FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” – UNIVEM
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOÃO LUIS BENITES JÚNIOR

FERRAMENTA PARA CONVERSÃO, REPLICAÇÃO E ATUALIZAÇÃO DE DADOS

MARÍLIA
2005

JOÃO LUIS BENITES JÚNIOR

FERRAMENTA PARA CONVERSÃO, REPLICAÇÃO E ATUALIZAÇÃO DE DADOS

Monografia apresentada ao Curso de Ciência da Computação do Centro Universitário Eurípides de Marília – UNIVEM, mantido pela Fundação Eurípides Soares da Rocha, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora:
Profª. Fátima L.S. Nunes

MARÍLIA
2005

A Deus, por me dar forças nesta jornada.

A meus pais João e Aparecida, minha namorada Cris, minhas irmãs Milena, Karina e Maira por estarem sempre a meu lado, me ajudando sempre a alcançar meus objetivos.

A todos os amigos presentes em todos os momentos.

AGRADECIMENTOS

A todos que de forma direta ou indireta contribuíram ou serviram de inspiração para produção deste trabalho.

Agradeço em especial:

À Profª. Fátima L.S. Nunes, pela compreensão e auxílio na orientação, transmitindo-me conhecimento de forma clara e objetiva, com muito profissionalismo e inteligência, tornando sua ajuda indispensável para a conclusão deste trabalho.

Ao sr. Vlad Karpov (Rússia), pela paciência que demonstrou ter dedicando parte de seu tempo no esclarecimento de algumas das dúvidas que surgiram ao longo do desenvolvimento deste trabalho.

Aos grandes amigos que fiz nesta instituição e que levarei para sempre em minha memória.

BENITES, João Luis, Jr. *Ferramenta para conversão, replicação e atualização de dados*. 2005. 110 f. Monografia (Bacharelado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

RESUMO

A finalidade deste projeto é o desenvolvimento de uma ferramenta que possibilite a conversão de tabelas e a replicação de dados entre uma base de dados nativa do Clipper e Sistemas Gerenciadores de Banco de Dados (MySQL e PostgreSQL). A ferramenta também disponibiliza de um sistema para o envio e recebimento de atualizações realizadas em ambas as bases de dados (Clipper e SGBD) mantendo-as idênticas, preservando a integridade dos dados mesmo após a realização da replicação.
Para atingir o resultado esperado, o projeto passou pelas seguintes etapas: 1 – Estudo de características dos SGBDs; 2 – Desenvolvimento de algoritmos para a conversão de tabelas e replicação e atualização dos dados; 3 – Realização de Testes de Desempenho e Integridade.
Após realização dos testes com a ferramenta, concluiu que os resultados obtidos foram satisfatórios, principalmente pela redução no número de mecanismos necessários para a execução destas tarefas.

Palavras-chave: Banco de dados, Replicação de dados, Conversão de dados

BENITES, João Luis, Jr. Tool for conversion, copy and update of data. 2005. 110 f. Monograph (Bachelor in Computer Science) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ABSTRACT

The purpose of this project is developing a tool that makes possible the conversion of tables and the copy of data between a Clipper native database and Database Management System(MySQL and PostgreSQL). The tool also provides a system for sending and receiving of updates in both databases (Clipper and DBMS) keeping them identical, preserving their integrity after the accomplishment of the copy.
Three steps were developed in the project: 1- Study of characteristics of DBMS´s; 2 – Development of programs for table conversion and copy of data and update of data; 3 – Execution of performance and integrity tests.
From the tests it was possible to verify that the tool works in a satisfactory way, without errors of conversion, copy and update processes. The tool decreases the amount of necessary mechanisms for execution of these tasks.

Key expression: Database, Data Copy, Data Conversion

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

INTRODUÇÃO

Os Sistemas Gerenciadores de Banco de Dados (SGBDs) são muito utilizados por se tratarem de ferramentas rápidas, seguras e eficientes no armazenamento, manutenção e recuperação de dados. Com o surgimento de Sistemas Gerenciadores de Banco de Dados gratuitos, muitas empresas, até mesmo de pequeno porte, estão adequando seus sistemas para trabalharem com SGBDs.

Grande parte destas organizações, porém, "adiam" seu projeto de migração devido a problemas encontrados com a linguagem de programação utilizada. Algumas linguagens de programação, por exemplo não oferecem suporte (*Driver* / Conectores) para que seja feita uma conexão com um SGBD desejado. Além disso, podem possuir seu próprio sistema de arquivos, o que vem a ser um problema adicional, pois torna-se necessária uma migração tanto dos dados da organização quanto dos programas em operação.

Esta mudança radical é ainda mais complexa quande se trata de sistemas legados, podendo trazer muitos problemas, ao invés de conveniências.

Sistemas legados não são simplesmente software antigos, são sistemas que incluem software, hardware, dados e processos corporativos. As modificações em uma parte do sistema, inevitavelmente, envolvem mais modificações em outros componentes (Sommerville, 2003).

Objetivos e Justificativa

Este trabalho tem como finalidade o desenvolvimento de uma ferramenta capaz de auxiliar organizações que buscam a migração de seus sistemas desenvolvidos especificamente em linguagem Clipper 5 (ou similares), com bases de dados no formato Xbase, para qualquer

outra linguagem que utiliza um SGBD MySQL ou PostgreSQL, de forma que esta migração ocorra de maneira simples, rápida e conveniente, mesmo quando se trata de sistemas legados, pois preserva a integração entre a base de dados antiga e o novo SGBD para que a migração ocorra de forma gradativa.

A ferramenta possui dois módulos básicos: conversão das estruturas e replicação dos dados.

O módulo de *Conversão das Estruturas* tem como finalidade capturar a estrutura física dos arquivos de tabelas (extensão .DBF) e convertê-la para tabelas do SGBD MySQL ou PostgreSQL, bem como, através da definição da chave primária pelo arquivo de índice (extensão .NTX) criá-la no SGBD. Na conversão serão analisados os tipos de dados suportados pelas tabelas Xbase e pelas tabelas dos SGBD´s, para que haja compatibilidade e não venha prejudicar a integridade dos dados na etapa de replicação.

O módulo de *Replicação dos Dados* é executado em duas etapas. A primeira consiste na alimentação dos dados da tabela pela primeira vez, ou seja, após a conversão da estrutura é feita a transferência dos dados existentes na tabela origem para a tabela destino. A segunda etapa ocorre em dois processos: o envio das modificações dos dados contidos na tabela origem para a tabela destino e o recebimento das modificações nos dados contidos na tabela destino para a tabela origem. Desta forma, os dados em ambas as bases se manterão atualizados, sem a necessidade de uma nova replicação geral da tabela para cada modificação realizada.

Com o presente trabalho, pretende-se obter o maior grau de conveniência possível para a realização do processo de migração, eliminando o número de mecanismos necessários para a realização destas operações.

Organização do trabalho

Além desta introdução, seguem os seguintes capítulos:

- O Capítulo 2 define o Sistema Gerenciador de Banco de Dados e apresenta suas principais vantagens.
- No capítulo 3 é mostrado um estudo sobre Modelo Entidade - Relacionamento, ou seja, uma explicação sobre objetos básicos de entidade e relacionamento.
- No capítulo 4 são apresentadas as características das tecnologias utilizadas: Sistema de Arquivos Banco de dados do Clipper 5 e SGBD PostgreSQL, além das informações necessárias sobre a definição da linguagem utilizada
- No capítulo 5 é descrito o funcionamento da ferramenta para a replicação.
- No capítulo 6 são descritos os resultados obtidos na conversão e replicação, além dos problemas encontrados.
- Para finalizar são apresentadas as conclusões obtidas através dos testes realizados, bem como as referências bibliográficas utilizadas na monografia.

2. SISTEMA GERENCIADOR DE BANCO DE DADOS

Neste capítulo será visto em detalhes o conceito de SGBD, suas vantagens e algumas das linguagens utilizadas para execução de operações nos SGBD´s.

Segundo Silberschatz, Korth e Sudarshan (1999), um Sistema Gerenciador de Banco de Dados (SGBD) é um sistema constituído por um conjunto de dados associados a um conjunto de programas que possibilitam acesso a esses dados.

Estes sistemas são criados para gerenciar um grande volume de informação de forma segura, conveniente e eficiente, atendendo à necessidades dos aplicativos. São usados para o armazenamento e manipulação de dados e devem garantir segurança das informações armazenadas, tanto contra eventuais problemas com o sistema quanto a tentativas de acessos não autorizadas.

2.1. Vantagens da utilização de um Sistema Gerenciador de Banco de Dados

Dentre as principais vantagens oferecidas podemos citar algumas mencionadas por Silberschatz, Korth e Sudarshan (1999):

Eliminação de Redundância: a redundância, que é a informação repetida em vários locais de armazenamentos diferentes é eliminada centralizando esta informação em apenas um sistema de banco de dados e compartilhando com todas as aplicações que a utilizam.

Eliminação de Inconsistências: em decorrência da eliminação da redundância dos dados, a inconsistência, que é a presença de um determinado dado de mesmo significado em locais diferentes com valores diferentes, também é eliminada com a centralização dos dados.

Por exemplo, centralizando o armazenamento de informações referentes a um determinado funcionário, a empresa não corre o risco de quando trocar por exemplo o cargo deste funcionário, este ficar com seu registro como “auxiliar administrativo” e em seu aviso de pagamento sair a antiga função que era “serviços gerais”.

Garantia de Integridade dos dados: a integridade dos dados é garantida quando são impostas certas restrições para a manutenção da consistência na entrada dos dados. Por exemplo, não é aceitável que qualquer aluno do curso de Ciência da Computação possua média final menor que zero ou maior que dez. Então, é imposta a regra que qualquer valor inserido para nota deve estar entre zero e dez.

Facilidade no Acesso aos Dados: não é obrigatória a construção de programas para acesso a um determinado tipo de informação ou emissão de relatório. Para isto basta apenas ter acesso ao banco e utilizar-se de linguagens de consulta própria do banco.

Segurança: nem todos os usuários do banco de dados estão autorizados ao acesso a todos os dados. Apesar da informação estar presente no mesmo banco de dados, é possível que configure o acesso aos dados de tal forma que um usuário só tenha acesso ao que lhe é de interesse. Por exemplo, o departamento pessoal de uma empresa deve ter acesso somente aos dados que se referem aos funcionários da empresa e não aos clientes da empresa.

Atomicidade: algumas operações no banco de dados devem ser feitas de forma única para assegurar a integridade e a consistência dos dados. Em algumas operações é crucial assegurar que, uma vez detectada uma falha, os dados sejam salvos em seu último estado consistente, ou seja, anterior à ocorrência da falha. Por exemplo, se em um processo de transferência de valores de uma conta A para uma conta B ocorrer falha no sistema durante

sua execução, é possível que o valor transferido seja debitado da conta A sem creditar na conta B, criando a inconsistência no banco de dados. A atomicidade neste caso vem garantir que ambas as operações (débito e crédito) sejam realizadas ou nenhuma delas.

Independência de dados: é a capacidade de modificar a definição dos esquemas em um determinado nível, sem afetar o esquema do nível superior. Existem dois níveis de independência de dados: independência de dados física e independência de dados lógica. A primeira é a capacidade de modificar o esquema físico dos dados para que se aprimore o desempenho sem que com isso haja necessidade de reescrever aplicações e a segunda é a mesma capacidade, referente ao nível lógico, sendo mais difícil de ser alcançada.

Integridade Semântica: A integridade semântica é o controle de valores válidos para os dados no SGBD. Ela existe para garantir que o dado em questão esteja correto mesmo se o usuário ou programa de aplicação tentar modificá-lo de forma incorreta. O valor do dado inserido ou atualizado é igual a algum valor pré-definido, ou está incluído em um intervalo de valores válidos. O SGBD monitora a atualização destes dados para que o dado seja tratado de acordo com o definidio, quando ocorrer a violação da integridade. Com isto tem-se a darantia de independência de gerenciamento de integridade de dados para as aplicações.

2.2. Implicações do Enfoque de Banco de Dados

Além das vantagens discutidas anteriormente, existem outras implicações na utilização do enfoque de SGBD, segundo Elmasri e Navathe (2002) que podem beneficiar a maior parte das organizações. São elas:

Potencial para impor padrões: o enfoque de banco de dados permite ao administrador do banco definir e impor padrões entre usuários de banco de dados numa organização. Isso facilita a comunicação e a cooperação entre vários departamentos, projetos e usuários dentro da organização. Padrões podem ser definidos para nomes e formatos de elementos de dados, formatos de apresentação, estruturas de relatórios, terminologia, etc.

Tempo reduzido para desenvolvimento de aplicações: o desenvolvimento de uma nova aplicação, como um novo relatório, por exemplo, leva muito menos tempo em relação a uma aplicação que utiliza arquivos tradicionais. Se partirmos desde a implementação de um novo banco de dados, este tempo pode ser maior em relação a um sistema de arquivos tradicionais, porém, na maioria das vezes, novas aplicações são desenvolvidas para bancos já existentes, onde o tempo gasto é de um sexto a um quarto do tempo necessário para um sistema de arquivos tradicionais.

Economias de escala: O enfoque do SGBD permite a consolidação de dados e aplicações, reduzindo desta forma a quantidade de desperdício por sobreposições entre atividades do pessoal de desenvolvimento e manutenção em diferentes projetos ou departamentos. Isto possibilita que toda a organização invista em processadores, dispositivos de armazenamento ou dispositivos de comunicação com maior capacidade, em vez de cada departamento ter que adquirir seu próprio equipamento (com menor capacidade). Isso reduz os custos gerais de operação e gerenciamento.

2.3. Abstração dos Dados

Segundo Silberschatz, Korth e Sudarshan (1999), a eficiência na recuperação das

informações de um SGBD está diretamente ligada à forma pela qual foram projetadas as complexas estruturas de representação desses dados. Muitos dos usuários de banco de dados necessitam de sistemas que omitem essa complexidade. Por este motivo foram desenvolvidos sistemas que, por meio de níveis de abstração, facilitam a interação dos usuários com o sistema.

A abstração dos dados é definida em três níveis:

Nível Físico (também conhecido como nível interno): é o mais baixo nível de abstração que descreve *como* esses dados estão de fato armazenados. Neste nível, estruturas de dados complexas de baixo nível são descritas em detalhes.

Nível Conceitual ou Lógico (também conhecido como nível lógico comunitário): descreve *quais dados* estão armazenados no banco de dados e *quais os inter-relacionamentos entre eles*. Assim, o banco de dados como um todo é descrito em termos de um número relativamente pequeno de estruturas simples. Este nível é utilizado por administradores do banco de dados que precisam decidir quais informações devem pertencer ao banco de dados.

Nível de Visão (também conhecido como nível externo ou nível lógico do usuário): é o mais alto nível de abstração, sendo o mais próximo do usuário. Descreve apenas parte do banco de dados, ou seja, a parte que é de interesse daquele determinado usuário. Por esta razão, o número de níveis de visão é determinado de acordo com a necessidade dos usuários do banco de dados, variando-se de 1 a n.

Na Figura 2.1 ilustra-se os níveis de abstração e seus relacionamentos:

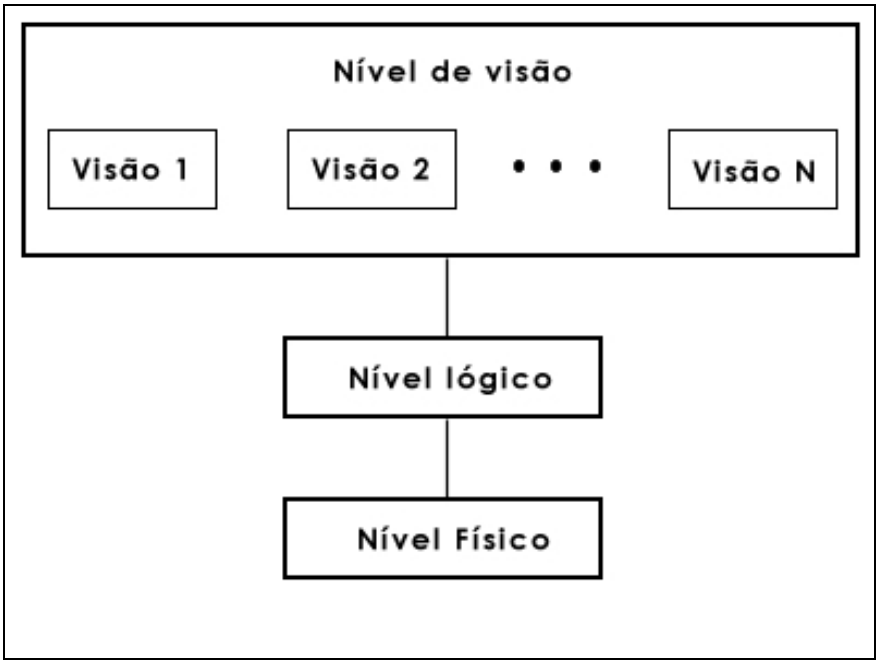

Figura 2.1 – Níveis de abstração dos dados.

2.4. Linguagens de Banco de Dados

Um sistema de banco de dados proporciona a utilização de dois tipos de linguagens que são descritas a seguir: uma específica para gerenciar os esquemas do banco de dados e outra para expressar consultas e atualizações dos dados.

2.4.1. Linguagem de Definição dos Dados

Segundo Silberschatz, Korth e Sudarshan (1999), um esquema de dados é especificado por um conjunto de definições expressas por uma linguagem especial chamada *Linguagem de Definição de Dados (Data Definition Language – DDL)*. O resultado da

compilação dos parâmetros DDL's é armazenado em um conjunto de tabelas que constituem um arquivo especial chamado *dicionário de dados* ou *diretório de dados*.

Um dicionário de dados é um arquivo de *metadados,* isto é, dados a respeito de dados. Esse arquivo ou diretório é consultado antes que o dado real seja modificado, garantindo que o tipo de dado que se deseja armazenar corresponde ou é compatível com o tipo de dados suportado.

2.4.2. Linguagem de Manipulação dos Dados

Segundo Silberschatz, Korth e Sudarshan (1999), a *recuperação das informações* armazenadas no banco de dados, ou a inserção de novas informações ou mesmo a remoção ou atualização de informações do banco de dados são consideradas técnicas de manipulação de dados.

A manipulação dos dados de forma compatível ao modelo de dados apropriado no banco de dados é feita através da *Linguagem de Manipulação de Dados (Data Manipulation Language – DML)*. Estas linguagens tem como objetivo proporcionar eficiente interação humana com o sistema de banco de dados. É basicamente de dois tipos:

DML's procedurais: exigem que o usuário especifique *quais* dados são necessários e *como* obtê-los.

DML's não procedurais: exigem que o usuário especifique *quais* dados são necessários *sem* especificar como obtê-los.

As DML's não procedurais são mais fáceis de aprender e de usar, porém, como o usuário não especifica como obter os dados, essas linguagens podem gerar código menos eficiente que os gerados por linguagens procedurais.

2.4.3. A Linguagem SQL

Segundo Elmasri e Navathe (2005), a linguagem SQL (*Structured Query Language*) pode ser considerada uma das maiores razões para o sucesso do banco de dados relacional, pois como se tornou um padrão, os usuários têm pouca preocupação durante a migração de suas aplicações que utilizam o banco de dados.

A Linguagem SQL possui um comando básico para a recuperação de informações de um banco de dados: o comando SELECT. Este comando retorna, após sua execução, uma tabela (relação) que pode possuir duas ou mais tuplas (linhas) idênticas em todos os valores de seus atributos, ou seja, uma tabela retornada pelo comando SELECT, em geral, não é um conjunto de tuplas porque este conjunto não permitiria dois membros idênticos; se isso ocorresse seria um **multiconjunto** (*multiset*) de tuplas. Em algumas destas relações, é imposto a restrição apenas aos conjuntos de tuplas e não aos multiconjuntos, como por exemplo, utilizando a opção DISTINCT no comando SELECT ou durante a inserção do resultado do comando SQL em uma tabela física do banco de dados.

2.4.4. Linguagem de Controle dos Dados

Os dados armazenados no banco de dados precisam ser protegidos de acessos não autorizados, alteração intencional ou introdução acidental de inconsistência.

Para que isto não ocorra, utiliza-se de uma linguagem para controle de permissões dos objetos de um SGBD chamada DCL (*Data Control Language*). Através do uso de comandos de controle como *Grant* e *Revoke*, o administrador do banco ou o criador da tabela

atribui ou remove direitos de acessos ao objeto desejado, garantindo desta forma que os comandos de modificação dos dados da tabela (*Insert, Delete, Update*) só serão realizados por usuários que possuírem autorização para a realização desta tarefa.

3. MODELO ENTIDADE-RELACIONAMENTO (MER)

Neste Capítulo é apresentado o Modelo Entidade-relacionamento e suas restrições, o conceito de chave e o modelo relacional. Ainda é descrito a redução de um esquema Entidade-Relacionamento a Tabelas.

Segundo Date (2000), um modelo de dados é como uma linguagem de programação – embora seja uma linguagem um tanto quanto abstrata – cujas construções podem ser usadas para resolver uma ampla variedade de problemas específicos, embora não tenham por si próprias nenhuma conexão direta com qualquer problema específico.

O Modelo Entidade-Relacionamento (MER) é um modelo de dados conceitual de alto nível bastante popular. Esse modelo e suas variações são freqüentemente utilizados para projeto conceitual de aplicações de banco de dados e muitas ferramentas de projeto de banco de dados empregam seus conceitos (Elmasri e Navathe, 2002).

O Modelo Entidade-Relacionamento descreve dados como entidades, seus relacionamentos e atributos.

Uma entidade é um objeto do mundo real com uma existência independente, ou seja, pode ser um objeto com uma existência física (pessoa, carro, produto, etc) ou pode ser um objeto com uma existência conceitual (uma empresa, um serviço, um curso, etc).

Os atributos são as propriedades específicas que descrevem uma entidade. Por exemplo, uma entidade cliente pode ser descrita através do nome, endereço e cidade. Uma entidade específica terá um valor para cada um de seus atributos.

Os relacionamentos representam a associação entre uma ou várias entidades. Por exemplo, podemos definir um relacionamento que associa o cliente Carlos com o representante de venda Manoel. Esse relacionamento especifica que o cliente Carlos é um cliente pertencente à carteira de clientes do representante Manoel.

No Modelo Entidade-Relacionamento, os conjunstos de entidades podem ser

representadas (variando de autor para autor) através do uso de retângulos, os atributos através do uso de elipses e os relacionamentos através do uso de losangos. São utilizadas também linhas que unem os atributos ao conjunto de entidades e o conjunto de entidades aos seus relacionamentos. Na Figura 3.1 é ilustrado um exemplo desta padronização. Nota-se que os conjuntos de entidades "cliente" e "representante" são representados pelo uso de retângulos. O conjunto de entidades "cliente", por exemplo, possui os atibutos "nome_cliente", "rua_cliente" e "cidade_cliente", já o conjunto "representante" possui os atributos "código_repres" e "nome_repres". Estes atributos estão representados por elipses. O conjunto de entidades "cliente" possui um relacionamento com representante chamado "pertence", este relacionamento está representado por um losango e significa que o conjunto de entidades "cliente" pode possuir uma entidade cliente que pertence à uma determinado entidade representante.

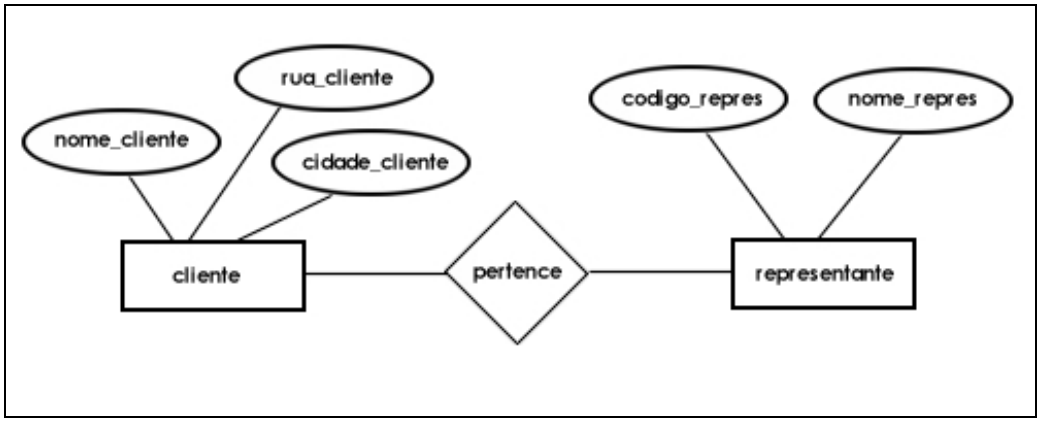

Figura 3.1 – Exemplo de uso do Modelo Entidade - Relacionamento.

3.1. Mapeamento de restrições

O Modelo Entidade – Relacionamento de uma organização pode conter certas

restrições, às quais o conteúdo do banco de dados deve respeitar. Dentre os mais importantes tipos de restrições pode-se ressaltar o mapeamento de cardinalidade e a dependência de existência.

3.1.1. Mapeamento de Cardinalidade

O mapeamento de cardinalidade expressa o número de entidades às quais outra entidade pode estar associada por meio de um relacionamento ou um conjunto de relacionamentos. É bastante útil na descrição dos conjuntos de relacionamentos entre dois conjuntos de entidades, embora possam contribuir para a descrição de conjuntos de relacionamentos que envolvam mais de dois conjuntos de entidades.

O mapeamento de cardinalidade pode ser, ou deve seguir as seguintes restrições:

Um para um: uma entidade em A está associada, no máximo a uma entidade em B, uma entidade em B está associada a, no máximo, uma entidade em A (Figura 3.2). Neste caso, como exemplo, podemos destacar que uma pessoa pode possuir apenas um número de CPF, assim como este CPF deverá ser de uma única pessoa.

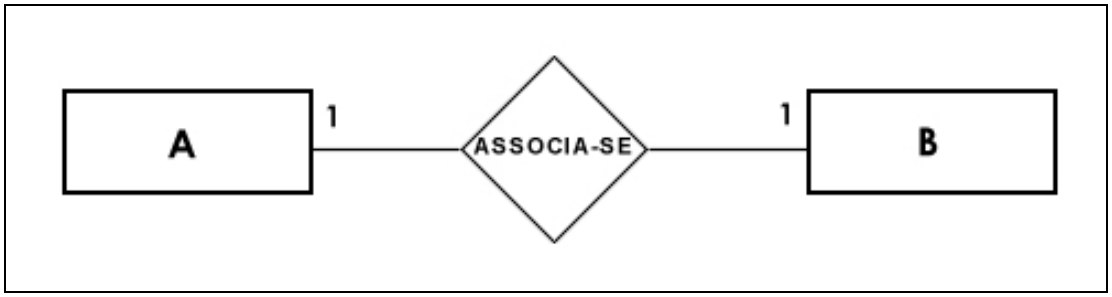

Figura 3.2 – Cardinalidade um para um.

Um para muitos: uma entidade em A está associada a várias entidades em B. Uma

entidade em B, entretanto, deve estar associada no máximo a uma entidade em A (Figura 3.3). Como exemplo pode-se citar que um único representante pode possuir em sua carteira de venda vários clientes e cada um destes clientes deve estar associado a somente um representante.

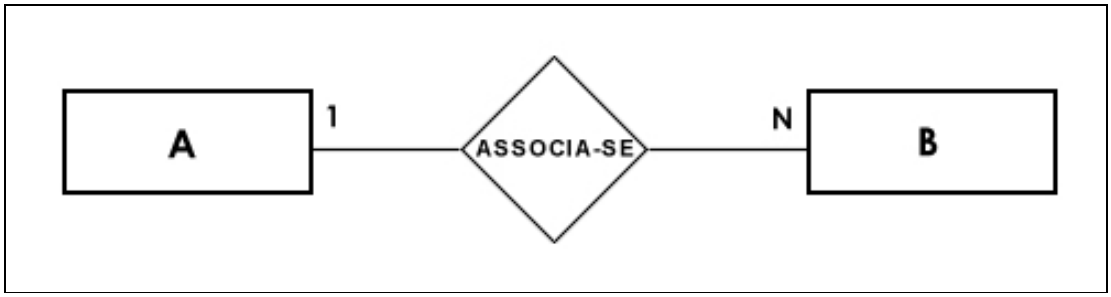

Figura 3.3 – Cardinalidade um para muitos.

Muitos para um: uma entidade em A está associada a no máximo uma entidade em B. Uma entidade em B, entretanto, pode estar associada a um número qualquer de entidades em A (Figura 3.4), ou seja, o inverso apresentado no mapeamento *um para muitos*. Por exemplo, um único cliente pode estar presente na carteira de vendas de vários representantes, porém, um representante poderá ter somente um cliente em sua carteira de vendas, o que operacionalmente não é nada comum, porém, foi citado somente para exemplo.

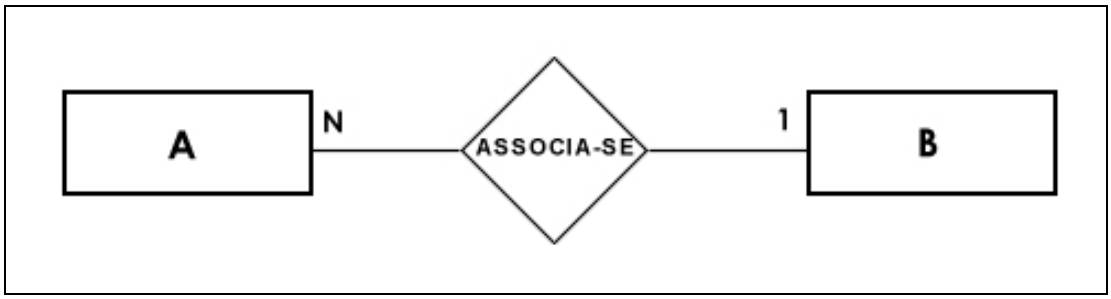

Figura 3.4 – Cardinalidade muitos para um.

Muitos para muitos: uma entidade em A está associada a qualquer número de

entidades em B e uma entidade em B está associada a um número qualquer de entidades em A (Figura 3.5). Por exemplo, um cliente pode estar presente na carteira de um ou mais representantes e um representante poderá vender para um ou vários clientes.

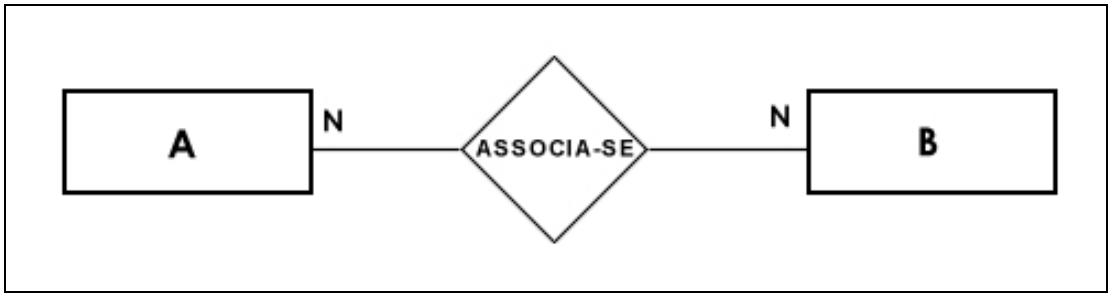

Figura 3.5 – Cardinalidade muitos para muitos.

Para uma melhor compreensão na Figura 3.6 ilustra-se a utilização em conjunto das restrições acima.

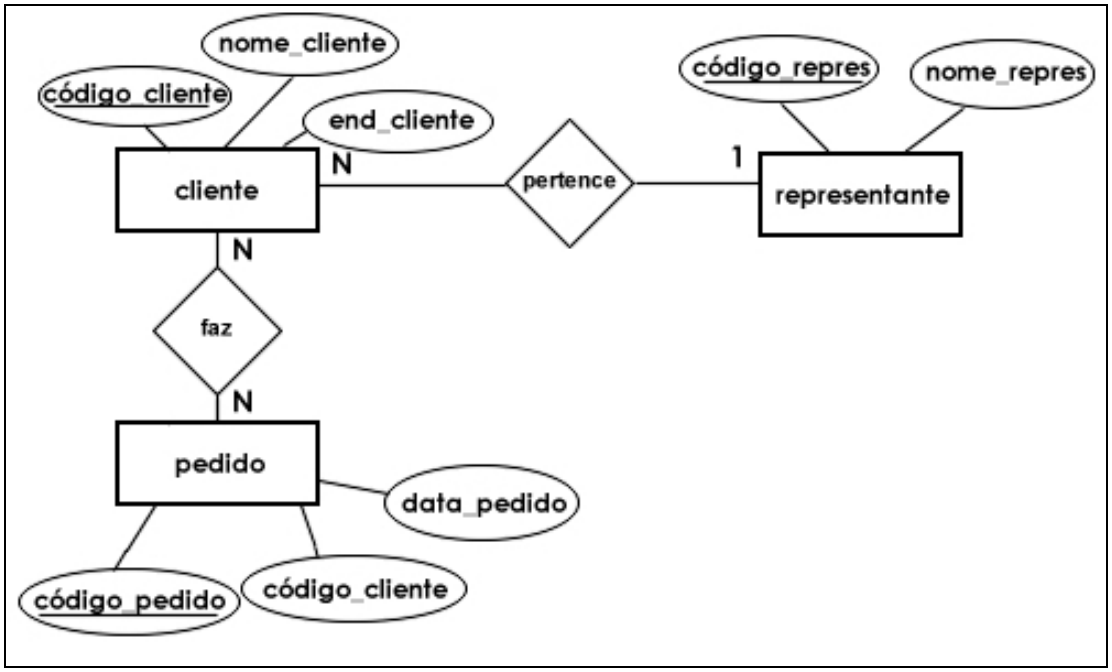

Figura 3.6 – Exemplo de uso do mapeamento de cardinalidade

Na Figura 3.6, cada entidade representante poderá ter em sua carteira de vendas

vários clientes, porém estes clientes devem pertencer a somente um representante. Estes clientes poderão então efetuar diversas compras junto a empresa através de uma entidade pedido, a qual poderá estar associada a vários clientes.

3.1.2. Dependência de Existência

A dependência de existência pode ser definida da seguinte forma: se a existência da entidade X depende da existência da entidade Y, então X é dito dependente da existência de Y. Operacionalmente, se Y for excluído, o mesmo deve acontecer com X. A entidade Y é chamada entidade dominante ou entidade forte e a entidade X é chamada de entidade subordinada ou entidade fraca (Silberschatz, Korth e Sudarshan, 1999) .

Pode-se considerar o mesmo exemplo aplicado anteriormente, adicionando-se apenas o conjunto de entidades **item de pedido**, conforme Figura 3.7. Este conjunto de entidades representa os itens ou produtos pertencentes a um determinado pedido, ou seja, quando um **cliente** efetua uma compra junto à empresa (através de um pedido), esta compra (**pedido**) é composta por vários produtos disponíveis para a venda pela empresa.

Diz-se então que a entidade **item de pedido** é dependente de existência da entidade **pedido**, pois sem a existência de uma venda a mesma não existirá. Caso a venda (ou o **pedido**) deixe de existir, os itens desta venda (**item de pedido**) também deverão deixar de existir.

Chama-se então a entidade **pedido** de entidade dominante ou entidade forte em relação a entidade **item de pedido** e a **entidade item de pedido** de entidade subordinada ou entidade fraca em relação a entidade **pedido**. A representação de entidade fraca no MER é padronizada por um retângulo de linhas duplas.

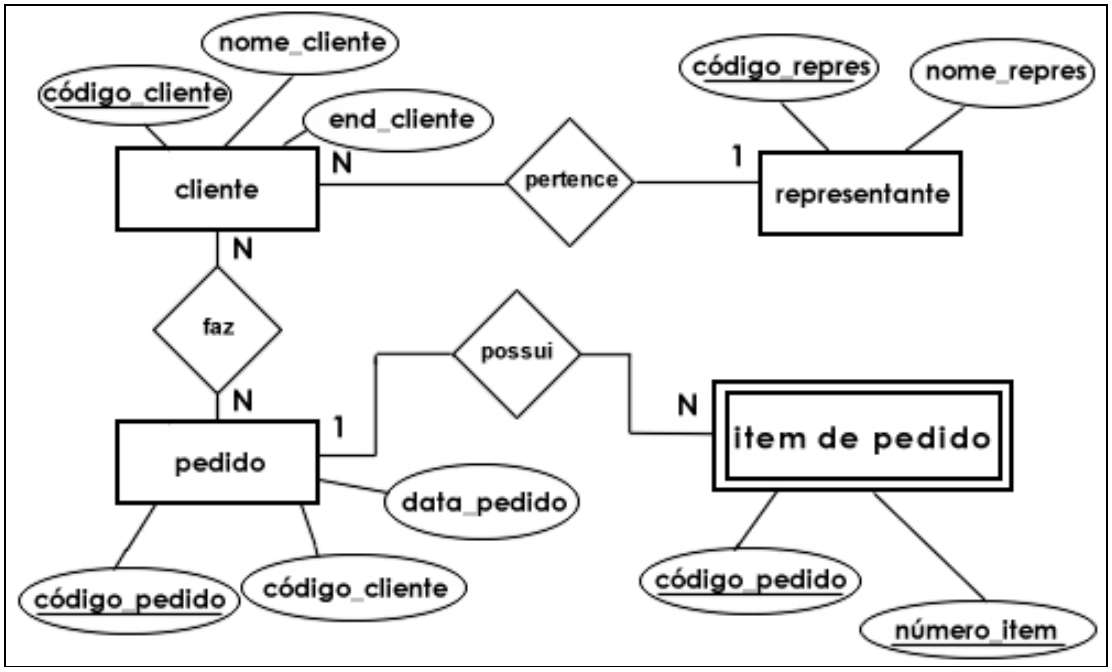

Figura 3.7 – Exemplo da representação de entidade forte e entidade fraca.

3.2. Chaves

O conceito de chave permite-nos fazer uma distinção de uma determinada entidade dentro de um conjunto de entidades através de um ou mais atributos.

Uma *superchave* é um agrupamento de um ou mais atributos que, tomados coletivamente, permitem identificar de maneira unívoca uma entidade em um conjunto de entidades. Em outras palavras: se K é uma superchave, qualquer grupo de atributos que contenham K é uma superchave (Silberschatz, Korth e Sudarshan, 1999).

Chamamos de *chave candidata* os casos onde a superchave é formada por subgrupos de atributos que não são superchaves.

Uma chave (primária, candidata ou super) é uma propriedade do conjunto de entidades e não de uma entidade individualmente. Quaisquer duas entidades individuais em um conjunto não podem ter, simultaneamente, mesmos valores em seus atributos–chave.

O termo chave primária é utilizado para caracterizar a chave candidata que é escolhida pelo projetista do banco de dados como de significado principal, ou seja, que possui identificador único, para a identificação de entidades dentro de um conjunto de entidades (Silberschatz, Korth e Sudarshan, 1999).

A utilização de chaves externas (ou estrangeiras) pode ocorrer principalmente em duas etapas:

Quando se trabalha com entidades fracas: neste caso a chave primária deste conjunto de entidades é a chave primária da entidade forte da qual é existencialmente dependente, adicionando-se um discriminador (ou identificador adicional) pertencente a esta entidade.

Quando se trabalha com um conjunto de relacionamentos: neste caso, a chave primária é composta por todos os atributos que constituem chaves primárias dos conjuntos de entidades envolvidas no relacionamento.

3.3. Modelo Relacional

Segundo Silberschatz, Korth e Sudarshan (1999), o modelo relacional usa um conjunto de tabelas para representar tanto os dados como a relação entre eles. Cada tabela possui múltiplas colunas e cada coluna possui um nome único. Uma linha em uma tabela representa um relacionamento entre um conjunto de valores. Uma vez que essa tabela é uma coleção de tais relacionamentos, há uma estreita correspondência entre o conceito tabela e o conceito matemático de relação, de onde se origina o nome desse modelo de dados. Na figura 3.8, é mostrado um exemplo de banco de dados relacional utilizando duas tabelas. A primeira

tabela representa os clientes da empresa e a segunda representa suas compras junto à empresa. O cliente João Carlos de Almeida, por exemplo, que está cadastrado sobre o código 122, que reside na Rua Aimorés, 200, na cidade de Tupã, estado de SP, efetuou uma compra junto à empresa sobre o pedido de número 587922. Este pedido foi efetuado no dia 02/05/2005 e possui data de entrega prevista para 07/05/2005.

Código	Nome	Endereço	Cidade	UF
120	Giovana Albuquerque	Rua do Ovo, 51	Bastos	SP
121	João Alberto da Silva	Av.15 de março, 10	Marília	SP
122	João Carlos de Almeida	Rua Aimorés, 200	Tupã	SP
123	José Francisco Alves	Rua Tupis, 1945	Tupã	SP

(a)

Numero	Cliente	Data_Pedido	Data_Entrega
587919	123	01/05/2005	06/05/2005
587920	121	01/05/2005	06/05/2005
587921	120	02/05/2005	07/05/2005
587922	122	02/05/2005	07/05/2005
587923	123	02/05/2005	06/05/2005

(b)

Figura 3.8 – Exemplo de tabelas no Modelo Relacional

3.4. Redução de um esquema Entidade-Relacionamento a Tabelas

Um banco de dados que utiliza-se de um esquema de banco de dados Entidade-Relacionamento pode ser representado por uma coleção de tabelas. Para cada conjunto de entidades e para cada conjunto de relacionamentos, existe uma tabela única dentro do banco de dados registrando o nome do conjunto de entidades / relacionamentos correspondente.

Tanto o Modelo Relacional quanto o modelo Entidade-Relacionamento são abstratos, ou seja, são representações lógicas de objetos reais. Como esses dois modelos empregam princípios de projeto similares, pode-se converter o projeto Entidade-Relacionamento em projeto relacional. Embora existam importantes diferenças entre uma relação e uma tabela, informalmente, uma relação pode ser considerada uma tabela de valores (Silberschatz, Korth e Sudarshan, 1999).

4. TECNOLOGIAS UTILIZADAS

Neste capítulo são apresentadas as tecnologias utilizadas, bem como em casos necessários, o motivo pelo qual a tecnologia foi escolhida.

4.1. Sistema de Banco de Dados do Clipper

Segundo Straley (1993), um arquivo de banco de dados no Clipper é um arquivo que possui campos e registros, de forma muito semelhante às linhas e colunas de uma planilha eletrônica. Cada registro (ou linha) quando acrescentado ao arquivo de banco de dados coloca informações em vários campos. Esses campos podem ser de diversos tipos de dados. Na Tabela 4.1 pode-se encontrar o símbolo, o nome e o comprimento dos tipos de dados suportados pelo Clipper.

Tabela 4.1 – Tipo de dados suportados pelo Clipper.

Símbolo	***Definição***	***Comprimento***
C	Caracter	1 a (64k – 1)
N	Numérico	32 (dígitos + casas decimais)
L	Lógico	1
M	Memo	até 64Kb
D	Date	8

Cada arquivo de banco de dados pode conter até 1024 campos. Em complemento aos campos definidos, cada registro possui também um sinalizador de eliminação – um byte unitário que será definido com um asterisco (*) se o registro estiver marcado para ser eliminado.

Segundo Spence (1994), o Clipper também permite que sejam criadas aplicações com independência de formato de dados. Isto é possível por meio do sistema de controladores

intercambiáveis de banco de dados (RDD – *Repleaceable Database Drivers*). Um RDD significa um nível de codificação que permite acessar tabelas de dados físicos com os mesmos elementos da linguagem.

Porém, este controlador é restrito a alguns formatos de dados. A Tabela 4.2 representa estes formatos, de acordo com a linguagem nativa e extensão de arquivo.

Tabela 4.2 – Formato de dados suportado pelo controlador RDD do Clipper

Extensão de Arquivo	***Nativo da Linguagem***
.DBF / .IDC /.CDX	FoxPro, da Microsoft
.DBF / .NDX	DBASE III e dBASE IV, da Borland
.DBF / .MDX	DBASE IV, da Borland
.DBF / .NTX	Clipper
.DB / .PDX	Paradox, da Borland

Apesar de ser uma meta do RDD tornar a aplicação independente do determinado RDD que ela esteja utilizando, até a Versão 5.2 do Clipper não havia sido implementado nenhum controlador que utilizasse manipulação da linguagem de consulta SQL.

Dentre os formatos acima mencionados será utilizado no presente trabalho o formato nativo do Clipper, ou seja, o formato no qual a extensão dos arquivos de banco de dados é *DBF*, com seus arquivos de índices com a extensão *NTX*.

4.1.1. Características de um Sistema de Banco de Dados no Clipper

Dentre as características existentes em um Sistema de Banco de Dados no Clipper, destacam-se duas principais, que são de extrema necessidade de conhecimento para a elaboração deste trabalho:

- *Utilização de uma lista de ordens:* quando são utilizados arquivos de índices para busca, ou mesmo para garantir que a atualização do arquivo de banco de dados reflita em seus índices, é necessária a utilização de uma lista de ordens que armazena a ordem de abertura de todos os índices. Sabendo-se a ordem do índice na lista de ordens, é possível definir por exemplo, qual o índice utilizado para realizar determinada busca na tabela (Straley, 1993).

- *Tipo de retorno de uma busca:* diferentemente dos sistemas de banco de dados convencionais, nos quais o retorno de uma busca é o registro referente à busca solicitada, no Clipper o retorno de uma busca é o número do registro referente à busca solicitada, o que vem a ser um problema, pois, por este motivo, pode-se afirmar que registros duplicados poderão ser aceitos em uma tabela (Straley, 1993).

4.1.2. Desvantagens do Sistema de Banco de Dados do Clipper

Apesar de possuir muitas das características de um SGBD, o Sistema de Banco de Dados do Clipper, não pode ser classificado como um deles devido à ausência de certas características básicas indispensáveis, as quais são destacadas algumas a seguir:

Controle de Integridade do Banco de Dados a Cargo do Programador: a integridade do banco de dados que é garantida a partir da restrição do domínio dos dados, da restrição da chave primária e da restrição de chaves externas, no Clipper fica a cargo do desenvolvedor da aplicação devido à ausência de um suporte para um dicionário de dados.

Inconsistência dos Dados: apesar de se poder trabalhar com a definição de um índice exclusivo para uma chave primária, no Clipper isto não é o bastante, pois um registro com uma chave primária duplicada poderá ser incluído no banco de dados, ele só não aparecerá no índice definido como chave primária, porém, ele estará em outros índices. Portanto os dados poderão estar inconsistentes (Spence, 1994).

Todas estas características e considerações são levadas em conta de acordo com o formato de dados trabalhado.

4.2. PostgreSQL

O PostgreSQL é um SGBD objeto-relacional de código aberto que implementa, além das características de um SGBD relacional, algumas características de orientação a objetos, como herança e tipos personalizados. O PostgreSQL possui uma grande compatibilidade com os padrões SQL92/SQL99. É extremamente robusto e confiável, além de ser extremamente flexível e rico em recursos.

Segundo o site do PostgreSQL, (http://pgdocptbr.sourceforge.net/pg80/preface.html), pela riqueza de recursos e conformidade com os padrões, o PostgreSQL é um SGBD muito adequado para o estudo universitário do modelo relacional, além de ser uma ótima opção para empresas implementarem soluções de alta confiabilidade sem altos custos de licenciamento. É um programa distribuído sob a licença BSD (*Berkeley Software Distribution*), o que torna o seu código fonte disponível e o seu uso livre para aplicações.

Alguns dos recursos do PostgreSQL presentes em sua Versão 8.0.0, são:

- Comandos complexos;
- Chaves estrangeiras;
- Gatilhos;
- Visões;
- Integridade transacional;
- Controle de simultaneidade multi-versão.

4.2.1. Características do PostgreSQL

É interessante citar algumas especificações técnicas sobre a capacidade de armazenamento do PostgreSQL, de acordo com a referência utilizada:

- Tamanho máximo de uma base de dados: Ilimitado (existem bases de mais de 4 Terabytes);
- Tamanho máximo de uma tabela: 16 Terabytes em todos os sitemas operacionais;
- Tamanho máximo de uma linha: 1,6 Terabytes;
- Tamanho máximo de um campo: 1 Gigabyte;
- Número máximo de linhas por tabela: ilimitado;
- Número máximo de colunas por tabela: 250 - 1600 dependendo dos tipos das colunas;
- Número máximo de índices por tabela: ilimitado;

4.2.2. Tipos de dados suportados pelo PostgreSQL

Segundo o site http://pgdocptbr.sourceforge.net/pg80/datatype.html, o PostgreSQL

disponibiliza uma variedade muita grande de tipos de dados. Na Tabela 4.3 são apresentados os principais tipos de dados suportados pelo PostgreSQL em sua Versão 8.0.0.

Tabela 4.3 – Tipos de dados suportados pelo PostgreSQL

Nome	Tamanho	Intervalo/Precisão	Descrição
Smallint	2 bytes	-32768 a +32767	Precisão fixa, com faixa mínima
Integer	4 bytes	-2147483648 a +2147483647	Precisão usual para precisão fixa
Bigint	8 bytes	-9223372036854775808 a +9223372036854775807	Precisão fixa, com faixa grande
Decimal	Variável	Sem limite	Precisão especificada pelo usuário, exata
Numeric	Variável	Sem limite	Precisão especificada pelo usuário, exata
Real	4 bytes	Precisão de 6 dígitos decimais	Precisão variável, inexata
Double precision	8 bytes	Precisão de 15 dígitos decimais	Precisão variável, inexata
Serial	4 bytes	1 a 2147483647	Inteiro autoincremental
Bigserial	8 bytes	1 a 9223372036854775807	Inteiro autoincremental grande
Money	4 bytes	-21474836.48 a +21474836.47	Valor monetário
character varying(*n*), varchar(*n*)	-	-	Tamanho da variável com limite
character(*n*), char(*n*)	-	-	Tamanho fixo, completado com brancos
Text	-	-	Tamanho variável não limitado
Bytea	4 bytes plus (o atual para textos binários)	-	Variável - tamanho de texto binário
timestamp [(*p*)] [without time zone]	8 bytes	4713 AC a 5874897 DC	Data ou hora
timestamp [(*p*)] with time zone	8 bytes	4713 AC a 5874897 DC	Data ou hora
interval [(*p*)]	12 bytes	-178000000 anos a 178000000 anos	Intervalos de tempo
Date	4 bytes	4713 AC a 32767 DC	Somente datas
time [(*p*)] [without time zone]	8 bytes	00:00:00.00 a 23:59:59.99	Somente a hora do dia
time [(*p*)] with time zone	12 bytes	00:00:00.00+12 a 23:59:59.99-12	Somente a hora do dia
Boolean	-	(‘TRUE’, ‘t’, ‘true’, ‘y’, ‘yes’, ‘1’) ou (‘FALSE’, ‘f’, ‘false’, ‘n’, ‘no’, ‘0’)	-
Cidr	12 a 24 bytes	-	Endereços de redes IPv4 ou IPv6
Inet	12 a 24 bytes	-	Hosts e Endereços de redes Ipv4 e Ipv6
Macaddr	6 bytes	-	Endereços MAC
Point	16 bytes	Ponto no plano	(x,y)
Line	32 bytes	Linha Infinita(não implementado totalmente ainda)	((x1,y1),(x2,y2))
Lseg	32 bytes	Segmento finito de uma linha	((x1,y1),(x2,y2))
Box	32 bytes	Pontos de uma caixa retangular	((x1,y1),(x2,y2))
Path	16+16n bytes	Caminho fechado (similar ao polígono)	((x1,y1),...)
Path	16+16n bytes	Caminho aberto	((x1,y1),...]

Tabela 4.3 – Tipos de dados suportados pelo PostgreSQL (continuação)

Nome	Tamanho	Intervalo/Precisão	Descrição
Polygon	40+16n bytes	Polígono (similar a um caminho fechado)	((x1,y1),...)
Circle	24 bytes	Círculo	<(x,y),r> (centro e raio)
Bit (n)	-	Bit inteiro positivo	-
Bit varying (n)	-	Bit inteiro positivo	-

Alguns tipos de dados não listados na Tabela 4.3, são listados a seguir, devido a sua complexidade:

Arrays: Segundo o site do PostgreSQL (http://pgdocptbr.sourceforge.net/pg80/arrays.html), como em várias linguagens de programação, a definição do tipo de dados *Array* é feita através da declaração do tipo do dado escolhido pelo usuário (integer, text, caracter, etc), seguido do número de elementos do *array* (linhas e colunas). Ex: integer[5][4], onde cada registro deste campo será um *array* de inteiros, com cinco linhas e quatro colunas. O seu acesso é feito através do uso da palavra ARRAY seguida de { } tendo os seus elementos separados por vírgula na declaração.Ex: ARRAY{{1, 2, 3, 4},{4, 3, 2, 1},{5, 6, 7, 8},{9, 3, 2, 2},{8, 5, 7, 8}}

Identificador de Objetos (OID´s): o tipo OID representa um identificador de objeto. São usados internamente pelo PostgreSQL como chaves primária para vários sistemas de tabelas. Possuem diversos tipos de apelidos, são eles: regproc, regprocedure, regoper, regoperator, regclass e regtype. Os OID's possuem seu tamanho implementado como um inteiro de 4 bytes sem sinal. A melhor forma para sua utilização é em referências a sistemas de tabelas. O tipo do OID tem poucas operações além da comparação. Pode ser moldado como um inteiro, e então ser manipulado usando os operadores padrões do inteiro.

4.3. MySQL

De acordo com o site http://dev.mysql.com/doc/refman/4.1/pt/what-is.html, o MySQL é um SGBD relacional de código aberto, rápido, subjetivo, e fácil de usar. Foi desenvolvido originalmente para trabalhar com bancos de dados muito grandes de maneira muito mais rápida que as soluções existentes e tem sido usado em ambientes de produção de alta demanda de maneira bem sucedida. A conectividade, a velocidade e a segurança fazem com que o MySQL seja altamente adaptável para acessar bancos de dados na Internet.

O MySQL também possui compatibilidade com o padrão SQL92/SQL99, porém em alguns casos o MySQL realiza operações de forma diferente, como por exemplo para campos VARCHAR, espaços extras são removidos quando o valor é armazenado.

Abaixo estão alguns aspectos do MySQL descritos no site do MySQL (http://dev.mysql.com/doc/refman/4.1/pt/features.html), presentes em sua Versão 4.1:

- Escrito em C e C++;
- Funciona em várias plataformas: Linux, Windows, Solaris entre outras;
- Suporte total a *multi-threads* usando *threads* diretamente no *kernel*, ou seja, caso haja necessidade pode-se usar múltiplas *CPU*s;
- Forrnece mecanismos de armazenamento transacional e não transacional;
- Um sistema de alocação de memória muito rápido e baseado em processo(*thread*);
- *Joins* muito rápidas usando uma *multi-join* de leitura única otimizada;
- Pode-se misturar tabelas de bancos de dados diferentes na mesma pesquisa.

4.3.1. Características do MySQL

Segundo o site do MySQL (http://dev.mysql.com/doc/refman/4.1/pt/table-size.html), a partir da Versão 3.23 o tamanho máximo para as tabelas do MySQL foi expandido para até 8 milhões de Terabytes. Com este tamanho de tabela maior permitido, o tamanho máximo efetivo das tabelas para o banco de dados MySQL é normalmente limitado pelas restrições do sistema operacional somente e não mais por limites internos do MySQL. Além disso, é possível utilizar se for o caso, ferramentas para compactar tabelas de tamanho muito grande, utilizadas somente para leitura.

Em relação ao número de índices, são permitidos até 32 índices por tabela, sendo cada índice de 1 a 16 colunas (ou partes de colunas). O tamanho máximo para cada item este índice é de 500 bytes, podendo ser alterado este valor de acordo com a compilação do MySQL.

4.3.2. Tipos de dados suportados pelo MySQL

O MySQL disponibiliza uma certa quantidade de tipos de campos que podem ser agrupados em três categorias: tipos numéricos, tipos de data e hora, e tipos *string* (caracteres). Os principais tipos disponíveis na sua Versão 4.1 encontram-se na Tabela 4.4.

Tabela 4.4 – Tipos de dados suportados pelo MySQL

Nome	Intervalo/Precisão	Descrição
TINYINT	Com sinal: de -128 até 127. Sem sinal: de 0 até 255	Um inteiro muito pequeno
BIT, BOOL, BOOLEAN	0 para falso e 1 para verdadeiro	Estes são sinônimos para TINYINT(1).
SMALLINT	Com sinal: -32768 até 32767. Sem sinal é de 0 a 65535.	Um inteiro pequeno
MEDIUMINT	Com sinal: de -8388608 a 8388607. Sem sinal é de 0 to 16777215	Um inteiro de tamanho médio
INT	Com sinal: de -2147483648 a 2147483647. Sem sinal: de 0 a 4294967295	Um inteiro de tamanho normal.
INTEGER		Este é um sinônimo para INT.
BIGINT	Com sinal: de -9223372036854775808 a 9223372036854775807. Sem sinal: de 0 a 18446744073709551615.	Um inteiro grande
FLOAT(M,D)	Entre -3.402823466E+38 e -1.175494351E-38, 0 e entre 1.175494351E-38 e 3.402823466E+38	Um número de ponto flutuante pequeno (precisão simples).
DOUBLE(M,D)	Entre -1.7976931348623157E+308 e -2.2250738585072014E-308, 0 e entre 2.2250738585072014E-308 e 1.7976931348623157E+308	Um número de ponto flutuante de tamanho normal (dupla-precisão)
DOUBLE PRECISION(M,D), REAL(M,D)		Estes são sinônimos para DOUBLE
DECIMAL(M,[D])	A faixa máxima do valor DECIMAL é a mesma do DOUBLE	Um número de ponto flutuante não empacotado (se comporta como um char)
DEC(M,[D]), NUMERIC(M,[D]), FIXED(M,[D])		Estes são sinônimos para DECIMAL
DATE	Entre '1000-01-01' e '9999-12-31'	Uma data no formato: 'AAAA-MM-DD'
DATETIME	Entre '1000-01-01 00:00:00' e '9999-12-31 23:59:59'	Um combinação de hora e data. Formato: 'AAAA-MM-DD HH:MM:SS'
TIMESTAMP	Entre '1970-01-01 00:00:00' e algum momento no ano 2037	Um timestamp
TIME	Entre '-838:59:59' e '838:59:59'	Uma hora no formato 'HH:MM:SS'
YEAR(2|4)	entre 1901 e 2155 para 4 digitos e 1970-2069 para 2 dígitos	Um ano no formato de 2 ou 4 digitos (padrão são 4 digitos).
CHAR	1 a 255 caracteres	Uma string de tamanho fixo
BIT, BOOL, CHAR		Sinônimo de char(1)
VARCHAR	1 a 255 caracteres. Se o tamanho especificado é maior que 255, o tipo de coluna é convertido para TEXT	Uma string de tamanho variável.
TINYBLOB, TINYTEXT	Tamanho máximo de 255 caracteres	Um campo BLOB ou TEXT
BLOB, TEXT	Tamanho máximo de 65535 caracteres	Um campo BLOB ou TEXT
MEDIUMBLOB, MEDIUMTEXT	Tamanho máximo de 16777215 caracteres	Um campo BLOB ou TEXT
LONGBLOB, LONGTEXT	Tamanho máximo de 4294967295 caracteres ou 4Gbytes	Um campo BLOB ou TEXT
ENUM('valor1','valor2',...)	Tamanho máximo de 65535 valores diferentes	Um objeto string que só pode ter um valor, selecionado da lista de valores
SET('valor1','valor2',...)	Até 64 membros	Um conjunto. Um objeto string que pode ter zero ou mais valores

4.4. Linguagem de Programação Delphi

Segundo Blue, Kaster, Lief e Scott (1997), Delphi é uma linguagem de programação baseada na linguagem Pascal por objetos (Object Pascal), que não é projetada especificamente para a programação de aplicativos de banco de dados. Entretanto, os componentes destinados a esses aplicativos disponibilizados pelo Delphi compensam a generalidade da linguagem Pascal. De fato, se for colocado simplesmente alguns poucos ícones de acesso aos dados na forma de um aplicativo, é possível criar aplicativos de banco de dados totalmente funcionais, embora simples, sem escrever sequer uma linha de código.

No Delphi, os aplicativos de banco de dados são criados combinando-se vários componentes visuais e não visuais. Esses componentes são conectados entre si através das suas propriedades *DataSource* ou *DataSet*. Todos os componentes de controle de dados no Delphi são projetados para ser *data-aware* (dados atualizados), isto é, eles exibem e atualizam os dados contidos em uma tabela.

O componente *DataSource* determina a origem dos dados a serem acessados. O componente *DataSet* é uma coleção de dados determinados por um componente *TTable*, *TQuery* ou *TStoreProc*. O *TTable* é usado nas tabelas do Paradox e do dBase, e inclui cada linha no conjunto de dados. O *TQuery* é usado para fazer uma seleção de registros de uma tabela, geralmente a partir de declarações SQL (*Structured Query Language*). Esses *DataSources* e *DataSets* podem ser passados diretamente para o BDE (*Borland Database Engine*), onde serão processados com operações que não são suportadas diretamente pelos componentes que os contêm.

O BDE (*Borland Database Engine*) do Delphi é o sistema que permite conectar o Delphi aos produtos Paradox e dBase da Borland. Consiste de um conjunto de chamadas de funções DLL que gerencia todo o tratamento de baixo nível dos dados e índices dentro do aplicativo Delphi. Como o BDE é parte integrante do Delphi, pode-se criar aplicativos de

banco de dados sem conhecer qualquer coisa a seu respeito.

Mesmo o Delphi possuindo todos estes recursos para tratamento de arquivos dBase e seus índices, ele não possui suporte para tratamento de arquivos nativos do Clipper, que apesar de serem os mesmos tipos de arquivos do dBase (DBF), possuem índices diferentes - em Dbase: arquivos .NDX e .MDX e em Clipper: arquivos .NTX.

Para solucionar este problema, foi criado pelo russo Vlad Karpov o componente terceirizado do Delphi chamado VKDBFNTX – Versão 1.0.7 disponível em http://vlad-karpov.narod.ru/Components.html, que é um componente gratuito para a utilização do Delphi com tabelas nativas do Clipper, utilizando seus índices. Possui em suas propriedades a opção para tratamento e atualização dos índices durante a utilização de tabelas Clipper. Assim, cada manutenção feita no arquivo de dados será refletida em seu índice.

4.5. Trabalhos correlatos

É muito importante mencionar que existem trabalhos correlatos a este, desenvolvidos em anos anteriores e que foram utilizados como fonte de pesquisa e conhecimento, um deles é intitulado “Ferramenta de Manutenção para Banco de Dados”, e foi apresentado nesta instituição pelo sr. Fagner Christian Paes, no ano de 2004, que propôs uma ferramenta para conversão de tabelas e replicação de dados entre Sistemas Gerenciadores de Banco de Dados (Interbase, MySQL e PostgreSQL) além da criação de Backup / Restore nestes SGBD´s.

5. FERRAMENTA PARA CONVERSÃO, REPLICAÇÃO E ATUALIZAÇÃO DE DADOS

Neste capítulo será descrita a implementação da ferramenta desenvolvida para a manutenção dos dados e criação das tabelas, bem como os métodos de acesso utilizados nos SGBD´s descritos anteriormente.

5.1. Definição da Ferramenta

A Figura 5.1 apresenta o diagrama de funcionamento da ferramenta desenvolvida.

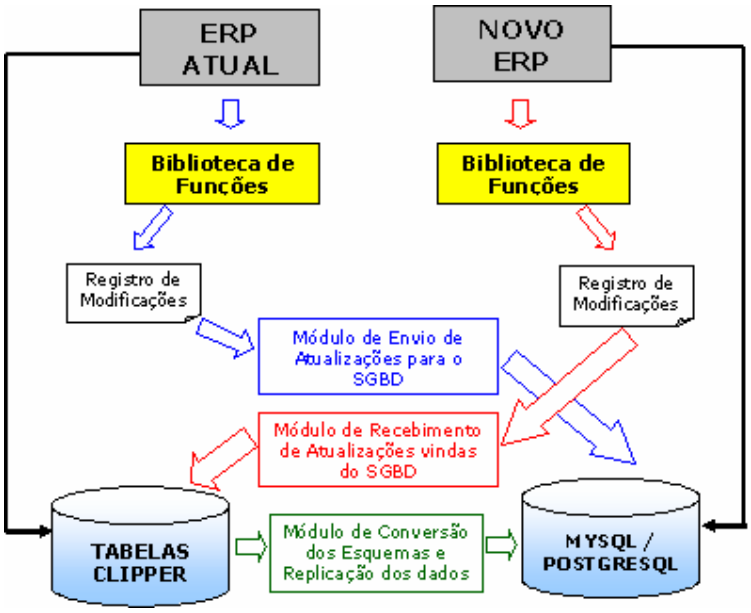

Figura 5.1 – Diagrama de funcionamento da Ferramenta

A ferramenta consiste de um mecanismo que possibilita a conversão de tabelas criadas no formato nativo do Clipper (DBF) para tabelas no formato de um dos SGBDs citados anteriormente. A ferramenta é responsável pela criação da estrutura da tabela,

alimentação inicial desta tabela e atualização das modificações realizadas nos dados desta tabela. Esta modificação deverá ser atualizada nos dois sentidos, tanto do lado nativo do Clipper quanto do lado do SGBD. São aceitas operações de inserção, atualização e / ou remoção dos dados na tabela, ficando a cargo das aplicações que executam estas operações na tabela a notificação da operação realizada para a ferramenta desenvolvida, através do uso de bibliotecas de funções pré-definidas.

Para uma melhor compreensão, são descritas as fases de desenvolvimento da ferramenta que basicamente compreende três aspectos: (1) Estudos dos tipos de dados nativos do Clipper e dos SGBDs, objetivando encontrar compatibilidade entre os mesmos; (2) Estudo das estruturas de dicionário de dados, apresentando o código confeccionado para conversão e replicação da base de dados e (3) Estudo de métodos para atualização de dados em Servidores de Banco de Dados diferentes.

5.2. Comparação entre tipos de dados suportados

A comparação entre os tipos de dados nativos do Clipper e do MySQL e PostgreSQL foi feita através da análise das Tabelas 4.1, 4.3 e 4.4 do capítulo anterior. A semelhança entre os tipos é muito importante para que não ocorra perda na integridade de dados.

Considerando que inicialmente a origem dos dados são tabelas nativas do Clipper, foi suficiente comparar apenas os tipos de dados existentes no Clipper com os tipos de dados compatíveis presentes no MySQL e no PostgreSQL.

Nas Tabelas 5.1 e 5.2 são mostradas as comparações realizadas entre Clipper/PostgreSQL e Clipper/MySQL, respectivamente. De acordo com estas informações foi implementada a conversão de tabelas nativas do Clipper para o formato do SGBD.

Tabela 5.1 – Comparação entre Clipper e PostgreSQL

Clipper	Intervalo	PostgreSQL
CHAR	Máximo de 1	CHAR
CHAR	Acima de 1 caracter	VARCHAR
NUMERIC	-	NUMERIC
LOGIC	-	BOOLEAN
MEMO	-	TEXT
DATE	-	DATE

Tabela 5.2 – Comparação entre Clipper e MySQL

Clipper	Intervalo	MySQL
CHAR	Máximo de 1	CHAR
CHAR	Acima de 1 caracter	VARCHAR
NUMERIC	-	NUMERIC
LOGIC	-	BOOLEAN
MEMO	-	TEXT
DATE	-	DATE

Nota-se que os tipos de dados utilizados tanto no PostgreSQL quanto no MySQL são os mesmos, variando-se apenas no valor armazenado para tipos lógicos no MySQL, sendo que este é um tipo de dado sinônimo de TINYINT, ou seja, quando verdadeiro, seu valor é igual a 1 no lugar de TRUE (utilizado no PostgreSQL), quando falso seu valor é igual a 0 em vez de FALSE.

5.3. Conversão de tabelas da base de dados Clipper

Nesta etapa foram pesquisadas rotinas a respeito do armazenamento dos metadados do Clipper, sendo desenvolvidos programas para acesso a estas estruturas de dados, a fim de que permitissem sua replicação.

Na Tabela 5.3 são descritos os métodos necessários para a obtenção do metadados de tabelas nativas do Clipper, utilizando o componente VKDBFNTX do Delphi 7.

Tabela 5.3 – Métodos utilizados para obtenção de informações do metadados do Clipper

Chamada do Método	Finalidade
TVKDBFNTX.DBFFieldDefs.Items[posição].Name	Retorna o nome do campo posicionado
TVKDBFNTX.DBFFieldDefs.Items[posição].len	Retorna o tamanho do campo posicionado
TVKDBFNTX.DBFFieldDefs.Items[posição].field_type	Retorna o tipo do campo posicionado, onde 'D' para datas, 'C' para caracter, 'L' para lógico, 'N' para numérico, 'M' para campo memo
TVKDBFNTX.Orders[ordem de abertura].KeyExpression	Retorna a expressão do índice posicionado

Por não possuir uma informação sobre chave primária definida em seus metadados, o fornecimento desta informação foi deixado a cargo do usuário, ou seja, quando o usuário seleciona o arquivo origem (nativo do Clipper), deve informar os índices utilizados para ele e definir um deles como sendo o portador da chave primária. A partir desta informação são acessados os dados referentes à expressão chave deste índice utilizando o método *KeyExpression*. A expressão chave, porém, pode possuir, além do nome dos campos utilizados na chave algumas palavras reservadas do Clipper, como por exemplo, a função STR que converte um determinado campo numérico em *string*. A Tabela 5.4 fornece informações sobre palavras reservadas que possivelmente podem estar presentes na expressão chave de um arquivo de índice nativo do Clipper.

Tabela 5.4 – Palavras reservadas presentes em uma expressão chave de um índice Clipper

Palavra reservada	Finalidade
DTOS(Nome do Campo)	Converte um campo do tipo data em string no formato AAAAMMDD
DTOC(Nome do Campo)	Converte um campo do tipo data em caracter no formato de data do computador (normalmente DD/MM/AAAA)
CTOD(Nome do Campo)	Converte um campo do tipo caracter em tipo data
VAL(Nome do Campo)	Converte um campo do tipo caracter em tipo numérico
STRING(Nome do Campo, Tamanho, [Casas Decimais])	Converte um campo do tipo numérico em tipo caracter adicionando espaços à esquerda
STR(Nome do Campo, Tamanho, [Casas Decimais])	Sinônimo de STRING
STRZ(Nome do Campo, Tamanho, [Casas Decimais])	Converte um campo do tipo numérico em tipo caracter adicionando zeros à esquerda
SUBSTR (Nome do Campo, Posição Inicial, Tamanho)	Extrai um número de caracteres (Tamanho) de um campo do tipo caracter à partir de uma posição inicial
SUBS	Sinônimo de SUBSTRING

Para tratamento das palavras presentes na Tabela 5.4 foi criado um algoritmo de eliminação de palavras reservadas, no qual, a partir de uma expressão chave consumia-se a palavra reservada (caso existisse) de acordo com a sua regra de produção.

O código apresentado na Figura 5.2 apresenta a implementação deste algoritmo. É interessante ressaltar que para a criação de um índice único no SGBD não é necessário o uso destas palavras reservadas (*str, dtos, val, etc*), portanto, é necessário que elas sejam descartadas.

O código fonte completo para esta operação pode ser verificado nos anexos apresentados ao final deste trabalho.

```
StrSQLIndex := 'ALTER TABLE '+LowerCase(NomeTabela)+' ADD PRIMARY KEY (';
StrIndex := VKDBFNTX.Orders[0].KeyExpresion;
nI := 1;
while (nI <= length(StrIndex)) do
begin
   Auxiliar := '';
   while ((copy(StrIndex, nI, 1) <> '+') and (nI <= length(StrIndex))) do
   begin
      Auxiliar := Auxiliar + copy(StrIndex, nI, 1);
      nI := nI + 1;
   end;
   nJ := 1;
   NomeCampo := '';
   //Para DTOS, DTOC, CTOD e VAL ignorou-se o ultimo parenteses, copiando
   //o string até a penultima posicao, ignorando a ultima
   if ((uppercase(copy(Auxiliar,1,4)) = 'DTOS') or
       (uppercase(copy(Auxiliar,1,4)) = 'DTOC') or
       (uppercase(copy(Auxiliar,1,4)) = 'CTOD')) then
      //Copia até a penultima posição
      NomeCampo := copy(Auxiliar, 6, (length(Auxiliar) - 6))
   else if (uppercase(copy(Auxiliar,1,3)) = 'VAL') then
      //Copia até a penultima posição
      NomeCampo := copy(Auxiliar, 5, (length(Auxiliar) - 5))
   else if (uppercase(copy(Auxiliar,1,3)) = 'STR') or
           (uppercase(copy(Auxiliar,1,6)) = 'STRING') then
   begin
      if (uppercase(copy(Auxiliar,1,3)) = 'STR') then
         Auxiliar := copy(Auxiliar, 5, (length(Auxiliar) - 4))
      else
         Auxiliar := copy(Auxiliar, 8,(length(Auxiliar) - 7));
         //Copia até o virgula (vem logo após o nome do campo)
      while (copy(Auxiliar, nJ, 1) <> ',') do
      begin
         NomeCampo := NomeCampo + copy(Auxiliar, nJ, 1);
         nJ := nJ + 1;
      end;
   end
   else if (uppercase(copy(Auxiliar,1,4)) = 'STRZ') then
   begin
      Auxiliar := copy(Auxiliar, 6, (length(Auxiliar) - 5));
      //Copia até o virgula (vem logo após o nome do campo)
      while (copy(Auxiliar, nJ, 1) <> ',') do
      begin
         NomeCampo := NomeCampo + copy(Auxiliar, nJ, 1);
         nJ := nJ + 1;
      end;
   end
   else if (uppercase(copy(Auxiliar,1,4)) = 'SUBS') or
           (uppercase(copy(Auxiliar,1,9)) = 'SUBSTRING') then
   begin
      if (uppercase(copy(Auxiliar,1,4)) = 'SUBS') then
         Auxiliar := copy(Auxiliar, 6, (length(Auxiliar) - 5))
      else
         Auxiliar := copy(Auxiliar, 11,(length(Auxiliar) - 10));
      //Copia até o virgula (vem logo após o nome do campo)
      while (copy(Auxiliar, nJ, 1) <> ',') do
      begin
         NomeCampo := NomeCampo + copy(Auxiliar, nJ, 1);
         nJ := nJ + 1;
      end;
   end
   else
      NomeCampo := Auxiliar;
   SetLength(aCamposChave, length(aCamposChave) + 1);
   aCamposChave[length(aCamposChave) - 1] := lowercase(NomeCampo);
   StrSQLIndex := StrSQLIndex + lowercase(NomeCampo) + ', ';
   StrCampoChave := StrCampoChave + lowercase(NomeCampo) + ';';
   nI := nI + 1;
end;

StrSQLIndex := Copy(StrSQLIndex,1,length(StrSQLIndex)-2);
StrSQLIndex := StrSQLIndex + ')';
StrCampoChave := Copy(StrCampoChave,1,length(StrCampoChave)-1);
```

Figura 5.2 – Código fonte para tratamento de palavras reservadas presentes em expressão chave do clipper

5.4. Confecção de algoritmos para conversão das estruturas

Após conhecer os metadados do Clipper e obter todas as informações referentes aos campos de suas tabelas é possível realizar a conversão da estrutura de uma tabela Clipper para uma tabela de um SGBD, de acordo com o sistema escolhido. Esta tabela é criada a partir da geração de um script DDL, ou seja, será gerado um conjunto de comandos SQL para que, quando executado, crie uma tabela no SGBD com a mesma estrutura da tabela Clipper respeitando a compatibilidade entre os campos.

Como visto anteriormente, as variações na criação de uma estrutura para o MySQL ou para o PostgreSQL são mínimas, o que trouxe uma conveniência muito grande na elaboração do algoritmo.

O programa apresentado na Figura 5.3 ilustra melhor o que está sendo relatado. É possível notar que o comando SQL para a criação da tabela é o mesmo, tendo em vista a compatibilidade dos campos na nomenclatura do tipo de dados utilizado.

```
StrSQLTabela := 'CREATE TABLE ' + NomeTabela + ' (';
for nI := 0 to VKDBFNTX.DBFFieldDefs.Count - 1 do
begin
   NomeCampo := VKDBFNTX.DBFFieldDefs.Items[nI].Name;
   StrSQLTabela := StrSQLTabela + lowercase(NomeCampo) + ' ';
   if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'C') then
   begin
      if (VKDBFNTX.DBFFieldDefs.Items[nI].len = 1) then
         StrSQLTabela := StrSQLTabela + ' char(1)'
      else
         StrSQLTabela := StrSQLTabela + ' varchar(' +
         InttoStr(VKDBFNTX.DBFFieldDefs.Items[nI].len) + ')';
   end
   else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'D') then
      StrSQLTabela := StrSQLTabela + ' date'
   else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'N') then
      StrSQLTabela := StrSQLTabela + ' numeric(' +
      InttoStr(VKDBFNTX.DBFFieldDefs.Items[nI].len) + ',' +
      InttoStr(VKDBFNTX.DBFFieldDefs.Items[nI].dec)  + ')'
   else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'L') then
      StrSQLTabela := StrSQLTabela + ' boolean'
   else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'M') then
      StrSQLTabela := StrSQLTabela + ' text'
   else
   begin
      MessageDlg('ERRO CRIANDO TABELA: ' + NomeTabela + #13 + #13 +
      ' Tipo de dado inválido para o campo: '+NomeCampo, mtError, [mbOk], 0);
   end;

   lAchou := false;
   for nJ := 0 to length(aCamposChave) - 1 do
   begin
      if (lowercase(NomeCampo) = lowercase(aCamposChave[nJ])) then
         lAchou := true;
   end;
   if (lAchou = true) then
      StrSQLTabela := StrSQLTabela + ' NOT NULL, '
   else
      StrSQLTabela := StrSQLTabela + ', ';
end;
StrSQLTabela := Copy(StrSQLTabela,1,length(StrSQLTabela)-2);
StrSQLTabela := StrSQLTabela + ')';
```

Figura 5.3 – Código fonte para criação de tabelas no SGBD à partir de tabelas do Clipper

O resultado da execução do código apresentado na Figura 5.3 pode ser verificado na Figura 5.4. O script SQL gerado é uma tabela extraída do Clipper e convertida para o MySQL. Como dito anteriormente, devido à compatibilidade entre os tipos de dados o mesmo script seria gerado para o PostgreSQL.

```
CREATE TABLE pedido (
numero       numeric(10,0) NOT NULL,
datapedido   date,
datadigita   date,
codcliente   numeric(5,0),
diflocentr   boolean,
codmovto     numeric(2,0),
cfop         numeric(4,0),
codrepres    numeric(3,0),
comissao     numeric(6,2),
datavenc1    date,
datavenc2    date,
datavenc3    date,
qtdedias1    numeric(3,0),
qtdedias2    numeric(3,0),
qtdedias3    numeric(3,0),
tipocobran   numeric(2,0),
datasaida    date,
unidatend    numeric(2,0),
situacao     numeric(1,0),
observacao   text,
pedestado    varchar(2),
pedregiao    numeric(2,0),
pedmicro     numeric(2,0),
dataentreg   date,
unidcadast   numeric(2,0));
ALTER TABLE pedido ADD PRIMARY KEY (numero);
```

Figura 5.4 – Exemplo de Script SQL de criação de tabela gerado pela ferramenta desenvolvida

5.5. Confecção de algoritmos para a Replicação dos Dados

Após a conversão da estrutura da tabela, o próximo passo é a replicação dos dados. Nesta fase os registros são selecionados e copiados a partir da base de dados do Clipper para a nova tabela no SGBD escolhido.

É interessante ressaltar que uma característica do Clipper quanto à remoção de seus registros é que ao serem eliminados da base de dados, estes apenas recebem um atributo de *situação*, não sendo removidos fisicamente da base de dados, podendo ser descartados, se necessário.

Para que seja possível evitar a duplicação de chaves, registros com chaves primárias repetidas são descartados através da atribuição do valor *true* à propriedade *SetDeleted* do componente *VKDBFNTX*, ou seja*, TKVDBFNTX.SetDeleted := true;*

Na Figura 5.5 é apresentado um trecho de código para a replicação dos dados entre o Clipper e o MySQL e entre o Clipper e o PostgreSQL. Devido à compatibilidade fornecida pelo componente Zeos Library, a criação do script SQL se torna mais prática, uma vez que o controle da conversão é passado para o componente através da utilização de parâmetros na criação do script SQL.

O código fonte completo para esta operação pode ser verificado nos anexos apresentados no final deste trabalho.

```
VKDBFNTX := TVKDBFNTX.Create(self);
VKDBFNTX.SetDeleted := true;
Prosseguir := AbreDbf(VKDBFNTX, LabelDbf.Caption, aIndices, 0);
if (Prosseguir = true) then
begin
   Conectou := ConectaBanco(ConexaoBanco);
   if (Conectou = true) then
   begin
      PopulouRegistro := true;
      NomeTabela := RetNomeTabela(LabelDbf.Caption);
      StrSQLCampos := 'INSERT INTO ' + NomeTabela + '(';
      for nI := 0 to VKDBFNTX.DBFFieldDefs.Count - 1 do
      begin
       StrSQLCampos := StrSQLCampos + lowercase(VKDBFNTX.DBFFieldDefs.Items[nI].Name) + ', ';
      end;
      StrSQLCampos := Copy(StrSQLCampos,1,length(StrSQLCampos)-2);
      StrSQLCampos := StrSQLCampos + ') ';

      StrSQLValores := 'VALUES(';
      for nI := 0 to VKDBFNTX.DBFFieldDefs.Count - 1 do
      begin
        StrSQLValores := StrSQLValores + ':p' + InttoStr(nI) + ', ';
      end;
      StrSQLValores := Copy(StrSQLValores, 1, length(StrSQLValores)-2);
      StrSQLValores := StrSQLValores + ') ';

      //Monta a string de inserção completa
      StrSQLPopula := StrSQLCampos + StrSQLValores;

      ...........


      for nI := 0 to VKDBFNTX.DBFFieldDefs.Count - 1 do
      begin
         if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'C') then
         begin
            if  (length(trim(VKDBFNTX.FieldByName(VKDBFNTX.DBFFieldDefs.Items[nI].Name).Text))
> 0) then
               DMLQuery.Params[nI].Value := VKDBFNTX.Fields[nI].Value
            else
            begin
               DMLQuery.Params[nI].AsString := '';
               if (TipoCaracter = 'NULL') then
                  DMLQuery.Params[nI].Value := NULL;
            end;
         end
         else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'N') then
         begin
            if  (length(trim(VKDBFNTX.FieldByName(VKDBFNTX.DBFFieldDefs.Items[nI].Name).Text))
> 0) then
               DMLQuery.Params[nI].Value := VKDBFNTX.Fields[nI].Value
            else
            begin
               if (VKDBFNTX.DBFFieldDefs.Items[nI].dec > 0) then
                  DMLQuery.Params[nI].AsFloat := 0
               else
                  DMLQuery.Params[nI].AsInteger := 0;
               if (TipoNumerico = 'NULL') then
                  DMLQuery.Params[nI].Value := NULL;
            end;
          end
          else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'D') then

      ...........

   end;
end;
```

Figura 5.5 – Trecho de código para a replicação de dados.

Ainda analisando a Figura 5.5 pode-se verificar que os dados que estão sendo replicados na nova tabela passam previamente por uma filtragem. Este filtro é aplicado como

configuração da ferramenta e foi desenvolvido para customizar a replicação dos campos que não possuem valores definidos na tabela origem (campos vazios). Estes dados podem ser adaptados a um formato do SGBD de acordo com a necessidade do usuário. Por exemplo, caso o usuário deseja que na replicação das tabelas, campos com valores vazios ao invés de serem vazios sejam nulos (NULL), é possível através da configuração mostrada na Figura 5.6.

Figura 5.6 – Tela para configurações de valores Nulos ou Vazios

5.6. Criação de catálogo de tabelas e seus índices

Durante o recebimento das modificações realizadas no SGBD, é necessária a atualização dos índices pertencentes à tabela que sofrerá a modificação. Para que a ferramenta tenha conhecimento destes dados, é feito no ato da criação da tabela no SGBD a geração de um catálogo que contém estas informações.

Após executar o script SQL da criação da tabela, este catálogo é atualizado e

posteriormente utilizado pelo processo de transmissão de modificações.

Na Tabela 5.5 é ilustrada a estrutura que deverá conter este arquivo de catálogo.

Tabela 5.5 – Estrutura do arquivo de catálogos das tabelas criadas no SGBD

Nome do Campo	Tipo / Tamanho	Informação Armazenada
TABELA	C(20)	Nome da Tabela no SGBD
CAMINHODBF	C(80)	Caminho Completo da localização do arquivo de tabela do Clipper
CAMINHONTX	C(80)	Caminho Completo da localização do arquivo de índice do Clipper
CAMPOCHAVE	C(100)	Nome de todos os campos que são chave primária, na ordem da chave, separados por ponto e vírgula
PRIMARYKEY	L	Verdadeiro se o índice em questão é a chave primária

A obrigatoriedade do uso de ponto e vírgula na separação dos nomes dos campos ocorre devido à exigência do componente VKDBFNTX.

O código fonte completo para esta operação se encontra nos anexos apresentados ao final deste trabalho.

5.7. Extraindo modificações realizadas na base de dados Clipper

Nesta etapa, foram desenvolvidos programas para a transmissão de cada modificação realizada na base de dados nativa do Clipper para o SGBD escolhido.

Para que seja realizada a replicação das modificações é necessário manter um registro de todas as alterações efetuadas na base de dados. Isto é possível de modo simples e rápido implementando-se bibliotecas de funções para a geração desta notificação. Esta biblioteca tem como finalidade gravar um novo registro na tabela de notificação para que a ferramenta desenvolvida neste trabalho possa percorrer esta tabela e atualizar o SGBD a partir de informações geradas na base de dados do Clipper.

Esta notificação deverá conter a estrutura mínima sugerida na Tabela 5.6:

Tabela 5.6 – Estrutura da tabela de notificações das modificações realizadas no Clipper

Nome do Campo	Tipo / Tamanho	Informação Armazenada
TABELA	C(20)	Nome da Tabela no SGBD
CHAVE	C(100)	Chave primária do registro modificado na tabela nativa do Clipper, usando ponto e vírgula como separador do conteúdo de cada campo
OPERACAO	C(1)	O tipo de modificação a ser realizada, onde de ser usado: 'U' para Update, 'I' para Insert e 'D' para Delete;
DATATRANS	D	Flag da data da transmissão da informação para o SGBD, se for vazio, a informação encontra-se pendente para transmissão
HORATRANS	C(8)	Hora da transmissão da informação para o SGBD

O penúltimo campo da Tabela 5.6 tem como finalidade informar se o registro já foi transmitido ou encontra-se pendente para transmissão. Este campo pode ser do tipo inteiro, caracter ou data. O tipo escolhido para utilização no desenvolvimento da ferramenta foi o tipo *date*, pois permite ter um controle melhor do "tempo de vida" da notificação.

Outra observação importante a ser feita é que a chave primária deverá ter um caracter separador de uso comum. Ou seja, ao gravar a chave primária do registro modificado na tabela de notificação, esta deverá ter seus valores separados por um caracter comum. Este caracter deve ser de conhecimento de todos os usuários da ferramenta. Para o desenvolvimento deste trabalho foi escolhido o caracter ";" (ponto e vírgula).

A biblioteca de funções deve ser programada na mesma linguagem dos aplicativos que irão acessar os dados nativos do Clipper e deve ser incorporada aos aplicativos que irão fazer a modificação dos dados destas tabelas. Seu algoritmo deve ser implementado da forma como apresentado na Figura 5.7. Neste caso, o algoritmo foi implementado utilizando a linguagem Clipper que, na maioria dos casos, deverá ser a linguagem padrão. Esta função deverá ser chamada sempre que for efetuada a uma modificação na tabela de dados. Devem

ser passados como parâmetros desta função o nome da tabela, a chave primária do registro modificado (separando os campos por ponto e vírgula) e o tipo de operação que foi realizada. Por exemplo, se houver um aplicativo que faz o cadastro de clientes, deve-se alterar o código fonte deste aplicativo para que após a inserção deste cliente na base de dados Clipper ele chame a função "EnviaSGBD" (conforme Figura 5.7) da seguinte forma: EnviaSGBD("cliente", "9551", "I"), onde "cliente" é o nome da tabela, "9551" é o código do cliente e "I" é a operação realizada, que neste caso foi a inserção de um novo cliente.

```
function EnviaSGBD(cTabela, cChave, cOperacao)

while !fuse("\projeto\dbf_sgbd",.f.,"notifica","Aguarde, Arquivo de notificacoes (DBF->SGBD)
nao liberado!")
end
notifica->(dbsetindex("\projeto\dbf_sgbd"))

cTabela := subs(cTabela,1,20)
cChave := subs(cChave,1,100)
cOperacao := subs(cOperacao,1,1)
notifica->(fAppend())
notifica->(trareg())
notifica->tabela := cTabela
notifica->chave := cchave
notifica->operacao := cOperacao
notifica->(dbunlock())

notifica->(dbclosearea())
return
```

Figura 5.7 – Código fonte em Clipper que gerará o registro para envio para o SGBD

Após estas considerações iniciais foi implementado na ferramenta o algoritmo de varredura que deve percorrer a tabela de notificações de forma contínua (*loop*) e, encontrando algum registro novo, deve seguir a rotina de tratamento deste registro de acordo com o tipo de modificação a ser realizada.

O trecho de código apresentado na Figura 5.8 ilustra uma inserção de um novo registro sendo realizada pela ferramenta de replicação. São feitas análises prévias para verificação da autenticidade da informação a ser inserida e após a validação destas informações, o registro é inserido. Havendo algum problema na inserção o usuário é notificado através de mensagens de erro.

```
if (DMLQuerySelect.RecordCount = 0) then
begin
   if (uppercase(ArquivoVala.FieldByName('OPERACAO').Text) = 'I') then
   begin
      StrSQLComando            :=            'INSERT            INTO            '            +
lowercase(trim(ArquivoVala.FieldByName('TABELA').Text)) + ' (';
      for nAux := 0 to (ArquivoOrigem.DBFFieldDefs.Count - 1) do
      begin
         //Pego o nome do campo que vou procurar na tabela resultado
         NomeCampo := lowercase(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name);
         StrSQLComando := StrSQLComando + NomeCampo + ', ';
      end;
      if (length(StrSQLComando) > 0) then
      begin
         StrSQLComando := Copy(StrSQLComando,1,length(StrSQLComando)-2);
         StrSQLComando := StrSQLComando + ') ';
         DMLQueryComando.SQL.Add(StrSQLComando);
      end;

      nCont := 0;
      StrSQLComando := 'VALUES(';
      for nAux := 0 to (ArquivoOrigem.DBFFieldDefs.Count - 1) do
      begin
         StrSQLComando := StrSQLComando + ':p' + InttoStr(nCont) + ', ';
         nCont := nCont + 1;
      end;
      if (length(StrSQLComando) > 0) then
      begin
         StrSQLComando := Copy(StrSQLComando,1,length(StrSQLComando)-2);
         StrSQLComando := StrSQLComando + ')';
         DMLQueryComando.SQL.Add(StrSQLComando);
      end;

      for nAux := 0 to (ArquivoOrigem.DBFFieldDefs.Count - 1) do
      begin
         if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type = 'M') then
         begin
            //É necessario gravar em um TMEMO antes do DBF
            CampoMemo.Clear;
            with CampoMemo.Lines do begin
               Add(ArquivoOrigem.Fields[nAux].Value);
            end;
            if (length(trim(CampoMemo.Text)) > 0) then
               DMLQueryComando.Params[nAux].Assign(CampoMemo.Lines)
            else
            begin
               DMLQueryComando.Params[nAux].AsString := '';
               if (TipoMemo = 'NULL') then
                  DMLQueryComando.Params[nAux].Value := NULL;
            end;
         end
         else
         begin
            if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type = 'C') then
            begin
            .......
            .......
            .......
         end;
      end;
   end;
end;
```

Figura 5.8 – Trecho de código ilustrando a inserção de um registro novo na tabela do SGBD

O código fonte completo para esta operação pode ser verificado nos anexos apresentados no final deste trabalho.

5.8. Extraindo modificações realizadas no SGBD

Nesta etapa foram desenvolvidos programas para que se pudesse realizar a transmissão de modificações realizadas no SGBD para a base de dados nativa do Clipper.

Como feito anteriormente, para que seja realizada a replicação das modificações também é necessário manter um registro de todas as alterações efetuadas no SGBD. Para isto é necessário utilizar também no SGBD uma tabela de notificações das modificações. Esta tabela deve possuir uma estrutura similar à apresentada na Tabela 5.6, porém com algumas modificações devido ao tipo de dados suportado no SGBD. Esta estrutura é apresentada na Tabela 5.7.

Tabela 5.7 – Estrutura da tabela de notificações das modificações realizadas no Destino

Nome do Campo	Tipo / Tamanho	Informação Armazenada
TABELA	VARCHAR(20)	Nome da Tabela no SGBD
CHAVE	VARCHAR(100)	Chave primária do registro modificado na tabela nativa do Clipper, usando ponto e vírgula como separador do conteúdo de cada campo
OPERACAO	CHAR(1)	O tipo de modificação a ser realizada, onde de ser usado: 'U' para Update, 'I' para Insert e 'D' para Delete;
DATATRANS	DATE	Flag da data da transmissão da informação para o SGBD, se for vazio, a informação encontra-se pendente para transmissão
HORATRANS	VARCHAR(8)	Hora da transmissão da informação para o SGBD

Tendo estas informações disponíveis, foi necessário realizar um estudo previamente à confecção de todos os algoritmos. Neste estudo foram levantadas informações a respeito de quais seriam as possíveis opções para executar a gravação de registros no arquivo de notificações no SGBD.

A primeira maneira encontrada seria implementar no algoritmo de criação das tabelas no SGBD a geração automática de gatilhos (*triggers*), que funcionariam como alimentadores da tabela de notificações. Para o usuário da ferramenta esta seria uma maneira bastante

interessante, pois, diferentemente da maneira apresentada anteriormente no envio de modificações do Clipper para o SGBD, não seria mais necessária a confecção de bibliotecas de funções. Porém apesar do SGBD PostgreSQL possuir suporte para triggers, no SGBD MySQL isto não seria possível pois segundo o site http://dev.mysql.com/doc/refman/4.1/pt/ansi-diff-triggers.html, a implementação de *triggers* está programada para ser implementada no MySQL Versão 5.1 e a versão que utilizada neste trabalho foi a 4.1.

Caso fosse possível ainda seriam necessários testes de performance, pois trabalhando com um banco de dados com muitas tabelas este algoritmo poderia ocasionar em lentidão no servidor de banco de dados o que o tornaria inviável.

A outra maneira encontrada é a mesma utilizada pelo envio de modificações da base de dados Clipper para o SGBD, ou seja, utilizando-se de bibliotecas de funções para o envio da informação modificada, porém no sentido contrário (SGBD para a base de dados Clipper).

A biblioteca de funções desta vez deve ser incorporada aos aplicativos que irão fazer a modificação dos dados nas tabelas do SGBD e deverá ter o seu algoritmo implementado em alguma linguagem que possui acesso ao SGBD escolhido, que no caso da Figura 5.9 foi a linguagem PHP. Esta função deve ser chamada sempre que for efetuada uma modificação na tabela de dados do SGBD. Devem ser passados como parâmetros o nome da tabela, a chave primária do registro modificado (separando os campos por ponto e vírgula) e o tipo de operação que foi realizada, idêntico ao exemplo visto anteriormente.

```
<?php

function EnviaClipper($cTabela, $cChave, $cOperacao)
{$parametros = "dbname=dbgranol port=5432 user=postgres password=postgres";

    if (!($con = pg_connect($parametros)))
    {   echo "<p align='center'><big><strong>";
  echo "Não foi possível estabelecer uma conexao com o banco de dados.<br>";
  echo "Favor contactar o Administrador.</strong></big></p>";
       exit;
 }

 $cTabela = substr($cTabela,0,20);
 $cChave = substr($cChave,0,100);
 $cOperacao = substr($cOperacao,0,1);

 $sql = "INSERT INTO sbgd_dbf(tabela, chave, operacao) VALUES('" . $cTabela . "','" . $cChave
. "','" . $cOperacao . "')";
 $resultado = pg_exec($con, $sql);

 return;
}
?>
```

Figura 5.9 – Código fonte em PHP que gera o registro para envio da modificação para a tabela nativa do Clipper

Após estas considerações iniciais foi implementado o algoritmo de varredura que deverá percorrer a tabela de notificações de forma contínua (*loop*) e caso encontre algum registro novo deverá seguir a rotina de tratamento deste registro de acordo com o tipo de modificação a ser realizada.

O trecho de código apresentado na Figura 5.10 ilustra uma inserção de um novo registro sendo realizada pela ferramenta de replicação. São feitas análises prévias para verificação da autenticidade da informação a ser inserida e após a validação destas informações, o registro é inserido. Havendo algum problema na inserção, o usuário é notificado através de mensagens de erro.

```
if (uppercase(DMLQueryTrans.FieldByName('operacao').Text) = 'I') then
begin
   ArquivoRec.Append;
   ArquivoRec.RLock;
   lInsere := true;
end
else
   MessageDlg('Tipo de Operação: ' + uppercase(DMLQueryTrans.FieldByName('operacao').Text) +
' para a Tabela: ' + uppercase(DMLQueryTrans.FieldByName('tabela').Text) +
   #13 + '    Inválida para a chave: ' + DMLQueryTrans.FieldByName('chave').Text +
   #13 + '                 Processo cancelado!                ', mtError, [mbOk], 0);
end;
if (lInsere or lAtualiza) then
begin
   MemoRecebe.Lines.Add('   Inserindo/Atualizando registro...');
   for nAux := 0 to (ArquivoRec.DBFFieldDefs.Count - 1) do
   begin
      //Pego o nome do campo que vou procurar na tabela resultado
      NomeCampo := ArquivoRec.DBFFieldDefs.Items[nAux].Name;
      //Pego a posicao deste campo na tabela resultado
      nPosCampo := ArquivoRec.FieldByName(NomeCampo).FieldNo;
      if (ArquivoRec.DBFFieldDefs.Items[nAux].field_type = 'M') then
      begin
         //É necessario gravar em um TMEMO antes do DBF
         CampoMemo.Clear;
         with CampoMemo.Lines do begin
            Add(DMLQueryRecord.Fields[nPosCampo - 1].Value);
         end;
         ArquivoRec.Fields[nPosCampo - 1].Assign(CampoMemo.Lines);
      end
      else if (ArquivoRec.DBFFieldDefs.Items[nAux].field_type = 'N') then
      begin
         if (DMLQueryRecord.Fields[nAux].Value <> NULL) then
         begin
            if (ArquivoRec.DBFFieldDefs.Items[nAux].dec > 0) then
               ArquivoRec.Fields[nPosCampo - 1].AsFloat := DMLQueryRecord.Fields[nAux].Value
            else
               ArquivoRec.Fields[nPosCampo            -            1].AsInteger            :=
DMLQueryRecord.Fields[nAux].Value;
         end;
      end
      else
      begin
         ArquivoRec.Fields[nPosCampo - 1].Value := DMLQueryRecord.Fields[nAux].Value;
      end;
   end;
   ArquivoRec.RUnLock;
   ArquivoRec.Post;
   FlegaRegistro := true;
end;
```

Figura 5.10 – Trecho de código ilustrando a inserção de um registro novo na tabela do Clipper

O código fonte completo para esta operação pode ser verificado nos anexos apresentados no final deste trabalho.

6. RESULTADOS

Este capítulo apresenta testes realizados na conversão da estrutura, replicação dos dados e no envio e recebimento de modificações realizadas nas tabelas, tanto com o Clipper, quanto com o SGBD.

Na replicação dos dados foi feita uma análise do desempenho e da autenticidade das informações inseridas no SGBD devido à quantidade de campos existentes em uma tabela. O mesmo teste foi realizado no envio dos dados da base de dados nativa do Clipper para o SGBD em questão e no recebimento dos dados das tabelas do SGBD escolhido para a base nativa do Clipper.

Os testes foram realizados em um computador Pentium IV 1.7 Ghz com 192 megabytes de memória RAM e 40 gigabytes de espaço total do disco rígido. Foi utilizado um Modelo Relacional representado na Figura 6.1 para que pudessem ser feitas a conversão das estruturas e a replicação dos dados.

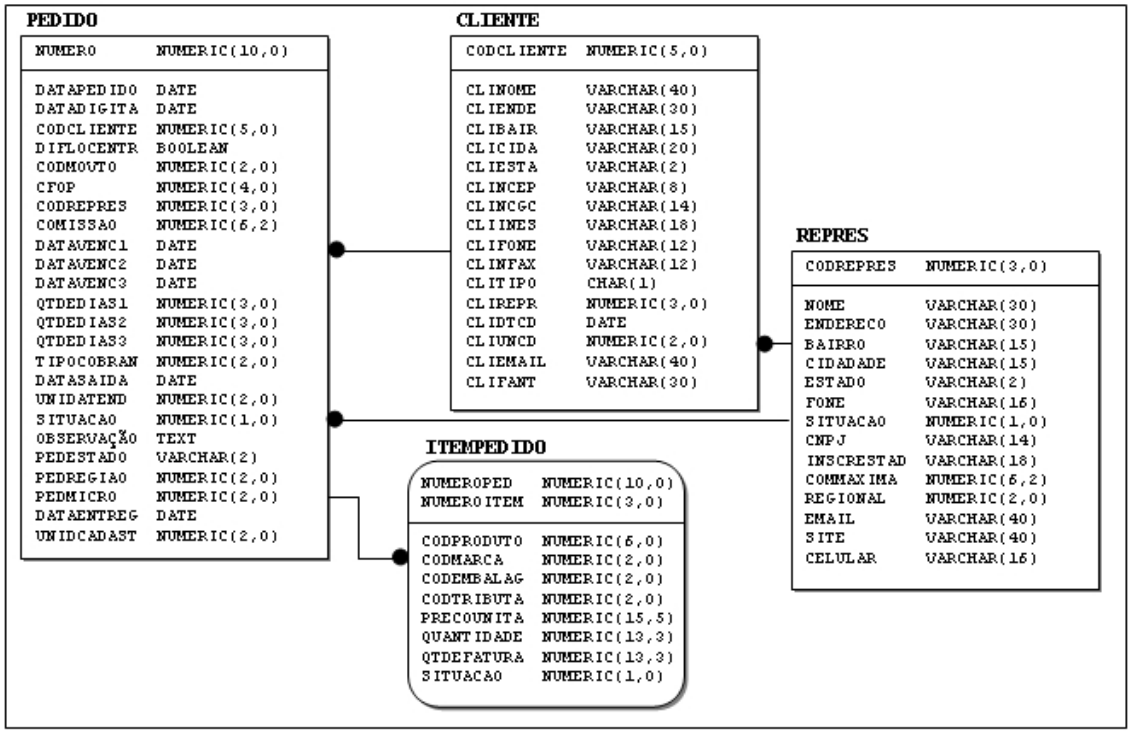

Figura 6.1 – Modelo Relacional dos dados utilizados nos testes

6.1. Funcionamento da ferramenta

Ao iniciar a ferramenta é apresentada a tela principal para escolha das opções disponíveis, ilustrada na Figura 6.2.

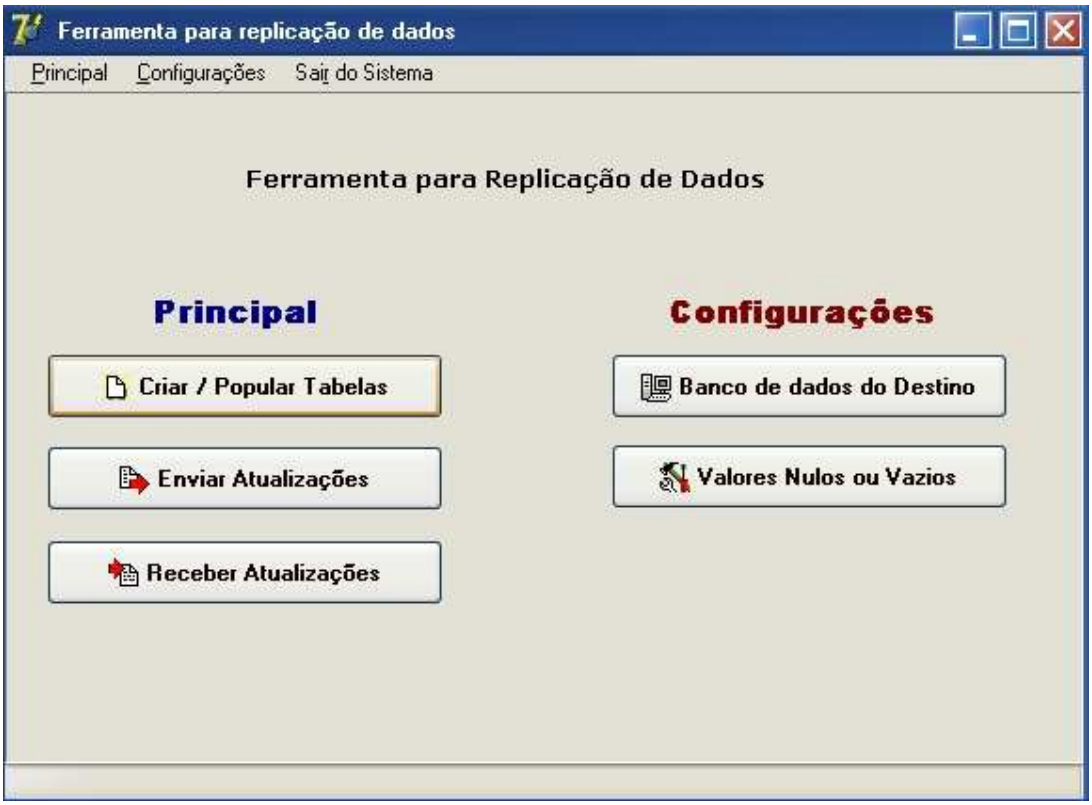

Figura 6.2 – Tela principal da ferramenta desenvolvida

Antes de iniciar o uso da ferramenta pela primeira vez, o usuário deverá escolher a opção de configuração do Banco de dados do destino. As demais opções não estarão disponíveis até que o usuário configure esta opção.

Quando o usuário escolhe esta opção é apresentada a tela mostrada na Figura 6.3, que tem como objetivo definir o tipo de SGBD para o destino das tabelas e de seus conteúdos e os dados necessárias para a conexão com este SGBD (IP, nome do banco de dados, número da porta de acesso, nome do usuário e senha para acesso ao SGBD). Após informados os dados o usuário deverá salvar estas informações e, caso julgue necessário, testar a conexão

entre a aplicação e o SGBD. No caso do SGBD estar localizado na própria máquina da aplicação é necessário informar o endereço IP “127.0.0.1”.

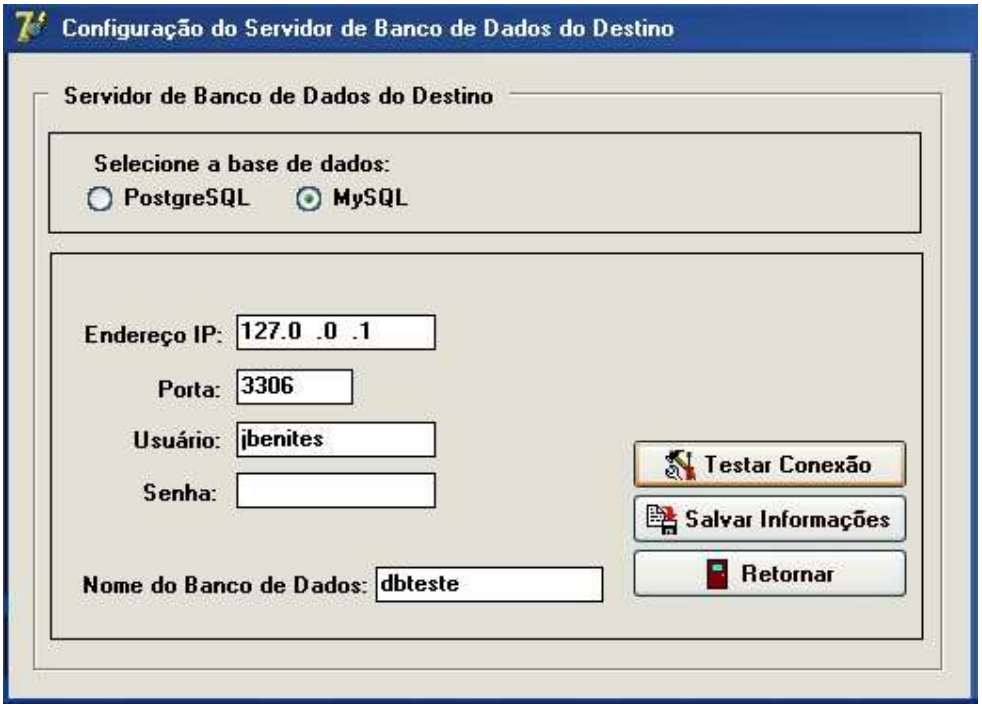

Figura 6.3 – Tela para configuração das informações para conexão com SGBD escolhido

Após a configuração do SGBD é necessário que o usuário faça a customização dos valores que serão replicados no SGBD escolhido. Isto é necessário devido ao fato de não existirem valores nulos na base de dados nativa do Clipper, apenas valor zero para tipos numéricos e vazios para tipos caracteres ou data. Para tipos lógicos, o Clipper assume como padrão o valor *false*. Por este motivo não é possível a customização deste tipo de dado ao usuário. Esta customização é de execução obrigatória na primeira vez em que for utilizar a ferramenta.

Na Figura 6.4 é ilustrada a tela de configuração de valores brancos ou nulos que serão replicados no SGBD.

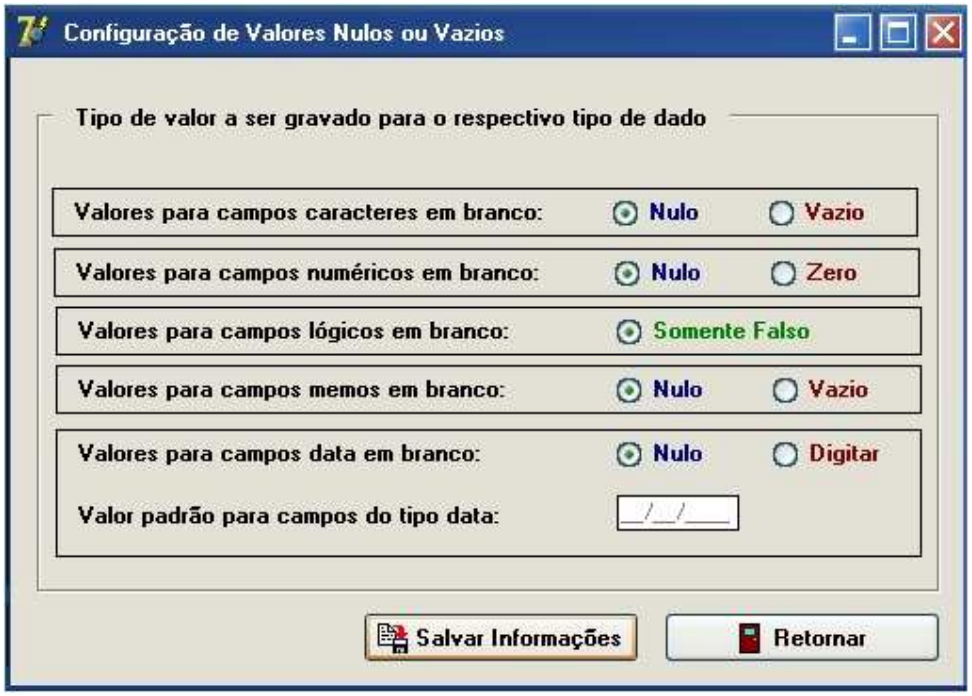

Figura 6.4 – Tela para customização de valores brancos ou nulos

Após execução dos dois passos anteriores o sistema está apto para que as tabelas possam ser replicadas e / ou atualizadas.

As opções de Envio de Atualizações e Recebimento de Atualizações, apesar de não serem necessárias neste momento, estão liberadas para que a ferramenta possa ser executada em máquinas diferentes da que foi feita a conversão da estrutura, devendo apenas anexar à instalação da aplicação o arquivo de catálogo de tabelas.

O próximo passo para que seja feita a criação da tabela e a replicação dos dados é acessar a opção “Criar / Popular Tabelas” da ferramenta.

Será apresentada a tela para criação de tabelas a qual se encontra ilustrada na Figura 6.5.

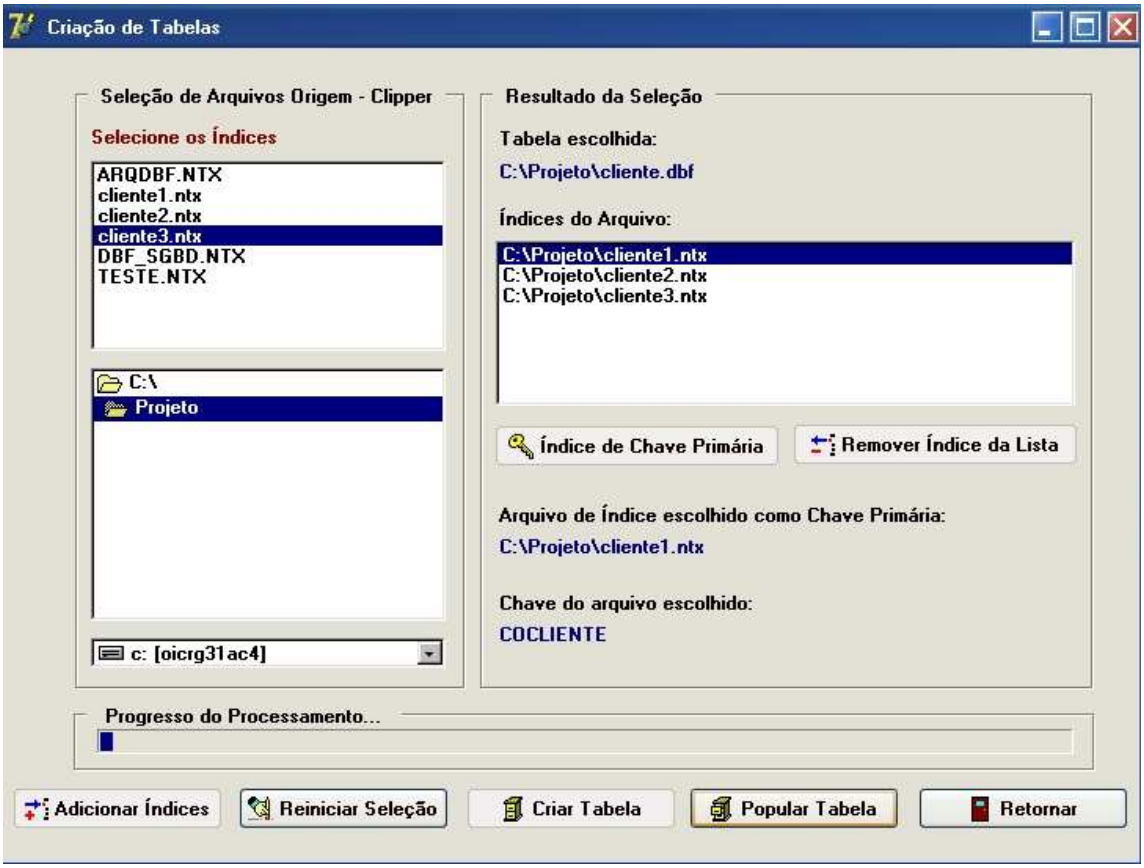

Figura 6.5 – Tela para criação de tabelas e replicação dos dados

Nesta opção o usuário deverá selecionar o arquivo de banco de dados nativo do Clipper que deseja replicar no SGBD (arquivo .DBF). Localizando a opção de *Seleção de arquivos origem - Clipper* (Figura 6.5) deverá ser selecionado o arquivo que corresponde à tabela a ser replicada. Caso seja necessário, o usuário deverá utilizar as opções para navegação que lhe estão disponíveis para acessar o caminho completo do arquivo. Uma vez localizado o arquivo, o usuário deverá clicar duas vezes sobre ele ou selecioná-lo e clicar no botão *Selecionar Tabela* que encontra-se disponível no canto inferior esquerdo da tela. Após esta ação, na mesma opção de navegação, serão disponibilizadas a opção para seleção dos índices pertencentes a este arquivo. Nesta fase de seleção deverão ser adicionados todos os índices pertencentes à tabela que será replicada, não só o que possui a chave primária da tabela, prosseguindo da mesma forma que na seleção de tabelas, só que o botão agora tem seu novo

nome como "*Adicionar Índices"*.

Após este procedimento, o usuário possui a opção de escolher a chave primária ou remover algum índice que possa ter sido adicionado erroneamente. Para escolher a chave primária deve-se selecionar o índice correspondente à chave e clicar no botão *"Índice de Chave Primária"*.

Neste momento o sistema está parametrizado para a criação da tabela no SGBD. Para que isto seja executado pela ferramenta, deverá ser pressionado o botão "*Criar Tabela"*. Será informado ao usuário se ocorreu algum erro na criação. Caso esteja livre de erros (operação ocorreu com sucesso), é liberado ao usuário o acesso ao botão "*Popular Tabela"*, que irá replicar os dados da tabela nativa do Clipper selecionada para o SGBD. O progresso do processamento das operações executadas (tanto criação como replicação) serão informados ao usuário através da barra de "*Progresso do Processamento*", localizada no canto inferior esquerdo.

O usuário ainda possui a opção de reiniciar uma seleção que tenha sido feita de forma incorreta através do uso do botão "*Reiniciar Seleção"*, ou mesmo cancelar a operação através do uso do botão *Retornar*.

Quanto ao envio das atualizações, este será realizado pela ferramenta de forma automática, porém deve ser iniciado pelo usuário através da opção *"Enviar Atualizações"*. Quando executada esta operação, o sistema inicia um *loop* infinito que poderá ser interrompido pelo usuário somente se não estiver efetuando nenhuma ação no momento da interrupção. Este *loop* inicia um processo de varredura tentando localizar informações que foram modificadas na base de dados Clipper e estão pendentes para serem replicadas no SGBD. Esta opção é apresentada na Figura 6.6.

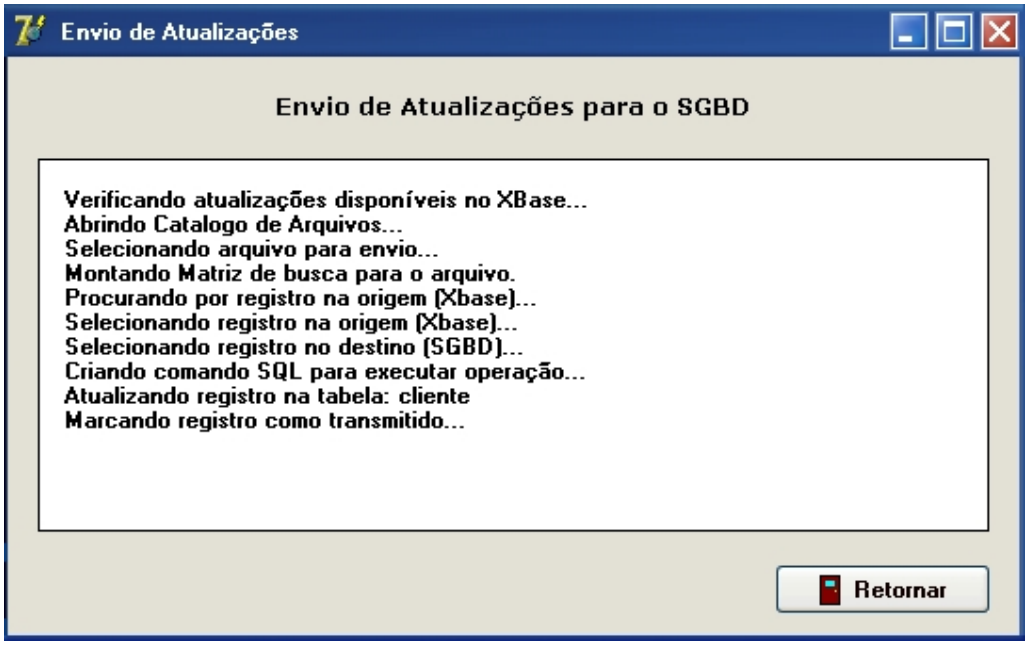

Figura 6.6 – Tela para envio de atualizações para o SGBD

A opção de envio de atualizações deverá ser iniciada e interrompida pelo usuário quando este julgar necessário. A interrupção da execução desta opção é feita através do botão Retornar, que fica desabilitado quando a ferramenta está executando alguma modificação.

A última opção disponível no sistema é o recebimento de atualizações vindas do SGBD. Este recebimento é executado através da opção "*Recebimento de Atualizações*", disponível na tela principal da ferramenta. Funciona de forma semelhante ao descrito para o Envio de Atualizações com a diferença de que são recebidas na base de dados nativa do Clipper as modificações realizadas nas tabelas do SGBD. Conforme apresentada na Figura 6.7.

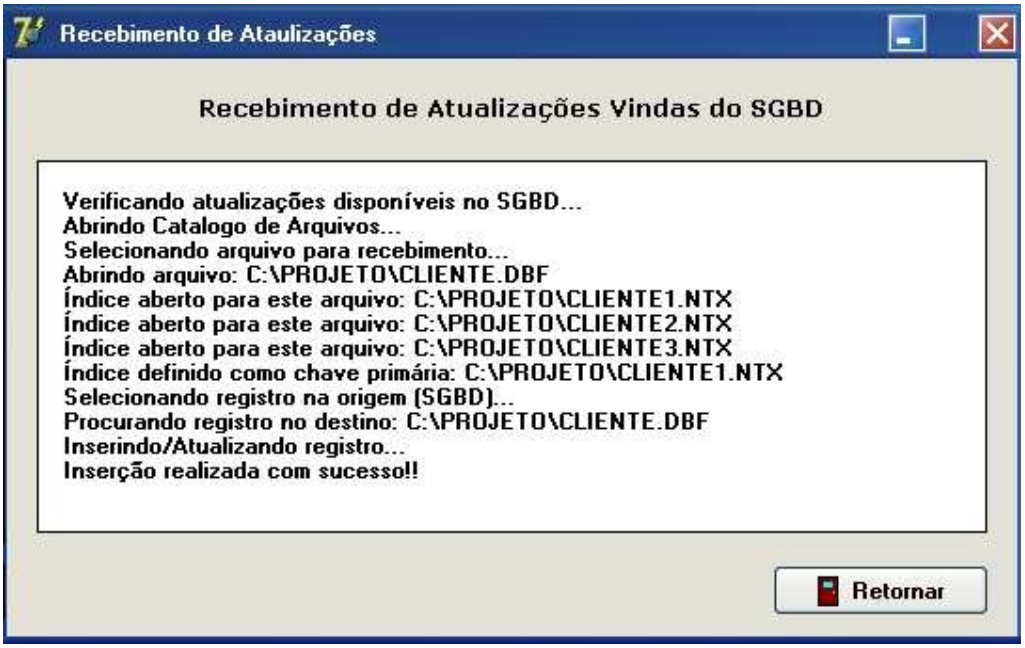

Figura 6.7 – Recebimento de atualizações vindas de um SGBD

6.2. Teste na conversão das Estruturas

Baseando-se no Modelo Relacional apresentado anteriormente na Figura 6.1, foi gerada a partir da tabela **pedido,** um arquivo de dados nativo do Clipper com a estrutura apresentada na Tabela 6.1.

Tabela 6.1 – Estrutura da Tabela no formato nativo do Clipper

Nome do Campo	Tipo
NUMERO	N(10,0)
DATAPEDIDO	D
DATADIGITA	D
CODCLIENTE	N(5,0)
TIPOENTREG	C(1)
CODMOVTO	N(2,0)
CFOP	N(4,0)
CODREPRES	N(3,0)
COMISSAO	N(6,2)
DATAVENC1	D
DATAVENC2	D
DATAVENC3	D
QTDEDIAS1	N(3,0)
QTDEDIAS2	N(3,0)
QTDEDIAS3	N(3,0)

Tabela 6.1 – Estrutura da Tabela no formato nativo do Clipper (continuação)

TIPOCOBRAN	N(2,0)
DATASAIDA	D
UNIDATEND	N(2,0)
SITUACAO	N(1,0)
OBSERVACAO	C(50)
PEDESTADO	C(2)
PEDREGIAO	N(2,0)
PEDMICRO	N(2,0)
DATAENTREG	D
UNIDCADAST	N(2,0)

Nas figuras 6.8 e 6.9 é apresentado o script SQL gerado pela ferramenta para criação da tabela de dados baseando-se na tabela nativa do Clipper "pedido.dbf" (apresentada na tabela 6.1) tanto para MySQL quanto para PostgreSQL.

```
CREATE TABLE pedido (
numero       numeric(10,0) NOT NULL,
datapedido   date,
datadigita   date,
codcliente   numeric(5,0),
diflocentr   boolean,
codmovto     numeric(2,0),
cfop         numeric(4,0),
codrepres    numeric(3,0),
comissao     numeric(6,2),
datavenc1    date,
datavenc2    date,
datavenc3    date,
qtdedias1    numeric(3,0),
qtdedias2    numeric(3,0),
qtdedias3    numeric(3,0),
tipocobran   numeric(2,0),
datasaida    date,
unidatend    numeric(2,0),
situacao     numeric(1,0),
observacao   text,
pedestado    varchar(2),
pedregiao    numeric(2,0),
pedmicro     numeric(2,0),
dataentreg   date,
unidcadast   numeric(2,0));
ALTER TABLE pedido ADD PRIMARY KEY (numero);
```

Figura 6.8 – Script SQL gerado pela ferramenta para criação da tabela no MySQL

```
CREATE TABLE pedido (
numero       numeric(10,0) NOT NULL,
datapedido   date,
datadigita   date,
codcliente   numeric(5,0),
diflocentr   boolean,
codmovto     numeric(2,0),
cfop         numeric(4,0),
codrepres    numeric(3,0),
comissao     numeric(6,2),
datavenc1    date,
datavenc2    date,
datavenc3    date,
qtdedias1    numeric(3,0),
qtdedias2    numeric(3,0),
qtdedias3    numeric(3,0),
tipocobran   numeric(2,0),
datasaida    date,
unidatend    numeric(2,0),
situacao     numeric(1,0),
observacao   text,
pedestado    varchar(2),
pedregiao    numeric(2,0),
pedmicro     numeric(2,0),
dataentreg   date,
unidcadast   numeric(2,0));
ALTER TABLE pedido ADD PRIMARY KEY (numero);
```

Figura 6.9 – Script SQL gerado pela ferramenta para criação da tabela no PostgreSQL

Note que o script gerado para os dois SGBDs são idênticos, ou seja, para os tipos de dados nativos do Clipper não é necessária nenhuma variação para sua conversão tanto em MySQL quanto em PostgreSQL. A única variação ocorrerá quando é inserida ou atualizada alguma informação. Enquanto que o PostgreSQL trabalha com valores *true* e *false* para seus campos do tipo booleano, o MySQL trabalha com valores 1 e 0, por exemplo para campos do mesmo tipo, pois este dado no MySQL foi derivado do tipo TINYINT, ou seja um inteiro sem sinal entre 0 e 255.

6.3. Desempenho na Conversão das Estruturas e Replicação da Base

Na realização dos testes durante a replicação dos dados foi analisado o desempenho e a consistência dos dados replicados nas diferentes bases de dados (SGBDs e Clipper). Para

análise de desempenho foram considerados os números de atributos e a quantidade de registros da tabela.

Para a realização dos testes foram utilizadas as tabelas **Cliente, Pedido e ItemPedido** conforme estrutura presente na Figura 6.1.

Em relação ao teste de desempenho da ferramenta foi analisado o tempo gasto para a replicação dos dados entre a base nativa do Clipper e os dois SGBDs.

É importante para a validação do teste a utilização de tabelas com número variado de colunas e tipos de dados. Os resultados obtidos podem ser verificados nas Figuras 6.10 e 6.11.

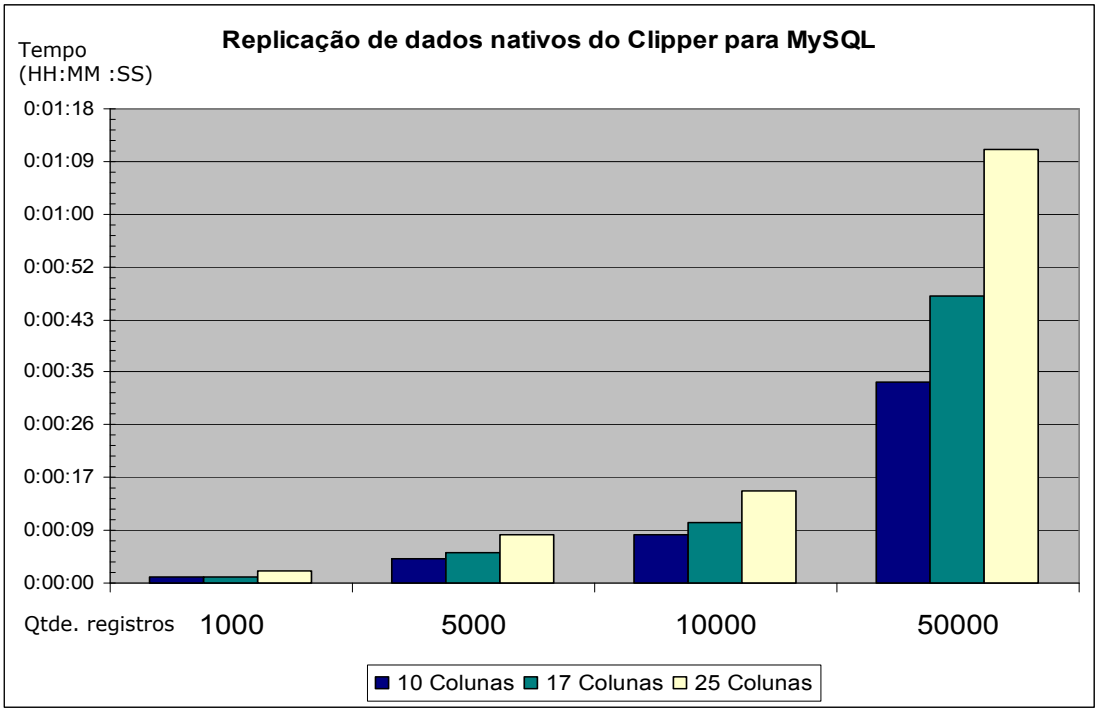

Figura 6.10 – Replicação de dados extraídos da base nativa do Clipper para o MySQL

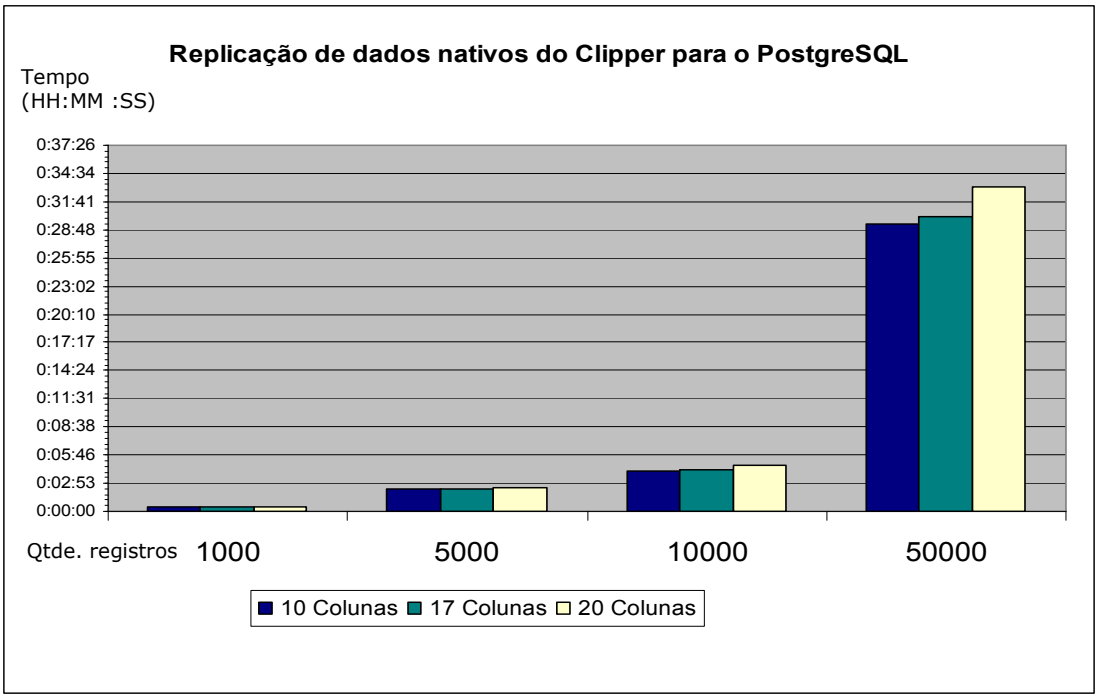

Figura 6.11 – Replicação de dados extraídos da base nativa do Clipper para o PostgreSQL

Após os testes é possível notar que em ambos os SGBDs a quantidade de atributos e o número de registros influenciaram no desempenho da replicação dos dados, no PostgreSQL porém esta influência é mínima. Como exemplo podemos analisar a Figura 6.10 em relação à Figura 6.11 e notar que para 50.000 registros obtivemos os seguintes resultados.

No MySQL, o tempo gasto para a replicação dos dados em uma tabela com 10 colunas foi de 33 segundos, já para uma tabela de 20 colunas foi gasto 1 minuto e 11 segundos, ou seja 115% a mais do tempos gasto com a tabela de 10 colunas.

No PostgreSQL porém, para esta mesma tabela de 10 colunas, o tempo gasto foi de 29 minutos e 25 segundos e para a mesma tabela de 20 colunas usada no MySQL foi gasto 33 minutos e 12 segundos, ou seja, 12,86% a mais do tempos gasto do que com a tabela de 10 colunas.

Analisando o desempenho dos SGBDs, nota-se que para a realização da replicação

dos dados o MySQL teve um desempenho muito superior ao PostgreSQL, chegando a ser, em alguns casos, cerca de 30 vezes mais rápido, isto ocorre pelo motivo do MySQL não possuir integridade referencial.

No envio e recebimento de modificações realizadas nas bases de dados, o desempenho em relação as modificações baseadas na criação e na execução de scripts SQL também foi satisfatório, eliminando o número de mecanismos que seriam necessários e com isto diminuindo consideravelmente o tempo gasto com o processamento da informação, pois a modificação é realiza de uma base de dados diretamente na outra. A variação do desempenho de um SGBD para o outro nesta etapa foi mínima no gasto de tempo geral do processamento. Isto deve ocorrer possivelmente devido ao número único de transação realizada nesta etapa.

Após realizada a replicação dos dados foram analisados alguns registros para verificação de consistência dos dados. Com relação à garantia de consistência, os resultados obtidos foram satisfatórios, pois não ocorreram perda ou corrupção dos dados. Quanto a verificação de campos com valores nulos ou vazios, os resultados obtidos foram os desejados, garantindo a padronização dos dados nulos ou vazios no SGBD de acordo com o desejado pelo usuário.

O teste de consistência dos dados foi realizado também durante o envio e recebimento das modificações realizadas nas bases de dados e como na replicação, através dos resultados obtidos é possível garantir sua consistência e afirmar que os resultados foram satisfatórios.

CONCLUSÃO

De acordo com as etapas propostas para a realização do projeto, concluiu-se que a conversão das estruturas, a replicação dos dados, tanto na criação quanto na modificação ocorreram de forma satisfatória, pois era desejada a execução destas tarefas da forma mais simples e conveniente possível. Sem o uso da ferramenta, a execução destas tarefas é realizada somente através da criação e execução de vários scripts SQL responsáveis pela criação das tabelas e replicação dos dados no SGBD e de vários scripts TXT responsáveis pela replicação dos dados no Clipper, tornando estes trabalhos demorados, complexos e desgastantes.

Utilizando a ferramenta confeccionada esta tarefa se torna mais fácil e é realizada automaticamente e em um período menor, porém o usuário deverá ter um conhecimento básico sobre SGBDs e da base de dados nativa do Clipper, para ser capaz de definir parâmetros necessários para a configuração da ferramenta.

Na realização da conversão de estruturas foi obtida a compatibilidade entre os tipos de dados através de uma análise prévia dos tipos de dados suportados pelos SGBDs e pelo Clipper.

A replicação dos dados, através do uso da ferramenta, teve sua consistência garantida, tendo apenas seu desempenho afetado de acordo com o número de registros e a quantidade de campos da tabela.

Quanto ao envio de informações a partir de uma base de dados Clipper para um SGBD MySQL ou PostgreSQL houve como única saída a criação de uma biblioteca de funções que deveria ser anexada ao programa e que criaria a notificação das modificações na base de dados Clipper, passando para a função o nome da tabela, a chave primária e a operação a ser realizada. O desenvolvedor que utilizar-se das bibliotecas de funções poderá ainda ter a implantação destas em seu sistema legado facilitada através do uso de recursos avançados da Linguagem Clipper que poderá substituir a chamada de funções como a de

inserções, por exemplo, por uma função do usuário. Nesta função do usuário será feito a inserção e após isto será cahamada a função da biblioteca de funções.

No recebimento de dados do SGBD para a base Clipper o procedimento foi o mesmo, porém, poderá ser estudada futuramente a utilização de gatilhos para a realização desta tarefa. Estes gatilhos deverão ser criados no ato de conversão da estrutura da tabela e deverá gerar a notificação da modificação automaticamente, facilitando ainda mais o uso da ferramenta.

REFERÊNCIAS BIBLIOGRÁFICAS

BLUE, Ted; KASTER, John; LIEF, Greg; SCOTT, Loren. *Desenvolvendo banco de dados em delphi.* Trad. de Lavio Pareschi. São Paulo, Makron Books, 1997.

DATE, C.J. *Introdução a sistemas de bancos de dados.* Trad.de Vandenberg Dantas de Souza. Rio de Janeiro, Campus, 2000.

Documentação do PostgreSQL 8.0.0. Disponível em: http://pgdocptbr.sourceforge.net/pg80/index.html. Acesso em: 16 Outubro 2005

ELMASRI, Ramez; NAVATHE, Shamkant B. *Sistema de banco de dados*: fundamentos e aplicações. 3.ed. Trad.de Teresa Cristina Padilha de Souza. Rio de Janeiro, LTC, 2002.

Manual de Referência do MySQL 4.1. Disponível em: http://dev.mysql.com/doc/refman/4.1/pt/index.html Acesso em: 16 Outubro 2005

PAES, Fagner C. *Ferramenta de Manutenção para Banco de Dados*. 2004. Trabalho de Conclusão de Curso (Ciência da Computação) – Fundação Eurípides Soares da Rocha - Univem, Marília, 2004.

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

SOMMERVILLE, Ian. *Engenharia de Software*. Trad.de André Maurício de Andrade. 6.ed. São Paulo, Addison Wesley, 2003

SPENCE, Rick *Clipper 5.2* Trad.de Teresa Cristina Félix de Souza. São Paulo, Makron Books, 1994.

STRALEY, Stephen J. *Clipper 5.2 – Ferramentas Poderosas*. Trad. de Geraldo Costa. Rio de Janeiro, Berkeley, 1993.

Anexo A – Código fonte confeccionado em Delphi para a conversão da estrutura extraída do Clipper

```
procedure TFormCriar.CriarTabela();
var
  NomeTabela, NomeCampo, Auxiliar, StrIndex : String;
  nI, nJ : Integer;
  StrSQLTabela, StrSQLIndex, StrCampoChave : String;
  DMLQuery : TZquery;
  Conectou, CriouTabela, Prosseguir, lAchou : boolean;
  aIndices, aCamposChave : Array of String;
begin
  CriouTabela := false;
  SetLength(aIndices, length(aIndices) + 1);
  aIndices[length(aIndices) - 1] := LabelNtx.Caption;
  VKDBFNTX := TVKDBFNTX.Create(self);
  VKDBFNTX.SetDeleted := true;
  Prosseguir := AbreDbf(VKDBFNTX, LabelDbf.Caption, aIndices, 0);
  if (Prosseguir = true) then
  begin
    NomeTabela := RetNometabela(LabelDbf.Caption);
    StrSQLIndex := '';
    if (length(LabelNtx.Caption) > 0) then
    begin
      //Montagem da String de Criação do Índice da Tabela
      StrCampoChave := '';
      StrSQLIndex := 'ALTER TABLE '+LowerCase(NomeTabela)+' ADD PRIMARY KEY (';
      StrIndex := VKDBFNTX.Orders[0].KeyExpresion;
      nI := 1;
      while (nI <= length(StrIndex)) do
      begin
        Auxiliar := '';
        while ((copy(StrIndex, nI, 1) <> '+') and (nI <= length(StrIndex))) do
        begin
          Auxiliar := Auxiliar + copy(StrIndex, nI, 1);
          nI := nI + 1;
        end;
        nJ := 1;
        NomeCampo := '';
        //Para DTOS, DTOC, CTOD e VAL ignorou-se o ultimo parenteses, copiando
        //o string até a penultima posicao, ignorando a ultima
        if ((uppercase(copy(Auxiliar,1,4)) = 'DTOS') or
            (uppercase(copy(Auxiliar,1,4)) = 'DTOC') or
            (uppercase(copy(Auxiliar,1,4)) = 'CTOD')) then
          //Copia até a penultima posição
          NomeCampo := copy(Auxiliar, 6, (length(Auxiliar) - 6))

        else if (uppercase(copy(Auxiliar,1,3)) = 'VAL') then
          //Copia até a penultima posição
          NomeCampo := copy(Auxiliar, 5, (length(Auxiliar) - 5))

        else if (uppercase(copy(Auxiliar,1,3)) = 'STR') or
                (uppercase(copy(Auxiliar,1,6)) = 'STRING') then
        begin
          if (uppercase(copy(Auxiliar,1,3)) = 'STR') then
            Auxiliar := copy(Auxiliar, 5, (length(Auxiliar) - 4))
          else
            Auxiliar := copy(Auxiliar, 8,(length(Auxiliar) - 7));
          //Copia até o virgula (vem logo após o nome do campo)
          while (copy(Auxiliar, nJ, 1) <> ',') do
          begin
            NomeCampo := NomeCampo + copy(Auxiliar, nJ, 1);
            nJ := nJ + 1;
          end;
        end

        else if (uppercase(copy(Auxiliar,1,4)) = 'STRZ') then
        begin
          Auxiliar := copy(Auxiliar, 6, (length(Auxiliar) - 5));
          //Copia até o virgula (vem logo após o nome do campo)
          while (copy(Auxiliar, nJ, 1) <> ',') do
          begin
            NomeCampo := NomeCampo + copy(Auxiliar, nJ, 1);
```

```
          nJ := nJ + 1;
        end;
      end

      else if (uppercase(copy(Auxiliar,1,4)) = 'SUBS') or
              (uppercase(copy(Auxiliar,1,9)) = 'SUBSTRING') then
      begin
        if (uppercase(copy(Auxiliar,1,4)) = 'SUBS') then
          Auxiliar := copy(Auxiliar, 6, (length(Auxiliar) - 5))
        else
          Auxiliar := copy(Auxiliar, 11,(length(Auxiliar) - 10));
        //Copia até o virgula (vem logo após o nome do campo)
        while (copy(Auxiliar, nJ, 1) <> ',') do
        begin
          NomeCampo := NomeCampo + copy(Auxiliar, nJ, 1);
          nJ := nJ + 1;
        end;
      end
      else
        NomeCampo := Auxiliar;

       SetLength(aCamposChave, length(aCamposChave) + 1);
       aCamposChave[length(aCamposChave) - 1] := lowercase(NomeCampo);

       StrSQLIndex := StrSQLIndex + lowercase(NomeCampo) + ', ';
       StrCampoChave := StrCampoChave + lowercase(NomeCampo) + ';';
      nI := nI + 1;
    end;
    StrSQLIndex := Copy(StrSQLIndex,1,length(StrSQLIndex)-2);
    StrSQLIndex := StrSQLIndex + ')';
    StrCampoChave := Copy(StrCampoChave,1,length(StrCampoChave)-1);
  end;


  //Montagem da String de criação da Tabela no SGBD
  StrSQLTabela := 'CREATE TABLE ' + NomeTabela + ' (';
  for nI := 0 to VKDBFNTX.DBFFieldDefs.Count - 1 do
  begin
    NomeCampo := VKDBFNTX.DBFFieldDefs.Items[nI].Name;
    StrSQLTabela := StrSQLTabela + lowercase(NomeCampo) + ' ';
    if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'C') then
    begin
      if (VKDBFNTX.DBFFieldDefs.Items[nI].len = 1) then
        StrSQLTabela := StrSQLTabela + ' char(1)'
      else
        StrSQLTabela := StrSQLTabela + ' varchar(' +
        InttoStr(VKDBFNTX.DBFFieldDefs.Items[nI].len) + ')';
    end
    else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'D') then
      StrSQLTabela := StrSQLTabela + ' date'
    else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'N') then
      StrSQLTabela := StrSQLTabela + ' numeric(' +
      InttoStr(VKDBFNTX.DBFFieldDefs.Items[nI].len) + ',' +
      InttoStr(VKDBFNTX.DBFFieldDefs.Items[nI].dec)  + ')'
    else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'L') then
      StrSQLTabela := StrSQLTabela + ' boolean'
    else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'M') then
      StrSQLTabela := StrSQLTabela + ' text'
    else
    begin
      MessageDlg('ERRO CRIANDO TABELA: ' + NomeTabela + #13 + #13 +
      ' Tipo de dado inválido para o campo: '+NomeCampo, mtError, [mbOk], 0);
    end;

    lAchou := false;
    for nJ := 0 to length(aCamposChave) - 1 do
    begin
      if (lowercase(NomeCampo) = lowercase(aCamposChave[nJ])) then
        lAchou := true;
    end;
    if (lAchou = true) then
      StrSQLTabela := StrSQLTabela + ' NOT NULL, '
    else
      StrSQLTabela := StrSQLTabela + ', ';
  end;
  StrSQLTabela := Copy(StrSQLTabela,1,length(StrSQLTabela)-2);
```

```
    StrSQLTabela := StrSQLTabela + ')';

    ProgressBar.Position := 0;
    if (length(LabelNtx.Caption) > 0) then
      ProgressBar.Max := 2
    else
      ProgressBar.Max := 1;

    CriouTabela := true;
    Conectou := ConectaBanco(ConexaoBanco);
    if (Conectou = true) then
    begin
      try
        DMLQuery := TZQuery.Create(self);
        DMLQuery.Active := false;
        DMLQuery.SQL.Clear;
        DMLQuery.Connection := ConexaoBanco;
        DMLQuery.SQL.Add(StrSQLTabela);
        DMLQuery.Active := true;
        ProgressBar.Position := ProgressBar.Position + 1;
      except
        on E: EZSQLException do
        begin
          if (lowercase(E.Message) = 'can not open a resultset') then
            ProgressBar.Position := ProgressBar.Position + 1
          else
          begin
            CriouTabela := false;
            BitBtnPopular.Enabled := false;
            MessageDlg(E.Message, mtWarning, [mbOk], 0);
          end;
        end;
      end;

      if ((length(StrSQLIndex) > 0) and (CriouTabela = true)) then
      begin
        try
          DMLQuery := TZQuery.Create(self);
          DMLQuery.Active := false;
          DMLQuery.SQL.Clear;
          DMLQuery.Connection := ConexaoBanco;

          DMLQuery.SQL.Add(StrSQLIndex);
          //DMLQuery.ExecSQL;
          DMLQuery.Active := true;
          ProgressBar.Position := ProgressBar.Position + 1;
        except
          on E: EZSQLException do
          begin
            if (lowercase(E.Message) = 'can not open a resultset') then
              ProgressBar.Position := ProgressBar.Position + 1
            else
            begin
              CriouTabela := false;
              BitBtnPopular.Enabled := false;
              MessageDlg(E.Message, mtWarning, [mbOk], 0);
            end;
          end;
        end;
        BitBtnPopular.Enabled := true;
      end;
      DesconectaBanco(ConexaoBanco);
      FechaDbf(VKDBFNTX, aIndices);
      if (CriouTabela = true) then
      begin
        BitBtnCriar.Enabled := false;
        GravaCatalogo(NomeTabela, StrCampoChave);
        MessageDlg('Tabela ' + NomeTabela + ' criada com sucesso!', mtInformation, [mbOk], 0)
      end
      else
        MessageDlg('Erros na criação da Tabela ' + NomeTabela + ' impedem a operação de ser
concluída!' +
                   #13 + '                          Processo Cancelado!', mtError, [mbOk], 0);
    end
    else
    begin
```

```
      MessageDlg('Não foi possível estabeler conexão com o Banco de Dados!' +
                 #13 + 'Processo de Criação de Tabela Cancelado!', mtError, [mbOk], 0);
      FechaDbf(VKDBFNTX, aIndices);
    end;
  end
  else
    MessageDlg('Houve Problemas com a abertura do arquivo ou indice selecionado' +
                 #13 + 'Processo cancelado! Reinicie a seleção e tente novamente!', mtError,
[mbOk], 0);
end;
```

Anexo B – Código fonte confeccionado em Delphi para a replicação dos dados extraídos do Clipper

```
procedure TFormCriar.BitBtnPopularClick(Sender: TObject);
var
  SeparadorDecimal : Char;
  StrSQLCampos, StrSQLValores, StrSQLPopula, NomeTabela : String;
  PopulouRegistro, Conectou, Prosseguir : boolean;
  DMLQuery : TZquery;
  nI : Integer;
  aIndices : Array of String;
  horainicio, horafim : String;
begin
  HoraInicio := timetostr(time);
  ProgressBar.Position := 0;
  SeparadorDecimal := DecimalSeparator;
  DecimalSeparator := '.';
  CarregaValoresDefinidos();
  SetLength(aIndices, length(aIndices) + 1);
  aIndices[length(aIndices) - 1] := LabelNtx.Caption;
  VKDBFNTX := TVKDBFNTX.Create(self);
  VKDBFNTX.SetDeleted := true;
  Prosseguir := AbreDbf(VKDBFNTX, LabelDbf.Caption, aIndices, 0);
  if (Prosseguir = true) then
  begin
    Conectou := ConectaBanco(ConexaoBanco);
    if (Conectou = true) then
    begin
      PopulouRegistro := true;
      NomeTabela := RetNomeTabela(LabelDbf.Caption);
      StrSQLCampos := 'INSERT INTO ' + NomeTabela + '(';
      for nI := 0 to VKDBFNTX.DBFFieldDefs.Count - 1 do
      begin
        StrSQLCampos := StrSQLCampos + lowercase(VKDBFNTX.DBFFieldDefs.Items[nI].Name) + ',
';
      end;
      StrSQLCampos := Copy(StrSQLCampos,1,length(StrSQLCampos)-2);
      StrSQLCampos := StrSQLCampos + ') ';

      StrSQLValores := 'VALUES(';
      for nI := 0 to VKDBFNTX.DBFFieldDefs.Count - 1 do
      begin
        StrSQLValores := StrSQLValores + ':p' + InttoStr(nI) + ', ';
      end;
      StrSQLValores := Copy(StrSQLValores, 1, length(StrSQLValores)-2);
      StrSQLValores := StrSQLValores + ') ';

      //Monta a string de inserção completa
      StrSQLPopula := StrSQLCampos + StrSQLValores;
      DMLQuery := TZQuery.Create(self);
      try
        DMLQuery.Active := false;
        DMLQuery.SQL.Clear;
        DMLQuery.Connection := ConexaoBanco;
        DMLQuery.SQL.Add(StrSQLPopula);
      except
        on E: EZSQLException do
        begin
          if (lowercase(E.Message) = 'can not open a resultset') then
            ProgressBar.Position := ProgressBar.Position + 1
          else
          begin
            PopulouRegistro := false;
            MessageDlg(E.Message, mtWarning, [mbOk], 0);
          end;
        end;
      end;
      ProgressBar.Max := VKDBFNTX.RecordCount;
      VKDBFNTX.First;
      while (not VKDBFNTX.Eof and PopulouRegistro = true) do
      begin
        if (VKDBFNTX.Deleted = true) then
        begin
```

```
          ProgressBar.Position := ProgressBar.Position + 1;
          VKDBFNTX.Next;
        end
        else
        begin
          try
            DMLQuery.Active := false;
            for nI := 0 to VKDBFNTX.DBFFieldDefs.Count - 1 do
            begin
              if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'C') then
              begin
                if
(length(trim(VKDBFNTX.FieldByName(VKDBFNTX.DBFFieldDefs.Items[nI].Name).Text)) > 0) then
                  DMLQuery.Params[nI].Value := VKDBFNTX.Fields[nI].Value
                else
                begin
                  DMLQuery.Params[nI].AsString := '';
                  if (TipoCaracter = 'NULL') then
                    DMLQuery.Params[nI].Value := NULL;
                end;
              end
              else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'N') then
              begin
                if
(length(trim(VKDBFNTX.FieldByName(VKDBFNTX.DBFFieldDefs.Items[nI].Name).Text)) > 0) then
                  DMLQuery.Params[nI].Value := VKDBFNTX.Fields[nI].Value
                else
                  begin
                    if (VKDBFNTX.DBFFieldDefs.Items[nI].dec > 0) then
                      DMLQuery.Params[nI].AsFloat := 0
                    else
                      DMLQuery.Params[nI].AsInteger := 0;
                    if (TipoNumerico = 'NULL') then
                      DMLQuery.Params[nI].Value := NULL;
                end;
              end
              else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'D') then
              begin
                if
(length(trim(VKDBFNTX.FieldByName(VKDBFNTX.DBFFieldDefs.Items[nI].Name).Text)) > 0) then
                begin
                  DMLQuery.Params[nI].asDate := VKDBFNTX.Fields[nI].Value;
                end
                else
                begin
                  if (TipoData = 'NULL') then
                  begin
                    DMLQuery.Params[nI].AsString := '';
                    DMLQuery.Params[nI].Value := NULL
                  end
                  else
                  begin
                    DMLQuery.Params[nI].asDate :=
strtoDate(copy(TipoData,9,2)+'/'+copy(TipoData,6,2)+'/'+copy(TipoData,1,4));
                  end;
                end;
              end
              else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'L') then
              begin
                if
(lowercase(VKDBFNTX.FieldByName(VKDBFNTX.DBFFieldDefs.Items[nI].Name).Text) = 'true') then
                begin
                  if (TipoBanco = 'MYSQL') then
                    DMLQuery.Params[nI].AsInteger := 1
                  else
                    DMLQuery.Params[nI].Value := true;
                end
                else
                begin
                  if (TipoBanco = 'MYSQL') then
                    DMLQuery.Params[nI].AsInteger := 0
                  else
                    DMLQuery.Params[nI].Value := false;
                end;
              end
              else if (VKDBFNTX.DBFFieldDefs.Items[nI].field_type = 'M') then
```

```
              begin
                CampoMemo.Clear;
                with CampoMemo.Lines do begin
                  Add(VKDBFNTX.Fields[nI].Value);
                end;
                if (length(trim(CampoMemo.Text)) > 0) then
                  DMLQuery.Params[nI].Assign(CampoMemo.Lines)
                else
                begin
                  DMLQuery.Params[nI].AsString := '';
                  if (TipoMemo = 'NULL') then
                    DMLQuery.Params[nI].Value := NULL;
                end;
              end
              else
              begin
                DMLQuery.Params[nI].AsString := '';
                DMLQuery.Params[nI].Value := NULL;
              end;
            end;

            DMLQuery.Active := true;
            ProgressBar.Position := ProgressBar.Position + 1;
          except
            on E: EZSQLException do
            begin
              if (lowercase(E.Message) = 'can not open a resultset') then
                ProgressBar.Position := ProgressBar.Position + 1
              else
              begin
                PopulouRegistro := false;
                MessageDlg(E.Message, mtWarning, [mbOk], 0);
              end;
            end;
          end;
          VKDBFNTX.Next;
        end;
      end;
      DMLQuery.Destroy;
      DesconectaBanco(ConexaoBanco);
      HoraFim := timetostr(time);

      if (PopulouRegistro = true) then
        MessageDlg('Tabela ' + NomeTabela + ' populada com sucesso! Inicio: '+ horainicio + '
Fim: ' + horafim, mtInformation, [mbOk], 0)
      else
      begin
        MessageDlg('Houve problemas durante o processo para popular a tabela ' + NomeTabela +
#13 + #13 +
                   '                              Registro: ' + InttoStr(VKDBFNTX.RecNo - 1) +
#13 +  #13 +
                   '                          Processo Cancelado!', mtWarning, [mbOk], 0);
      end;
    end
    else
      MessageDlg('Não foi possível estabeler conexão com o Banco de Dados!' +
                 #13 + 'Processo para Popular Tabela Cancelado!', mtError, [mbOk], 0);

    FechaDbf(VKDBFNTX, aIndices);
  end
  else
    MessageDlg('Houve Problemas com a abertura do arquivo ou indice selecionado' +
               #13 + 'Processo cancelado! Reinicie a seleção e tente novamente!', mtError,
[mbOk], 0);

  DecimalSeparator := SeparadorDecimal;
end;
```

Anexo C – Código fonte confeccionado em Delphi para a geração de um catálogo de arquivos

```
procedure TFormCriar.GravaCatalogo(NomeTabela : String; StrCampoChave : String);
var
  nRegistro, nI : Integer;
  aIndices : Array of String;
  Prosseguir : boolean;
begin
  SetLength(aIndices, length(aIndices) + 1);
  aIndices[length(aIndices) - 1] := 'arqdbf.ntx';
  VKDBFNTX := TVKDBFNTX.Create(self);
  VKDBFNTX.SetDeleted := true;
  Prosseguir := AbreDbf(VKDBFNTX, 'arqdbf.dbf', aIndices, 0);
  if (Prosseguir = true) then
  begin
    for nI := 0 to (ListBoxNtx.Items.Count - 1) do
    begin
      nRegistro := VKDBFNTX.Orders[0].FindKeyFields('TABELA;CAMINHONTX',
                   [uppercase(NomeTabela),uppercase(ListBoxNtx.Items.Strings[nI])]);
      if (nRegistro = 0) then
        VKDBFNTX.Append
      else
      begin
        VKDBFNTX.RecNo := nRegistro;
        VKDBFNTX.Edit;
      end;
      VKDBFNTX.RLock;
      VKDBFNTX.FieldByName('TABELA').Value := uppercase(NomeTabela);
      VKDBFNTX.FieldByName('CAMINHODBF').Value := uppercase(LabelDbf.Caption);
      VKDBFNTX.FieldByName('CAMINHONTX').Value := uppercase(ListBoxNtx.Items.Strings[nI]);
      if (uppercase(LabelNtx.Caption) = uppercase(ListBoxNtx.Items.Strings[nI])) then
      begin
        VKDBFNTX.FieldByName('CAMPOCHAVE').Value := uppercase(StrCampoChave);
        VKDBFNTX.FieldByName('PRIMARYKEY').Value := true;
      end
      else
        VKDBFNTX.FieldByName('PRIMARYKEY').Value := false;
      VKDBFNTX.RUnLock;
      VKDBFNTX.Post;
    end;
    FechaDbf(VKDBFNTX, aIndices);
  end
  else
    MessageDlg('Houve Problemas com a abertura do arquivo de catalogos' +
               #13 + 'Processo de gravação de catalogo cancelado!', mtError, [mbOk], 0);
end;
```

Anexo D – Código fonte confeccionado em Delphi para o envio de atualizações para o SGBD

```
procedure TFormEnvia.Envio();
var
  Conectou, Prosseguir, FlegaRegistro, AchouCampo : Boolean;
  nRegistro, nOrdemKey, nIni, nAux, nAux2, nCont, nPosCampo : Integer;
  StrCampoChave, CaminhoDbf, Caracter, NomeCampo : String;
  StrSQLWhere, StrSQLComando : String;
  aIndexVala, aIndexCatalogo, aIndices, aSeek, aCamposChave : Array of String;
  DMLQuerySelect, DMLQueryComando : TZQuery;
begin
  TimerEnvia.Enabled := false;
  BitBtnRetornar.Enabled := false;
  MemoEnvia.Lines.Clear;
  aIndexCatalogo := nil;
  SetLength(aIndexVala, length(aIndexVala) + 1);
  aIndexVala[length(aIndexVala) - 1] := 'c:\projeto\dbf_sgbd.ntx';
  ArquivoVala := TVKDBFNTX.Create(self);
  ArquivoVala.SetDeleted := true;

  Prosseguir := AbreDbf(ArquivoVala, 'c:\projeto\dbf_sgbd.dbf', aIndexVala, 0);
  if (Prosseguir = true) then
  begin
    Conectou := ConectaBanco(ConexaoBanco);
    if (Conectou = true) then
    begin
      MemoEnvia.Lines.Add('');
      MemoEnvia.Lines.Add('   Verificando atualizações disponíveis no XBase...');
      ArquivoVala.First;
      nRegistro := ArquivoVala.Orders[0].FindKeyFields('DATATRANS', []);
      if (nRegistro > 0) then
      begin
        ArquivoVala.RecNo := nRegistro;
        //Abre Catalogo para poder abrir os índices do arquivo
        MemoEnvia.Lines.Add('   Abrindo Catalogo de Arquivos...');
        SetLength(aIndexCatalogo, length(aIndexCatalogo) + 1);
        aIndexCatalogo[length(aIndexCatalogo) - 1] := 'arqdbf.ntx';
        Catalogo := TVKDBFNTX.Create(self);
        Catalogo.SetDeleted := true;
        Prosseguir := AbreDbf(Catalogo, 'c:\projeto\arqdbf.dbf', aIndexCatalogo, 0);
        if (Prosseguir = true) then
        begin
          while ((not ArquivoVala.Eof) and (ArquivoVala.FieldByName('DATATRANS').Text = ''))
do
          begin
            MemoEnvia.Lines.Add('   Selecionando arquivo para envio...');
            nRegistro :=
Catalogo.Orders[0].FindKeyFields('TABELA;CAMINHONTX',[trim(uppercase(ArquivoVala.FieldByName('
TABELA').Text))]);
            if nRegistro > 0 then
              Catalogo.RecNo := nRegistro
            else
              Prosseguir := false;
            if (Prosseguir = true) then
            begin
              aIndices := nil;
              StrCampoChave := '';
              nOrdemKey := 0;
              CaminhoDbf := trim(Catalogo.FieldByName('CAMINHODBF').Text);
              while ((not Catalogo.Eof) and (trim(Catalogo.FieldByName('TABELA').Text) =
trim(uppercase(ArquivoVala.FieldByName('TABELA').Text)))) do
              begin
                SetLength(aIndices, length(aIndices) + 1);
                aIndices[length(aIndices) - 1] :=
trim(Catalogo.FieldByName('CAMINHONTX').Text);
                if (Catalogo.FieldByName('PRIMARYKEY').Value = true) then
                begin
                  StrCampoChave := trim(Catalogo.FieldByName('CAMPOCHAVE').Text);
                  nOrdemKey := length(aIndices) - 1;
                end;
                Catalogo.Next;
              end;
```

```
              ArquivoOrigem := TVKDBFNTX.Create(self);
              ArquivoOrigem.SetDeleted := true;
              Prosseguir := AbreDbf(ArquivoOrigem, CaminhoDbf, aIndices, nOrdemKey);
              if (Prosseguir = true) then
              begin
                //Cria matriz com chave de busca (aseek)
                MemoEnvia.Lines.Add('   Montando Matriz de busca para o arquivo.');
                nIni := 1;
                nAux := 1;
                aSeek := nil;
                while (nAux <= length(trim(ArquivoVala.FieldByName('chave').Text))) do
                begin
                  Caracter := Copy(ArquivoVala.FieldByName('chave').Text, nAux, 1);
                  if (Caracter = ';') or (nAux =
length(trim(ArquivoVala.FieldByName('chave').Text))) then
                  begin
                    if (nAux = length(trim(ArquivoVala.FieldByName('chave').Text))) then
                      nAux := nAux + 1;
                    SetLength(aSeek, length(aSeek) + 1);
                    aSeek[length(aSeek) - 1] := Copy(ArquivoVala.FieldByName('chave').Text,
nIni, (nAux - nIni));
                    nIni := nAux + 1;
                  end;
                  nAux := nAux + 1;
                end;

                MemoEnvia.Lines.Add('   Procurando por registro na origem (Xbase)...');
                //Se possuir indice
                if (length(StrCampoChave) > 0) then
                  nRegistro := ArquivoOrigem.Orders[nOrdemKey].FindKeyFields(StrCampoChave,
aSeek)
                else
                begin
                  //Se nao possuir, verifica se existe somente um registro, se existir
atualiza
                  //para mais de um registro ou nenhum registro - mensagem de erro
                  if (ArquivoOrigem.RecordCount = 1) then
                    nRegistro := 1
                  else
                    nRegistro := 0;
                end;
                if nRegistro > 0 then
                  ArquivoOrigem.RecNo := nRegistro
                else
                  Prosseguir := false;
                if (Prosseguir = true) then
                begin
                  MemoEnvia.Lines.Add('   Selecionando registro na origem (Xbase)...');

                  //Montando Select a partir do arquivo Xbase
                  try
                    MemoEnvia.Lines.Add('   Selecionando registro na origem (SGBD)...');
                    DMLQuerySelect := TZQuery.Create(self);
                    DMLQuerySelect.Active := false;
                    DMLQuerySelect.SQL.Clear;
                    DMLQuerySelect.Connection := ConexaoBanco;
                    DMLQuerySelect.SQL.Add('SELECT * FROM ');

DMLQuerySelect.SQL.Add(trim(lowercase(ArquivoVala.FieldByName('TABELA').Text)));
                    //Se possuir indice
                    if (length(StrCampoChave) > 0) then
                    begin
                      DMLQuerySelect.SQL.Add(' WHERE ');
                      StrSQLWhere := RetornaSQLWhere(StrCampoChave, 0);
                      DMLQuerySelect.SQL.Add(StrSQLWhere);

                      nCont := 0;
                      nIni := 1;
                      for nAux := 1 to length(trim(StrCampoChave)) do
                      begin
                        Caracter := Copy(StrCampoChave, nAux, 1);
                        if (Caracter = ';') then
                        begin
                          if (Copy(StrCampoChave, nIni, (nAux - nIni)) <> '') then
                          begin
                            //Guardando Campos em uma matriz
```

```
                          SetLength(aCamposChave, nCont + 1);
                          aCamposChave[nCont] := Copy(StrCampoChave, nIni, (nAux - nIni));
                          nCont := nCont + 1;
                        end;
                        nIni := nAux + 1;
                      end;
                    end;
                    if ((nAux - 1)  = length(StrCampoChave)) then
                    begin
                      SetLength(aCamposChave, nCont + 1);
                      aCamposChave[nCont] := Copy(StrCampoChave, nIni, (nAux - nIni));
                    end;
                  end;

                  nCont := 0;
                  nIni := 1;
                  nAux := 1;
                  while (nAux <= length(trim(ArquivoVala.FieldByName('chave').Text))) do
                  begin
                    Caracter := Copy(ArquivoVala.FieldByName('chave').Text, nAux, 1);
                    if (Caracter = ';') or (nAux =
length(trim(ArquivoVala.FieldByName('chave').Text))) then
                    begin
                      if (nAux = length(trim(ArquivoVala.FieldByName('chave').Text))) then
                        nAux := nAux + 1;
                      if (Copy(ArquivoVala.FieldByName('chave').Text, nIni, (nAux - nIni))
<> '') then
                      begin
                        ParametroSelect(Arquivovala, ArquivoOrigem, DMLQuerySelect,
aCamposChave[nCont], nCont, nIni, nAux);
                      end;
                      nCont := nCont + 1;
                      nIni := nAux + 1;
                    end;
                    nAux := nAux + 1;
                  end;
                  DMLQuerySelect.Active := true;
                except
                  on E: EZSQLException do
                  begin
                    if (lowercase(E.Message) <> 'can not open a resultset') then
                    begin
                      MessageDlg(E.Message, mtWarning, [mbOk], 0);
                      Prosseguir := false;
                    end;
                  end;
                end;
                if (Prosseguir = true) then
                begin
                  try
                    MemoEnvia.Lines.Add('   Criando comando SQL para executar operação...');
                    DMLQueryComando := TZQuery.Create(self);
                    DMLQueryComando.Active := false;
                    DMLQueryComando.SQL.Clear;
                    DMLQueryComando.Connection := ConexaoBanco;
                  except
                    on E: EZSQLException do
                    begin
                      MessageDlg(E.Message, mtWarning, [mbOk], 0);
                    end;
                  end;
                  FlegaRegistro := false;
                  StrSQLComando := '';
                  //Se encontrou o registro, é necessário atualiza-lo ou remove-lo (Depende
                  //do Flag de opcao), caso nao encontrou é necessário inseri-lo
                  if (DMLQuerySelect.RecordCount = 1) then
                  begin
                    //Se for um 'U'pdate ou um 'D'elete
                    if (uppercase(ArquivoVala.FieldByName('OPERACAO').Text) = 'U') or
                       (uppercase(ArquivoVala.FieldByName('OPERACAO').Text) = 'D') then
                    begin
                      //Se for Update
                      if (uppercase(ArquivoVala.FieldByName('OPERACAO').Text) = 'U') then
                      begin
                        StrSQLComando := 'UPDATE ' +
lowercase(trim(ArquivoVala.FieldByName('TABELA').Text)) + ' SET ';
```

```
                          nCont := 0;
                          for nAux := 0 to (ArquivoOrigem.DBFFieldDefs.Count - 1) do
                          begin
                            AchouCampo := false;
                            //Pego o nome do campo que vou procurar na tabela resultado
                            NomeCampo :=
lowercase(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name);
                            for nAux2 := 0 to (length(aCamposChave) - 1) do
                            begin
                              if (trim(lowercase(aCamposChave[nAux2])) = NomeCampo) then
                                AchouCampo := true;
                            end;
                            //Se não é campo chave
                            if (not AchouCampo) then
                            begin
                              StrSQLComando := StrSQLComando + NomeCampo + '=:p' +
InttoStr(nCont) + ', ';
                              nCont := nCont + 1;
                            end;
                          end;
                          if (length(StrSQLComando) > 0) then
                          begin
                            StrSQLComando := Copy(StrSQLComando,1,length(StrSQLComando)-2);
                            DMLQueryComando.SQL.Add(StrSQLComando);
                          end;

                          nCont := 0;
                          for nAux := 0 to (ArquivoOrigem.DBFFieldDefs.Count - 1) do
                          begin
                            AchouCampo := false;
                            //Pego o nome do campo que vou procurar na tabela resultado
                            NomeCampo :=
lowercase(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name);
                            for nAux2 := 0 to (length(aCamposChave) - 1) do
                            begin
                              if (trim(lowercase(aCamposChave[nAux2])) = NomeCampo) then
                                AchouCampo := true;
                            end;
                            //Se não é campo chave
                            if (not AchouCampo) then
                            begin
                              if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type = 'M')
then
                              begin
                                //É necessario gravar em um TMEMO antes do DBF
                                CampoMemo.Clear;
                                with CampoMemo.Lines do begin
                                  Add(ArquivoOrigem.Fields[nAux].Value);
                                end;
                                if (length(trim(CampoMemo.Text)) > 0) then
                                  DMLQueryComando.Params[nCont].Assign(CampoMemo.Lines)
                                else
                                begin
                                  DMLQueryComando.Params[nCont].AsString := '';
                                  if (TipoMemo = 'NULL') then
                                    DMLQueryComando.Params[nCont].Value := NULL;
                                end;
                              end
                              else if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type =
'C') then
                              begin
                                if
(length(trim(ArquivoOrigem.FieldByName(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name).Text)) >
0) then
                                  DMLQueryComando.Params[nCont].Value :=
ArquivoOrigem.Fields[nAux].Value
                                else
                                begin
                                  DMLQueryComando.Params[nCont].AsString := '';
                                  if (TipoCaracter = 'NULL') then
                                    DMLQueryComando.Params[nCont].Value := NULL;
                                end;
                              end
                              else if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type =
'N') then
```

```
                              begin
                                if
(length(trim(ArquivoOrigem.FieldByName(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name).Text)) >
0) then
                                  DMLQueryComando.Params[nCont].Value :=
ArquivoOrigem.Fields[nAux].Value
                                else
                                begin
                                  if (ArquivoOrigem.DBFFieldDefs.Items[nAux].dec > 0) then
                                    DMLQueryComando.Params[nCont].AsFloat := 0
                                  else
                                    DMLQueryComando.Params[nCont].AsInteger := 0;
                                  if (TipoNumerico = 'NULL') then
                                    DMLQueryComando.Params[nCont].Value := NULL;
                                end;
                              end
                              else if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type =
'D') then
                              begin
                                if
(length(trim(ArquivoOrigem.FieldByName(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name).Text)) >
0) then
                                  DMLQueryComando.Params[nCont].Value :=
ArquivoOrigem.Fields[nAux].Value
                                else
                                begin
                                  if (TipoData = 'NULL') then
                                  begin
                                    DMLQueryComando.Params[nCont].AsString := '';
                                    DMLQueryComando.Params[nCont].Value := NULL
                                  end
                                  else
                                  begin
                                    DMLQueryComando.Params[nCont].asDate :=
strtoDate(copy(TipoData,9,2)+'/'+copy(TipoData,6,2)+'/'+copy(TipoData,1,4));
                                  end;
                                end;
                              end
                              else if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type =
'L') then
                              begin
                                if
(lowercase(ArquivoOrigem.FieldByName(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name).Text) =
'true') then
                                begin
                                  if (TipoBanco = 'MYSQL') then
                                    DMLQueryComando.Params[nCont].AsInteger := 1
                                  else
                                    DMLQueryComando.Params[nCont].Value := true;
                                end
                                else
                                begin
                                  if (TipoBanco = 'MYSQL') then
                                    DMLQueryComando.Params[nCont].AsInteger := 0
                                  else
                                    DMLQueryComando.Params[nCont].Value := false;
                                end;
                              end;
                              nCont := nCont + 1;
                            end;
                          end;
                        end
                        else
                        begin
                          nCont := 0;
                          StrSQLComando := 'DELETE FROM ' +
lowercase(trim(ArquivoVala.FieldByName('TABELA').Text)) + ' ';
                          DMLQueryComando.SQL.Add(StrSQLComando);
                        end;
                        if (length(trim(StrCampoChave)) > 0) then
                        begin
                          try
                            DMLQueryComando.SQL.Add('WHERE ');
                            StrSQLWhere := RetornaSQLWhere(StrCampoChave, nCont);
                            DMLQueryComando.SQL.Add(StrSQLWhere);
                            nIni := 1;
```

```
                            nAux := 1;
                            nAux2 := 0;
                            while (nAux <=
length(trim(ArquivoVala.FieldByName('chave').Text))) do
                            begin
                              Caracter := Copy(ArquivoVala.FieldByName('chave').Text, nAux,
1);
                              if (Caracter = ';') or (nAux =
length(trim(ArquivoVala.FieldByName('chave').Text))) then
                              begin
                                if (nAux =
length(trim(ArquivoVala.FieldByName('chave').Text))) then
                                  nAux := nAux + 1;
                                if (Copy(ArquivoVala.FieldByName('chave').Text, nIni, (nAux -
nIni)) <> '') then
                                begin
                                  ParametroSelect(Arquivovala, ArquivoOrigem, DMLQueryComando,
aCamposChave[nAux2], nCont, nIni, nAux);
                                end;
                                nAux2 := nAux2 + 1;
                                nCont := nCont + 1;
                                nIni := nAux + 1;
                              end;
                              nAux := nAux + 1;
                            end;
                            FlegaRegistro := true;
                            DMLQueryComando.Active := true;
                          except
                            on E: EZSQLException do
                            begin
                              if (lowercase(E.Message) <> 'can not open a resultset') then
                              begin
                                FlegaRegistro := false;
                                MessageDlg(E.Message, mtWarning, [mbOk], 0);
                              end;
                            end;
                          end;
                        end
                        else
                        begin
                          MessageDlg('Tipo de Operação: ' +
uppercase(ArquivoVala.FieldByName('OPERACAO').Text) + ' para a Tabela: ' +
uppercase(ArquivoVala.FieldByName('TABELA').Text) +
                                     #13 + '    Inválida para a chave: ' +
ArquivoVala.FieldByName('CHAVE').Text +
                                     #13 + '                 Processo cancelado!                ',
mtError, [mbOk], 0);
                        end;
                      end
                      else
                      begin
                        if (DMLQuerySelect.RecordCount = 0) then
                        begin
                          if (uppercase(ArquivoVala.FieldByName('OPERACAO').Text) = 'I') then
                          begin
                            StrSQLComando := 'INSERT INTO ' +
lowercase(trim(ArquivoVala.FieldByName('TABELA').Text)) + ' (';
                            for nAux := 0 to (ArquivoOrigem.DBFFieldDefs.Count - 1) do
                            begin
                              //Pego o nome do campo que vou procurar na tabela resultado
                              NomeCampo :=
lowercase(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name);
                              StrSQLComando := StrSQLComando + NomeCampo + ', ';
                            end;
                            if (length(StrSQLComando) > 0) then
                            begin
                              StrSQLComando := Copy(StrSQLComando,1,length(StrSQLComando)-2);
                              StrSQLComando := StrSQLComando + ') ';
                              DMLQueryComando.SQL.Add(StrSQLComando);
                            end;

                            nCont := 0;
                            StrSQLComando := 'VALUES(';
                            for nAux := 0 to (ArquivoOrigem.DBFFieldDefs.Count - 1) do
                            begin
```

```
                            StrSQLComando := StrSQLComando + ':p' + InttoStr(nCont) + ', ';
                            nCont := nCont + 1;
                          end;
                          if (length(StrSQLComando) > 0) then
                          begin
                            StrSQLComando := Copy(StrSQLComando,1,length(StrSQLComando)-2);
                            StrSQLComando := StrSQLComando + ')';
                            DMLQueryComando.SQL.Add(StrSQLComando);
                          end;

                          for nAux := 0 to (ArquivoOrigem.DBFFieldDefs.Count - 1) do
                          begin
                            if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type = 'M') then
                            begin
                              //É necessario gravar em um TMEMO antes do DBF
                              CampoMemo.Clear;
                              with CampoMemo.Lines do begin
                                Add(ArquivoOrigem.Fields[nAux].Value);
                              end;
                              if (length(trim(CampoMemo.Text)) > 0) then
                                DMLQueryComando.Params[nAux].Assign(CampoMemo.Lines)
                              else
                              begin
                                DMLQueryComando.Params[nAux].AsString := '';
                                if (TipoMemo = 'NULL') then
                                  DMLQueryComando.Params[nAux].Value := NULL;
                              end;
                            end
                            else
                            begin
                              if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type = 'C')
then
                              begin
                                if
(length(trim(ArquivoOrigem.FieldByName(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name).Text)) >
0) then
                                  DMLQueryComando.Params[nAux].Value :=
ArquivoOrigem.Fields[nAux].Value
                                else
                                begin
                                  DMLQueryComando.Params[nAux].AsString := '';
                                  if (TipoCaracter = 'NULL') then
                                    DMLQueryComando.Params[nAux].Value := NULL;
                                end;
                              end
                              else if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type =
'N') then
                              begin
                                if
(length(trim(ArquivoOrigem.FieldByName(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name).Text)) >
0) then
                                  DMLQueryComando.Params[nAux].Value :=
ArquivoOrigem.Fields[nAux].Value
                                else
                                begin
                                  if (ArquivoOrigem.DBFFieldDefs.Items[nAux].dec > 0) then
                                    DMLQueryComando.Params[nAux].AsFloat := 0
                                  else
                                    DMLQueryComando.Params[nAux].AsInteger := 0;
                                  if (TipoNumerico = 'NULL') then
                                    DMLQueryComando.Params[nAux].Value := NULL;
                                end;
                              end
                              else if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type =
'D') then
                              begin
                                if
(length(trim(ArquivoOrigem.FieldByName(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name).Text)) >
0) then
                                  DMLQueryComando.Params[nAux].Value :=
ArquivoOrigem.Fields[nAux].Value
                                else
                                begin
                                  if (TipoData = 'NULL') then
                                  begin
                                    DMLQueryComando.Params[nAux].AsString := '';
```

```
                                  DMLQueryComando.Params[nAux].Value := NULL
                                end
                                else
                                begin
                                  DMLQueryComando.Params[nAux].asDate :=
strtoDate(copy(TipoData,9,2)+'/'+copy(TipoData,6,2)+'/'+copy(TipoData,1,4));
                                end;
                              end;
                            end
                            else if (ArquivoOrigem.DBFFieldDefs.Items[nAux].field_type =
'L') then
                            begin
                              if
(lowercase(ArquivoOrigem.FieldByName(ArquivoOrigem.DBFFieldDefs.Items[nAux].Name).Text) =
'true') then
                              begin
                                if (TipoBanco = 'MYSQL') then
                                  DMLQueryComando.Params[nAux].AsInteger := 1
                                else
                                  DMLQueryComando.Params[nAux].Value := true;
                              end
                              else
                              begin
                                if (TipoBanco = 'MYSQL') then
                                  DMLQueryComando.Params[nAux].AsInteger := 0
                                else
                                  DMLQueryComando.Params[nAux].Value := false;
                              end;
                            end;
                          end;
                        end;

                        try
                          FlegaRegistro := true;
                          DMLQueryComando.Active := true;
                        except
                          on E: EZSQLException do
                          begin
                            if (lowercase(E.Message) <> 'can not open a resultset') then
                            begin
                              FlegaRegistro := false;
                              MessageDlg(E.Message, mtWarning, [mbOk], 0);
                            end;
                          end;
                        end;
                      end
                      else
                      begin
                        MessageDlg('Tipo de Operação: ' +
uppercase(ArquivoVala.FieldByName('OPERACAO').Text) + ' da Tabela: ' +
uppercase(ArquivoVala.FieldByName('TABELA').Text) +
                                   #13 + '    Inválida para a chave: ' +
ArquivoVala.FieldByName('CHAVE').Text +
                                   #13 + '                Processo cancelado!
', mtError, [mbOk], 0);
                      end;
                    end
                    else
                      MessageDlg('Inserção não pode ser realizada na Tabela: ' +
uppercase(ArquivoVala.FieldByName('TABELA').Text) +
                                   #13 + '        Chave: ' +
ArquivoVala.FieldByName('CHAVE').Text + ' - Registro já existe!' +
                                   #13 + '                Processo cancelado!
', mtError, [mbOk], 0);
                  end;
                  //Se tudo ocorreu normalmente flega registro como já transmitido
                  if (FlegaRegistro = true) then
                  begin
                    ArquivoVala.RLock;
                    ArquivoVala.Edit;
                    MemoEnvia.Lines.Add('   Flegando registro como transmitido...');
                    ArquivoVala.FieldByName('DATATRANS').AsString := DateToStr(Date);
                    ArquivoVala.FieldByName('HORATRANS').AsString := TimetoStr(Time);
                    ArquivoVala.RUnLock;
                    ArquivoVala.Post;
                  end;
```

```
                  end;
                end
                else
                  MessageDlg('Tabela: ' + uppercase(ArquivoVala.FieldByName('TABELA').Text) +
' com mais de 1 registro não possui chave primária' +
                             #13 + '                          Processo de atualização
cancelado!                        ', mtError, [mbOk], 0);
                FechaDbf(ArquivoOrigem, aIndices);
              end;
            end
            else
            begin
              MessageDlg('Tabela: ' + uppercase(ArquivoVala.FieldByName('TABELA').Text) + '
não cadastrada no arquivo de catalogo' +
                #13 + '                          Processo de atualização cancelado!
', mtError, [mbOk], 0);
            end;
            ArquivoVala.Next;
          end;
          FechaDbf(Catalogo, aIndexCatalogo);
        end;
      end
      else
        MemoEnvia.Lines.Add('   Nenhum registro disponível para atualização');
      DesconectaBanco(ConexaoBanco);
    end
    else
    begin
      MessageDlg('Não foi possível estabeler conexão com o Banco de Dados!' +
                 #13 + 'Processo de atualização de tabelas cancelado!', mtError, [mbOk], 0);
    end;
    FechaDbf(ArquivoVala, aIndexVala);
  end;
  BitBtnRetornar.Enabled := true;
  TimerEnvia.Enabled := true;
  TimerEnvia.Interval := 10000;  {10 segundos}
end;
```

Anexo E – Código fonte confeccionado em Delphi para o recebimento de atualizações vindas do SGBD

```
procedure TFormRecebe.Recebimento();
var
  nRegistro, nOrdemKey, nAux, nIni, nCont, nPosCampo : integer;
  CaminhoDbf, StrCampoChave, Caracter, StrSQLSelect, StrAuxiliar, NomeCampo : String;
  Conectou, Prosseguir, lInsere, lAtualiza, FlegaRegistro : boolean;
  DMLQueryTrans, DMLQueryRecord, DMLQueryFlag : TZQuery;
  UpdateFlagTrans : TZUpdateSQL;
  data_transmissao : String;
  aIndices, aIndexCatalogo, aCamposChave, aSeek : Array of String;
begin
  MemoRecebe.Lines.Clear;
  TimerRecebe.Enabled := false;
  BitBtnRetornar.Enabled := false;
  //Prosseguir := true;
  aIndexCatalogo := nil;

  DMLQueryFlag  := TZQuery.Create(self);
  UpdateFlagTrans := TZUpdateSQL.Create(self);

  //Insere := false;
  Conectou := ConectaBanco(ConexaoBanco);
  if (Conectou = true) then
  begin
    MemoRecebe.Lines.Add('');
    MemoRecebe.Lines.Add('   Verificando atualizações disponíveis no SGBD...');
    DMLQueryTrans := TZQuery.Create(self);
    try
      DMLQueryTrans.Active := false;
      DMLQueryTrans.SQL.Clear;
      DMLQueryTrans.Connection := ConexaoBanco;
      DMLQueryTrans.SQL.Add('SELECT * FROM sgbd_dbf where datatrans is NULL');
      //DMLQueryTrans.ExecSQL;
      DMLQueryTrans.Active := true;
    except
      on E: EZSQLException do
      begin
        if (E.ErrorCode <> -1) then
        begin
          MessageDlg(E.Message, mtWarning, [mbOk], 0);
        end;
      end;
    end;
    DMLQueryTrans.First;
    if (not DMLQueryTrans.Eof) then
    begin
      //Abre Catalogo para poder abrir os índices do arquivo
      MemoRecebe.Lines.Add('   Abrindo Catalogo de Arquivos...');
      SetLength(aIndexCatalogo, length(aIndexCatalogo) + 1);
      aIndexCatalogo[length(aIndexCatalogo) - 1] := 'arqdbf.ntx';
      Catalogo := TVKDBFNTX.Create(self);
      Catalogo.SetDeleted := true;
      Prosseguir := AbreDbf(Catalogo, 'c:\projeto\arqdbf.dbf', aIndexCatalogo, 0);
      if (Prosseguir = true) then
      begin
        while (not DMLQueryTrans.Eof) do
        begin
          MemoRecebe.Lines.Add('   Selecionando arquivo para recebimento...');
          nRegistro :=
Catalogo.Orders[0].FindKeyFields('TABELA;CAMINHONTX',[trim(uppercase(DMLQueryTrans.FieldByName
('TABELA').Text))]);
          if nRegistro > 0 then
            Catalogo.RecNo := nRegistro
          else
            Prosseguir := false;
          if (Prosseguir = true) then
          begin
            aIndices := nil;
            StrCampoChave := '';
            nOrdemKey := 0;
            CaminhoDbf := trim(Catalogo.FieldByName('CAMINHODBF').Text);
```

```
            while ((not Catalogo.Eof) and (trim(Catalogo.FieldByName('TABELA').Text) =
trim(uppercase(DMLQueryTrans.FieldByName('TABELA').Text)))) do
            begin
              SetLength(aIndices, length(aIndices) + 1);
              aIndices[length(aIndices) - 1] := trim(Catalogo.FieldByName('CAMINHONTX').Text);
              if (Catalogo.FieldByName('PRIMARYKEY').Value = true) then
              begin
                StrCampoChave := trim(Catalogo.FieldByName('CAMPOCHAVE').Text);
                nOrdemKey := length(aIndices) - 1;
              end;
              Catalogo.Next;
            end;

            //Abre o arquivo neste instante para que possa pegar o tipo do campo
            ArquivoRec := TVKDBFNTX.Create(self);
            Arquivorec.SetDeleted := true;
            Prosseguir := AbreDbf(ArquivoRec, CaminhoDbf, aIndices, nOrdemKey);
            MemoRecebe.Lines.Add('   Abrindo arquivo: ' + CaminhoDbf);

            for nAux := 0 to length(aIndices) - 1 do
              MemoRecebe.Lines.Add('   Índice aberto para este arquivo: ' + aIndices[nAux]);

            if (length(aIndices) > 0) then
              MemoRecebe.Lines.Add('   Índice definido como chave primária: ' +
aIndices[nOrdemKey]);


            if (Prosseguir = true) then
            begin
              //Seleciona registro na origem (SGBD)
              DMLQueryRecord := TZQuery.Create(self);
              try
                MemoRecebe.Lines.Add('   Selecionando registro na origem (SGBD)...');
                DMLQueryRecord.Active := false;
                DMLQueryRecord.SQL.Clear;
                DMLQueryRecord.Connection := ConexaoBanco;
                DMLQueryRecord.SQL.Add('SELECT * FROM ');

DMLQueryRecord.SQL.Add(trim(lowercase(DMLQueryTrans.FieldByName('TABELA').Text)));
                //Se tabela possuir indice pega a chave primaria e gera clausula where
                if (length(StrCampoChave) > 0) then
                begin
                  DMLQueryRecord.SQL.Add(' WHERE ');
                  nIni := 1;
                  nCont := 0;
                  StrSQLSelect := '';
                  for nAux := 1 to length(trim(StrCampoChave)) do
                  begin
                    Caracter := Copy(StrCampoChave, nAux, 1);
                    if (Caracter = ';') then
                    begin
                      if (Copy(StrCampoChave, nIni, (nAux - nIni)) <> '') then
                      begin
                        StrSQLSelect := StrSQLSelect + lowercase(Copy(StrCampoChave, nIni,
(nAux - nIni)));
                        StrSQLSelect := StrSQLSelect + '=:p' + InttoStr(nCont) + ' AND ';
                        //Guardando Campos em uma matriz
                        SetLength(aCamposChave, nCont + 1);
                        aCamposChave[nCont] := Copy(StrCampoChave, nIni, (nAux - nIni));
                        nCont := nCont + 1;
                      end;
                      nIni := nAux + 1;
                    end;
                  end;
                  if ((nAux - 1)  = length(StrCampoChave)) then
                  begin
                    StrSQLSelect := StrSQLSelect + lowercase(Copy(StrCampoChave, nIni, (nAux -
nIni))) + '=:p' + inttoStr(nCont);
                    SetLength(aCamposChave, nCont + 1);
                    aCamposChave[nCont] := Copy(StrCampoChave, nIni, (nAux - nIni));
                  end;
                  DMLQueryRecord.SQL.Add(StrSQLSelect);

                  nCont := 0;
                  nIni := 1;
                  nAux := 1;
```

```
                  aSeek := nil;
                  //Montagem dos parametros do Where da chave primária (portabilidade
alcançada)
                  while (nAux <= length(DMLQueryTrans.FieldByName('chave').Text)) do
                  begin
                    Caracter := Copy(DMLQueryTrans.FieldByName('chave').Text, nAux, 1);
                    if (Caracter = ';') or (nAux =
length(trim(DMLQueryTrans.FieldByName('chave').Text))) then
                    begin
                      if (nAux = length(trim(DMLQueryTrans.FieldByName('chave').Text))) then
                        nAux := nAux + 1;
                      if (Copy(DMLQueryTrans.FieldByName('chave').Text, nIni, (nAux - nIni))
<> '') then
                      begin
                        if
(ArquivoRec.DBFFieldDefs.Items[ArquivoRec.FieldByName(aCamposChave[nCont]).FieldNo -
1].field_type = 'C') then
                          begin
                            if (length(trim(Copy(DMLQueryTrans.FieldByName('chave').Text, nIni,
(nAux - nIni)))) > 0) then
                              DMLQueryRecord.Params[nCont].AsString :=
Copy(DMLQueryTrans.FieldByName('chave').Text, nIni, (nAux - nIni))
                            else
                            begin
                              DMLQueryRecord.Params[nCont].AsString := '';
                              if (TipoCaracter = 'NULL') then
                                DMLQueryRecord.Params[nCont].Value := NULL;
                            end;
                          end
                          else if
(ArquivoRec.DBFFieldDefs.Items[ArquivoRec.FieldByName(aCamposChave[nCont]).FieldNo -
1].field_type = 'N') then
                          begin
                            if (length(trim(Copy(DMLQueryTrans.FieldByName('chave').Text, nIni,
(nAux - nIni)))) > 0) then
                            begin
                              if
(ArquivoRec.DBFFieldDefs.Items[ArquivoRec.FieldByName(aCamposChave[nCont]).FieldNo - 1].dec >
0) then
                                DMLQueryRecord.Params[nCont].AsFloat :=
StrtoFloat(Copy(DMLQueryTrans.FieldByName('chave').Text, nIni, (nAux - nIni)))
                              else
                                DMLQueryRecord.Params[nCont].AsInteger :=
StrtoInt(Copy(DMLQueryTrans.FieldByName('chave').Text, nIni, (nAux - nIni)));
                            end
                            else
                            begin
                              //Tipos numericos nos arquivos DBF´s Nativos do Clipper sempre são
0 (mesmo configurando como nulo)
                              if
(ArquivoRec.DBFFieldDefs.Items[ArquivoRec.FieldByName(aCamposChave[nCont]).FieldNo - 1].dec >
0) then
                                DMLQueryRecord.Params[nCont].AsFloat := 0
                              else
                                DMLQueryRecord.Params[nCont].AsInteger := 0;
                              DMLQueryRecord.Params[nCont].Value := NULL;
                            end;
                          end
                          else if
(ArquivoRec.DBFFieldDefs.Items[ArquivoRec.FieldByName(aCamposChave[nCont]).FieldNo -
1].field_type = 'D') then
                          begin
                            if (length(trim(Copy(DMLQueryTrans.FieldByName('chave').Text, nIni,
(nAux - nIni)))) > 0) then
                            begin
                              StrAuxiliar := Copy(DMLQueryTrans.FieldByName('chave').Text, nIni,
(nAux - nIni));
                              DMLQueryRecord.Params[nCont].AsString := Copy(StrAuxiliar,1,4) +
'-' + Copy(StrAuxiliar,5,2) + '-' + Copy(StrAuxiliar,7,2);
                            end
                            else
                            begin
                              //Não é necessario testar se o tipodata é null pois se for data
padrao nao entra neste if
                              DMLQueryRecord.Params[nCont].AsString := '';
                              DMLQueryRecord.Params[nCont].Value := NULL;
```

```
                          end;
                        end
                        else if
(ArquivoRec.DBFFieldDefs.Items[ArquivoRec.FieldByName(aCamposChave[nCont]).FieldNo -
1].field_type = 'L') then
                        begin
                          if (Copy(DMLQueryTrans.FieldByName('chave').Text, nIni, (nAux -
nIni)) = 'true') then
                            DMLQueryRecord.Params[nCont].AsBoolean := true
                          else
                          begin
                            if (Copy(DMLQueryTrans.FieldByName('chave').Text, nIni, (nAux -
nIni)) = 'false') then
                              DMLQueryRecord.Params[nCont].AsBoolean := false
                            //Não é necessario testar se o tipologico é null pois se for null
automaticamente é false no Clipper
                            else
                            begin
                              DMLQueryRecord.Params[nCont].AsString := '';
                              DMLQueryRecord.Params[nCont].Value := NULL;
                            end;
                          end;
                        end
                        else if
(ArquivoRec.DBFFieldDefs.Items[ArquivoRec.FieldByName(aCamposChave[nCont]).FieldNo -
1].field_type = 'M') then
                        begin
                          if (length(trim(Copy(DMLQueryTrans.FieldByName('chave').Text, nIni,
(nAux - nIni)))) > 0) then
                            DMLQueryRecord.Params[nCont].AsString :=
Copy(DMLQueryTrans.FieldByName('chave').Text, nIni, (nAux - nIni))
                          else
                          begin
                            DMLQueryRecord.Params[nCont].AsString := '';
                            if (TipoMemo = 'NULL') then
                              DMLQueryRecord.Params[nCont].Value := NULL;
                          end;
                        end;
                      end;
                      SetLength(aSeek, nCont + 1);
                      //Monta matriz para busca no DBF
                      aSeek[nCont] := Copy(DMLQueryTrans.FieldByName('chave').Text, nIni,
(nAux - nIni));
                      nCont := nCont + 1;
                      nIni := nAux + 1;
                    end;
                    nAux := nAux + 1;
                  end;
                end;
                DMLQueryRecord.Active := true;
              except
                on E: EZSQLException do
                begin
                  if (lowercase(E.Message) <> 'can not open a resultset') then
                  begin
                    MessageDlg(E.Message, mtWarning, [mbOk], 0);
                  end;
                end;
              end;
              lInsere := false;
              lAtualiza := false;
              FlegaRegistro := false;
              //Verifica existencia de registro na tabela origem
              if (DMLQueryRecord.RecordCount > 0) then
              begin
                MemoRecebe.Lines.Add('   Procurando registro no destino:
'+uppercase(trim(CaminhoDbf)));

                //Se possuir indice
                if (length(StrCampoChave) > 0) then
                BEGIN
                  nRegistro := ArquivoRec.Orders[nOrdemKey].FindKeyFields(StrCampoChave,
aSeek);
                END
                else
                begin
```

```
                  //Se nao possuir, verifica se existe somente um registro, se existir
atualiza
                  //para mais de um registro ou nenhum registro appenda
                  if (ArquivoRec.RecordCount = 1) then
                    nRegistro := 1
                  else
                    nRegistro := 0;
                end;
                //Se encontrou registro
                if (nRegistro > 0) then
                begin
                  if (uppercase(DMLQueryTrans.FieldByName('operacao').Text) = 'U') then
                  begin
                    ArquivoRec.RecNo := nRegistro;
                    if (ArquivoRec.Deleted = false) then
                    begin
                      ArquivoRec.RLock;
                      ArquivoRec.Edit;
                      lAtualiza := true;
                    end
                    else
                    begin
                      ArquivoRec.Next;
                      MessageDlg('Tipo de Operação: ' +
uppercase(DMLQueryTrans.FieldByName('operacao').Text) + ' para a Tabela: ' +
uppercase(DMLQueryTrans.FieldByName('tabela').Text) +
                                 #13 + '    Inválida para a chave: ' +
DMLQueryTrans.FieldByName('chave').Text +
                                 #13 + '        Registro encontra-se deletado - Processo
cancelado!                 ', mtError, [mbOk], 0);
                    end;
                  end
                  else  if (uppercase(DMLQueryTrans.FieldByName('operacao').Text) = 'D') then
                  begin
                    MemoRecebe.Lines.Add('   Removendo registro...');
                    ArquivoRec.RecNo := nRegistro;
                    ArquivoRec.RLock;
                    ArquivoRec.Edit;
                    ArquivoRec.Deleted := true;
                    ArquivoRec.RUnLock;
                    ArquivoRec.Post;
                    FlegaRegistro := true;
                    MemoRecebe.Lines.Add('   Registro removido com sucesso!!')
                  end
                  else
                    MessageDlg('Tipo de Operação: ' +
uppercase(DMLQueryTrans.FieldByName('operacao').Text) + ' para a Tabela: ' +
uppercase(DMLQueryTrans.FieldByName('tabela').Text) +
                                 #13 + '    Inválida para a chave: ' +
DMLQueryTrans.FieldByName('chave').Text +
                                 #13 + '                 Processo cancelado!                ',
mtError, [mbOk], 0);
                end
                else
                begin
                  if (uppercase(DMLQueryTrans.FieldByName('operacao').Text) = 'I') then
                  begin
                    ArquivoRec.Append;
                    ArquivoRec.RLock;
                    lInsere := true;
                  end
                  else
                    MessageDlg('Tipo de Operação: ' +
uppercase(DMLQueryTrans.FieldByName('operacao').Text) + ' para a Tabela: ' +
uppercase(DMLQueryTrans.FieldByName('tabela').Text) +
                                 #13 + '    Inválida para a chave: ' +
DMLQueryTrans.FieldByName('chave').Text +
                                 #13 + '                 Processo cancelado!                ',
mtError, [mbOk], 0);
                end;

                if (lInsere or lAtualiza) then
                begin
                  MemoRecebe.Lines.Add('   Inserindo/Atualizando registro...');

                  for nAux := 0 to (ArquivoRec.DBFFieldDefs.Count - 1) do
```

```
                  begin
                    //Pego o nome do campo que vou procurar na tabela resultado
                    NomeCampo := ArquivoRec.DBFFieldDefs.Items[nAux].Name;
                    //Pego a posicao deste campo na tabela resultado
                    nPosCampo := ArquivoRec.FieldByName(NomeCampo).FieldNo;
                    if (ArquivoRec.DBFFieldDefs.Items[nAux].field_type = 'M') then
                    begin
                      //É necessario gravar em um TMEMO antes do DBF
                      CampoMemo.Clear;
                      with CampoMemo.Lines do begin
                        Add(DMLQueryRecord.Fields[nPosCampo - 1].Value);
                      end;
                      ArquivoRec.Fields[nPosCampo - 1].Assign(CampoMemo.Lines);
                    end
                    else if (ArquivoRec.DBFFieldDefs.Items[nAux].field_type = 'N') then
                    begin
                      if (DMLQueryRecord.Fields[nAux].Value <> NULL) then
                      begin
                        if (ArquivoRec.DBFFieldDefs.Items[nAux].dec > 0) then
                          ArquivoRec.Fields[nPosCampo - 1].AsFloat :=
DMLQueryRecord.Fields[nAux].Value
                        else
                          ArquivoRec.Fields[nPosCampo - 1].AsInteger :=
DMLQueryRecord.Fields[nAux].Value;
                      end;
                    end
                    else
                    begin
                      ArquivoRec.Fields[nPosCampo - 1].Value :=
DMLQueryRecord.Fields[nAux].Value;
                    end;
                  end;
                  ArquivoRec.RUnLock;
                  ArquivoRec.Post;
                  FlegaRegistro := true;
                  if lInsere then
                    MemoRecebe.Lines.Add('   Inserção realizada com sucesso!!')
                  else
                    MemoRecebe.Lines.Add('   Atualização realizada com sucesso!!');
                end;
              end;
              FechaDbf(ArquivoRec, aIndices);
              //Neste ponto é necessario flegar o registro da vala como transmitido
              if (FlegaRegistro) then
              begin
                try
                  DMLQueryFlag.Active := false;
                  DMLQueryFlag.SQL.Clear;
                  DMLQueryFlag.Connection := ConexaoBanco;
                  DMLQueryFlag.SQL.Add('UPDATE sgbd_dbf SET datatrans = :p0, horatrans = :p1
WHERE id_processo = :p2');
                  //data_transmissao
                  DMLQueryFlag.Params[0].asDate := date;
                  DMLQueryFlag.Params[1].asString := timetostr(time);
                  //id_processo
                  DMLQueryFlag.Params[2].asInteger :=
DMLQueryTrans.FieldByName('id_processo').Value;
                  DMLQueryFlag.Fields.Clear;
                  DMLQueryFlag.Active := true;
                except
                  on E: EZSQLException do
                  begin
                    if (lowercase(E.Message) <> 'can not open a resultset') then
                    begin
                      MessageDlg(E.Message, mtWarning, [mbOk], 0);
                    end;
                  end;
                end;
              end;
            end;
          end
          else
            MessageDlg('Tabela: ' + uppercase(DMLQueryTrans.FieldByName('TABELA').Text) + '
não cadastrada no arquivo de catalogo' +
                  #13 + '                        Processo de atualização cancelado!
', mtError, [mbOk], 0);
```

```
          DMLQueryTrans.Next;
        end;
        FechaDbf(Catalogo, aIndexCatalogo);
      end;
    end
    else
    begin
      MemoRecebe.Lines.Add('   - SGBD sem atualizações diponíveis!!!');
    end;
    DMLQueryTrans.Destroy;
    DesconectaBanco(ConexaoBanco);
  end
  else
  begin
    MessageDlg('Não foi possível estabeler conexão com o Banco de Dados!' +
               #13 + 'Processo de atualização de tabelas cancelado!', mtError, [mbOk], 0);
  end;
  UpdateFlagTrans.Destroy;
  DMLQueryFlag.Destroy;
  BitBtnRetornar.Enabled := true;
  TimerRecebe.Enabled := true;
  TimerRecebe.Interval := 10000;  {10 segundos}
end;
```