentação oficial (2013e). Segundo o documento, o SGBD não utiliza os índices das colunas ao utilizar este operador. Realizando testes no *Firebird* é possível identificar que este SGBD utiliza índices nesta situação específica. Este complicador se torna crítico no momento em que é possível identificar o uso deste operador no sistema de automação comercial, forçando uma mudança no momento da conversão dos módulos do sistema.

Outra diferença encontrada entre os SGBD em relação aos índices é quanto ao operador "*Starting With"*, utilizado pelo *Firebird*, e em grande quantidade dentro do sistema de automação comercial. Este operador foi introduzido no sistema de automação para contornar um problema do *Firebird* onde os comandos do operador "*like"* não usavam os índices. Comandos como por exemplo "*select * from TABELA where tabela.campo like 'TESTE%'"* não fazem uso do índice do campo. O operador "*starting with*" retorna os mesmos resultados do comando "*like*" acima, porém usando o índice.

Concluindo o comparativo é possível verificar que serão necessárias alterações pontuais em comandos SQL do sistema de automação comercial quanto ao uso dos operadores "*Starting with*" que não é implementado e "*is null*" que não utiliza índices no SGBD *PostgreSQL*. Quanto a conversão dos índices não será necessária uma manutenção inicial, pois todos os índices criados no *PostgreSQL* são por default do tipo *BTree* que é o mesmo utilizado atualmente pelo SGBD *Firebird.*

[3] Site oficial GiST. Disponível em: <http://gist.cs.berkeley.edu/>. Acesso em 15 maio 2013

1.3.3 Tipo de dados

Para realizar a conversão de um banco de dados é de grande importância conhecer os tipos de dados utilizados em cada um dos SGBD envolvidos. Esta informação é necessária para avaliar nas migrações de dados se são necessárias modificações estruturais em campos de tabelas por causa de tipos de dados não equivalentes ou incompatíveis.

O SGBD *Firebird* possui uma boa diversidade de tipos de dados implementada, porém baixa em comparação com os tipos de dados do *PostgreSQL*. Segundo Cantu (2005), no *Firebird*, os tipos de dados são divididos em nove grandes grupos: *Char* e *Varchar*; *Date, Time* e *Timestamp; Integer* e *Smallint; Bigint; Float; Double Precision; Numeric e Decimal; Array; Blob.* Já o *PostgreSQL*, segundo a sua documentação oficial (2013f), possui os seus tipos de dados divididos em 13 grupos: numéricos; monetários; cadeia de caracteres; binários; data hora; booleano; geométricos; endereços de rede; cadeias de bits; matrizes; tipos compostos; identificadores de objetos; pseudotipos.

Comparando os grupos entre os dois SGBD é possível identificar que os tipos de dados comuns (*char, varchar, integer, date, numeric, double, blob*) são atendidos por ambos. Para um ambiente de conversão é necessário que o banco de dados de destino possua todos tipos de dados do banco de origem para que não existam incompatibilidades. Para assegurar que problemas não ocorram no processo de conversão, deve ser executada uma comparação entre os tipos de dados utilizados pelo banco de origem e o de destino e verificar se cada um deles possui um tipo correspondente. Esta comparação pode ser observada na tabela 1.1.

Tabela 1.1 - Comparação entre os tipos de dados.

Tipo Dado *Firebird*	Tipo Dado *PostgreSQL*
Char	Char
Varchar	Varchar, text
Date	Date
Time	Time
Timestamp	Timestamp
Integer	Integer
Smallint	Smallint
Bigint	Bigint
Float	Float
Double Precision	Double
Numeric	Numeric
Decimal	Decimal
Array	Array
Blob	Bytea

Além de comparar os tipos de dados, deve ser analisada detalhadamente a função de cada um destes tipos e verificar se existe concordância entre o seu funcionamento, armazenamento e faixa de valores.

Os tipos *char* e *Varchar* no *Firebird* possuem respectivamente um tamanho máximo de 32767 e 32765 bytes (CANTU, 2005), já estes campos no *PostgreSQL* possuem seu limite máximo em cadeias de caracteres de até 1GB. Ainda em relação a estes tipos de campos, o *PostgreSQL* disponibiliza um tipo nomeado "*text"* que armazena cadeias de caracteres de qualquer tamanho (*POSTGRESQL*, 2013g). Na conversão do banco de dados deste trabalho não foi necessário modificar estes campos pois as configurações padrões deste tipo de dado no *PostgreSQL* atendem aos requisitos.

Nos tipos de campo *Blob*, dentro do *Firebird*, existe uma peculiaridade que pode trazer problemas em uma conversão. Este tipo de campo no *Firebird* pode ser definido como um tipo de armazenamento para valores binários e informações textuais. Se no banco de dados *Firebird* existir um campo *Blob* contendo texto ele deve ser convertido para o *PostgreSQL* como tipo de dado *text*, caso o *Blob* seja um binário ele deve ser convertido como *Bytea.*

Outro fator importante a ser verificado em uma conversão de banco de dados de um sistema relacionado aos tipos de dados é a função de *cast*. No caso deste trabalho, ambos SGBDs aceitam esta função de conversão de tipo de dado e em ambos a forma utilizada é semelhante. O quadro 1.1 apresenta um exemplo de *cast* aceito pelos dois SGBDs.

```
SELECT CAST (varchar '1234' AS text);
```

Quadro 1.1 – Exemplo de Cast.

Concluindo a análise dos tipos de dados dos SGBDs *Firebird* e *PostgreSQL*, é possível notar uma quantidade maior de tipos de dados disponíveis no *PostgreSQL*. Com isso seriam abertas novas possibilidades para o sistema de automação comercial após a conversão. Outro fator importante que deve ser considerado na conversão são com relação aos campos *Blob*, que antes devem ser analisados e classificados, para serem corretamente convertidos aos seus respectivos tipos de campo no banco de dados *PostgreSQL*. O restante dos tipos de dados se aproxima muito em comportamento e definições, portanto não apresentam problemas na conversão.

1.3.4 Operadores e funções internas

A análise detalhada de operadores lógicos, aritméticos, de concatenação, comparação e funções internas são de grande importância para um trabalho de conversão de um sistema. Caso exista uma diferença entre estes operadores e funções nos bancos de dados envolvidos, as cláusulas SQL deverão receber uma atenção especial pois certamente é necessário alterá-las para garantir o seu funcionamento.

Com relação às funções internas, é possível observar que as principais possuem a sua nomenclatura e seu funcionamento semelhantes. Como por exemplo as funções "*extract*", "*lpad*", "*rpad*", "*cast*", funções matemáticas como a "*sum*", "*abs*", "*mod*". Existem alguns exemplos de funções que divergem tanto em funcionamento como na sua nomenclatura, tais como as funções "*truncate"* do *Firebird* e "*trunc*" do *PostgreSQL* que possuem a mesma finalidade porém com assinaturas diferentes. Outro exemplo de função divergente entre os dois SGBD é "*char_length*" do *Firebird* e o "*length*" do *PostgreSQL*.

O caso dos operadores é semelhante ao das funções. Os principais operadores são equivalentes e atendem as necessidades da conversão deste trabalho, porém existem alguns exemplos de operadores que divergem. O *PostgreSQL* possui uma grande variedade de funções e operadores abrindo novas possibilidades para o sistema de automação comercial após a conversão.

Apesar de existirem semelhanças entre a maior parte das funções e operadores, todos os SQLs executados pelo sistema de automação comercial foram revisados e testados no processo de conversão.

1.3.5 Stored Procedures

Uma *Stored Procedure* (SP) é um conjunto de comandos e rotinas executadas diretamente dentro do banco de dados. É possível através de uma SP, informar parâmetros de entrada e receber retornos de valores. Esta prática, por ser executada dentro do Banco de Dados, evita o tráfego de rede, aumenta a agilidade e a segurança dos dados. (CANTU, 2006).

No *Firebird* a linguagem utilizada para desenvolver estas *Stored Procedures* é a PSQL. Dentro de uma SP, através da linguagem PSQL é possível declarar variáveis de qualquer tipo, utilizar cláusulas para delimitar blocos de código como o *Begin-End*, introduzir comentários, exceções, condições, *loops*, *cases*, eventos e comandos de saída e suspensão

(CANTU, 2006). Abaixo segue um exemplo básico de uma *Stored Procedure* escrita na linguagem PSQL para o *Firebird*, utilizada pelo sistema de automação comercial.

```
CREATE OR ALTER PROCEDURE QTDESTOQUE (
    CDEMPA VARCHAR(3),
    CDITEA VARCHAR(15),
    TAITEA VARCHAR(5),
    DATA DATE)
RETURNS (
    QTD DOUBLE PRECISION,
    CUMED DOUBLE PRECISION)
AS
DECLARE VARIABLE DTMOVD DATE;
BEGIN
  /* BUSCA A DATA DO ULTIMO MOVIMENTO */
  SELECT MAX(ITE_DTMOVD000) DTMOVD
  FROM TB_EIT
  WHERE EMP CDEMPA003 =:CDEMPA
  AND ITE_CDITEA015 =:CDITEA
  AND TIT_TAITEA005 =:TAITEA
  AND ITE_DTMOVD000 <= :DATA
  INTO :DTMOVD;
  /* RETORNA OS VALORES DO ESTOQUE */
  SELECT EIT_QTESTN011 QTD, EIT_CUMEDN017 CUMED
  FROM TB_EIT
  WHERE EMP_CDEMPA003 =:CDEMPA
  AND ITE CDITEA015 =:CDITEA
  AND TIT_TAITEA005 =:TAITEA
  AND ITE_DTMOVD000 =:DTMOVD
  INTO :QTD, :CUMED;
  SUSPEND;
END
```

Quadro 1.2 – Exemplo de *Stored Procedure* para *Firebird*

No *PostgreSQL* as *Stored Procedures*, chamadas internamente de *functions*, seguem o mesmo padrão de funcionalidade do *Firebird*, porém utilizam outras linguagens de programação. A linguagem padrão para desenvolver uma SP neste SGBD é a PL/PgSQL e existem outras linguagens suportadas: PL/Tcl; PL/Perl; PL/Python; C. Para fazer uso destas linguagens é necessário realizar uma instalação à parte para cada uma delas, utilizando o comando *createlang* (MATTHEW, STONES, 2005). As SPs escritas em *PostgreSQL* possuem as mesmas características apresentadas da PSQL do *Firebird*, apesar de serem escritas em linguagens diferentes. Abaixo segue o exemplo da mesma SP do exemplo anterior, escrita em PL/PgSQL.

```
CREATE TYPE EST_CUSTO AS(
  EIT_QTESTN011 FLOAT,
  EIT CUMEDN017 FLOAT
);

CREATE OR REPLACE FUNCTION QTDESTOQUE (CDEMPA VARCHAR(3), CDITEA
VARCHAR(15), TAITEA VARCHAR(5),DATA_AUX DATE)
RETURNS EST CUSTO AS '
DECLARE
      DTMOV DATE;
      RESULT EST_CUSTO;
BEGIN
      SELECT max(ITE DTMOVD000) INTO DTMOV FROM tb eit
      where emp_cdempa003 = CDEMPA
      and ite_cditea015 = CDITEA
      and tit_taitea005 = TAITEA
      and ite_dtmovd000 <= DATA_AUX;

      SELECT EIT_QTESTN011, EIT_CUMEDN017 INTO RESULT FROM TB_EIT
      WHERE emp_cdempa003 = CDEMPA
      and ite_cditea015 = CDITEA
      and tit taitea005 = TAITEA
      and ite_dtmovd000 = DTMOV;

      return result;
END;
' LANGUAGE plpgsql;
```

Quadro 1.3 – Exemplo de *Stored Procedure* para *PostgreSQL*.

Apesar de existirem semelhanças entre as funções das *Stored Procedures* nos dois SGBDs em questão, é necessária uma conversão manual de todas as SPs do SGBD de origem para o SGBD de destino devido à diferença entre as linguagens. Esta conversão será de extrema importância para que os módulos do sistema de automação comercial convertidos funcionem corretamente.

1.3.6 Triggers

Assim como as *Stored Procedures*, as *Triggers* também são comandos e rotinas executadas no banco de dados. Estas, porém, são executadas vinculadas às tabelas. Assim que são criadas as *triggers*, elas são vinculadas a um tipo de gatilho, que pode ser um *insert*, *update* ou *delete*. Quando uma tabela possui uma *trigger* vinculada a um destes tipos de gatilho, como por exemplo a um *update*, ao ser executado, qualquer SQL de *update* em qualquer registro desta tabela, a *trigger* será acionada automaticamente. (DATE, 2003)

Tanto no *Firebird* como no *PostgreSQL* as *Triggers* possuem operadores internos para acessarem os valores antigos e novos. Um exemplo pode ser visto em uma *trigger* de

update, onde se é necessário concatenar um valor do registro antigo com o novo que está sendo atualizado, assim como é apresentado no quadro 1.4.

```
CREATE OR ALTER TRIGGER EXEMPLO_TRIGGER FOR TABELA1
AS
BEGIN
NEW.CAMPO1 = NEW.CAMPO1 || OLD.CAMPO1;
END
```

Quadro 1.4 – Exemplo de *trigger* para *Firebird*.

A linguagem para programar uma *trigger* é igual ao da programação de uma *Stored Procedure* para os dois SGBDs, como citado na seção 1.3.5. Assim como no caso das SPs este fato é um ponto determinante para o processo de conversão do banco de dados, pois também será necessário converter todas as *triggers* manualmente para possibilitar o uso do sistema de automação comercial.

1.3.7 Generators x Sequences

O *Firebird* possui em sua definição o conceito de *Generators*, herdado do *Interbase.* Um *generator* nada mais é do que um nome ou um apelido ao nome oficial *Sequences*, definido pelo padrão SQL. (CANTU, 2006)

Sequences são utilizadas para gerar sequências de números inteiros incrementados ou decrementados (DATE, 2003). Podem ser utilizadas também para gerar valores únicos de um campo chave de uma tabela que não possua colunas que se apliquem adequadamente a este conceito.

A criação de *Sequences* é semelhante nos dois SGBDs, porém assim como no exemplo das *Stored Procedures* e *Triggers* também precisa ser implementado manualmente no banco de dados de destino para permitir o funcionamento adequado dos módulos convertidos do sistema de automação comercial.

1.3.8 Backup e Restore

Apesar de as rotinas de *Backup* e *Restore* não afetarem diretamente o processo de conversão do banco de dados, é interessante iniciar uma comparação destes processos entre os SGBDs envolvidos para se ter ideia de como deverão ser implementadas estas rotinas nos clientes da empresa XYZ.

Segundo a documentação oficial do *Firebird* (*FIREBIRD*, 2013e), o processo de *Backup* e *Restore* é realizado por uma ferramenta a parte que é instalada juntamente com o SGBD. Esta ferramenta é o executável *gbak* encontrado dentro da pasta de binários do *Firebird.* O *gbak* é um executável de linha de comando usado tanto em sistemas operacionais Linux quanto Windows.

Os *backups* do *Firebird*, a partir da versão 2.0, podem ser executados de 3 formas: Cópia de arquivo; *Backup* total; *Backup* incremental. O arquivo de *backup* contém os dados compactados e todas as estruturas do banco de dados, porém destas estruturas ele armazena apenas as suas definições, fazendo com que o arquivo de *backup* seja muito menor em comparação ao arquivo original. Além disso, no processo de *backup/restore* do *Firebird* os dados marcados como lixo (apagados e temporários) são descartados.

O *backup* por cópia de arquivo é uma possibilidade aceita pelo *Firebird*, pois a base de dados é um arquivo com extensão ".fdb", e a cópia pode ser realizada diretamente pelo sistema operacional. Esta prática apesar de simples não é recomendada, pois a cópia física de um banco de dados em uso pode corromper o arquivo.

O tipo de *backup* total é o mais comum utilizado, onde é realizado um *backup* de todos os dados e definições gerando um arquivo do tipo *gbk.* Este arquivo como mencionado anteriormente possui um tamanho inferior ao arquivo da base de dados e é criado pelo programa de linha de comando *gbak,*

O tipo de *backup* incremental é gerado pelo programa *nBackup* que foi introduzido no *Firebird* na sua versão 2.0. As funções de coleta de lixo e descarte de conteúdo, que são executadas pelo *gbak*, não são executadas pelo *nBackup.*

O *PostgreSQL* também possui 3 formas de *backup* de dados: *pg_dump*; cópia de arquivos de sistema; cópia de segurança em linha.

O método *pg_dump* é um comando que ao ser selecionado gera um arquivo contendo todos os comandos SQL que ao serem processados recriam o banco de dados no mesmo estado que ele se encontrava quando o arquivo foi criado.

O método de cópia de arquivos do sistema, assim como a cópia do arquivo físico do *Firebird* não é recomendada, pois se o banco de dados estiver sendo usado podem ocorrer inconsistências com os dados.

O último método de *backup* do *PostgreSQL* é o de cópia de segurança em linha. Este método tem seu funcionamento baseado no registro de escrita prévia (WAL – *Write Ahead Log*). O *PostgreSQL* armazena em um diretório todas as alterações realizadas nos arquivos de banco de dados. O WAL tem a finalidade de fornecer segurança contra quedas, pois se o sistema cair, o banco de dados pode retornar a um estado consistente refazendo as entradas gravadas neste último ponto de verificação. Se for necessário realizar um *backup*, pode ser feita a recuperação da cópia de segurança do banco de dados no nível de sistema de arquivos e, depois, refeitas as alterações a partir da cópia dos arquivos de segmento do WAL, para trazer a restauração para o tempo presente.

Concluindo a análise de *backup/restore* dos SGBDs envolvidos é possível verificar que as rotinas atuais definidas para o *Firebird* poderão ser convertidas normalmente para as rotinas utilizadas pelo *PostgreSQL*, já que ambas possuem ferramentas que realizam estas funções de forma semelhante.

1.3.9 Garbage Collection x VACUUM

Garbage Collection (GC) é o processo de coleta de lixo. Todos os bancos de dados ao longo do seu uso acabam gerando registros de lixo e temporários, que necessitam ser excluídos para diminuir espaço em disco ocupado e melhorar o desempenho. (CANTU, 2006)

A exemplo das rotinas de *backup* e *restore*, o *garbage collection* também não influencia a conversão dos bancos de dados, mas se faz necessária uma análise deste tópico para comparação comportamental dos dois SGBDs quanto a esta rotina, a fim de verificar se as configurações realizadas nos clientes da empresa XYZ poderão permanecer ou deverão ser reconfiguradas.

Ao atualizar ou remover um registro de uma tabela o *Firebird* cria uma cópia temporária deste registro, isto é feito para que as transações concorrentes enxerguem estas informações corretamente ou para reverter os dados para a forma original caso a transação

que os manipulou seja desfeita. Quando o *Firebird* detecta que uma versão temporária de um registro não é mais necessária ele a marca como “lixo”. (CANTU, 2006)

O processo interno do *Firebird* responsável por realizar o *Garbage Collection* é o *Sweep.* O *sweep* procura por todo o banco de dados registros marcados como lixo e libera o espaço utilizado por eles. Este processo por *default* é marcado como automático, porém é possível configurá-lo para o modo manual. Esta configuração é permitida, pois este processo, dependendo do momento que é acionado, pode comprometer o desempenho do banco de dados e por esta causa aconselha-se acioná-lo manualmente em um horário onde o banco está em *stand-by.*

O *PostgreSQL* não possui uma rotina automática de GC assim como o *Firebird*, porém possui um comando que executa esta função e que pode ser agendado para ser rodado em um horário definido pelo DBA. O comando em questão é o *VACUUM*. Este comando executa a limpeza dos registros marcados como lixo e os libera para uso. Uma opção a mais em comparação com o *Firebird* é que o comando *VACUUM* pode ser rodado com o parâmetro *ANALYSE* onde é retornada uma listagem por tabela de quantos registros foram limpos no processo. (MATTHEW, STONES, 2005)

Apesar de o *PostgreSQL* não possuir uma rotina automática como o *Firebird*, não serão necessárias mudanças na rotina dos clientes da empresa XYZ na questão de GC pois atualmente está sendo utilizado a configuração manual para não comprometer o desempenho. Portanto apenas será necessário adaptar a execução agendada do novo comando *VACUUM* ao invés do comando *sweep* do *Firebird.*

Concluindo o estudo conceitual dos SGBDs é possível obervar grandes diferenças em suas características. Divergências como as linguagens das *Stored Procedures* e operadores e funções internas devem ser tratadas com cautela para garantir o funcionamento do sistema de automação comercial.

2 CONVERSÕES DE DADOS

A migração de um SGBD para outro implica em uma série de preocupações e problemas para os desenvolvedores e DBA. Desde a simples conversão das tabelas, índices e chaves estrangeiras, até a preservação da integridade dos dados, tudo deve ser avaliado e considerado para diminuir as perdas e problemas futuros que este processo pode gerar.

Partindo deste cenário de dificuldade, faz-se necessária uma análise aprofundada em softwares de conversão e suas funcionalidades. A expectativa é que estes softwares sejam capazes de além de executar a tarefa básica de converter os dados mantendo a sua integridade, também migrar características específicas dos SGBDs.

O sistema de automação comercial convertido neste trabalho utiliza um banco de dados *Firebird.* Para convertê-lo para *PostgreSQL* por completo será necessário migrar ou adaptar características de grande importância ao banco de dados atual como *procedures, views, triggers e generators* que são amplamente usados no sistema de automação.

Para a execução desta tarefa, foram escolhidos 3 softwares de conversão de banco de dados disponíveis no mercado. A partir de testes de conversões com o banco de dados do sistema de automação comercial, estes softwares são avaliados através dos seguintes requisitos: características gerais, desempenho, interface, comparativo de funcionalidades e, principalmente, são realizadas comparações entre os dados convertidos e os originais para comprovar que a integridade foi mantida.

Os softwares de conversão escolhidos foram *Firebird2PostgreSQL*, *FullConvert* e o *Datapump* for *PostgreSQL*. O primeiro foi escolhido por ser um software livre e de código fonte aberto, os dois últimos são os softwares pagos mais conhecidos para esta tarefa.

2.1 Ambiente para testes de conversão

Como o principal requisito dos testes é a integridade dos dados e o desempenho é um indicador secundário, não foram considerados quais os sistemas operacionais são os mais performáticos para cada um dos SGBDs. Como os dois Sistemas de Gerenciamento de banco de dados possuem versões para o sistema operacional Windows, esta plataforma foi escolhida para executar as conversões.

2.1.1 Configurações de Hardware

Para realizar os testes de conversão dos dados foi utilizada a seguinte configuração de hardware: Sistema Operacional Windows 8 Pro 64bits, Dual Core Intel I5 540M 2800MHz, 4GB Memória RAM DIMM3, HD Ssd raid 0.

2.1.2 Versões de Sistema de Gerenciamento de Banco de Dados

As versões escolhidas para serem instaladas no ambiente de testes foram as mais atuais que estavam disponíveis nos sites oficiais dos desenvolvedores. A última versão disponível do *Firebird* e que foi instalada é a 2.5.2.26539_0_x64 *superclassic*, já a última versão disponível para *PostgreSQL* que foi instalada é a 9.2.3.

Para o funcionamento de um dos softwares de conversão foi necessário também a instalação à parte de um *driver* ODBC para *Firebird* na versão 2.0.1.152 x64.

O monitoramento dos bancos de dados, consultas e manutenções foram executados pelos softwares IBExpert para o *Firebird* e o pgAdmin para o *PostgreSQL*.

2.1.3 Banco de Dados

A base de dados de um cliente da empresa XYZ do software de automação comercial escolhido para os testes de conversão é uma base de médio porte. Este cliente possui uma rede de lojas com 50 filiais e é utilizada há, aproximadamente, 4 anos. Considerando o volume de dados produzido por estas 50 filiais ao longo deste período informado, esta base é considerada de médio porte dentro da empresa XYZ.

Esta base possui o tamanho de aproximadamente 4,5GB. A partir de um comando SQL é possível identificar 18.609.949 registros divididos dentro de 191 tabelas. Dentro da distribuição destas tabelas são encontrados um total de 632 índices. Além dos dados comuns este banco utiliza outras funcionalidades do *Firebird*, listadas a seguir: 11 *views*, 101 *procedures*, 166 t*riggers*, 16 *generators.*

O banco de dados do sistema de automação não faz uma utilização completa dos tipos de dados do SGBD *Firebird*, mas nos seus 3107 campos de tabelas possui uma boa divisão dos tipos de dados. Conforme a tabela 2.1, é possível visualizar todos os tipos de dados utilizados atualmente e sua quantidade de campos.

Tabela 2.1 - Tipos de campos e suas respectivas quantidades

Tipo de Dado	*SmallInt*	*Integer*	*Float*	*Char*	*Double*	*TimeStamp*	*Varchar*	*Blob*
Quantidade de campos	62	730	8	463	444	367	1001	32

2.2 *Full Convert*

O software de conversão *Full Convert* foi criado pela empresa Spectral Core para ser uma das mais robustas soluções de conversão de banco de dados. Segundo o fabricante, ele pode converter mais de 10 tipos de bancos de dados diferentes para os maiores SGBDs disponíveis no Mercado, como *Oracle*, *PostgreSQL*, *SQL Server* e *MySQL*. O *Full Convert* possui uma versão *trial*, porém esta versão não converte todos os dados e tabelas da base de origem, apenas um percentual fixo de dados para avaliação. Para atingir o objetivo proposto de migrar todos os dados do banco de origem, foi adquirida a licença Enterprise do software e instalada a versão 5.23.

A primeira impressão que o software apresenta é de uma *interface* geral moderna e muito bem definida, os menus e tutoriais auto-explicativos e intuitivos. O idioma disponível no programa é o inglês e não possui opção para download de outras linguagens.

Foram realizadas 2 conversões com o *Full Convert*, a primeira executada com todos os parâmetros e configurações *default*, que o próprio software sugere a partir da base escolhida do software de automação comercial. A segunda conversão modificando as configurações do *Full Convert* de acordo com as características do banco de dados do software de automação comercial.

2.2.1 Conversão de Banco de Dados do Software de Automação Comercial com parâmetros *default*

Ao iniciar uma nova conversão, o software *Full Convert* abre uma listagem ao usuário de todos os bancos de dados que são possíveis copiar os dados. Ao todo são 14 opções disponíveis de origem dos dados. Ao lado desta listagem são solicitados os dados de acesso a este banco de dados. No caso deste trabalho as informações solicitadas para conectar em uma base do *Firebird* são: servidor, nome do usuário, senha e o caminho do arquivo ".fdb". Existem também algumas configurações avançadas como protocolos de acesso, portas,

charset, e dll que podem ser modificadas caso necessário. Na figura 2.1 é apresentada a tela inicial de seleção de banco de dados de origem.

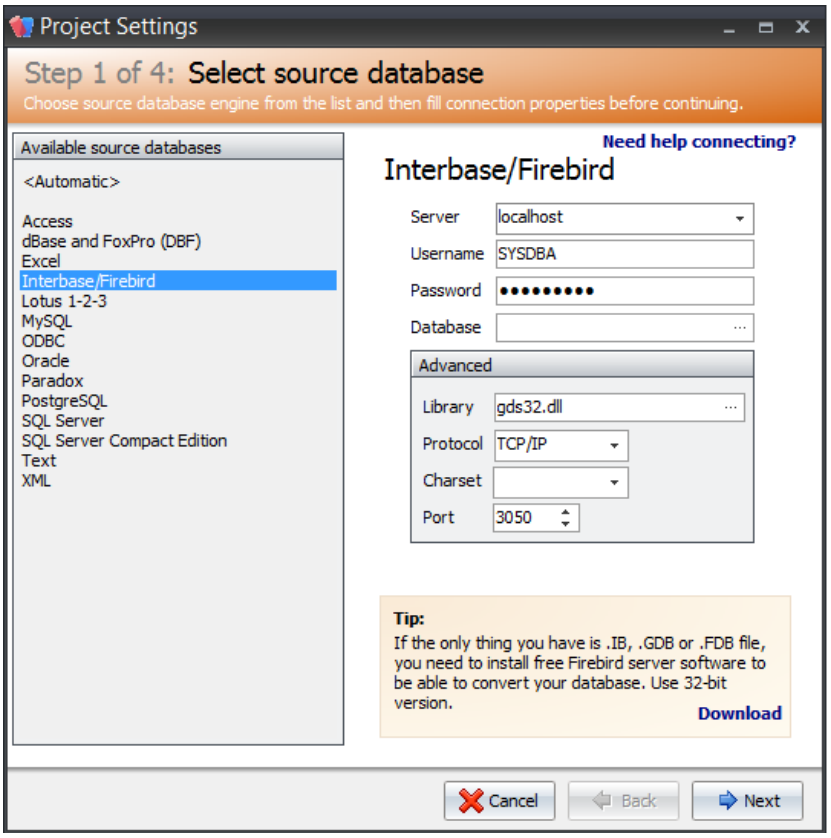

Figura 2.1 - Tela de seleção de banco de dados de origem.

A segunda tela do processo de conversão solicita a conexão com a base de dados de destino, que no caso deste trabalho é uma base previamente criada no *PostgreSQL*. Esta base foi criada com o nome *Full_Convert_teste1*. Após informar as opções corretamente, a tela é apresentada como mostra a figura 2.2.

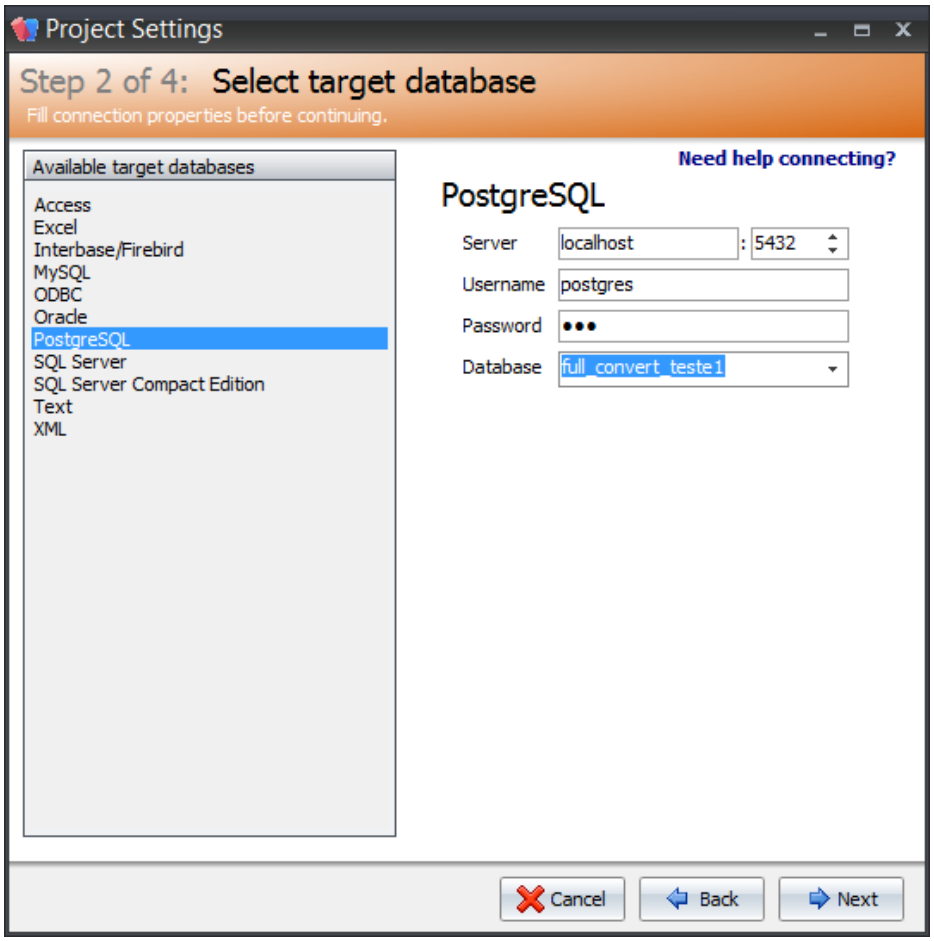

Figura 2.2 - Tela de seleção do banco de dados de destino.

No próximo passo é solicitada a seleção das tabelas da base de origem que devem ser convertidas. Existe a possibilidade de deixar tabelas de fora, modificar campos e estruturas das tabelas antes da conversão e criar scripts para filtrar apenas os dados necessários para ser convertidos para a base de destino.

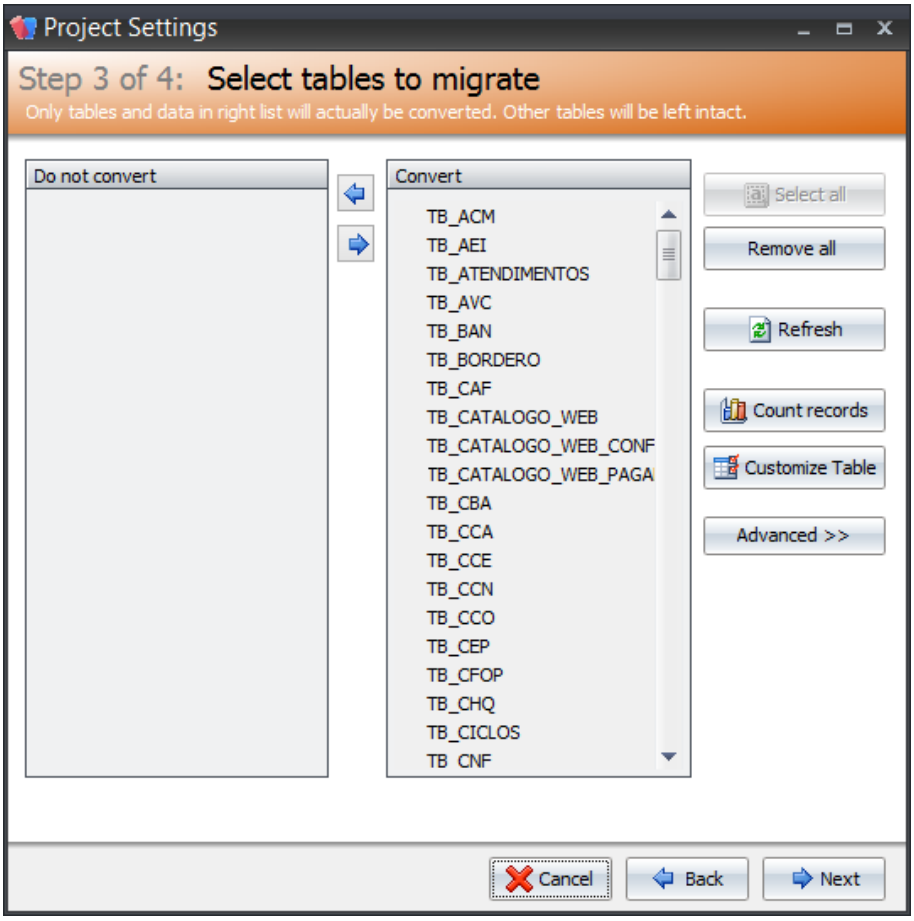

Figura 2.3 - Seleção das tabelas que devem ser convertidas.

Após confirmadas estas informações será solicitado o início da conversão. Esta conversão com as configurações *default*, levou 1:53:19 com uma taxa de 3076 registros por segundo. Os resultados obtidos nesta conversão estão listados na tabela 2.2.

2.2.2 Conversão do Banco de Dados do software de Automação Comercial com parâmetros modificados

O software *Full Convert* apresenta algumas configurações que podem ser aplicadas especificamente para uma base de dados de origem. A partir destas opções é possível configurar a conversão de uma forma que melhor se adapte ao cenário e as características da base. No caso do banco de dados escolhido para este trabalho foi necessário configurar algumas opções para garantir que todos os dados fossem migrados.

Na conversão realizada com as configurações *default* do programa os resultados foram abaixo do esperado conforme a tabela 2.2 demonstra (coluna Conv.1 - F). Para resolver este problema e aumentar o número de registros convertidos foram modificadas as opções

gerais de execução do software *Full Convert.* Os parâmetros modificados foram para utilizar instruções de *insert* ao invés do comando *copy* e a utilização do *charset* ISO8859_1.

Na primeira situação, o software faz uso por *default* do comando *copy from* para cadastrar os registros no banco de dados. Este comando realiza uma cópia dos dados de um arquivo e o copia as tabelas do banco de dados. Esta forma de inclusão de registros no banco de dados é mais rápida que uma série de instruções de *insert* em sequência (MATTHEW, STONES, 2005). Na tentativa de melhorar o seu desempenho o software utiliza esta opção, porém, quando utilizada a opção de *inserts* a quantidade de registros importados aumentou consideravelmente.

Na situação do *charset,* mudando esta configuração, foi garantido que a leitura do software se daria na mesma codificação de caracteres que o banco de origem foi criado e mantido. Os resultados após estas modificações podem ser observados na tabela 2.2.

Utilizando as configurações modificadas o *Full Convert* realizou a conversão em 5:30:02. Os resultados obtidos neste processo foram consideravelmente melhores como podem ser observados na tabela 2.2 (coluna Conv.2 - F).

2.3 *Datapump* for *PostgreSQL*

O software *Datapump*, ou mais especificamente a versão *Datapump* for *PostgreSQL*, é desenvolvido pela empresa ESM Database Management Solutions. Assim como o *Full Convert*, este software não possui uma versão *trial* que atinja as exigências desta pesquisa, portanto foi adquirida a distribuição completa do programa e instalada no ambiente de testes a versão 3.1.0.8.

O *Datapump for PostgreSQL* é diferente dos outros dois softwares testados na questão de conexão. Ele utiliza suporte ADO (ActiveX Data Objects) da Microsoft para realizar as conexões com qualquer banco de dados que possua esta opção. O ADO possibilita que uma aplicação acesse e manipule os dados de uma diversidade de fontes sem a necessidade de saber como o banco de dados foi implementado, isso através de uma camada intermediária entre a linguagem de programação e OLE DB. (MICROSOFT, 2013).

A sua *interface* é simples e ágil e assim como o *Full Convert* este aplicativo também utiliza a forma de *wizard* para guiar o usuário por todos os passos da conversão. O idioma original do programa é inglês, porém ele disponibiliza formas de realizar *downloads* de outras linguagens para atualizar a sua biblioteca.

Para este *software* foram realizados os mesmos 2 testes aplicados ao *Full Convert.* Um teste com banco de dados do sistema de automação comercial e com os parâmetros *default* do *Datapump* e outro com o banco de dados do sistema de automação e com os parâmetros modificados de acordo com as características da base.

2.3.1 Conversão do Banco de Dados do software de Automação Comercial com parâmetros default

Ao iniciar uma nova conversão, o *Datapump for PostgreSQL* apresenta a tela da figura 2.4. Para iniciar a conversão é necessário informar uma base de dados de origem, esta base é passada através de uma *string* de conexão. O *software* fornece uma ferramenta para ajudar a montar esta conexão para qualquer banco de dados que possua suporte à ADO. Abaixo é necessário informar um banco de dados de destino, este em *PostgreSQL*. Neste ponto o *Datapump* possui configurações fáceis e rápidas para ajudar a conectar no banco de dados de destino (que já deve estar previamente criado).

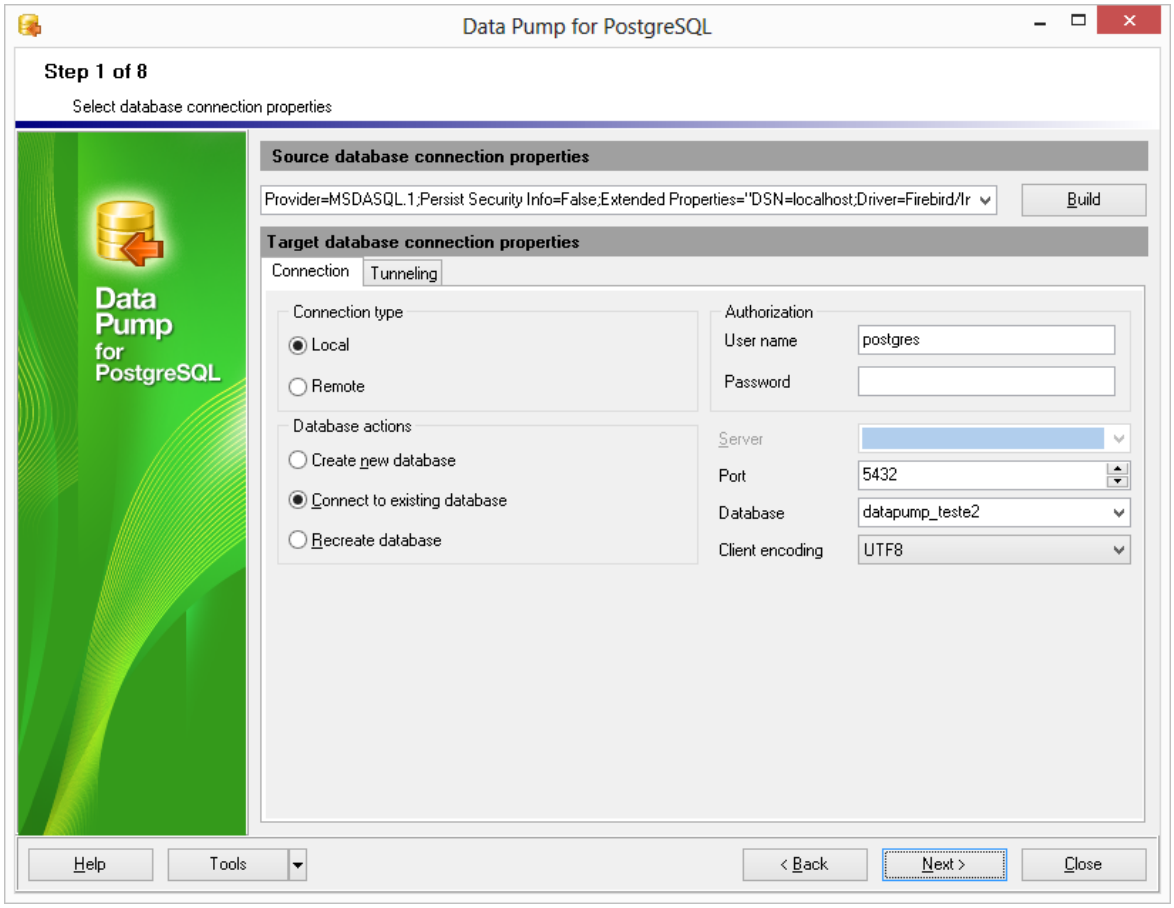

Figura 2.4 - Seleção dos bancos de origem e destino.

Seguindo o fluxo que o software apresenta, é solicitada a escolha das tabelas que devem ser convertidas como é mostrado na figura 2.5. Um detalhe a parte desta seleção é que aqui também são listadas além das tabelas, as *views* do banco de dados. Opção que até agora nenhum dos softwares testados havia disponibilizado. Com as informações básicas de conversão definidas, o *Datapump* abre uma tela com as opções avançadas, nesta parte devem ser selecionados como serão importados os nomes de tabelas, os valores *defaults*, se as tabelas devem ser "zeradas" antes de iniciar o processo e a forma como serão copiados os dados de uma base à outra. Esta tela está exemplificada pela figura 2.6.

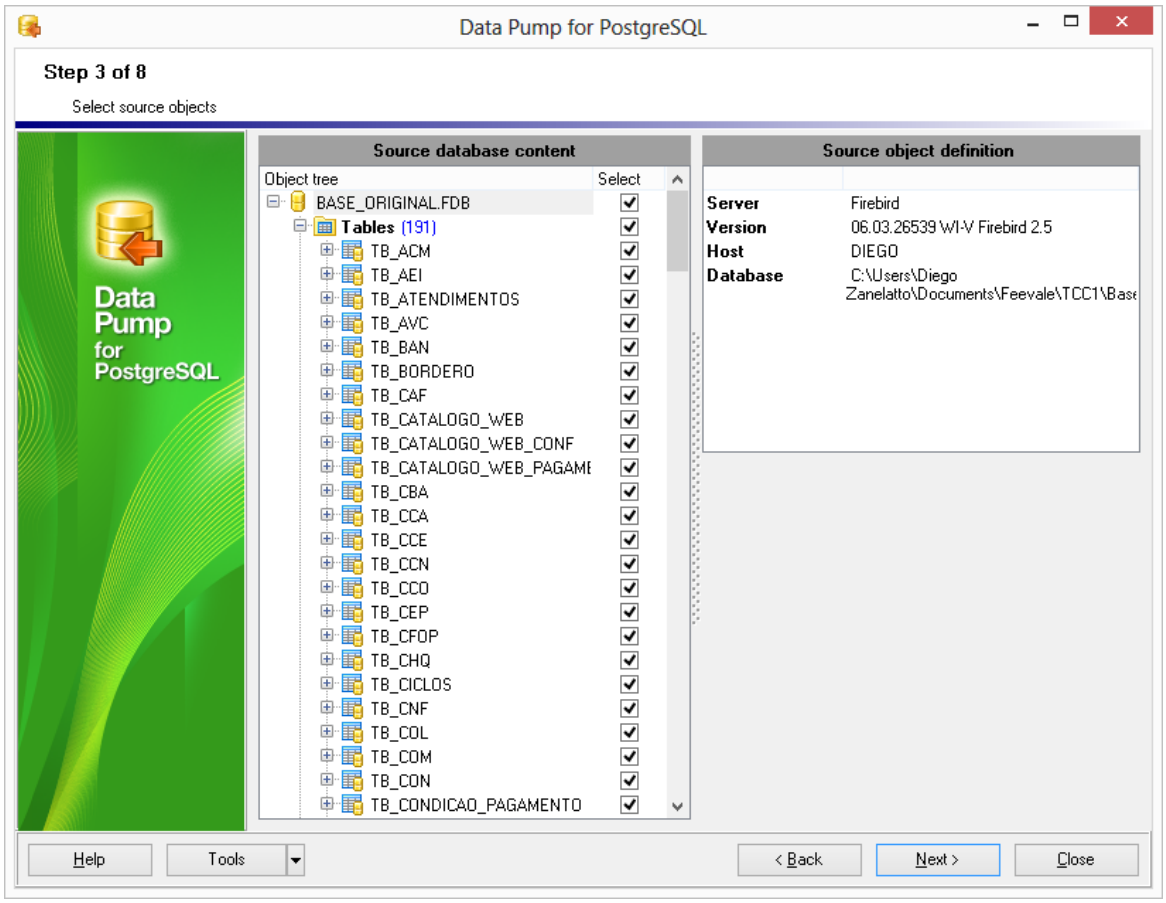

Figura 2.5 - Seleção das tabelas e views.

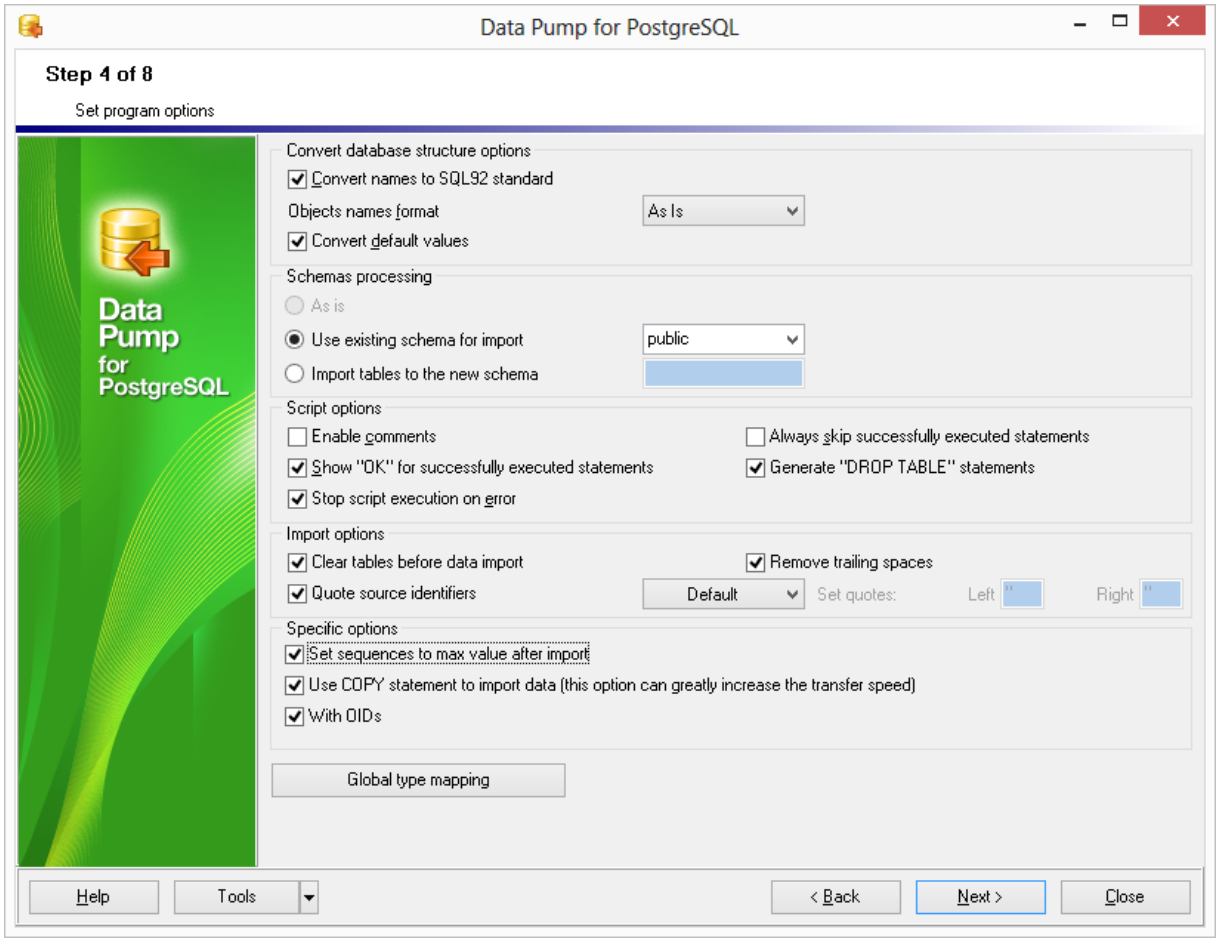

Figura 2.6 - Parâmetros avançados.

Após todas as opções avançadas definidas o *Datapump* apresenta uma tela contendo uma visualização prévia de como deve ficar o banco de dados após a conversão, mostrada na figura 2.7. É possível fazer alguns ajustes finais com relação aos nomes de tabelas e campos e *schemas* do banco de destino neste momento.

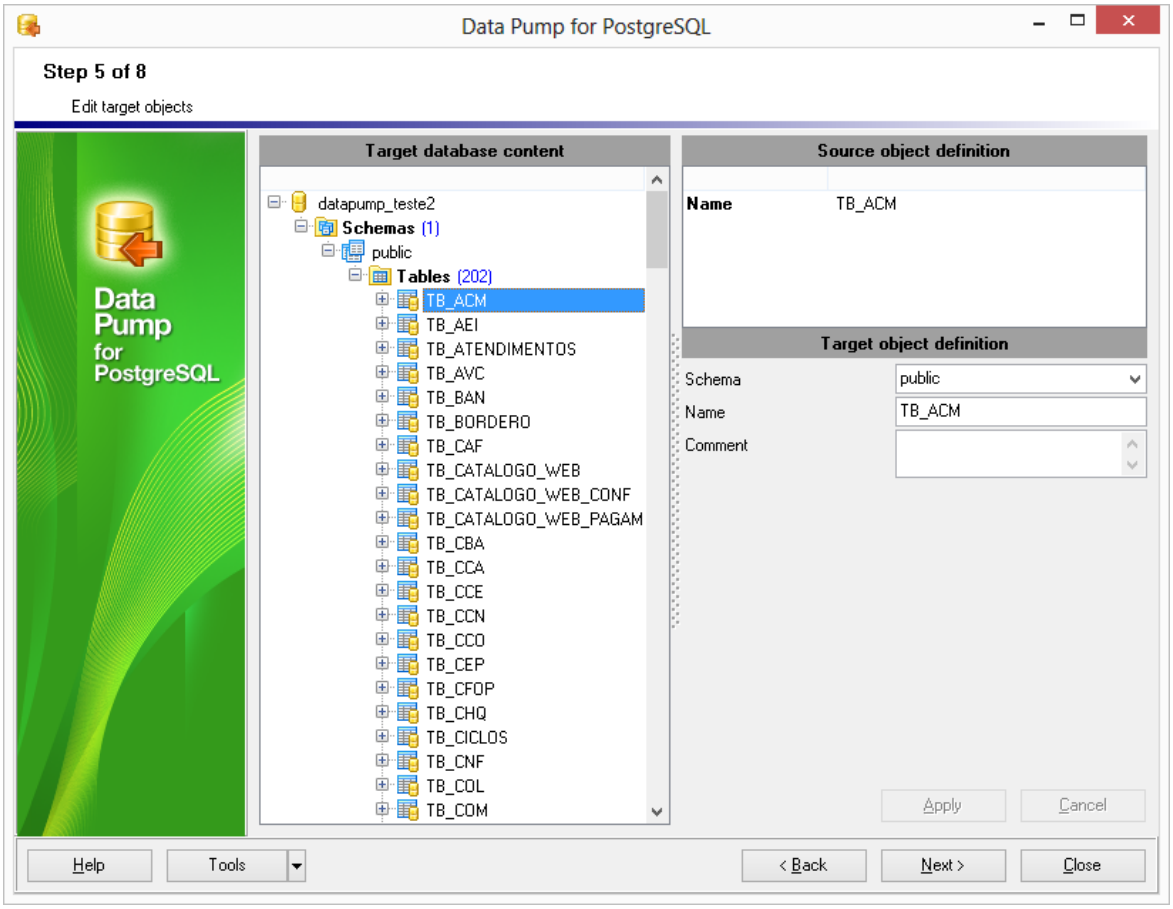

Figura 2.7 - *Preview* das tabelas que serão criadas.

No próximo passo o *Datapump* gera uma listagem com todos os scripts de criação de tabelas, indices e chaves primárias e estrangeiras. O usuário pode modificar e analisar todos eles antes de rodá-los. A sua execução para o banco de dados da automação comercial foi realizada rapidamente e sem nenhum problema, como pode ser vizualizado na figura 2.8.

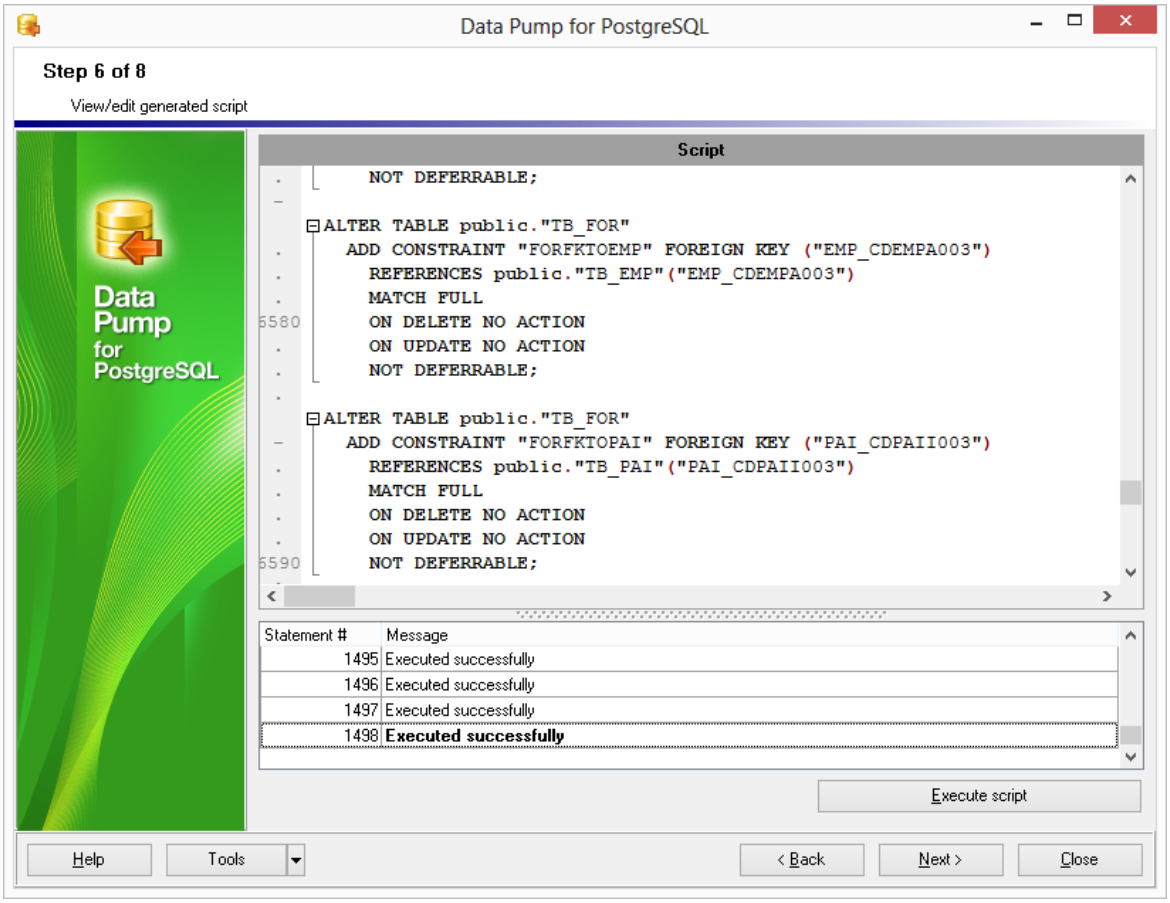

Figura 2.8 - Execução dos scripts.

Com as tabelas criadas o *Datapump* apresenta uma tela onde é possível selecionar e filtrar os dados que serão importados por tabela. Neste filtro por tabela o usuário pode inserir cláusulas de SQL para refinar o filtro dos dados a serem convertidos, como por exemplo, filtrar apenas períodos mais atuais descartando registros muito antigos ou desatualizados. A figura 2.9 mostra as opções da tela de refinamento de consultas dos dados a serem convertidos.

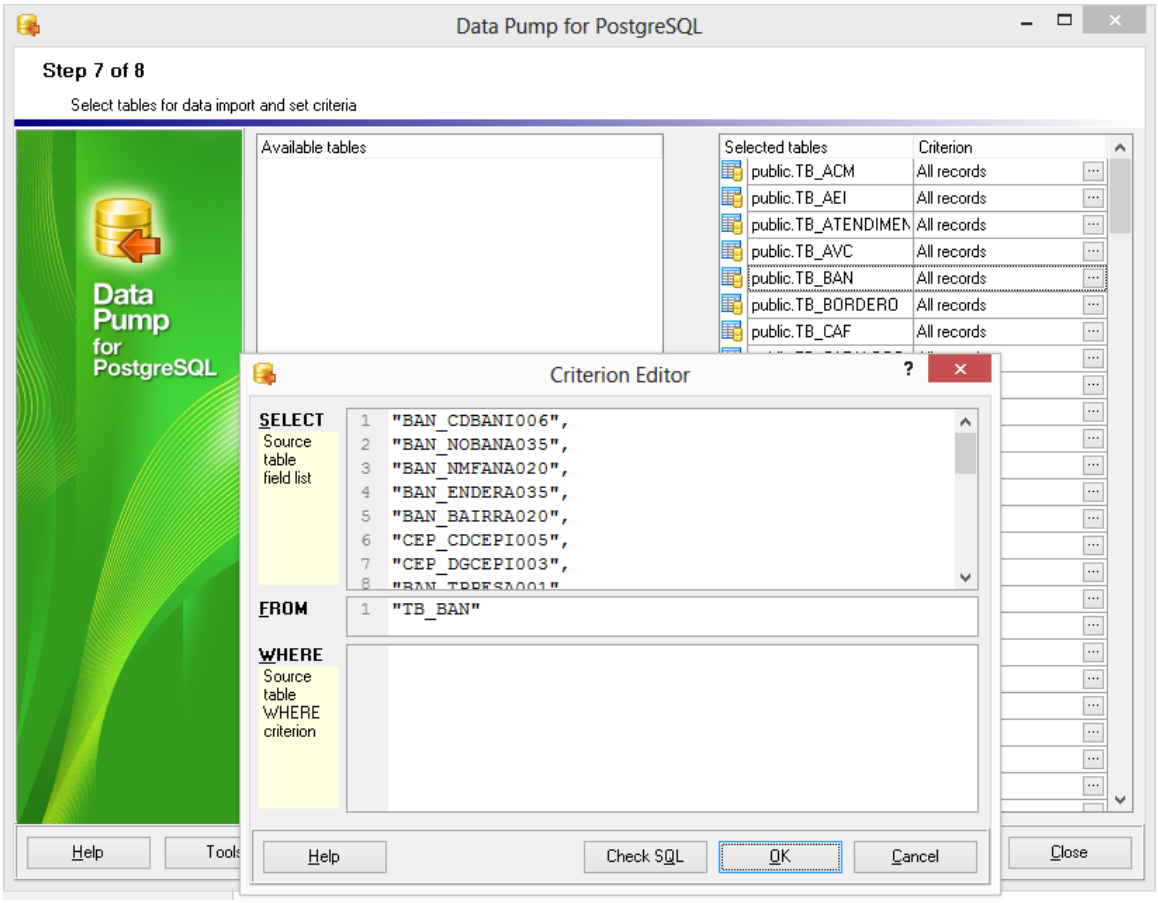

Figura 2.9 - Tela de afinamento das consultas.

A última tela do *wizard* possui apenas um *display* para exibição das mensagens do *log* e o progresso da conversão. A execução da conversão do banco de dados do sistema de automação comercial com os parâmetros do *Datapump for PostgreSQL* em *default* levou 1:45:06 e gerou um *log* de 4322 erros. Os resultados desta conversão podem ser visualizados na tabela 2.2 (coluna Conv.1 - D).

2.3.2 Conversão do Banco de Dados do software de Automação Comercial com os parâmetros modificados

Assim como o *software Full Convert* o *Datapump for PostgreSQL* não conseguiu converter todos os registros utilizando apenas os parâmetros *default*, como é mostrado na tabela 2.2 (coluna Conv.1 - D). Na tentativa de melhorar o resultado da conversão e aumentar o número de registros convertidos foram realizadas algumas modificações nos parâmetros do *Datapump*.

Uma das modificações foi alterar o parâmetro para usar *INSERT* ao invés de *COPY*. Assim como na conversão do *Full Convert*, a utilização do comando *Copy* fez com que ocorressem perdas de registros. Ao usar o comando *insert* o programa começou a apresentar instabilidade na conversão e gerar muitos erros de conversão de dados *timestamp* para *date*. Ao acionar a empresa desenvolvedora do *software* foi constatado que esta anomalia era um *bug*. Este foi corrigido e novos executáveis foram enviados. Estes novos executáveis foram instalados e a partir destes foi realizada uma nova conversão.

Alguns campos de tabelas também tiveram que ser modificados antes de iniciar a conversão, pois na primeira execução foram apresentados erros de tamanho de campo. Estes campos foram alterados e após esta modificação não apresentaram mais problemas.

A duração desta segunda conversão foi acima do esperado. Foram necessárias mais de 8 horas de execução para finalizar o processo. Após a conclusão os resultados obtidos foram muito superiores em comparação à execução com os parâmetros *default* e com isso se aproximando muito do que era esperado no quesito de manutenção da integridade dos dados (tabela 2.2 – coluna Conv.2 - D).

2.4 *Firebird2PostgreSQL*

Os dois primeiros softwares testados neste trabalho são de origem privada e possuem o seu código fonte fechado e suas distribuições dependentes de aquisição de licenças. Através desta afirmação, notou-se importante acrescentar ao trabalho de pesquisa um software de distribuição livre. O escolhido que se enquadra nestes requisitos é o *Firebird2PostgreSQL*.

Este software faz parte do site de projetos de *PostgreSQL pgFoundry* (PGFOUNDRY, 2013). Neste repositório de projetos relacionados ao banco de dados *PostgreSQL*, o autor desenvolveu seu software e disponibilizou a sua última versão na data 23/01/2009. É um software livre e possui o seu código fonte disponível para download juntamente com o seu executável. Após uma rápida instalação, o programa apresenta uma interface básica para realizar a sua tarefa (Figura 2.10). Na sua tela principal é solicitado o caminho de um arquivo ".fdb", da base de origem do *Firebird* e a seleção da base de destino previamente criada no *PostgreSQL*. Como ponto positivo, este programa não necessita nenhuma configuração de ODBC ou linha de acesso para conectar em um dos SGBDs.

Após configurado, a única opção disponível para o usuário é o botão Start. Selecionando esta opção, o programa começa a execução. Porém, ao iniciá-la, o erro 42601 é

retornado, alertando que o tipo de campo “float8” não é permitido. Para tirar a prova de que o banco de dados do sistema de automação comercial utilizado não era o problema, foi criado um novo banco mais simples, com apenas 3 tabelas relacionadas entre si, com alguns registros gerados aleatoriamente por um programa. Mesmo com esta base simples, de alguns MBs, o programa não finalizou a sua tarefa básica de conversão.

Após uma busca por sites e no próprio fórum destinado ao projeto, não foi possível encontrar uma forma de contornar este problema. Foram encontrados outros desenvolvedores com os mesmos problemas e nenhuma resposta do autor do projeto. Devido a estes fatos, não foi possível finalizar a análise deste software.

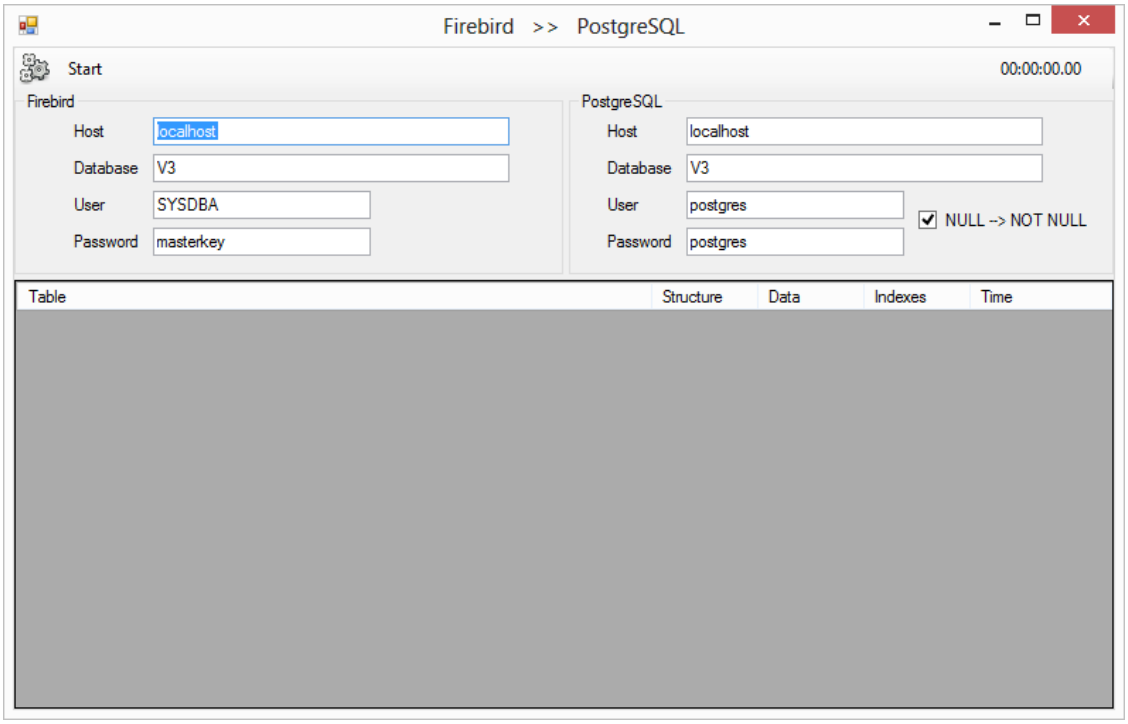

Figura 2.10 - Tela inicial do *software Firebird2PostgreSQL*.

2.5 Resultados

Para analisar os resultados, foram consideradas as quantidades de registros importados, velocidade de importação e integridade dos dados. Apesar de várias tentativas, não foi possível realizar uma conversão com o software *Firebird2PostgreSQL*, então não foi possível inclui-lo nesta análise.

O software *Full Convert* em sua configuração *default* apresentou um resultado insatisfatório, convertendo 88% dos dados. Destes 12% que não foram convertidos, a maior parte dos dados foi de tabelas críticas para o sistema como as de controle de movimentações de produtos, tabela de CEP, e tabelas de dados complementares de notas fiscais. Devido à

importância dos dados não convertidos e do alto índice de problemas, esta configuração foi considerada abaixo da expectativa.

A segunda execução do software *Full Convert* com os seus parâmetros modificados de forma que a conversão se adaptasse às características da base de dados de origem resultou em um aumento de registros migrados. Foram convertidos 99,92% dos registros da base de dados deixando de converter apenas alguns registros de tabelas de *log* e alguns registros de tabelas menos importantes.

Em ambas execuções o software *Full Convert* se mostrou rápido, executando suas conversões em tempos abaixo de 2 horas de duração. Quanto à integridade dos dados, em nenhuma das situações apresentadas ele atingiu 100%, porém em sua segunda execução atingiu índices satisfatórios.

O software *Datapump* teve a sua primeira execução com um desempenho muito abaixo do esperado, convertendo apenas 76,84% dos dados da base original. Com quase 30% de perda de dados é obvio que há uma perda significante da integridade dos dados na base convertida, descartando-se totalmente o uso da mesma em um sistema.

Na segunda execução, modificando os parâmetros de acordo com as características da base de dados de origem, o *Datapump* teve seus resultados aprimorados. Obteve um índice de 99,88% de conversão dos dados. Assim como o *Full Convert*, deixou de fora alguns registros de tabelas de *log*, mas os dados críticos e importantes foram mantidos e convertidos corretamente.

Outro ponto positivo do software *Datapump* foi o seu suporte eficiente que, quando acionado, resolveu o problema e liberou uma nova versão em menos de uma semana. Um ponto negativo a ser avaliado é a lentidão do software em sua execução com os parâmetros modificados. Não utilizando o comando COPY, a execução durou mais de 8 horas, um tempo elevado em comparação com o primeiro software testado o *Full Convert*.

Apesar de os dois softwares conseguirem converter mais de 99% dos registros e atingir um índice satisfatório para a avaliação deste trabalho deve-se destacar que eles não possuem ferramentas para converter situações específicas do banco de dados *Firebird* como procedures e triggers. O software que mais se aproximou deste parâmetro foi o *Datapump* que converteu as *views* do banco de dados de origem.

Como pode ser visualizado na tabela 2.2, os resultados dos testes de conversão estão apresentados da seguinte forma: A primeira coluna representa os nomes das tabelas do

sistema; a segunda coluna apresenta os registros totais de cada uma das tabelas do banco de dados de origem; e as colunas seguintes representam os testes respectivamente do *Full Convert* com parâmetros default, *Full Convert* com os parâmetros modificados, *Datapump* com parâmetros default e *Datapump* com parâmetros modificados. As células com cores em vermelho apresentam diferenças entre o número de registros convertidos e a quantidade de registros total esperado para a tabela. Apenas as tabelas que houveram divergências foram listadas nesta tabela.

Tabela 2.2 - Resultados das conversões.

Tabelas	Registros	Conv.1 - F	Conv.2 - F	Conv.1 - D	Conv.2 - D
TB_ACM	567734	0	567734	563734	567234
TB_AEI	112930	0	107931	112930	112930
TB_ATENDIMENTOS	38272	0	38272	0	38272
TB_AVC	11992	0	11992	0	11992
TB_BAN	3	0	3	3	3
TB_BORDERO	275	0	275	275	275
TB_CATA*LOG*O_WEB	63	0	63	63	63
TB_CATA*LOG*O_WEB_CONF	21	0	21	21	21
TB_CCA	56	0	56	56	56
TB_CEP	645344	0	645344	605344	645344
TB_CFOP	5	0	5	5	5
TB_CHQ	1436	0	1436	0	1436
TB_CNF	773970	0	773970	193000	773970
TB_COL	1839	0	1839	1839	1839
TB_CPF	307957	307957	307957	24000	307957
TB_CPJ	307982	307482	306982	71000	307982
TB_CVE	163	163	163	0	163
TB_DCL	16009	16009	16009	0	16009
TB_EIT	3029331	3029331	3029331	2946331	3020631
TB_IPC	518200	518200	518200	0	518200
TB_ITENS_PEDIDO_E_COMMERCE	1978	1978	1978	0	1978
TB_LID	84156	84156	84156	0	84156
TB_LID_CODIGO_ETIQUETA	163	163	163	0	163
TB_METAS_VEN	3954	3954	3954	0	3954
TB_MFC	174806	167307	167307	174806	174806
TB_MNF	2287984	2287984	2287984	0	2277484
TB_MNFC	12995	12995	12995	0	12995

TB_MNF_CODIGO_ETIQUETA	53444	53444	53444	0	52044
TB_NFA	11992	11992	11992	0	11992
TB_NFC	12016	12016	12016	0	12016
TB_NFP	5511	5511	5511	0	5511
TB_NF_E_COMMERCE	687	687	687	0	687
TB_PCO	17819	17819	17819	0	17819
TB_PEDIDO_E_COMMERCE	1024	1024	1024	0	1024

Com base na análise realizada neste trabalho o software escolhido para dar continuidade a este processo de conversão do sistema de automação comercial foi o *Datapump for PostgreSQL*. Este, além de converter os dados, índices e chaves conseguiu também converter as *views* e, sendo assim, foi o que mais se aproximou dos requisitos definidos para este projeto, mesmo tenho atingido um índice de conversão um pouco menor em comparação com o software *Full Convert*.

Com o software de conversão escolhido é necessário definir o que deverá ser convertido manualmente pois este software somente converterá os dados, índices, chaves primárias e estrangeiras e *views*. No sistema de automação comercial atualmente são utilizadas as *procedures* e *triggers* do banco de dados *Firebird*, e assim como foi constatado no capítulo 1 deste trabalho, estas deverão ser convertidas manualmente.

3 MIGRAÇÃO DOS MÓDULOS SELECIONADOS

Após concluídos os estudos de cada um dos bancos de dados envolvidos neste trabalho e finalizadas as conversões dos dados foi possível dar início a migração do sistema e seus módulos. Para fins de testes e avaliações, este trabalho converteu 3 módulos que apresentam situações de extremo uso dos bancos de dados envolvidos. Um módulo é responsável por uma grande inserção de dados no SGBD, outro módulo é responsável por grandes consultas nas tabelas mais populosas, e o último realiza uma rotina completa de venda, acessando as principais tabelas do banco de dados, por meio de rotinas alternadas de inserção e consulta.

3.1 Sistema

O sistema de automação é desenvolvido pela empresa XYZ desde 1999. Em seus primeiros 7 anos foi programado em COBOL e teve seus recursos apontados para a área de automação comercial e industrial. Após 2005 foi convertido para uma plataforma gráfica utilizando Delphi 5 e *log*o após desenvolvido em Delphi 7, mudando o seu foco especificamente para automação comercial. Atualmente a versão de Delphi utilizada para o seu desenvolvimento é o Delphi 2007.

O sistema é dividido atualmente em uma estrutura de 245 módulos. Estes módulos abrangem todo o funcionamento de uma rede de lojas, desde o controle de estoque, controles fiscais, vendas e gestão. Cada um destes módulos trabalha com as informações do usuário, captando dados, processando-os se necessário e recebendo novos dados e informações para exibição.

Estes módulos possuem um funcionamento integrado com 7 *dll* principais do sistema. Elas realizam todas as definições de conexões com o banco de dados, possuem as instalações e configurações de componentes necessários para executar o programa e implementam todas as regras de negócios que os módulos do sistema utilizarão em qualquer transação com o banco de dados. Esta forma de organização das *dll* criada internamente facilitou todo o processo de conversão do sistema. Com toda a regra de negócios centralizada, as alterações ficam focadas apenas nestas *dll* principais e os módulos não são alterados.

O sistema de automação realiza as conexões com o banco de dados através de uma conexão *DBExpress*[4], que é uma arquitetura da empresa Embarcadero[5], desenvolvedora do Delphi. Esta arquitetura é composta por *drivers* que acessam o banco de dados de forma rápida e eficiente. Para cada banco de dados existe uma biblioteca independente de *interfaces* para o processamento de comandos SQL. Sendo assim esta arquitetura permite que a aplicação se conecte a uma série de banco de dados sem a necessidade de alterar a tecno*log*ia de acesso ao banco.

A arquitetura *DBExpress* possibilita a escolha de *driver*s para usufruir de suas funcionalidades. O *driver* utilizado pelo sistema atualmente é o Devart[6], este desenvolvido pela empresa que dá nome ao *driver*. Anteriormente o *driver* utilizado pelo sistema de automação era o *driver default DBExpress*, disponibilizado pelo próprio Delphi, porém após testes de desempenho realizados pela empresa, foi constatado que o *driver* Devart é mais performático.

3.2 Definições Técnicas dos Módulos Escolhidos

Como citado na introdução deste capítulo, os módulos escolhidos para este trabalho apresentam características específicas para identificar possíveis problemas nas rotinas básicas de um SGBD como inserções, consultas e atualizações. O módulo de Relatório de Desempenho testa uma consulta com grande quantidade de dados relacionando tabelas críticas do banco de dados. A consulta de clientes realiza uma consulta consolidada em uma das maiores tabelas do cliente e por fim o *StressTest* que é módulo que testa os limites do banco de dados e realiza uma grande quantidade de consultas, inserções e atualizações.

3.2.1 Relatório de Desempenho

O Relatório de Desempenho é um dos relatórios mais utilizados no sistema de automação comercial pelos clientes da empresa XYZ. Nesta listagem são realizados os cruzamentos de informações de compra e venda, além de também possibilitar a análise de índices de PA (Produtos Atendidos), Ticket Médio (valor médio entre as vendas realizadas), atendimentos, Taxa de Conversão e outros indicadores essenciais para o varejo. Abaixo segue a figura 3.1 apresentando algumas informações do Relatório de Desempenho.

---

[4] Fonte do que é *DBExpress****
[5] www.embarcadero.com
[6] Informações sobre o DevArt (site, acesso)***

	Compras	%Compras	NF Compras	Vendas	%Vendas	NF Venda	PA	Media PA	Ticket Médio	Atend.	TaxaConv.	Vlr.M2	%M2
Janeiro/2012	37.186,57	27,48	39	76.536,34	33,96	433	1310	3,03	176,76	862,00	50,00	2.341,99	33,96
Fevereiro/2012	84.314,02	62,31	32	47.167,80	20,93	346	616	1,78	136,32	808,00	43,00	1.443,32	20,93
Março/2012	13.814,02	10,21	29	101.635,24	45,10	712	1133	1,59	142,75	1.367,00	52,00	3.110,01	45,10
Total	135.314,62	100,00	100	225.339,38	100,00	1491	3059	2,05	151,13	3.037,00	48,33	6.895,33	99,99

Figura 3.1 – Informações e indicadores do Relatório de Desempenho

Para possibilitar o cruzamento destas informações, este relatório realiza algumas instruções de SQL (Quadro 3.1) com um alto nível de complexidade. Estas instruções possuem relacionamentos de grandes tabelas, *selects* dentro de *selects* e agrupamentos definidos dinamicamente pela aplicação.

Outro fator que faz com que este relatório seja um bom parâmetro para os testes deste trabalho é que ele necessita acessar uma grande quantidade de dados das maiores tabelas do sistema. Como este relatório é executado diariamente pelas lojas, em horários de fechamento, ele resulta em um aumento no processamento compartilhado no servidor, onde o banco de dados está rodando. Com o processamento e I/O do servidor sendo compartilhados por vários *clients*, acessando este relatório ao mesmo tempo, é interessante verificar se, com troca do SGBD, o sistema ganhará em desempenho.

```
Select EMP_CDEMPA003,
ANO, MES,
Sum(Notas) Notas, Sum(Valor) Valor, Sum(QtdItens) QtdItens,
sum(ValorFrete) ValorFrete from (
select m.EMP_CDEMPA003 EMP_CDEMPA003,
EXTRACT(YEAR FROM m.mnf_DTMOVD000) ANO, EXTRACT(MONTH FROM
m.mnf_DTMOVD000) MES,
m.nfi_seriea003, m.nfi_cdpora006, m.nfi_nunofi006,
0 notas, sum(m.mnf quantn011)qtditens,
sum((m.mnf_quantn011 * m.mnf_prunin017)-(m.MNF_PRUNIN017 *
m.MNF_QUANTN011 * m.MNF_PEDESN005/100)
  +(m.MNF_PRUNIN017 * m.MNF_QUANTN011 * m.MNF_PEACRN005/100)) valor,
 sum(TB_NFI.NFI_VLFREN017 / (SELECT COUNT(MNF.ITE_CDITEA015) FROM
                             TB_MNF MNF WHERE tb_nfi.emp_cdempa003 =
mnf.emp_cdempa003
                             and tb_nfi.nfi_cdpora006 = mNF.nfi_cdpora006
                             and tb_nfi.nfi_seriea003 = mNF.nfi_seriea003
                             and tb nfi.nfi nunofi006 =
mNF.nfi_nunofi006)) VALORFRETE
from tb_mnf m
inner join tb_nfi on tb_nfi.emp_cdempa003 = m.emp_cdempa003
                 and tb nfi.nfi cdpora006 = m.nfi cdpora006
                 and tb_nfi.nfi_seriea003 = m.nfi_seriea003
                 and tb_nfi.nfi_nunofi006 = m.nfi_nunofi006
                 and  (tb_nfi.nfi_tpnfia001 = 'V' or (
(tb nfi.nfi tpnfia001 = 'G') and (tb nfi.nop cdnata004 in
('5102','5101','6101','6102')) ) ) and tb nfi.nfi seriea003 <> 'VLP'
inner join TB_TMO TMO on m.TMO_CDMOVI003 = TMO.TMO_CDMOVI003
AND TMO.TMO_SITUAA001 = 'S'
where 0=0
```

```
 AND TB_NFI.NFI_DTMOVD000 >= '01.03.2013' AND TB_NFI.NFI_DTMOVD000 <=
'31.03.2013 23:59:59'
 AND TB NFI.EMP CDEMPA003 IN ('001')

GROUP BY 1,2,3,4,5,6
union all
select m.EMP_CDEMPA003 EMP_CDEMPA003,
EXTRACT(YEAR FROM m.mnf DTMOVD000) ANO, EXTRACT(MONTH FROM
m.mnf_DTMOVD000) MES,
m.nfi_seriea003, m.nfi_cdpora006, m.nfi_nunofi006,
1 notas, 0 qtditens, 0 Valor, 0 VALORFRETE
from tb_mnf m
inner join tb nfi on tb nfi.emp cdempa003 = m.emp cdempa003
                 and tb_nfi.nfi_cdpora006 = m.nfi_cdpora006
                 and tb_nfi.nfi_seriea003 = m.nfi_seriea003
                 and tb_nfi.nfi_nunofi006 = m.nfi_nunofi006
                 and  (tb_nfi.nfi_tpnfia001 = 'V' or (
(tb_nfi.nfi_tpnfia001 = 'G') and (tb_nfi.nop_cdnata004 in
('5102','5101','6101','6102')) ) ) and tb_nfi.nfi_seriea003 <> 'VLP'
inner join TB_TMO TMO on m.TMO_CDMOVI003 = TMO.TMO_CDMOVI003
AND TMO.TMO_SITUAA001 = 'S'
where 0=0
 AND TB_NFI.NFI_DTMOVD000 >= '01.03.2013' AND TB_NFI.NFI_DTMOVD000 <=
'31.03.2013 23:59:59'
 AND TB_NFI.EMP_CDEMPA003 IN ('001')

    and (exists /* existe algum pagamento que nao e bonus */
     (select * from tb_mca mca
       inner join tb_tmc tmc on tmc.tmc_cdmovi006 = mca.tmc_cdmovi006
      where mca.mca cdemoa003 = TB NFI.emp cdempa003
            and mca.mca ordesa006 = TB NFI.nfi cdpora006
            and mca.mca_seriea003 = TB_NFI.nfi_seriea003
            and mca.mca_nunofi006 = TB_NFI.nfi_nunofi006
            and ((tmc.tmc_tpmova001 <> 'B') or (tmc.tmc_tpmova001 is
null) ))
         or (not exists /* ou nao existe nenhum pagamento */
             (select * from tb_mca mca
               inner join tb_tmc tmc on tmc.tmc_cdmovi006 =
mca.tmc_cdmovi006
              where mca.mca cdemoa003 = TB NFI.emp cdempa003
                    and mca.mca_ordesa006 = TB_NFI.nfi_cdpora006
                    and mca.mca_seriea003 = TB_NFI.nfi_seriea003
                    and mca.mca_nunofi006 = TB_NFI.nfi_nunofi006)
            )
        )
GROUP BY 1,2,3,4,5,6
) as Desempenho
GROUP BY 1,2,3
```

Quadro 3.1 - SQL do Relatório de Desempenho

Este relatório possui vários tipos de agrupamentos, modos de exibições e cruzamentos de dados. Cada um dos tipos de agrupamento ou resumos modifica significantemente as instruções SQL, aumentando ainda mais a diversidade das consultas que podem ser realizadas por este módulo e a complexidade da consulta. Cada uma destas

variações foi testada para validar se a integridade das informações consultadas foi mantida após a migração do módulo e também verificar se existem alterações no desempenho do mesmo.

3.2.2 Consulta de Clientes

O módulo de consulta de clientes, assim como todos os módulos que realizam consultas de dados do sistema, segue um padrão de interface e funcionalidades específicas. Por padrão todos os módulos de consulta possibilitam a edição dos campos que compõem o filtro. Caso o escopo de dados do módulo possua 30 campos diferentes, todos estes campos estão disponíveis ao usuário para compor o filtro. Esta montagem de filtro e a montagem do SQL para a consulta é realizada pelo próprio *framework* do sistema de forma automática, portanto as modificações necessárias para migrar de *Firebird* para *PostgreSQL* serão especificamente realizadas nesta camada do sistema, não sendo necessário modificar a interface.

Outro fator impactante para a conversão é a estrutura de paginação aplicada nos módulos de consulta do sistema. A consulta de cliente, assim como em todos os módulos de consulta do sistema possui uma funcionalidade de paginação, a qual verifica se o resultado de uma consulta SQL possui mais de 50 registros, caso isso ocorra as instruções SQL são modificadas para filtrar apenas de 50 e 50 registros. Caso o usuário avance a sua seleção em tela o sistema identifica este avanço e gera novas instruções de consulta. No *Firebird* as funções utilizadas para realizar esta tarefa nos comandos SQL são o FIRST e o SKIP.

O cliente de um estabelecimento comercial pode ser considerado tanto como uma pessoa Física ou Jurídica. Para esta situação ser atendida pelo sistema foi criada uma definição em que, cada um destes tipos de cliente, é gravado em uma tabela diferente. Foram então criadas as tabelas TB_CPJ (pessoa jurídica) e TB_CPF (pessoa física).

Por possuir menos registros que um cliente pessoa física, a tabela de pessoa jurídica possui menos campos, porém todos são campos básicos para os dois tipos de cliente. Um cliente pessoa jurídica possui o seu registro apenas na tabela TB_CPJ, porém um cliente de pessoa física possui os seus registros tanto na tabela TB_CPJ como na TB_CPF. Partindo desta situação todos os clientes de pessoa física possuem 2 registros no banco de dados ligados por uma chave estrangeira do seu campo código.

Alguns exemplos de consultas podem ser visualizados nos quadros 3.2 e 3.3. O quadro 3.2 apresenta uma consulta dos dados de um cliente selecionado pelo seu código, já o quadro 3.3 apresenta uma consulta paginada, onde o filtro executado é pelo campo de razão social no qual se solicita ao SGBD que sejam consultados todos os clientes que comecem com a *string* "ANA".

```
SELECT
TB_CPJ.*,
TB_CPF.*
FROM TB CPJ
LEFT JOIN TB_CPF ON (TB_CPF.CPJ_CDCLII006 = TB_CPJ.CPJ_CDCLII006)

WHERE TB_CPJ.CPJ_CDCLII006 = 313189
```

Quadro 3.2 – Exemplo de SQL do Módulo de Consulta de Cliente

```
SELECT
FIRST 50 SKIP 0
TB_CPJ.*,
TB_CPF.*,
TB_SCL.*
FROM
TB_CPJ
INNER JOIN TB_SCL ON (TB_SCL.SCL_CDOBSI003 = TB_CPJ.SCL_CDOBSI003)
LEFT JOIN TB_CPF ON (TB_CPF.CPJ_CDCLII006 = TB_CPJ.CPJ_CDCLII006)

WHERE TB CPJ.CPJ CDCLII006 > 0 AND (TB CPJ.CPJ RZSOCA060) STARTING WITH
'ANA'

ORDER BY CPJ_RZSOCA060
```

Quadro 3.3 – Exemplo de SQL de paginação do módulo de Consulta de Clientes

3.2.3 *StressTest*

O *StressTest* não é classificado como um módulo do sistema e sim um programa auxiliar de grande importância para os testes de versão, utilizado pela equipe de desenvolvimento. Este programa testa toda rotina de venda do sistema de forma que é possível escalar sua execução, simulando várias lojas realizando vendas simultaneamente e sem interferências.

Mesmo não sendo um módulo, ele foi escolhido para ser convertido pois executa uma das tarefas mais completas em questão de uso de banco de dados. Ele realiza uma série de consultas e inserções nas tabelas mais acessadas e populosas do sistema, sendo assim, capaz de testar a capacidade da estrutura de servidores de um cliente.

Este aplicativo roda por meio de *threads*, podendo ser escalável ao ponto de utilizar o máximo de recursos de um servidor, tanto do ponto de vista do banco de dados, como também do processamento e tráfego de rede. Geralmente o teste realizado nos clientes é seguido por alguns parâmetros, como por exemplo o número de pontos de venda que estarão abertos ao mesmo tempo em um dia de pico da rede de lojas. Com este número definido, são instanciados *StressTest* na mesma quantidade de pontos de venda, a fim de simular um pico de vendas e movimentações.

A tela inicial do programa define todas as configurações e comportamentos da rotina de venda. A figura 3.2 apresenta a tela inicial do *StressTest*.

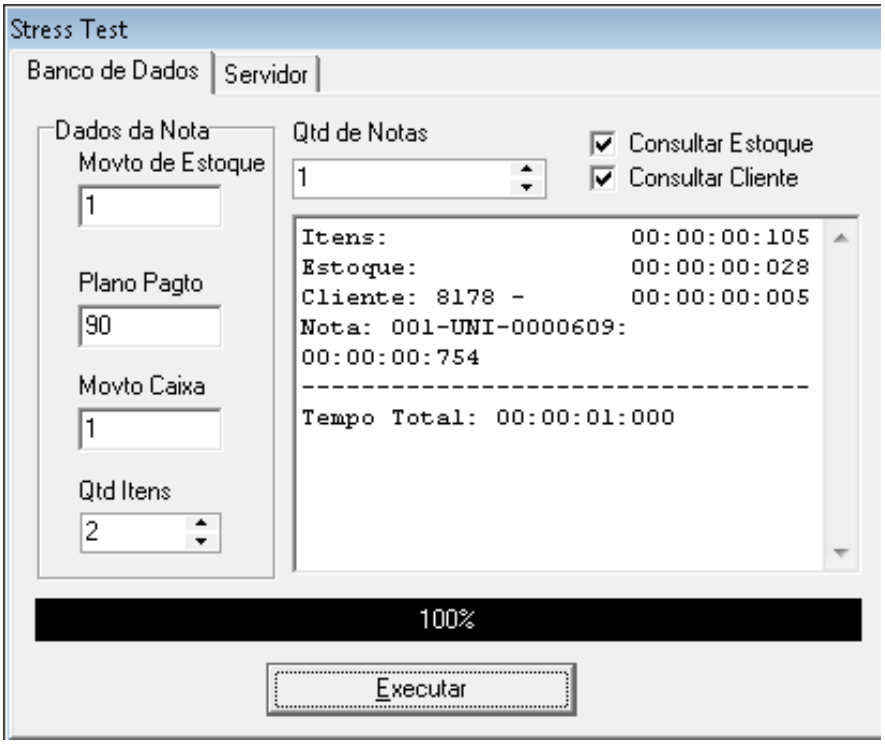

Figura 3.2 – Tela inicial do programa *StressTest*.

As informações que podem ser configuradas na tela inicial do programa são a quantidade de notas que ele vai executar em sequência, as informações básicas de pagamento da nota que são o movimento de estoque, plano de pagamento e movimento de caixa e também pode ser configurada a quantidade de produtos que ele incluirá na nota fiscal. Outros parâmetros que podem ser configurados são: se ele deve consultar o estoque dos produtos e também um cliente aleatório para a nota.

Ao rodar o programa, o contador de notas inicia, de acordo com o parâmetro definido em tela, e começa a cadastrar os documentos fiscais no banco de dados. Para cadastrar uma

nota fiscal ele executa uma série de consultas, atualizações e inserções que podem ser acompanhadas na listagem abaixo:

1 – Consulta os dados da empresa, os seus parâmetros e configurações relacionadas à venda. As tabelas envolvidas nesta consulta são pequenas e possuem poucos registros.

2 – Consulta de forma randômica a quantidade de produtos configurada na tela inicial do programa. Geralmente este valor é medido com antecedência utilizando a média de PA (Produtos Atendidos por nota) da rede de lojas em questão. As tabelas envolvidas nesta consulta possuem grandes quantidades de registros e um vínculo simples entre a tabela de produto e a tabela de grade. Apesar da grande quantidade de registros, esta consulta não apresenta nenhuma instrução de SQL mais complexa.

3 – Avalia a quantidade em estoque dos produtos selecionados. Este processo é realizado na rotina de venda para não permitir a saída de um produto com estoque zerado. Mesmo sendo um programa de teste e não tendo a necessidade de fazer esta validação, esta rotina foi mantida no fluxo do programa, pois a tabela consultada neste processo possui uma grande quantidade de registros e a instrução SQL necessária para retornar os registros é de nível avançado. Por ser uma tabela que registra o estoque do produto por empresa, grade e por data, mantendo o registro de estoque diário, esta tabela se tornou uma das maiores do banco de dados e por este motivo é de extrema importância testar o seu desempenho.

4 – Consulta os preços dos produtos selecionados. A tabela de preços, a exemplo da tabela de estoque, também possui uma grande quantidade de registros, porém não tem a característica de registrar os preços por data, diminuindo assim consideravelmente o seu tamanho em relação à tabela de estoque. Esta consulta, por utilizar algumas *Stored Procedures*, deixa o seu funcionamento um pouco mais complexo. Desta forma ela traz aos testes um diferencial importante, pois será necessário verificar se estas *Procedures* manterão o desempenho, quando convertidas às funções no *PostgreSQL*.

5 – Consulta de forma randômica o cliente para a nota. Apesar de a tabela de clientes ser grande, na maior parte das redes de lojas, esta consulta não possui nenhuma complexidade que agregue um fator novo aos testes. Por consultar apenas um código aleatório utilizando o campo chave da tabela, o banco de dados acaba usando o índice para reduzir o tempo de execução. Esta rotina foi mantida no fluxo do programa, pois o cliente faz parte da chave da tabela de nota e é necessário informá-lo para possibilitar a gravação do documento.

6 – Consulta as formas de pagamento e movimentos de caixa. Ambas informações são armazenadas em tabelas simples, cada uma destas tabelas são consultadas por instruções básicas de SQL, mas que aumentam consideravelmente o número de transações com o banco de dados.

7 – Após todas as informações necessárias para a gravação do documento estarem armazenadas, o documento é gravado. Esta rotina executa inserções em 4 tabelas distintas, a tabela de notas, a tabela de itens de nota, a tabela de complemento de nota e a tabela de formas de pagamento. Juntamente com as inserções, é realizada também uma atualização na tabela de estoque do produto. As tabelas envolvidas nesta rotina são as maiores tabelas do banco de dados, sendo a tabela de notas uma das mais utilizadas pelo sistema.

Os dados destas tabelas são utilizados pela maior parte dos relatórios e indicadores do sistema, onde para ganhos de performance foram criados uma série de índices e triggers que facilitam as consultas e análises. A consequência negativa de ter uma grande quantidade de índices nas tabelas principais do sistema vem na inserção dos dados, pois cada índice novo prejudica o tempo e o desempenho da gravação, fazendo deste um bom medidor para os testes de performance que são realizados neste trabalho.

8 – Após finalizada a gravação, o programa identifica o tempo total de execução da rotina e passa para o próximo documento.

3.3 Processo de Migração

O processo de migração consiste em modificar os módulos escolhidos para que os mesmos acessem o banco de dados convertido. Para fazer isto o modelo de acesso aos dados do sistema foi modificado. Após esta etapa, todas as instruções SQL realizadas pelos módulos, são revisadas e alteradas para que funcionem corretamente com o SGBD *PostgreSQL*. Com estas etapas iniciais resolvidas, é possível comparar os resultados das instruções SQL e os resultados nos módulos, a fim de verificar se os módulos convertidos continuam executando corretamente, de acordo com os próprios módulos antes da conversão.

Para modificar o modelo de acesso aos dados foi necessário alterar o *driver* da arquitetura de conexão *DBExpress* utilizada atualmente. É possível também alterar o modo de acesso, utilizando ODBC ou outra tecno*log*ia, porém como o intuito do trabalho é verificar o desempenho do sistema com a troca de SGBD, o escolhido foi manter a tecno*log*ia de acesso para não gerar outros gargalos no processo comparativo. Caso seja verificado que o *driver*

utilizado para o acesso ao *PostgreSQL* esteja causando lentidão ou problemas, será estudada a troca desta tecno*log*ia.

Para manter ainda mais a integridade das informações dos testes foi possível utilizar um *driver* da mesma empresa que desenvolve o *driver* atual para *Firebird*, o Devart. Como o sistema possui um único ponto onde as configurações de conexão são realizadas, esta mudança foi relativamente simples, o *driver* foi modificado no código fonte e uma nova *dll* de acesso foi acrescentada ao projeto e à pasta de instalação do programa. Desta forma o sistema ficou apto a conectar no novo banco de dados.

A simplicidade da troca de tecno*log*ia de acesso não pode ser aplicada ao problema dos SQLs que não funcionaram no novo banco de dados. Tentando apenas realizar a autenticação no sistema, surgiram problemas relacionados às instruções que o SGBD *PostgreSQL* não conseguiu interpretar. Cada uma destas instruções teve que ser interpretada e reprogramada.

Um destes problemas foi quanto à questão dos SQLs com aspas duplas. No sistema de automação comercial, no início de sua programação, os SQLs tinham suas *strings* escritas com aspas duplas, ao invés de aspas simples. Um exemplo de um SQL nestes moldes pode ser visualizado no quadro 3.4. Esta situação é aceita pelo *Firebird*, porém o *PostgreSQL* apresenta erros ao rodar instruções com esta característica. Para contornar esta situação foi necessário substituir manualmente todas as instruções que utilizavam aspas duplas por aspas simples.

```
SELECT * FROM TB_ITE WHERE ITE_CDITEA015 = "123456789012345"
```

Quadro 3.4 – Exemplo SQL com aspas duplas

Outros problemas encontrados foram relacionados às *stored procedures* e *triggers*, que não foram convertidas pelos softwares de conversão. Novas funções e *triggers* tiveram que ser implementadas manualmente no banco de dados *PostgreSQL*, necessitando de alguns estudos e analises para colocar tudo em funcionamento novamente.

Para melhor exemplificar as dificuldades encontradas para converter cada um dos módulos, as próximas seções deste capítulo abordaram separadamente cada um deles, apontando os seus problemas e soluções encontradas.

3.3.1 Relatório de Desempenho

O módulo de relatório de desempenho realiza em média 70 instruções SQL para gerar uma listagem. Este número varia de acordo com o layout e filtros escolhidos, mas nunca alterando drasticamente este número. Grande parte destas 70 instruções são consultas de parâmetros e informações necessárias para alimentar os cálculos do relatório e os cabeçalhos dos relatórios. O SQL apresentado no quadro 3.1 é um exemplo da complexidade da consulta realizada por esta rotina. Para converter este SQL foi necessário realizar algumas mudanças em relação ao original rodado no *Firebird*.

Uma destas mudanças foi a necessidade de acrescentar uma descrição para a cosulta que é realizada após o indicador "*from*". No *Firebird* esta rotina executa sem problemas, mas no *PostgreSQL* é apresentado um erro informando que a *subquery* necessita de um *alias* válido.

Outro ponto modificado foram as aspas duplas que eram utilizadas nas consultas. Por ser um relatório antigo dentro do sistema de automação comercial, esta técnica estava sendo utilizada em vários SQLs e foi necessário modificar todas as instruções que tinham este problema.

Cada uma das 70 instruções foi rodada separadamente e manualmente em cada um dos bancos de dados, com a finalidade de comparar os seus resultados. Como são todas instruções de consulta, elas devem retornar os mesmos resultados quando executadas com os mesmos filtros e parâmetros de entrada.

Realizadas as devidas mudanças no sistema de automação comercial, o módulo de Relatório de Desempenho ficou apto a funcionar, utilizando a base de dados convertida em *PostgreSQL*. Após comparativos de valores foi possível observar que os resultados das diversas variações do relatório estavam de acordo com os valores apresentados pelo sistema que estava conectando ao banco de dados *Firebird*, possibilitando o inicio dos testes de desempenho com base no módulo convertido.

3.3.2 Consulta de Clientes

Assim como citado na introdução deste capítulo, o módulo de consulta de clientes possui uma característica onde qualquer campo do seu cadastro pode ser usado como filtro nesta consulta, ou seja, a instrução SQL vai ser modificada de acordo com a preferência do

usuário de forma dinâmica. Seguindo este princípio foi necessário revisar todos os tipos de filtro possíveis para certificar-se que, ao converter o sistema para *PostgreSQL*, todos os campos possam ser utilizados no filtro da consulta.

O registro de cliente no banco de dados (tabelas de pessoa física e jurídica) possui 110 campos. Portanto foi necessário reproduzir todos os tipos de filtro nos SQLs contendo estes campos. Após executada esta análise foi constatado que todos os campos estavam aptos a serem filtrados nas rotinas de consulta no banco de dados *PostgreSQL* com exceção da variação da consulta de campos *varchar* pelo método "começando com".

Um problema atualmente no *Firebird* é o uso da função *LIKE* nas consultas de campos do tipo *Varchar*. Uma consulta comum no sistema de automação é buscar registros cujos campos "começam com". Para fazer esta instrução é necessário usar a função *LIKE* e filtrar o campo, concatenando o caractere "%" ao final. Porém esta consulta no *Firebird* não utiliza o índice do campo, caso ele possua. Para contornar esta situação no *Firebird*, existe a função interna *"Starting With*". Todos os campos, que são filtrados na consulta de clientes do sistema pela opção "começando com", tem esta função adicionada ao seu SQL pelo sistema de automação.

Como citado no capitulo 1.3.2 deste trabalho, o *PostgreSQL* não implementa a função "*Starting With*", porém utilizando o comando *LIKE*, a sua execução trabalha corretamente com os índices dos campos filtrados. Com base nestes fatos foi necessário revisar os SQLs de cada campo do tipo *Varchar,* das tabelas de cliente, para que onde fosse chamada a função "*Starting With*" fosse usado o comando *LIKE*.

Outra situação encontrada onde foi necessário intervir para manter o funcionamento do módulo após a conversão do banco de dados, foi o caso da paginação. Assim como explicado anteriormente neste trabalho, as consultas do sistema possuem uma paginação automática. Toda a rotina que gera as instruções de paginação precisou ser revisada. No *Firebird* as funções usadas para obter os resultados paginados eram o *FIRST* e o *SKIP*, já no *PostgreSQL* as funções internas que são utilizadas nestas situações são o *LIMIT* e o *OFFSET*. Nos quadros 3.5 e 3.6 seguem exemplos de um SQL paginado no *Firebird* e seguido de sua conversão para o *PostgreSQL* respectivamente.

```
SELECT
FIRST 50 SKIP 0
TB_CPJ.*,
TB_CPF.*,
FROM
TB_CPJ
LEFT JOIN TB_CPF ON (TB_CPF.CPJ_CDCLII006 = TB_CPJ.CPJ_CDCLII006)
ORDER BY CPJ_RZSOCA060
```

Quadro 3.5 - SQL Paginado *Firebird*

```
SELECT
TB_CPJ.*,
TB_CPF.*,
FROM
TB_CPJ
LEFT JOIN TB_CPF ON (TB_CPF.CPJ_CDCLII006 = TB_CPJ.CPJ_CDCLII006)
ORDER BY CPJ_RZSOCA060
LIMIT 50 OFFSET 0
```

Quadro 3.6 - SQL paginado *PostgreSQL*

Após realizadas todas as modificações necessárias no sistema de automação comercial, o módulo de consulta de clientes teve todas as suas características e funcionalidades mantidas com o novo acesso ao SGBD *PostgreSQL*. Mesmo com as mudanças citadas acima o módulo permaneceu funcional após a conversão. Para comprovar estas afirmações foram realizadas comparações nos dados obtidos tanto na versão que estava executando em *Firebird* quanto na nova versão convertida para *PostgreSQL* e os resultados permaneceram os mesmos. Desta forma habilitando a continuidade do trabalho e fazendo com que os testes de desempenho sejam possíveis.

3.3.3 *StressTest*

O *StressTest* por ser um programa que realiza o complexo processo de venda possui 124 instruções SQL a serem revisadas, quase o dobro de instruções em relação ao módulo de Relatório de Desempenho. Estes SQLs estão distribuídos em consultas, rotinas de atualização de registros e inclusões nas maiores tabelas do banco de dados.

Este programa, por ser uma aplicação antiga das rotinas de teste do sistema, utiliza um padrão nas instruções SQL que atualmente não é mais usado. Este padrão é o de utilizar aspas duplas na identificação dos campos “*varchar*” e “*char*”. O *Firebird* aceita instruções que possuem aspas duplas nestes campos, ele é capaz de identificar este parâmetro e tratá-lo

corretamente, porém o *PostgreSQL* não permite este tipo de instrução. Para resolver este problema todos os SQL foram modificados para utilizar aspas simples.

Outro problema encontrado nos SQLs revisados do programa foi com relação aos *alias* de tabelas. Um *alias* é uma funcionalidade dos SGBDs relacionais que visam facilitar o desenvolvimento de uma instrução SQL. Um *alias* pode renomear uma tabela, uma coluna ou até mesmo um *sub-select*, facilitando o entendimento e a organização da instrução. O problema que foi encontrado no processo de conversão foi em relação aos *alias* utilizados nos updates. No *Firebird* é comum utilizar *alias* até mesmo em updates, porém esta opção não funcionou corretamente no *PostgreSQL*. Para contornar este problema, todos os *updates* que utilizavam *alias* para tabelas tiveram que ser retirados.

Assim como exemplificado no capitulo 3.2.3, a rotina de venda necessita consultar os preços de venda e saldos em estoque, para isso foi necessário converter uma série de *Stored Procedures* que são usadas atualmente no sistema. A linguagem utilizada para concluir estas conversões foi a padrão PL/PgSQL, a qual atendeu todos os requisitos necessários para retornar os mesmos resultados obtidos pelas *Stored Procedures*, escritas em PSQL do *Firebird*.

Reescrevendo as funções para *PostgreSQL*, foi possível notar uma diferença grande no modo de funcionamento entre ela e uma *Stored Procedure* do *Firebird*. No *Firebird*, uma *Stored Procedure* pode retornar uma série de valores, sendo que eles podem variar os seus tipos de campos. Já nas funções do *PostgreSQL* não é possível retornar mais de um valor. Para resolver este problema é necessário criar um novo tipo de campo interno no banco de dados do *PostgreSQL* e fazer com que a função retorne este tipo de campo.

Estes tipos podem ser criados com qualquer número de campos, e estes campos podem ser de qualquer tipo suportado pelo *PostgreSQL*. No quadro 3.6 pode ser observado um tipo criado para ser usado em uma das funções convertidas.

```
CREATE TYPE EST_CUSTO AS(
  QUANTIDADE_EST FLOAT,
  CUSTO_MEDIO FLOAT
);
```

Quadro 3.7 - Exemplo Tipo criado para uso em Funções do *PostgreSQL*

Outra diferença entre o PL/PgSQL e o PSQL é no momento de dar nomes aos *alias* de tabelas e campos. O PL/PGSQL possui algumas palavras reservadas que não podem ser

utilizadas como *alias*. Dois exemplos de *alias* que estavam sendo utilizados nas *Stored Procedures* do *Firebird* são as palavras reservadas *DATA* e *TEMP*. Todas as funções que usavam estas referências tiveram que ser modificadas.

Concluindo todas as modificações o programa *StressTest* ficou apto a conectar e rodar suas rotinas no banco de dados convertido *PostgreSQL*. Mesmo após as modificações, os dados gravados e consultados pelo programa foram os mesmos em comparação com o *StressTest* rodado no *Firebird*. Assim como os outros dois módulos convertidos, o *StressTest* também ficou apto a execução de testes de desempenho.

4 TESTES

Um dos objetivos principais deste trabalho é aplicar o que foi estudado e verificar na prática os resultados das conversões, tanto do banco de dados como dos módulos do sistema de automação. Neste capitulo são apresentados todos os passos para atingir os objetivos do trabalho de pesquisa, exemplificando o ambiente, os testes e os resultados obtidos.

4.1 Ambiente de Testes

A escolha de um ambiente de testes adequado é de grande importância para manter a integridade dos resultados de um trabalho de pesquisa. Ela faz com que a seleção dos parâmetros e configurações seja essencial para garantir que a informação coletada seja confiável e possa ser utilizada em futuras consultas. Nas próximas seções são configurados os ambientes que foram utilizados para os testes, demonstrando e explicando o motivo da escolha de cada uma das configurações e opções a fim de garantir todos os resultados obtidos.

4.1.1 Arquitetura do Ambiente de testes

A arquitetura do ambiente de testes foi montada grande parte em máquinas virtuais. Mesmo sabendo que uma máquina virtual por estar rodando dentro de outro sistema operacional pode apresentar perdas de desempenho, esta escolha se deu pelo fato de que este trabalho não precisa medir o máximo de desempenho que um SGBD pode oferecer, e sim deve realizar um comparativo entre o desempenho dos mesmos. Sendo assim, com máquinas virtuais, é possível garantir que serão emuladas as mesmas configurações para os dois ambientes.

Cada SGBD foi instalado em uma máquina virtual separada, mas com as mesmas configurações e rodando na mesma máquina física. Para isso foi criada uma máquina virtual *template* que foi clonada para cada um dos casos. Em caso de perda da máquina virtual de teste, por qualquer motivo, basta iniciar um novo processo de cópia que já seria possível dar continuidade aos testes.

A configuração das máquinas virtuais atende os seguintes parâmetros listados na tabela 4.1.

Tabela 4.1 - Configuração Máquinas Virtuais

Sistema Operacional	Linux CentOS 6.0
Memória	2GB
Processador	Intel I5
Disco	100GB – SSd - NTFS

O sistema operacional das máquinas virtuais escolhido foi o Linux. Para esta escolha foi levado em consideração qual o SO que cada um dos SGBD é mais performático. No trabalho de PLETSCH (2005), é possível concluir que tanto o *PostgreSQL* como o *Firebird* tem melhores resultados quando rodados em sistemas operacionais Linux. Outro motivador para esta escolha é de que 95% dos clientes da empresa XYZ já utilizam SO Linux em seus servidores de Bancos de Dados atualmente.

Outro fator que atribui uma nova característica ao ambiente de testes é o sistema operacional no qual rodará o sistema de automação comercial. Este não possui uma versão liberada para Linux, rodando apenas em SO Windows. Para atender este requisito, enquanto os SGBDs estão rodando em máquinas virtuais Linux, o sistema de automação rodará no SO que estiver emulando as máquinas virtuais de teste.

O sistema de automação será executado em um sistema operacional *Windows 7 Professional*, que possui as mesmas configurações das máquinas virtuais. O acesso aos SGBDs se dará por conexão TCP/IP nas portas 3050 e 5432, portas de acesso *default* dos SGBDs *Firebird* e *PostgreSQL* respectivamente, via *driver DBExpress*.

A figura 4.1 resume a configuração da estrutura do ambiente de testes.

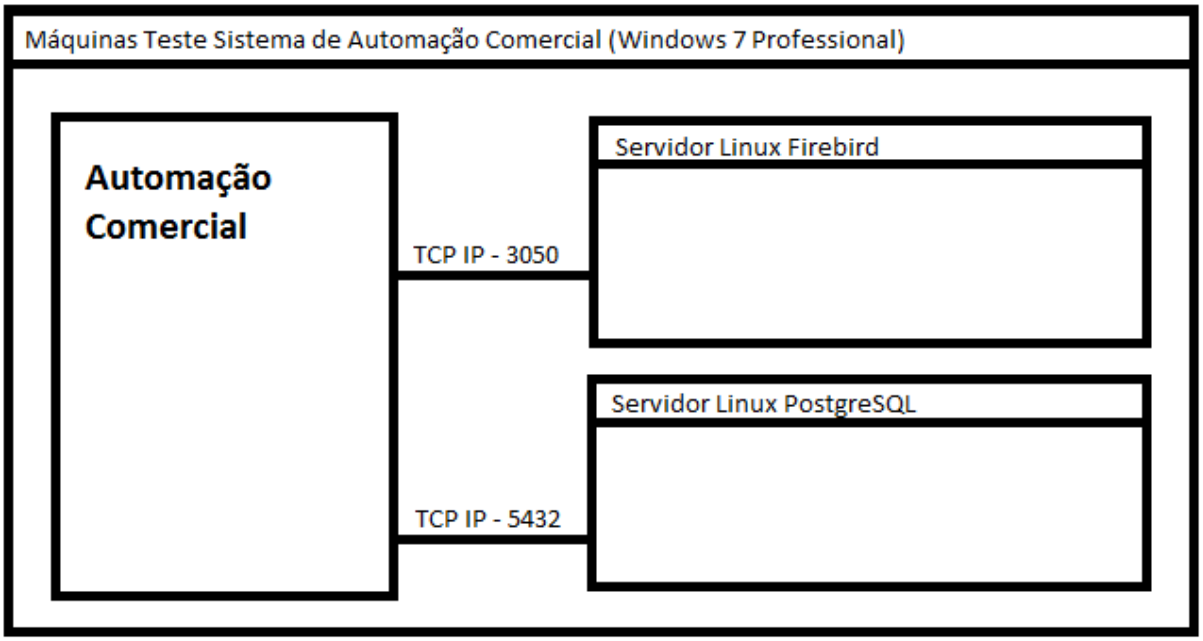

Figura 4.1 - Estrutura final de Testes

4.1.2 Instalação *Firebird*

A instalação do *Firebird* em Linux é muito simples. Bastando rodar o arquivo *install.sh* encontrado no diretório da pasta de instalação, o SGBD configura tudo o que é necessário para iniciar o funcionamento do serviço. Porém nesta instalação básica o tipo de servidor instalado é a versão *Classic*. Como verificado nos capítulos iniciais deste trabalho, a versão mais completa do *Firebird* disponível é a *SuperClassic*. Sendo assim foi necessário configurar esta instalação para que fosse trocado de *Classic* para *SuperClassic*. Para fazer isso foi necessário rodar um script à parte que se encontra na diretório de instalação do *Firebird* chamado *changeMultiConnectMode.sh*.

Com o SGBD instalado uma cópia do backup da base de dados foi copiada para um diretório escolhido para os testes. Este backup foi restaurado para garantir que os dados dos testes sejam os mesmos que serão acessados na base convertida do *PostgreSQL*.

4.1.3 Instalação *PostgreSQL*

A instalação do *PostgreSQL* na máquina virtual de testes é um pouco mais extensa em comparação com a instalação do *Firebird*. Para realizã-la foi necessário fazer o download do arquivo *.rpm* diretamente do site oficial do *PostgreSQL*. Após instalar o *RPM,* iniciar o serviço do SGBD no Linux foi necessário configurar o arquivo *PostgreSQL.conf* para dar acesso à máquina que rodou os testes. Para fazer isso foi necessário modificar o parâmetro

listen_addresses e adicionar o *IP* da máquina de testes. Outro arquivo que precisou ser modificado para dar permissão de acesso à máquina de testes ao *PostgreSQL* foi o *pg_hba.conf*, onde foi necessário adicionar as configurações da rede que seriam permitidas para acessar os bancos de dados do SGBD.

Após a instalação do SGBD, uma nova conversão, com os mesmos parâmetros definidos no capitulo 2 deste trabalho, foi executada gerando um novo banco de dados, garantindo assim a igualdade dos dados entre a base de dados do *Firebird* com a do *PostgreSQL*.

4.2 Execução dos testes

Nesta parte do trabalho os testes são explicados e exemplificados. Cada teste foi executado e seus resultados armazenados. Ao final deste processo cada teste é avaliado individualmente, analisando seus tempos e comparando com os testes equivalentes executados no outro SGBD.

As execuções dos testes foram montadas de forma que facilitassem o entendimento das informações e resultados obtidos. Para isso cada teste foi executado três vezes, recuperando-se sua média de tempo de execução, garantindo assim que os valores estão os mais próximos da realidade. Outro ponto importante a ressaltar da execução destes testes é a sua execução sem o uso de *cache* de memória, e a execução utilizando este *cache.* Foi optado por este tipo de teste na tentativa de verificar casos em que o SGBD possui um melhor desempenho em comparação ao outro SGBD, quando utilizado estes dados em memória.

4.2.1 Organização dos testes

Foram criados 10 testes diferentes para os módulos convertidos neste trabalho. Cada um dos testes foi executado duas vezes seguidas, a primeira vez após reiniciar a máquina virtual e por consequência limpando todo o *cache* da memória e a segunda execução *log*o após esta primeira utilizando os dados guardados na memória e no *cache* do SGBD. Cada um destes testes foi executado três vezes, totalizando 60 testes por SGBD e 120 no total geral, executados ao final deste trabalho.

A figura 4.2 apresenta um diagrama exemplificando a execução dos testes.

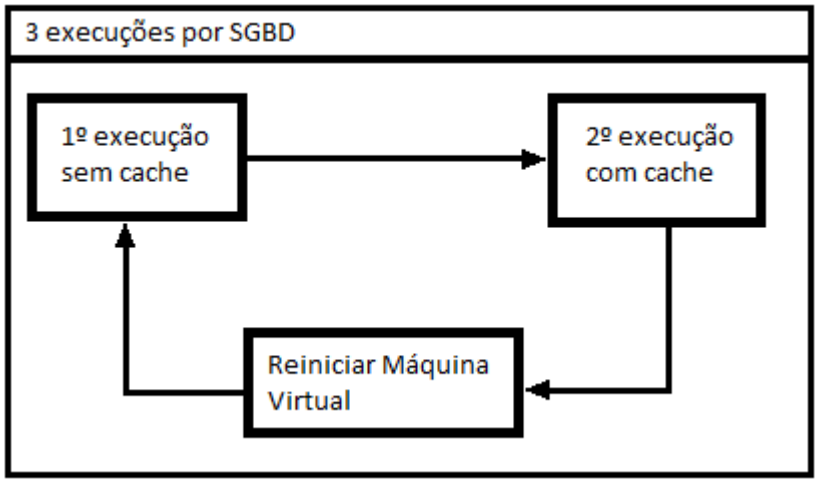

Figura 4.2 - Diagrama de execução dos testes.

A divisão dos testes foi dada da seguinte forma: 3 testes no programa *StressTest*, 5 testes no módulo Consulta de Clientes e 2 testes no módulo de Relatório de Desempenho. Abaixo segue a listagem com cada um dos testes e suas respectivas funções.

1 - *StressTest* Teste 1: Neste teste o programa será configurado para registrar 1000 notas fiscais no banco de dados contendo 2 produtos por documento. Este teste será importante para verificar como os SGBDs trabalham com uma única conexão inserindo, atualizando e consultando por um longo período de tempo.

2 - *StressTest* Teste 2: Neste segundo teste o programa *StressTest* deve registrar 100 notas com 10 produtos em cada uma. Também será um teste com apenas uma conexão ativa com o SGBD. O motivo deste teste é de verificar como o banco de dados se comporta com uma quantidade alta de atualizações e consultas nas maiores tabelas do sistema que são a tabela de estoque e de produtos vinculados a nota.

3 - *StressTest* Teste 3: O terceiro teste consiste em rodar 10 programas *StressTest* ao mesmo tempo. Cada um destes programas vai cadastrar 200 documentos com 2 produtos vinculados. Este teste mostrará como o SGBD se comporta com o acesso concorrente às mesmas tabelas, mas com várias transações simultâneas.

4 - Consulta de Clientes Teste 1: Este teste consiste em consultar 1 cliente pelo seu campo de CPF. O CPF é um campo do banco de dados que o seu valor não pode ser repetido, do tipo VARCHAR com tamanho fixo de 14 caracteres e que possui um índice próprio. Este teste tem a importância de verificar como uma consulta simples, por um campo de texto que possua índice, se comporta em ambos os SGBDs, utilizando ou não o *cache* de memória.

5 – Consulta de Clientes Teste 2: Nesta situação o módulo será utilizado para consultar clientes que o seu nome comece com a letra ‘A’. Este teste será relevante para verificar o uso do comando “*Starting with”* do *Firebird* em comparação com o comando *Like* utilizado pelo *PostgreSQL* em campos que possuem índices a serem utilizados.

6 – Consulta de Clientes Teste 3: Neste teste o módulo de consulta de cliente deve consultar todos os clientes que possuam a letra ‘Y’ em seu nome. Nesta situação o comando *Like* será utilizado por ambos os SGBDs, podendo assim comparar o uso do mesmo em uma instrução de consulta por um campo que possua índice.

7 – Consulta de Clientes Teste 4: Este é o teste mais simples de toda a rotina. O cliente será consultado pelo seu código principal, que é na verdade a chave primária da tabela de clientes. Este teste apresentará qual o SGBD possui a consulta mais otimizada em campos chave de tabelas.

8 – Consulta de Clientes Teste 5: Neste teste será consultado todos os clientes em que o seu nome comece com a letra ‘A’, mas nesta situação a paginação do sistema de automação comercial será utilizada. Assim que for consultado o sistema paginará automaticamente as primeiras 50 paginas (cada uma contendo 50 registros), fazendo com que o *Firebird* utilize os seus comandos *First* e *Skip* e o *PostgreSQL* os comandos *Limit* e *Offset.*

9 – Relatório de Desempenho Teste 1: O primeiro teste no Relatório de Desempenho tem como finalidade buscar as informações de uma loja, porém de um período grande (6 meses de movimentações), fazendo com que a consulta trabalhe para buscar valores em campos que não possuem índice e trabalhando bastante com a ordenação dos resultados.

10 – Relatório de Desempenho Teste 2: O segundo teste no módulo de Relatório de Desempenho procura executar uma consulta diferenciada do teste anterior. Neste caso o período consultado é de apenas um mês, porém a consulta das empresas é consolidada, pesquisando os dados de toda a rede (50 lojas). Como a exibição do relatório é apresentada por agrupamento de valores por empresa, o teste acaba por testar como os agrupamentos dos SGBD trabalham em situações com grande quantidade de dados para organizar.

4.2.2 Testes

Para facilitar a visualização dos resultados dos testes foi implementada uma nova funcionalidade no sistema de automação comercial para que no momento em que uma instrução SQL fosse rodada, esta instrução fosse gravada em um *log* de texto. Neste *log* além

de ter a instrução gravada, foram salvos os tempos de início e fim das execuções de cada uma das instruções. Com esta nova camada nas execuções das instruções SQL foi possível medir com precisão todos os testes executados. O quadro 4.1 demonstra o modo como este *log* ficou organizado.

```
***18/10/2013 10:15:50:113

select e.emp_cdempa003, e.emp_cgcmfa014, e.emp_tppesa001,
  e.emp_nmempa035, e.emp_nmfana020, e.EMP_MATRIA003,
  e.emp situaa001 Situacao, e.sub matriz
from tb_emp e
where e.emp_cdempa003 not in ("000", "   ", "")
order by emp cdempa003

***18/10/2013 10:15:50:124

***18/10/2013 10:15:50:129

select e.emp_cdempa003
from tb_emp e
where e.emp_cdempa003 <> "000" and e.emp_cdempa003 <> "" and
e.emp_cdempa003 <> "   "
order by emp_cdempa003

***18/10/2013 10:15:50:133

***18/10/2013 10:15:50:136

SELECT * FROM TB_PEE WHERE EMP_CDEMPA003 = '001' AND PEE_DCPEEA060 =
'MetrosQuadrados'

***18/10/2013 10:15:50:139
```

Quadro 4.1 – Exemplo *Log* execução SQL

Com base no que foi definido quanto aos testes, após suas execuções os resultados obtidos foram coletados e organizados. As tabelas 4.2, 4.3, 4.4 e 4.5 apresentam os resultados das três execuções dos testes em ambos os SGBDs.

Tabela 4.2 - 1ª execução dos testes

	Firebird		PostgreSQL	
	1ª Execução (Sem cache)	2ª Execução (Com cache)	1ª Execução (Sem cache)	2ª Execução (Com cache)
StressTest - Teste 1	22 minutos 22 segundos	...	16 minutos 50 segundos	...
StressTest - Teste 2	6 minutos 14 segundos	...	5 minutos 11 segundos	...
StressTest - Teste 3	21 minutos 50 segundos	...	6 minutos 32 segundos	...
Consulta Cliente - Teste 1	194 ms	57 ms	90 ms	35 ms
Consulta Cliente - Teste 2	3 segundos 881 ms	837 ms	651 ms	161 ms
Consulta Cliente - Teste 3	11 segundos 604 ms	11 segundos 391 ms	644 ms	175 ms
Consulta Cliente - Teste 4	116ms	50ms	84 ms	27 ms
Consulta Cliente - Teste 5	15 segundos 263ms	12 segundos 168 ms	8 segundos 763 ms	6 segundos 967 ms
Relatório de Desempenho - Teste 1	6 segundos 467 ms	3 segundos 405 ms	9 segundos 526 ms	3 segundos 823 ms
Relatório de Desempenho - Teste 2	13 segundos 196 ms	12 segundos 185 ms	18 segundos 621 ms	13 segundos 449 ms

Tabela 4.3 - 2ª execução dos testes

	Firebird		PostgreSQL	
	1ª Execução (Sem cache)	2ª Execução (Com cache)	1ª Execução (Sem cache)	2ª Execução (Com cache)
StressTest - Teste 1	26 minutos 48 segundos	...	17 minutos 37 segundos	...
StressTest - Teste 2	7 minutos e 26 segundos	...	5 minutos 25 segundos	...
StressTest - Teste 3	21 minutos e 27 segundos	...	6 minutos 50 segundos	...
Consulta Cliente - Teste 1	192 ms	55 ms	92 ms	31 ms
Consulta Cliente - Teste 2	3 segundos 988 ms	800 ms	570 ms	165 ms
Consulta Cliente - Teste 3	16 segundos 461 ms	13 segundos 92 ms	651 ms	167 ms
Consulta Cliente - Teste 4	179 ms	50 ms	86 ms	33 ms
Consulta Cliente - Teste 5	13 segundos 537 ms	10 segundos 665 ms	9 segundos 308 ms	6 segundos 799 ms
Relatório de Desempenho - Teste 1	6 segundos 553 ms	3 segundos 327 ms	10 segundos 106 ms	3 segundos 845 ms
Relatório de Desempenho - Teste 2	12 segundos 911 ms	11 segundos 575 ms	18 segundos 710 ms	13 segundos 538 ms

Tabela 4.4 - 3ª execução dos testes

	Firebird		PostgreSQL	
	1ª Execução (Sem cache)	2ª Execução (Com cache)	1ª Execução (Sem cache)	2ª Execução (Com cache)
StressTest - Teste 1	29 minutos 14 segundos	...	17 minutos 42 segundos	...
StressTest - Teste 2	7 minutos e 38 segundos	...	5 minutos 40 segundos	...
StressTest - Teste 3	22 minutos e 2 segundos	...	7 minutos 9 segundos	...
Consulta Cliente - Teste 1	205 ms	57 ms	92 ms	37 ms
Consulta Cliente - Teste 2	3 segundos 739 ms	774 ms	634 ms	159 ms
Consulta Cliente - Teste 3	11 segundos 280 ms	10 segundos 59 ms	645 ms	169 ms
Consulta Cliente - Teste 4	185 ms	49 ms	86 ms	35 ms
Consulta Cliente - Teste 5	13 segundos 660 ms	10 segundos 719 ms	9 segundos 357 ms	6 segundos 928 ms
Relatório de Desempenho - Teste 1	6 segundos 693 ms	3 segundos 288 ms	10 segundos 75 ms	3 segundos 864 ms
Relatório de Desempenho - Teste 2	12 segundos 919 ms	11 segundos 598 ms	18 segundos 642 ms	13 segundos 549 ms

Tabela 4.5 - Média (em segundos) das 3 execuções dos testes

	Firebird		PostgreSQL	
	1ª Execução (Sem cache)	2ª Execução (Com cache)	1ª Execução (Sem cache)	2ª Execução (Com cache)
StressTest - Teste 1	1568,00	...	1043,00	...
StressTest - Teste 2	426,00	...	325,33	...
StressTest - Teste 3	1306,33	...	410,33	...
Consulta Cliente - Teste 1	0,20	0,06	0,09	0,03
Consulta Cliente - Teste 2	3,87	0,80	0,62	0,16
Consulta Cliente - Teste 3	13,12	11,51	0,65	0,17
Consulta Cliente - Teste 4	0,16	0,05	0,09	0,03
Consulta Cliente - Teste 5	14,15	11,18	9,14	6,90
Relatório de Desempenho - Teste 1	6,57	3,34	9,90	3,84
Relatório de Desempenho - Teste 2	13,01	11,79	18,66	13,51

As execuções dos testes com *cache* do programa *StressTest* não foram contabilizadas para o trabalho pois seu fluxo possui rotinas que buscam randomicamente códigos de produtos para consultar estoque e preços. Desta forma os resultados do teste poderiam ser alterados em casos onde uma das execuções consulte mais produtos que já estejam na memória do que em outra execução, alterando assim o tempo final do teste consideravelmente.

Juntamente com os testes foi verificado também o comportamento dos SGBDs no sistema operacional durante os processos. Foram encontradas diferenças no tratamento das conexões, transações, uso de memória e uso de processamento. Estas diferenças

comportamentais foram analisadas no intuito de verificar como os SGBDs utilizaram os recursos disponíveis a eles durante a execução dos testes.

As execuções do programa *StressTest* deixaram clara a grande diferença no uso dos recursos e tratamento das transações pelos SGBDs no sistema operacional. As figuras 4.3 e 4.4 apresentam a diferença do uso de recursos dos dois SGBDs no primeiro teste do *StressTest*, onde existe apenas uma conexão com o banco porém realizando uma grande carga de inserções, atualizações e consultas. Já as figuras 4.5 e 4.6 demonstram o comportamento dos SGBDs no sistema operacional quando existem 10 conexões com o banco de dados, realizando constantes cargas de inserção, atualização e consulta.

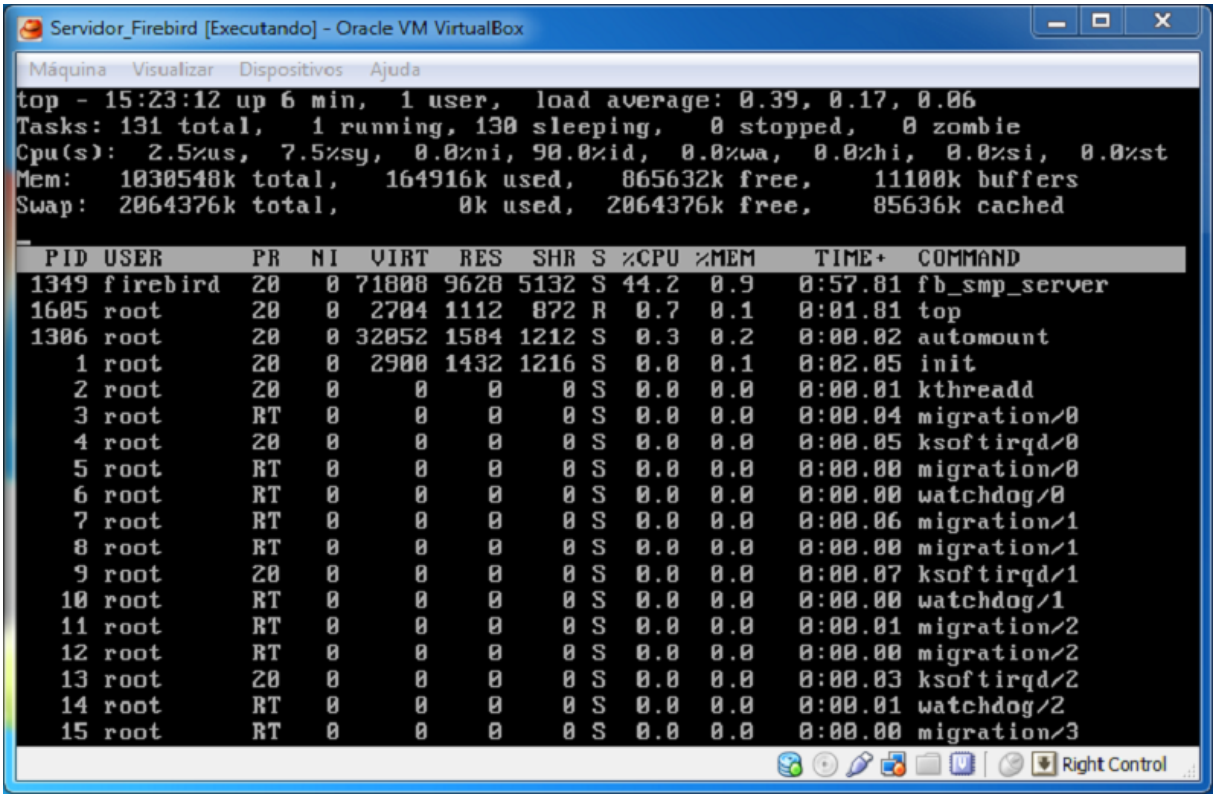

Figura 4.3 - Execução Teste 1 *StressTest* no servidor *Firebird*

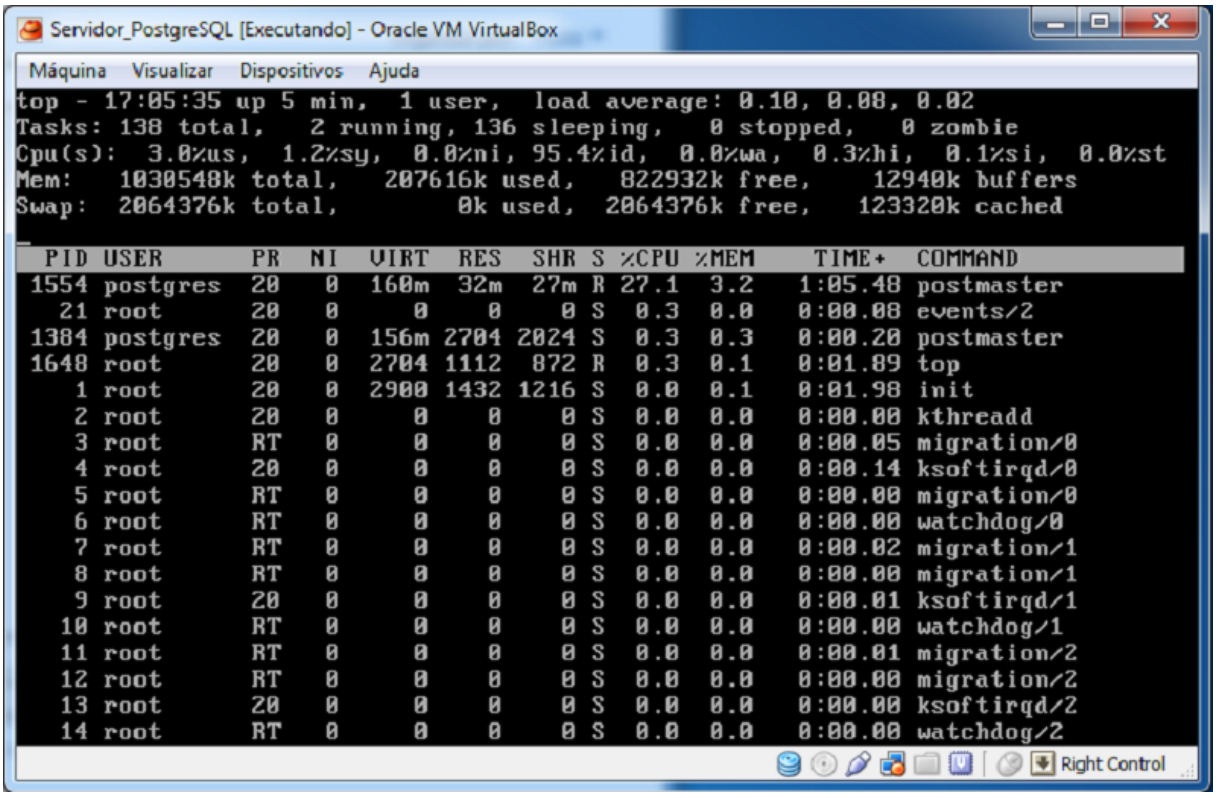

Figura 4.4 - Execução Teste 1 *StressTest* no servidor *PostgreSQL*

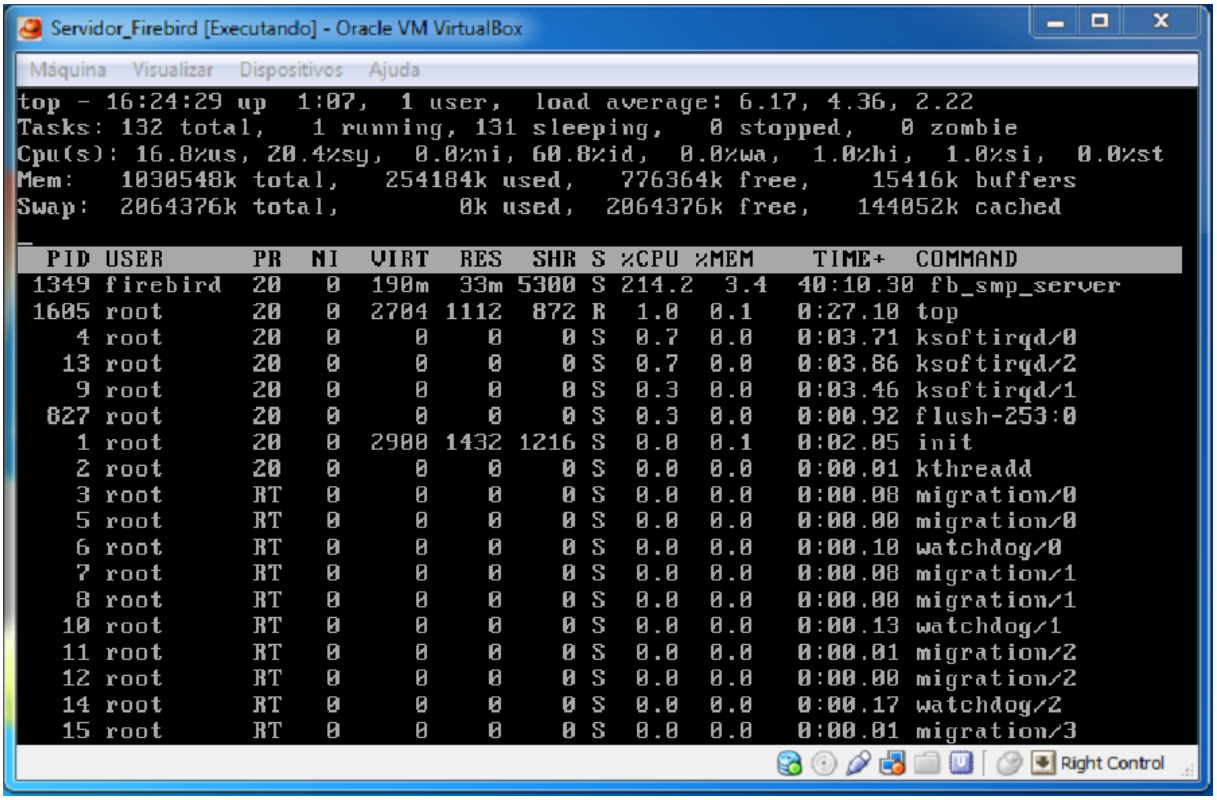

Figura 4.5 - Execução Teste 3 *StressTest* no servidor *Firebird*

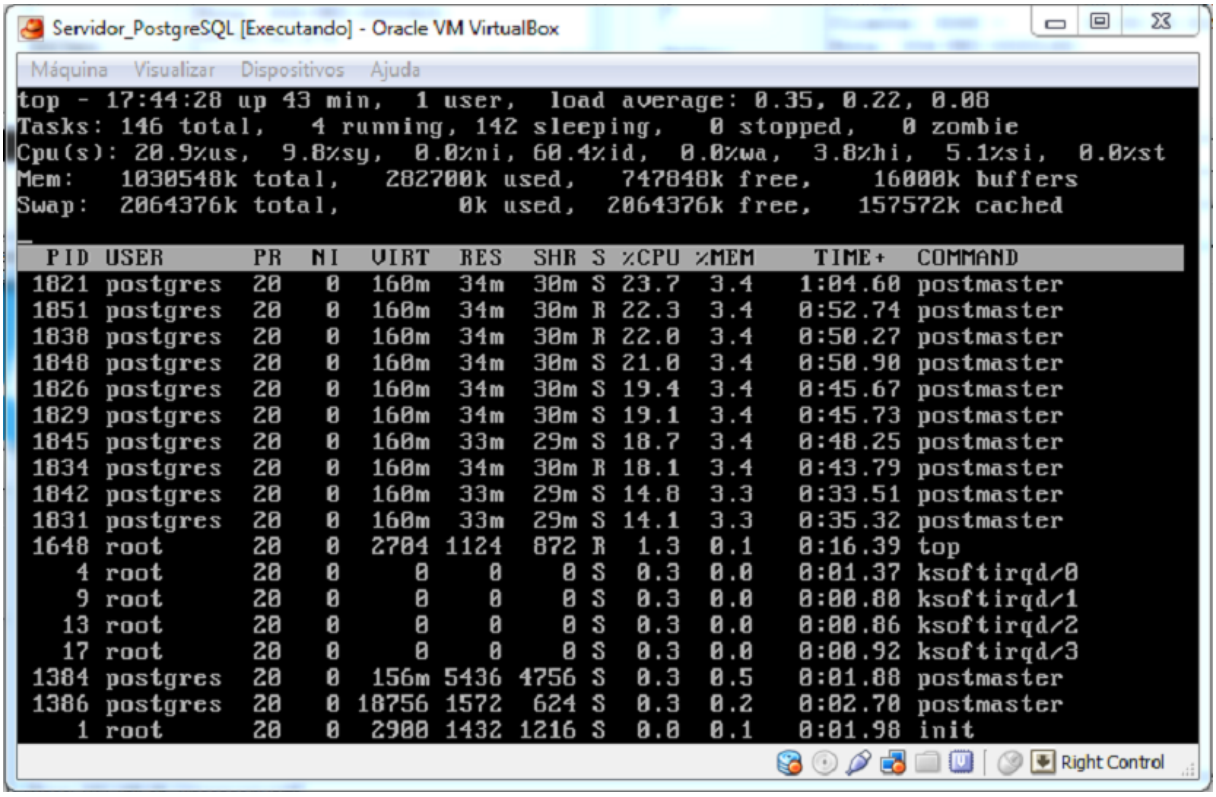

Figura 4.6 - Execução Teste 3 *StressTest* servidor *PostgreSQL*

4.2.3 Analises dos Resultados

Com base nos dados obtidos nos testes (tabelas 4.2, 4.3, 4.4 e 4.5) foi possível verificar uma grande melhora no desempenho geral do sistema de automação comercial utilizando o SGBD *PostgreSQL*. Sem maiores cálculos é possível observar uma grande vantagem do *PostgreSQL* em relação ao *Firebird* em quase todos os requisitos analisados neste trabalho. Mas mesmo que esta vantagem seja visível, faz-se necessária uma análise aprofundada nestes dados coletados e concluir uma análise comparativa entre o desempenho dos dois bancos de dados.

Os testes executados no módulo de Relatório de Desempenho foram os únicos onde o *PostgreSQL* apresentou resultados inferiores ao *Firebird*. Os resultados do Relatório de Desempenho apontam uma vantagem geral do *Firebird* de 30,96% em relação ao *PostgreSQL*. Em uma situação sem a utilização de *cache* no primeiro teste do Relatório de Desempenho a vantagem chegou a 50,7%.

O módulo de consulta de clientes foi testado em 5 situações diferentes simulando consultas usuais que os usuários do sistema de automação comercial executam diariamente e rotinas com consultas mais pesadas para verificar o desempenho entre os dois SGBDs.

Revisando os dados obtidos nos resultados destes testes é possível concluir que o SGBD *PostgreSQL* obteve um desempenho quase 1000% superior ao *Firebird*.

Este número extremamente superior se deve ao fato de no terceiro teste da consulta de clientes, o *PostgreSQL* tem um desempenho acima do esperado. Ao consultar clientes que contenham uma letra específica em seu nome o *Firebird* levou em média 13,115 segundos sem o uso de *cache* e 11,514 segundos utilizando o seu *cache* de memória. Já o *PostgreSQL* no mesmo teste levou em média 0,64 segundos sem *cache* e 0,17 segundos utilizando o seu *cache.* Este resultado demonstra claramente que um algoritmo de consulta de fragmentos de *strings*, em campos com índice, é muito superior no SGBD *PostgreSQL*.

O teste 3 da consulta de clientes por apresentar resultados muito superiores, destoando da média geral, poderia em uma análise consolidada mudar o conceito final do desempenho do módulo, desta forma também deve-se analisar os dados retirando os resultados deste teste especificamente. Recalculando novamente as médias sem a inclusão deste teste, na análise geral, o SGBD *PostgreSQL* ainda possui um desempenho 170,49% superior ao SGBD *Firebird*, demonstrando que em situações normais de consulta de dados, tanto por campos com índice, como consultas paginadas, ele foi o mais rápido.

A análise dos três testes realizados no programa *StressTest* foram importantes para avaliar o comportamento dos dois SGBDs em situações com uma grande carga de dados, tanto em inclusão como atualizações e consultas. Neste quesito o *PostgreSQL* foi superior em todos os testes. Tanto com uma conexão com o banco de dados, como com múltiplas conexões. Foi possível observar que o *PostgreSQL* foi 99,88% mais rápido que o *Firebird* em uma média geral de tempo. No teste onde são simuladas várias conexões com o banco de dados, o SGBD *PostgreSQL* teve um desempenho 218,36% superior ao *Firebird*, comprovando assim a sua superioridade em acessos simultâneos.

Relacionado aos testes do *StressTest* foi possível acompanhar o comportamento dos dois SGBD durante as execuções. Nos testes com um único acesso o *Firebird* utilizou um percentual maior de processamento e um menor uso de memória, já o *PostgreSQL* utilizou em média mais memória e menos processamento como pode ser visualizado nas figuras 4.3 e 4.4.

Nos testes com múltiplos acessos ao banco de dados nota-se a diferença no funcionamento entre os SGBDs. O *Firebird* instancia apenas 1 processo em memória para gerenciar todas as suas conexões e o *PostgreSQL* instancia 1 processo *Postmaster* para cada conexão. Estes detalhes podem ser observados nas figuras 4.5 e 4.6 que, além destas

particularidades, pode ser observado também que o uso de processamento do *Firebird* está em 214%, obviamente por estar utilizando várias *threads* para gerenciar as conexões.

Todos os dados e considerações citadas acima podem ser acompanhados na tabela 4.6. Os dados desta tabela foram gerados a partir das médias de todos os tempos coletados nas três execuções de testes.

Tabela 4.6 - Percentual de Desempenho do SGBD *PostgreSQL* em relação ao SGBD *Firebird*

	% de Desempenho PostgreSQL em relação Firebird	
	Sem cache	Com cache
StressTest - Teste 1	50,34	...
StressTest - Teste 2	30,94	...
StressTest - Teste 3	218,36	...
Consulta Cliente - Teste 1	115,69	64,08
Consulta Cliente - Teste 2	525,77	397,11
Consulta Cliente - Teste 3	1928,09	6659,69
Consulta Cliente - Teste 4	87,50	56,84
Consulta Cliente - Teste 5	54,81	62,13
Relatório de Desempenho - Teste 1	-50,70	-15,09
Relatório de Desempenho - Teste 2	-43,42	-14,64

Um detalhe que pode ser observado através dos resultados dos testes é a eficiência dos dois SGBDs no aproveitamento dos dados mantidos na memória. Em uma média geral o *Firebird* obteve o desempenho de 142,98% melhor ao utilizar o *cache*, já o *PostgreSQL* alcançou o desempenho de 160,83% melhor utilizando esta memória. Estes dados podem ser acompanhados na tabela 4.7.

Tabela 4.7 - Percentual de melhoria no desempenho na utilização do *Cache*

	Firebird	PostgreSQL
Consulta Cliente - Teste 1	249,70	166,02
Consulta Cliente - Teste 2	381,46	282,47
Consulta Cliente - Teste 3	13,90	279,65
Consulta Cliente - Teste 4	222,15	169,47
Consulta Cliente - Teste 5	26,55	32,54
Relatório de Desempenho - Teste 1	96,74	157,60
Relatório de Desempenho - Teste 2	10,37	38,08
Média total Aproveitamento	142,98	160,83

Para melhor visualização dos resultados dos testes, foram criados gráficos confrontando os dados obtidos. Estes gráficos foram organizados de forma que realcem os resultados de acordo com o grupo de testes que este trabalho executou. Os três grupos de testes são testes do *StressTest*, testes do módulo de Consulta de Documentos e os testes do módulo Relatório de Desempenho. Dentro destes grupos temos a análise do uso do *cache* em memória.

O gráfico apresentado na figura 4.7 mostra os resultados dos três testes do *StressTest* sem a utilização do *cache*. É possível observar a superioridade do *PostgreSQL* em todas as situações e principalmente no teste 3 onde foi executado o teste com múltiplas conexões com o banco de dados. Esta superioridade é comprovada no gráfico da figura 4.8 onde todos os testes são contabilizados e somados, resultando em um comparativo total.

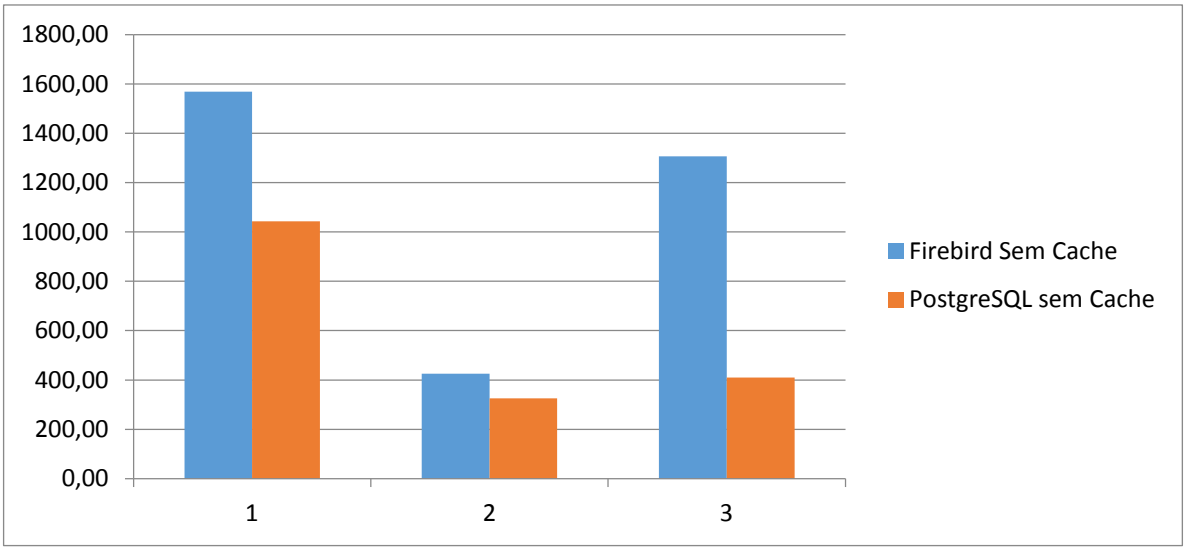

Figura 4.7 – Testes sem *cache* do *StressTest*

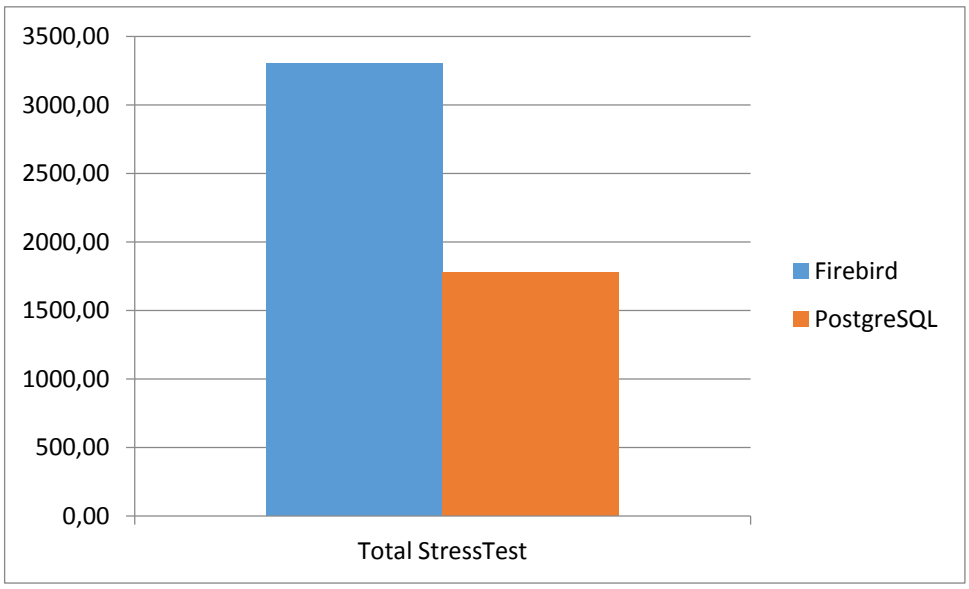

Figura 4.8 – Testes sem *cache* do StresTest totalizados

O gráfico da figura 4.9 apresenta os resultados obtidos na execução sem *cache* dos 5 testes do módulo de Consulta de Clientes. Nele é possível observar a grande vantagem do *PostgreSQL* em rotinas de consulta que buscam fragmentos de uma *string*.

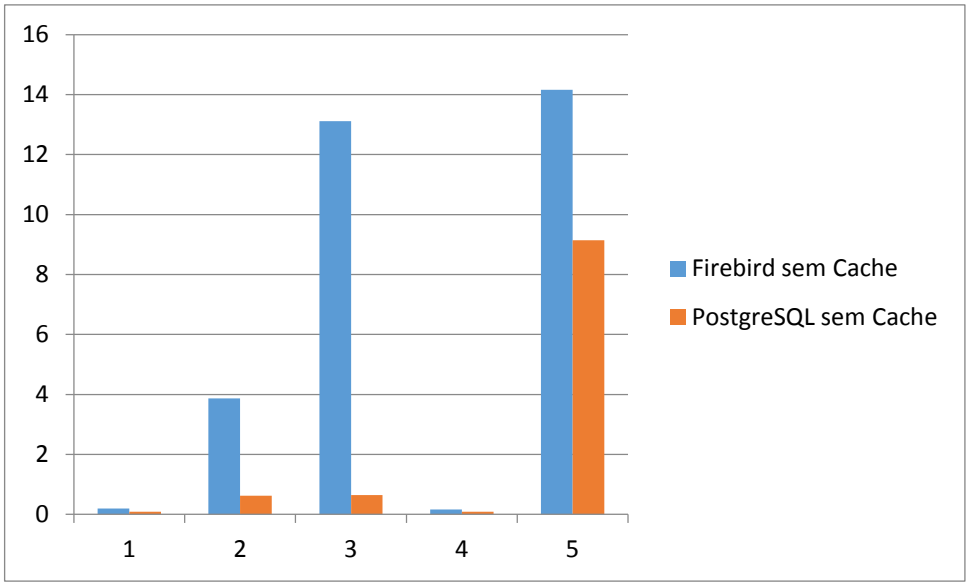

Figura 4.9 – Testes sem *cache* Consulta de Clientes

A figura 4.10 mostra os resultados dos mesmos 5 testes do módulo de Consulta de Clientes porém com a utilização dos dados em memória. Nele pode ser observado que o *Firebird* diminuiu seus tempos em relação ao *PostgreSQL* em 3 dos 5 testes. Isso demonstra um bom uso da memória pelo *Firebird*, porém ainda não sendo suficiente para alcançar o *PostgreSQL*.

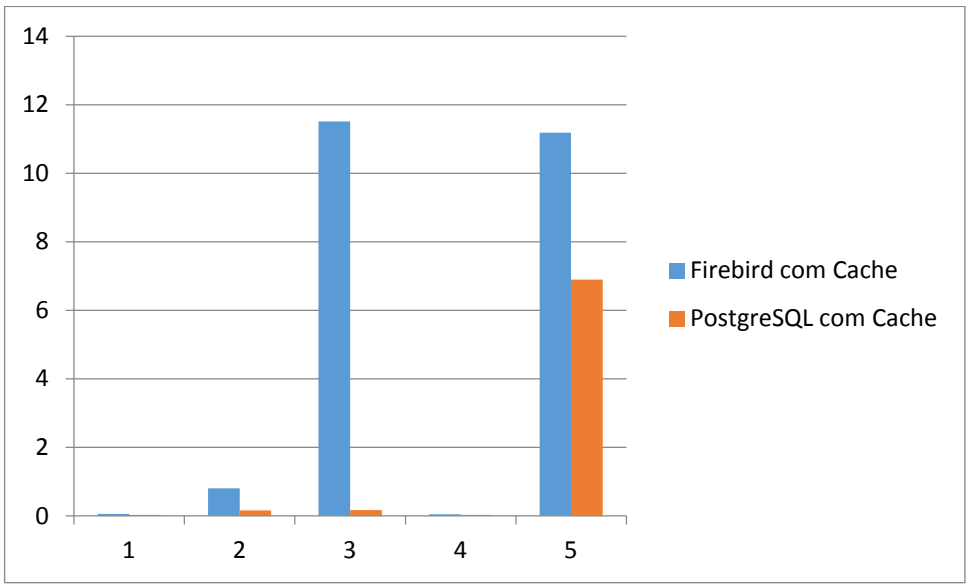

Figura 4.10 – Testes com *cache* Consulta de Clientes

Na figura 4.11 podem ser observados os resultados obtidos nos dois testes aplicados no módulo de Relatório de Desempenho sem a utilização de c*ache*. O resultado foi desfavorável para o *PostgreSQL* que apresentou tempos superiores ao *Firebird*.

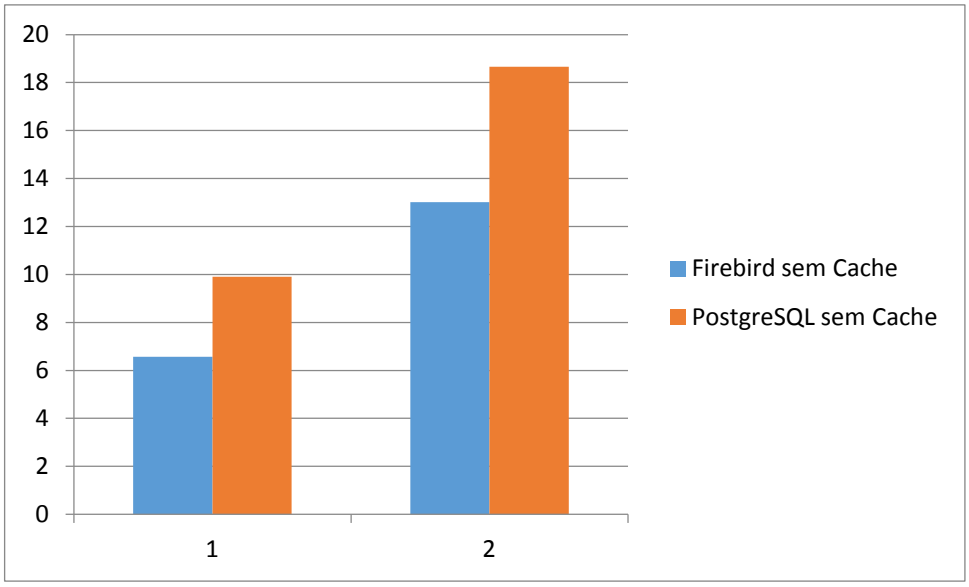

Figura 4.11 – Testes sem *cache* Relatório de Desempenho

Já na figura 4.12 o gráfico introduz os tempos obtidos ao utilizar os dados na memória nos dois SGBD. Nesta situação o percentual de uso do *cache* no *PostgreSQL* foi superior ao apresentado pelo *Firebird* deixando os resultados muito próximos, porém não foram suficientes para o *PostgreSQL* alcançar o desempenho do *Firebird*.

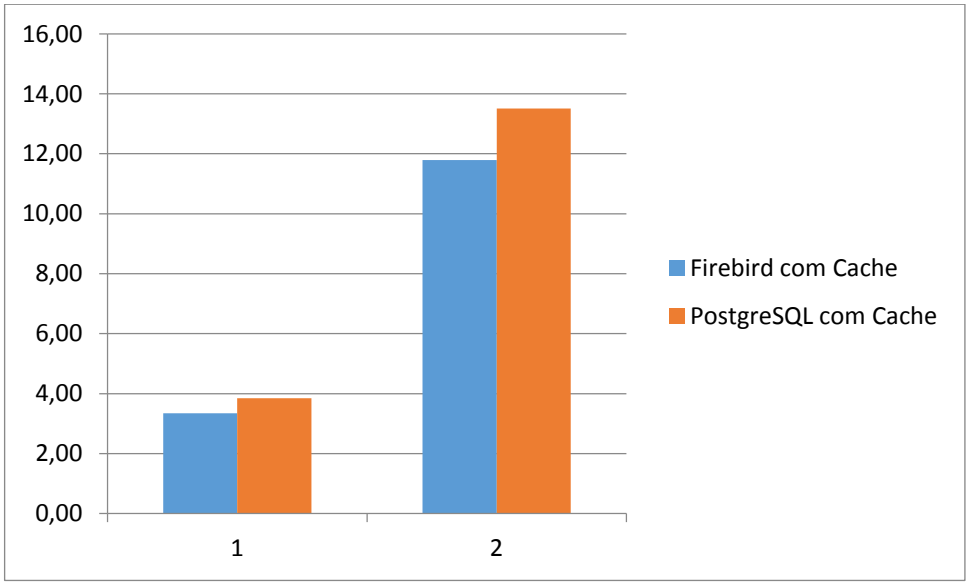

Figura 4.12 – Testes com *cache* Relatório de Desempenho

Com as informações e dados obtidos neste capítulo é possível montar um gráfico apresentando um total geral do desempenho dos 2 SGBDs. O gráfico da figura 4.13 apresenta os tempos totalizados dos testes da consulta de documentos e relatório de desempenho, separados pelo uso ou não de *cache*. Não foi acrescentado os tempos dos testes do *StressTest* a este gráfico pois eles possuem valores muito altos e poderiam tirar a noção geral que o

gráfico apresentaria e também por não possuir um resultado com o uso do *cache*. O total dos tempos em separado dos testes do *StressTest* pode ser observado na figura 4.8.

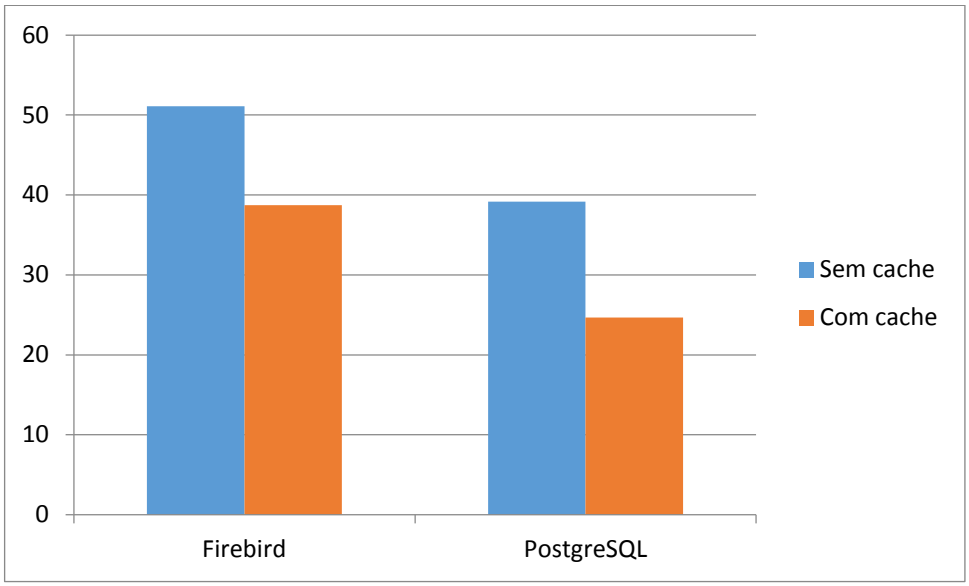

Figura 4.13 – Resultado final dos testes nos SGBD utilizados

Um fato interessante que pode ser observado através do gráfico da figura 4.13 é que o desempenho final do *Firebird* utilizando os dados em memória é levemente superior ao desempenho sem a utilização desta memória pelo *PostgreSQL*. Esta informação é importante para concluir que o *PostgreSQL* teve um desempenho satisfatório para os requisitos deste trabalho.

CONCLUSÃO

Através da pesquisa teórica detalhada realizada referente aos dois SGBDs envolvidos neste projeto foi possível observar uma grande semelhança entre as suas principais características e funções, possibilitando a continuidade do trabalho. Outros fatos importantes descobertos por meio desta pesquisa é que no processo de conversão dos módulos do sistema de automação comercial propriedades como *Stored Procedures*, índices e processos como *backup* e *restore* deveriam ter uma atenção especial, pois possuem definições distintas e que poderiam resultar em divergências nos resultados esperados para este trabalho caso não tratadas corretamente.

Com a base teórica formada foi possível criar um ambiente de testes adequado aos dois SGBDs e que possibilitou um bom desempenho nos processos de conversão e teste trazendo resultados mais precisos e seguros. Esta base teórica também foi importante para auxiliar nas rotinas de conversão de SQL do sistema de automação, pois muitas destas instruções tiveram que receber uma manutenção para garantir o seu funcionamento.

As conversões de dados realizadas pelos *softwares* pesquisados trouxeram um resultado aceitável para os padrões definidos neste trabalho. Com os índices de conversão dos dados chegando próximos a 100% foi possível corrigir os erros pontualmente e dar continuidade ao processo de conversão dos módulos do sistema. O software escolhido para realizar esta tarefa foi o *Datapump for PostgreSQL* e através dele os dados da base de dados original foram convertidos para a base de dados em *PostgreSQL* que foi usada no decorrer desta pesquisa.

Apesar de algumas dificuldades encontradas, como mudanças de SQL, modificações no código fonte e complicações nas conversões de *Stored Procedures*, as migrações dos módulos para funcionarem em *PostgreSQL* foram concluídas com sucesso. Os 3 módulos permaneceram com os seus comportamentos e valores inalterados, resultando em uma base confiável para a execução dos testes de desempenho.

Os testes realizados nos módulos trouxeram resultados claros quanto aos objetivos desta pesquisa. Através dos dados adquiridos foi possível confirmar a hipótese inicial do trabalho, onde convertendo o banco de dados do sistema de automação comercial para *PostgreSQL* haveria um ganho de desempenho. Este ganho foi mensurado e a partir desta análise é possível concluir que a migração de SGBD será benéfica ao sistema de automação.

Além da vantagem de desempenho, a pesquisa apresentou outros benefícios que o sistema ganhará ao migrar o banco de dados, como uma maior robustez em questões de acessos simultâneos, um melhor uso de recursos do sistema operacional e uma documentação mais completa para futuras consultas. Questões que podem fazer a diferença no futuro do software, onde existe uma ambição de entrar em clientes de grande porte, e a garantia de um desempenho superior com uma grande quantidade de conexões com o banco de dados daria uma segurança maior ao sistema de automação comercial.

Outra grande vantagem será nas consultas por campos *strings*, necessárias nas rotinas mais comuns e utilizadas do sistema. Quase todos os módulos do sistema possuem consultas que buscam *strings* ou fragmentos de *strings*, onde ficou comprovado neste trabalho que após a conversão haverão ganhos maiores que 170% nestas situações.

Com base no que foi citado acima é possível concluir que este trabalho apresentou uma solução adequada para um problema real de um sistema de automação comercial. Com uma base teórica formada, foram criadas soluções práticas para migrar um sistema de um banco de dados *Firebird* para *PostgreSQL*, assim como ficou comprovado que o seu desempenho após esta conversão ficou superior ao antigo SGBD utilizado.

REFERÊNCIAS BIBLIOGRÁFICAS

CANTU, Carlos H. ***Firebird* Essencial.** Rio de Janeiro: Ciência Moderna, 2005.

CANTU, Carlos H. ***Firebird* 2.0 – O Banco de Dados do Novo Milênio.** Rio de Janeiro: Ciência Moderna, 2006.

COLARES, Flávio Martins. **Análise comparativa de bancos de dados gratuitos.** Disponível em:<http://www.flf.edu.br/revista-flf/monografias-computacao/monografia_flaviocolares.pdf>. Acesso em: 11 março 2013.

DATE, C. J. **Introdução a Sistemas de Banco de Dados.** Rio de Janeiro: Elsevier, 2003.

Documentação oficial *PostgreSQL*. **Site oficial do SGBD *PostgreSQL* – Features**. 2013. Disponível em: <http://www.*PostgreSQL*.org/docs/9.1/static/features.html>. Acesso em: 12 maio 2013.

FIREBIRD. **Site oficial do SGBD *Firebird* – Team Members.** 2013a. Disponível em: <http://www.*Firebird*sql.org/en/team-members/ >. Acesso em: 12 março 2013.

FIREBIRD. **Site oficial do SGBD *Firebird* – Release Description.** 2013b. Disponível em:

<http://www.*Firebird*sql.org/en/*Firebird*-2-5-release-description/>. Acesso em: 12 março 2013.

FIREBIRD. **Site oficial do SGBD *Firebird* – Classic or Super.** 2013c. Disponível em: <http://www.*Firebird*sql.org/manual/qsg15-classic-or-super.html>. Acesso em: 12 maio 2013.

FIREBIRD. **Site oficial do SGBD *Firebird* – Classic or Super.** 2013d. Disponível em: <http://www.*Firebird*sql.org/manual/qsg25-classic-or-super.html>. Acesso em: 12 maio 2013.

FIREBIRD. **Site oficial do SGBD *Firebird* – Team Members.** 2013e. Disponível em: < http://www.*Firebird*sql.org/manual/gbak.html#gbak-intro>. Acesso em: 12 maio 2013.

IB-AID. **1 TB *Firebird* databases: Preliminary Report.** Disponível em: <http://http://www.ib-aid.com/articles/item104> Acesso em: 02 maio 2013

MATTHEW, Neil; STONES, Richards. **Beginning Databases with *PostgreSQL* from Novice to Professional.** Springer-Verlag. New York. 2005.

MICROSOFT. **Site oficial Microsoft – ADO.** 2013. Disponível em: < http://msdn.microsoft.com/en-us/library/windows/desktop/ms675532(v=vs.85).aspx>. Acesso em: 12 março 2013.

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

PGFOUNDRY. **PgFoundry Forge.** 2013. Disponível em: <http://www.pgfoundry.org>. Acesso em: 15 março 2013.

PLETSCH, Edson Luis. **Avaliação das técnicas de desempenho em sistemas de gerenciadores de banco de dados relacionais.** Novo Hamburgo. 2005.

POSTGRESQL. **Site oficial do SGBD *PostgreSQL* – Contributors**. 2013a. Disponível em: <http://www.*PostgreSQL*.org/community/contributors/>. Acesso em: 11 março 2013.

POSTGRESQL. **Site oficial do SGBD *PostgreSQL* – Versioning.** 2013b. Disponível em: <http://www.*PostgreSQL*.org/support/versioning/>. Acesso em: 12 março 2013.

Site oficial da Embarcadeiro.

POSTGRESQL. **Site oficial do SGBD *PostgreSQL* – History**. 2013c. Disponível em: <http://pgdocptbr.sourceforge.net/pg80/history.html>. Acesso em: 12 maio 2013.

POSTGRESQL. **Site oficial do SGBD *PostgreSQL* – Tutorial**. 2013d. Disponível em: <http://pgdocptbr.sourceforge.net/pg80/tutorial-arch.html>. Acesso em: 12 maio 2013.

POSTGRESQL. **Site oficial do SGBD *PostgreSQL* – Indexes Types**. 2013e. Disponível em: <http://pgdocptbr.sourceforge.net/pg80/indexes-types.html>. Acesso em: 12 maio 2013.

POSTGRESQL. **Site oficial do SGBD *PostgreSQL* – Datatype**. 2013f. Disponível em: <http://pgdocptbr.sourceforge.net/pg80/datatype.html>. Acesso em: 12 maio 2013.

POSTGRESQL. **Site oficial do SGBD *PostgreSQL* – Datatype Character**. 2013g. Disponível em: <http://pgdocptbr.sourceforge.net/pg80/datatype-character.html>. Acesso em: 12 maio 2013.

SOUZA, Maurício de Oliveira; MATIOSKI, Murilo Estevam; NEVES, Antônio Pereira. **Análise de desempenho dos bancos de dados Mysql, PosgreSQL e *Firebird*: Estudo de caso.** Curitiba: Gráfica Expoente, 2008.