UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO PARA ADMINISTRAÇÃO DE SGBD POSTGRESQL

PAULO FERNANDO ODWAZNY

BLUMENAU
2003

2003/2

PAULO FERNANDO ODWAZNY

PROTÓTIPO PARA ADMINISTRAÇÃO DE SGBD POSTGRESQL

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Alexander Roberto Valdameri

BLUMENAU
2003

2003/2

PROTÓTIPO PARA ADMINISTRAÇÃO DE SGBD POSTGRESQL

Por

PAULO FERNANDO ODWAZNY

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Alexander Roberto Valdameri
Membro:	Prof. Marcel Hugo, FURB
Membro:	Prof. Marcelo Ferrari, FURB

Blumenau, 05 de Novembro de 2003

Dedico este trabalho as pessoas que me ajudaram diretamente e indiretamente na sua realização. Meus Pais Asnelda Odwazny e Carlos Romeu Odwazny, minha irmã Vania Mara Odwazny e minha namorada Petra Cristina Selke por sempre acreditarem no meu potencial e terem sempre me incentivado para a realização do mesmo.

Uma vida não questionada não merece ser vivida.

Platão

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me proporcionado a oportunidade de estar concluindo mais uma importante etapa na minha vida.

Aos meus Asnelda Odwazny e Carlos Romeu Odwazny por terem me dado muita força durante esta jornada.

Agradeço a minha Irmã, Vania Mara Odwazny e minha Namorada, Petra Cristina Selke, pela paciência e confiança depositada.

Ao meu orientador Prof Alexander Roberto Valdameri por toda a atenção e incentivo na orientação disponibilizada para o desenvolvimento deste trabalho.

Agradeço a todos que ajudaram diretamente e indiretamente na minha formação universitária, tornando a universidade mais culta e amistosa.

RESUMO

O presente trabalho apresenta o desenvolvimento um protótipo de uma ferramenta para a administração de um SGBD PostgreSQL. Tendo como principal objetivo gerenciar bases de dados, permitindo criar, alterar e apagar tabelas. Foram estudados as técnicas da linguagem de programação Java, mais especificamente o software JBuilder da Borland e o SGBD free PostgreSQL.

Palavras Chave: Banco de Dados; PostgreSQL; Java; Borland JBuilder.

ABSTRACT

The present work displays the development of a prototype of a tool to a SGBD PostgreSQL administration. As main objective, it will manage database, allowing one to create, change or delete tables. The language techniques of Java program were studied, more specificaly the Borland JBuilder software and the SGBD free PostgreSQL.

Key-Words: Data Base; PostgreSQL; Java; Borland JBuilder.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

SGBD – Sistema de Gerenciamento de Banco de Dados

SQL – Linguagem Estruturada de Consulta (*Structured Query Language*)

SUMÁRIO

1 INTRODUÇÃO

Este capítulo fornece uma visão geral do assunto abordado neste trabalho. Na contextualização, ainda tem-se o relato dos principais objetivos e a estrutura do trabalho.

1.1 CONTEXTUALIZAÇÃO

Atualmente a informação é muito importante para todo e qualquer ramo de atividade, por conseqüência, os dados devem ser armazenados de forma segura, proporcionando assim informações corretas. Uma das formas de se armazená-los é em um banco de dados.

De acordo com Suehring (2002), bancos de dados mantêm as informações em tabelas. Uma tabela é uma estrutura que consiste em pelo menos uma coluna, mas normalmente mais de uma. Um banco de dados é, portanto, uma coleção de uma ou mais tabelas de informações relacionadas.

Há inúmeros bancos de dados atualmente no mercado. Para gerenciar esses bancos de dados surgiram os Sistemas Gerenciadores de Bancos de Dados (SGBD). Os SGBDs são uma coleção de ferramentas que possibilitam criar e manter um banco de dados, em outras palavras, é a ferramenta que trata todo e qualquer acesso ao banco de dados.

Um dos SGBDs que mais tem evoluído em termos de funcionalidades e utilização mercadológica é o PostgreSQL (GESCHWINDE, 2002). Aliado a tais fatos e, considerando que o mesmo é um sistema livre e com o código fonte aberto, vislumbra-se uma significativa ascensão na sua utilização no desenvolvimento de sistemas de informações comerciais.

Segundo Geschwinde (2002) em 1986 foram feitas as implementações iniciais do PostgreSQL, com base em outro banco de dados da Universidade da Califórnia chamado Ingres. Foram feitas várias mudanças no PostgreSQL, dentre as quais se destacam:

a) tipos de dados para operações geométricas foram adicionados;
b) a velocidade de resposta foi aumentada significativamente;
c) características ANSI SQL92 foram adicionadas.

O PostgreSQL é considerado o mais avançado sistema de banco de dados com código fonte aberto do mundo (WORSLEY, 2002). Há um significativo número de desenvolvedores que trabalham efetivamente com o PostgreSQL e este número está em constante crescimento (GESCHWINDE, 2002).

Todavia, o banco de dados PostgreSQL não apresenta em sua estrutura de funcionamento uma interface "amigável" para gerenciamento das funcionalidades, tornando a sua utilização um desafio para os projetistas e desenvolvedores. Neste sentido, este trabalho apresenta o desenvolvimento de um software de suporte, cuja funcionalidade seja facilitar a interação entre o usuário (administrador de banco de dados) e o banco de dados, oferecendo em um ambiente gráfico as funcionalidades que o mesmo dispõe.

Para a realização deste trabalho fez-se necessário a utilização de uma linguagem de programação para o desenvolvimento de uma interface entre o PostgreSQL e o usuário. A utilização de uma linguagem de programação orientada a objeto é sugerida por diversos autores pela facilidade do desenvolvimento, grande capacidade de reutilização, confiabilidade, robustez, extensibilidade e armazenabilidade.

Segundo Newman (1997) Java é uma linguagem de programação simples, orientada a objetos, distribuída, interpretada, potente, segura, de arquitetura neutra, portátil, de alto desempenho, multiencadeada e dinâmica.

Para o desenvolvimento do trabalho utilizou-se a metodologia de Orientação a Objeto através da *Unified Modeling Language* (UML). Como ferramenta de desenvolvimento com suporte a linguagem Java utilizou-se o ambiente de desenvolvimento Borland JBuilder.

1.2 OBJETIVOS

O objetivo do Trabalho de Conclusão de Curso é desenvolver uma ferramenta multiplataforma, com uma interface amistosa para o usuário, capaz de manipular as estruturas e os dados em um SGBD PostgreSQL.

Os objetivos específicos do trabalho são:

a) disponibilizar uma interface gráfica para interação do usuário com o SGBD;

b) disponibilizar as principais funcionalidades do SGBD na ferramenta em nível de *Data Definition Language* (DDL) e *Data Manipulation Language* (DML).

1.3 ESTRUTURA DO TRABALHO

Este trabalho está divido em forma de capítulos descritos a seguir.

Primeiro capítulo expõe na introdução uma justificativa do que originou este trabalho como também uma síntese do que será tratado no desenvolvimento do trabalho. Os objetivos a serem alcançados.

O segundo capítulo apresenta o SGBD PostgreSQL, suas principais características e funcionalidades.

O terceiro capítulo faz uma breve exposição do gerenciador gráfico do SGBD PostgreSQL chamado pgAdmin.

O quarto capítulo apresenta a especificação feita para o desenvolvimento do trabalho e apresenta o protótipo com suas funcionalidades.

O quinto capítulo expõe a que conclusão chegou-se após o desenvolvimento do trabalho e algumas sugestões para continuação.

2 SGBD POSTGRESQL

Este capítulo descreve um pouco da história do SGBD PostgreSQL, trazendo também suas principais características, tecnologias implementadas e também um pouco sobre sua instalação.

2.1 HISTÓRICO

Segundo Worsley et al (2003), PostgreSQL é um *Object-Relational Database Management System* (ORDBMS), ou seja, um Sistema Gerenciador de Base de Dados Objeto Relacional desenvolvido desde 1977. Ingres foi o nome dado ao projeto inicial, desenvolvido na Universidade da Califórnia em Berkeley. Mais tarde foi atualizado, atendendo o universo comercial, pela Rational Technologe/Ingres Corporation.

Em 1986 um grupo liderado por Michael Stonebraker de Berkeley, continuou desenvolvendo o Ingres com o propósito de torná-lo uma Base de Dados Objeto-Relacional chamado Postgres. Foi em 1996 que o Postgres ficou pronto, chamando-se PostgreSQL, que está em constante atualização (WORSLEY, 2002).

2.2 PRINCIPAIS CARACTERÍSTICAS

Segundo Worsley et al (2003), algumas características do PostgreSQL são:

a) objeto-relacional (ORDBMS) - PostgreSQL trata os dados como um modelo Objeto-Relacional e possibilita a manipulação de rotinas e regras complexas. Exemplos destas avançadas funcionalidades são o "SQL query" declarativo, controle de concorrência multi-versão, suporte a multi-usuário, otimização de "query", herança e listas (*arrays*);
b) altamente extensível - PostgreSQL suporta operadores, funções, métodos de acesso e tipos de dados;
c) suporte SQL compreensível - PostgreSQL suporta a especificação SQL99 e inclui características avançadas desde SQL92;
d) integridade referencial - é usado para validar os dados de uma base de dados;
e) API flexível - a flexibilidade da API do PostgreSQL determina fácil suporte ao desenvolvimento do PostgreSQL ORDBMS. Tais interfaces incluem: Object Pascal (Delphi), Python, Pearl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++ e Pike;
f) linguagem de procedimentos (*Procedure*) - suporta linguagem procedural interna, incluindo uma linguagem nativa chamada PL/pgSQL. Outra vantagem do PostgreSQL e a capacidade de usar Pearl, Python ou TCL como linguagem de procedimentos;
g) MVCC - *Multi-Version Concurrency Control*, é a tecnologia que PostgreSQL usa para não precisar de trava (*locking*). Outros Sistemas Gerenciadores de Bancos de Dados (SGBD) como MySQL ou Access, as vezes bloqueiam os dados fazendo o usuário esperar uma resposta, isto é, o "Leitor" (*Reader*) do PostgreSQL fica bloqueado para atualizar os registros. PostgreSQL mantém-se a

par de todas as transações executadas pelo usuário na base de dados, por isso está apto a manusear os registros sem causar o atraso;

h) cliente / servidor - PostgreSQL utiliza um processo por usuário, a arquitetura cliente / servidor tem um processo principal que dispara outros processos adicionas para cada conexão;

i) write ahead log (WAL) - se houver um erro na base de dados, este será um registro de uma transação que será restaurada, isto traz bons benefícios para o evento do erro. Se qualquer mudança for feita na base de dados, pode ser recuperado usando os arquivos de log. Uma vez restaurado, o usuário continua trabalhando do ponto antes do erro ter ocorrido.

2.3 TECNOLOGIAS

De acordo com Worsley (2002), sendo comparado com os diversos SGBDs que existem no mercado, principalmente com os livres, o PostgreSQL agrega um grande nicho de tecnologias e funcionalidades, antes existentes somente em SGBDs corporativos e próprios, como por exemplo o Oracle.

Além de criar, alterar e apagar base de dados, usuários, tabelas e dados, existem ainda outras características, como:

a) agregações "*Aggregate"* que permitem ao usuário criar funções de agregação como por exemplo a média de todas as linhas de uma determinada coluna *avg(idade)*;

b) grupos de usuários *"Group"*, onde operadores de mesmas características são cadastrados;

c) transformações "*Cast"*, onde são permitidos fazer transformações do tipo texto para inteiro e vice versa;

d) conversões *"Conversion"*, são conversões de tipo de codificação como por exemplo a codificação UNICODE para LATIN1;

e) domínio *"Domain"*, tipo de dado subjacente ao domínio que o operador pode criar, sendo assim mais padronizado, por exemplo, a criação de um tipo "código" que será um inteiro de 4 posições;

f) funções *"Function"*, para descrever um comando SQL que irá ser usado diversas vezes;
g) índices *"Index"*, utilizados para ter um retorno mais rápido das informações;
h) linguagem procedural *"Language"*, linguagens procedurais são locais a cada banco de dados. Para tornar uma linguagem disponível a todos os bancos de dados por padrão, esta linguagem deve ser instalada no banco de dados "template1";
i) operador *"Operator"*, pode ser criado um operador definido pelo usuário, assim como "=", ">" ou "<"; j) regra *"Rule"*, permite a definição de uma ação alternativa a ser realizada para inclusões, atualizações ou exclusões nas tabelas do banco de dados. As regras também são utilizadas para implementar as visões das tabelas;
k) esquema *"Shema"*, um espaço de nomes: contém objetos nomeados (tabelas, tipos de dado, funções e operadores) cujos nomes podem duplicar os nomes de outros objetos existentes em outros esquemas, seqüência *"Sequence"*, gerador de números seqüenciais no banco de dados em uso;
l) gatilho *"Trigger"*, os gatilhos são usados normalmente para dispararem antes ou depois de algum comando, por exemplo, antes de criar uma tabela, verificar se ela já não existe;
m) tipo de dados *"Type"*, o usuário pode criar um novo tipo de dados, como por exemplo, um inteiro de uma posição;
n) visões *"View"* , que é uma abreviatura de uma seleção (WORSLEY, 2002).

2.4 INSTALAÇÃO

De acordo com Worsley (2002), a instalação do PostgreSQL não é difícil, todavia, é preciso alguns softwares para a sua instalação. Se a plataforma Linux é a utilizada, existem grandes chances de que todas os softwares já estejam instalados. Se for utilizar derivações do BSD, como FreeBSD ou MacOS X, deverá ser feito o *download* desses softwares que são:

a) GNU make – programas distribuídos na forma de pacotes tar.gz de código fonte seguem um procedimento de instalação próprio. Estes programas precisam ser

compilados no sistema do usuário, para serem posteriormente instalados. Utilitários como o GNU make, também conhecido como gmake, automatizam este procedimento;

b) ISO/ANSI C Compiler – existem vários compiladores ISO/ANSI C, mas o mais recomendável para o PostgreSQL é o GNU C Compiler;

c) GNU zip and tar – conhecido com gzip, é um utilitário para comprimir e descomprimir arquivos. Todos os arquivos, "zipados", compressos pelo gzip tem a extensão *.gz*. Junto com o gzip será preciso uma cópia do tar, O tar funciona como um "arquivador", transferindo conjuntos de arquivos para um determinado arquivo de destino. Os arquivos tar normalmente tem a extensão *.tar* e compresso com o gzip tem-se a extensão *.tar.gz*, que é o caso do PostgresSQL.

Na versão Win32 do PostgreSQL, faz se necessário a utilização do software Cygwin, que emula o Linux no ambiente Windows. Apesar do Cygwin ser usado em muitas situações, o uso do PostgreSQL junto com o Cygwin não é recomendado. Existem previsões de que a versão 7.5 poderá ser instalada no Windows sem essa camada (WORSLEY, 2002).

2.5) METADADOS

O *metadados* do SBGD PostgreSQL se localiza na base de dados *template1* e as tabelas estão assim divididas:

a) *pg_aggregate*, tabela responsável pelos dados de agregações;
b) *pg_am*, tabela de índices de métodos de acesso;
c) *pg_amop*, tabela de métodos de acesso dos operadores;
d) *pg_amproc*, tabela de procedimentos;
e) *pg_attrdef*, tabela de valores padrão das colunas;
f) *pg_attribute*, tabela de atributos de colunas;
g) *pg_cast*, tabela de conversão de tipos de dados
h) *pg_class*, tabela de tabelas, índices, relacionamentos;
i) *pg_constraint*, tabela de chaves primarias, chaves estrangeiras;
j) *pg_conversion*, tabela de informações sobre conversões;

k) *pg_database*, tabela de bases de dados;
l) *pg_depend*, tabela de dependências de objetos de base de dados;
m) *pg_description*, tabela de comentários sobre os objetos de base de dados;
n) *pg_group*, tabela de grupos de usuários;
o) *pg_index*, tabela de informações adicionais dos índices;
p) *pg_inherits*, tabela de hierarquia de herança;
q) *pg_language*, tabela de linguagens de funções;
r) *pg_largeobject*, tabela de objetos;
s) *pg_listener*, tabela de notificações de suporte assíncrono;
t) *pg_namespace*, tabela de esquemas do SGBD;
u) *pg_opclass*, tabela de índices das classes de métodos de acesso dos operadores;
v) *pg_operator*, tabela de operadores;
w) *pg_proc*, tabela de procedimentos e funções;
x) *pg_shadow*, tabela de usuários das bases de dados;
y) *pg_trigger*, tabela de gatilhos;
z) *pg_type*, tabela de tipos de dados.

3 FERRAMENTAS DE ADMINISTRAÇÃO EXISTENTES

Existe uma ferramenta administrativa de SGBD´s no mercado, por exemplo, para o SGBD PostgreSQL, existe o pgAdmin III.

Em 1998 iniciou-se o projeto de construção da ferramenta pgAdmin III, que hoje em dia é utilizado por mais de 50000 usuários no mundo. O pgAdmin III é atualizado com base mas sugestões destes usuários. Segundo o *site* do pgAdmin III, desenvolvedores com mais de 10 anos de experiência em ferramentas para banco de dados estão trabalhando no projeto.

O pgAdmin III é um software livre e tem versões Windows, FreeBSD, Linux e Mac OS.

Para demonstrar a funcionalidade do pgAmin III, seguem algumas ilustrações.

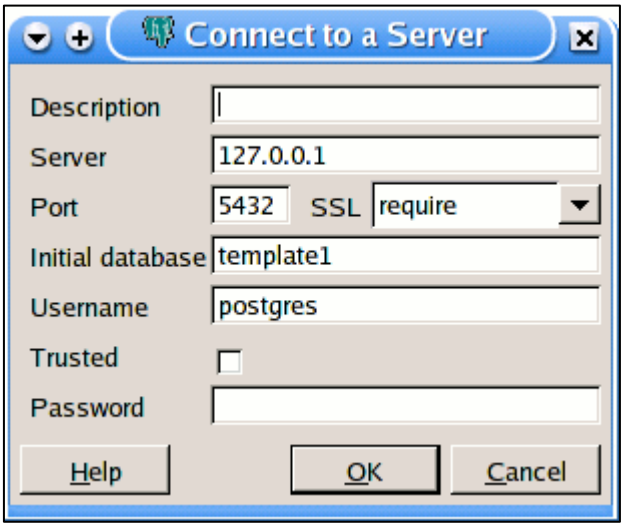

FIGURA 1 – Conexão do pgAdmin

A figura 1 mostra a tela de conexão do software com o banco de dados. O pgAdmin pode se conectar as versões 7.3 e 7.4 do PostgreSQL.

Na figura 2 pode-se notar os objetos do banco de dados, mostrando por exemplo as bases de dados e as tabelas.

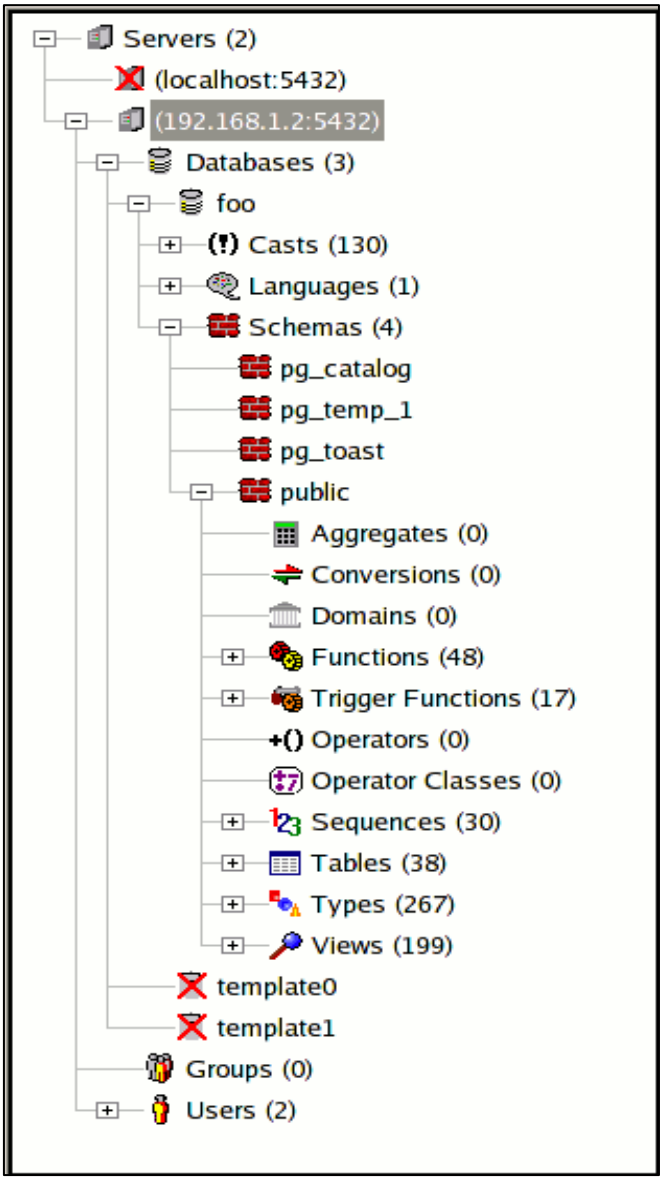

FIGURA 2 – Árvore de objetos

Na figura 3 ve-se as propriedades de cada objeto, como por exemplo, as propriedades da tabela “translationforge_changelog”.

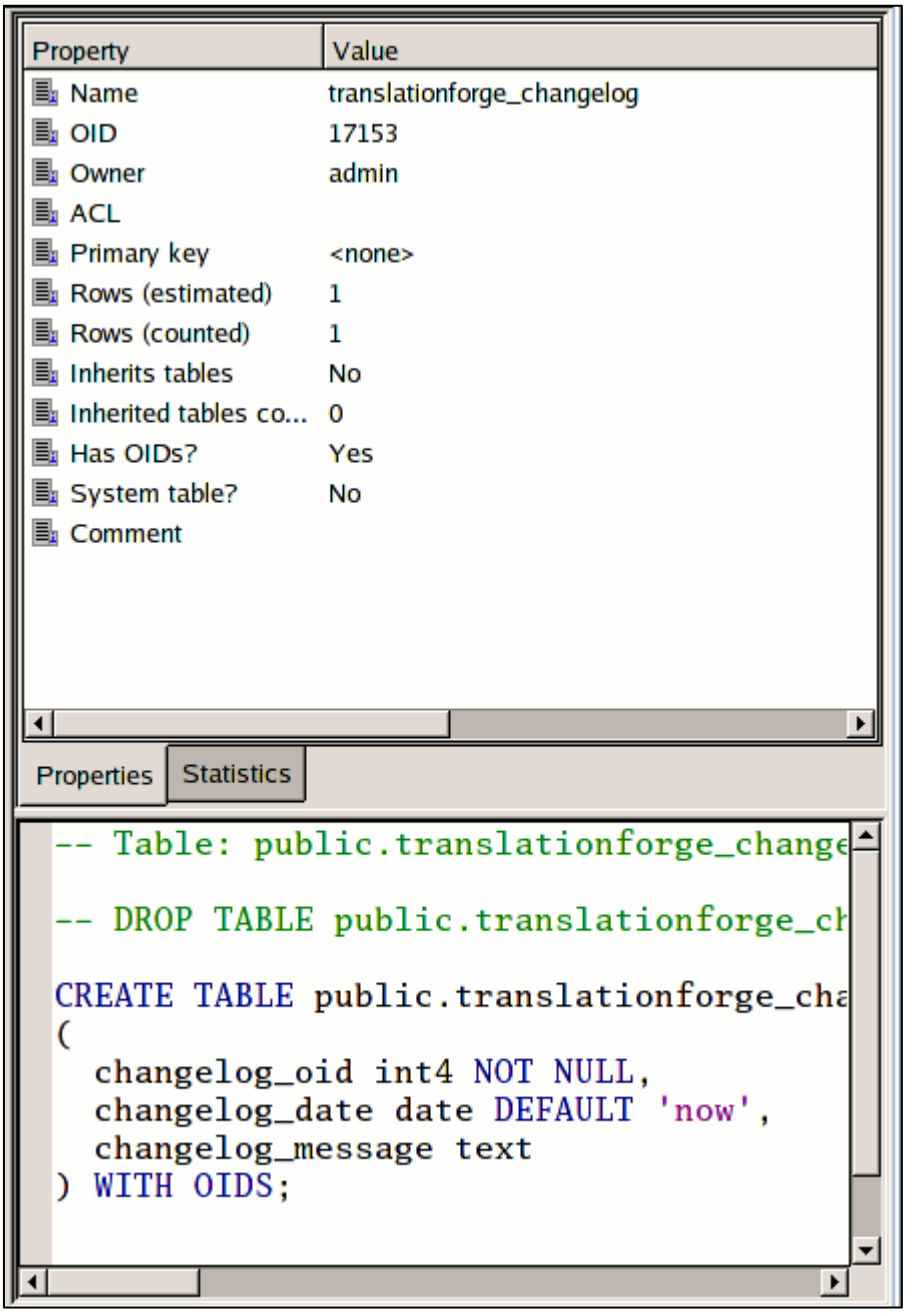

FIGURA 3 – Propriedades da tabela translationforge_changelog

Ao se clicar 2 vezes na árvore de objetos, pode-se modificar as propriedades deste objeto, como se pode observar na figura 4.

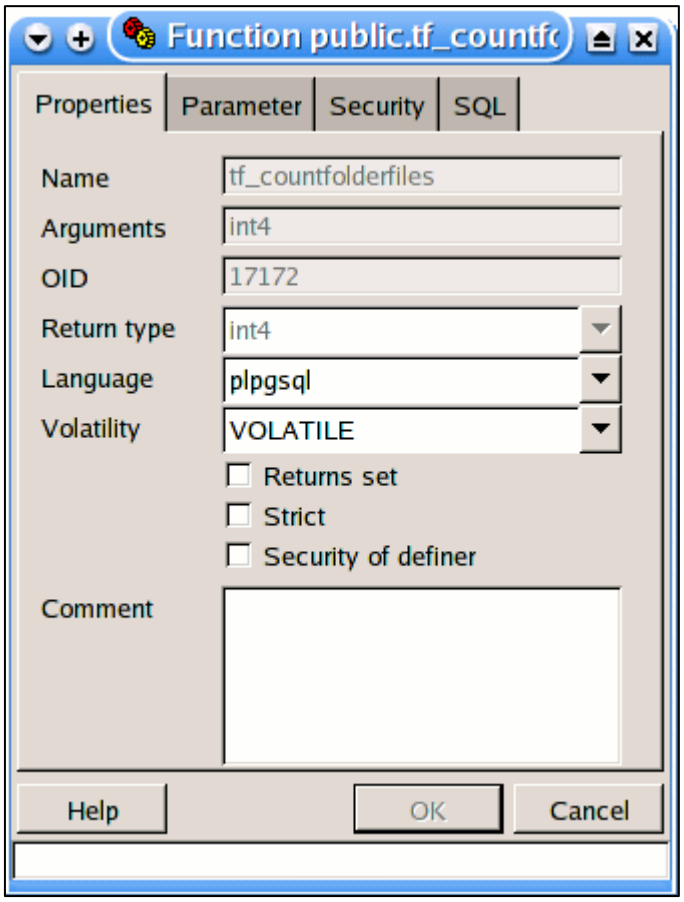

FIGURA 4 – Alteração de propriedades de um objeto

A figura 5 ilustra a criação de uma tabela no pgAdmin III, apresentando especificamente a criação dos campos.

No pgAdmin III foi englobado todas as funcionalidades do PostgreSQL, como a manipulação de dados, a manipulação da estrutura de dados e o controle de usuários.

Uma boa interface gráfica, estar em constante atualização e ser um software livre são as principais características do pgAdmin III.

No pgAdmin III utiliza-se a mesma interface com o usuário para a criação e a alteração das tabelas.

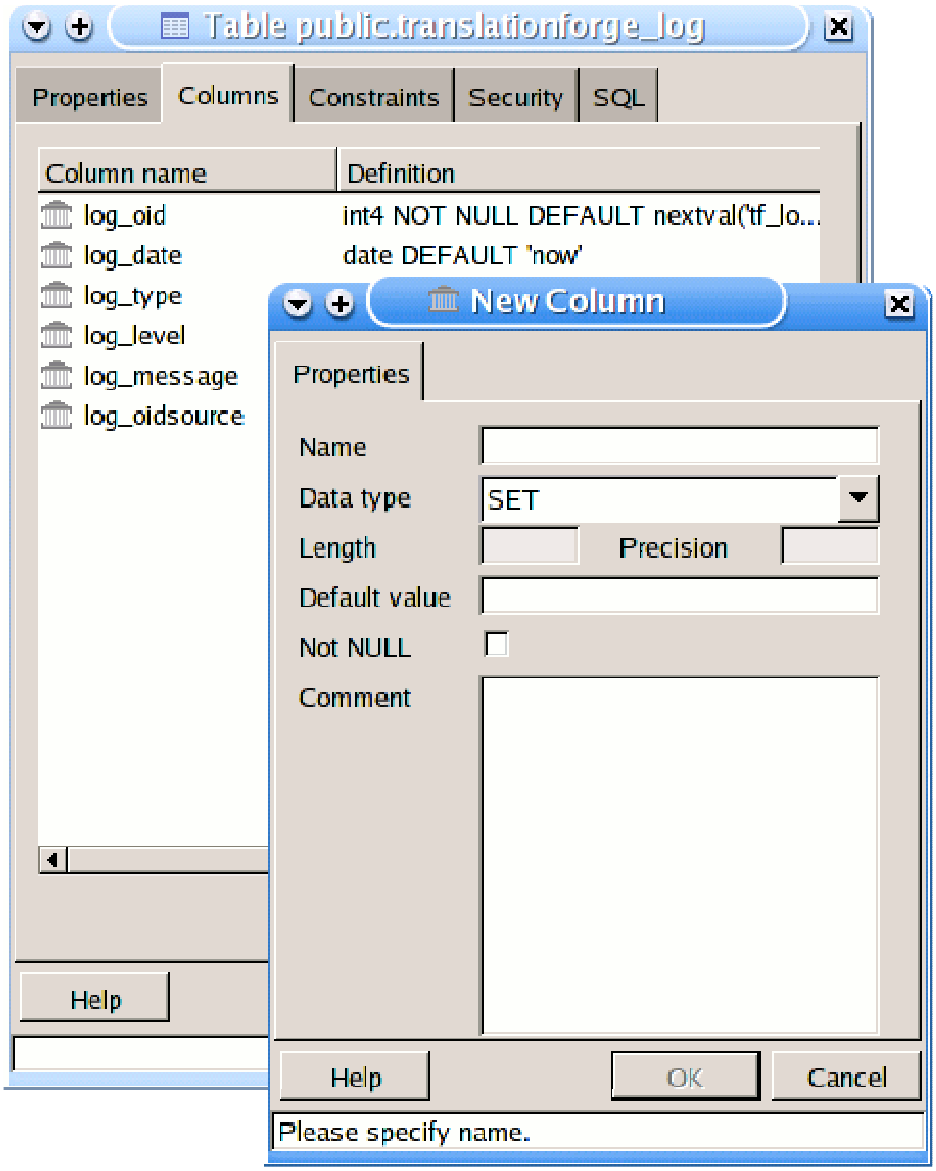

FIGURA 5 – Criação de tabelas

O pgAdmin é uma das ferramentas mais indicadas para a administração de um banco de dados PostgreSQL, porém tem algumas restrições como, por exemplo, o fato de não ser multi-conexão, podendo-se abrir somente uma conexão no pgAdmin.

O software pode ser adquirido pela internet ou através da distribuidora oficial do PostgreSQL no Brasil chamada DBExperts. Esta empresa oferece, a um determinado preço, o pgAdmin III e livros traduzidos para o português sobre o PostgreSQL.

4 DESENVOLVIMENTO DO TRABALHO

O desenvolvimento do trabalho é a parte fundamental do trabalho, onde é descrito todo o processo de desenvolvimento do protótipo, iniciando pela especificação, implementação e apresentação.

4.1 ESPECIFICAÇÃO

A ferramenta administrativa do SGBD visa atender uma categoria específica de usuários: o administrador de banco de dados (*Database Administrator* - DBA).

O DBA é responsável por (DATE, 1989):

a) decidir qual o conteúdo do banco de dados;

b) definir a estrutura e estragias de acesso do banco de dados;

c) definir verificações de autorização e níveis de acesso;

d) definir estratégia de backup e recuperação;

e) monitorar o desempenho do bando de dados.

Nesta primeira versão, o Frog atenderá as necessidades de DBA, mais diretamente relacionadas a linguagem de manipulação de dados (LMD) e de definição de dados (LDD).

Todas as funcionalidades tem por objetivo deixar o banco de dados PostgreSQL mais fácil de ser manipulado, oferecendo uma interface gráfica onde se pode ter uma melhor visão dos dados e da estrutura do banco de dados.

As funcionalidades estão mais aparentes no protótipo, ou seja, os comandos SQL podem ser executados clicando sobre um botão no protótipo. O protótipo por sua vez não analisa o comando, apenas repassa esse comando para o SGBD que então faz a análise verificando eventuais erros. O retorno do SGBD também passa pelo protótipo que então mostra para o usuário do protótipo.

4.1.1 REQUISITOS

A ferramenta Frog, tem como objetivo:

a) facilitar a visualização de bases de dados e tabelas;

b) permitir que o usuário escreva comandos no formato SQL;

c) criar, apagar e ver as propriedades de uma base de dados;

d) criar, apagar e alterar as propriedades de uma tabela;

e) ter a característica multi-plataforma;

f) ter a capacidade de múltiplas conexões com o SGBD.

Tendo seu principal foco voltado para os dados, com o Frog o usuário, ou melhor, desenvolvedor, pode acessar os dados de uma tabela com maior facilidade.

Por ser um SGBD multi-plataforma, o Frog atinge um maior alvo de usuários, fazendo a ferramenta administrativa ter uma maior popularidade na comunidade de desenvolvedores em PostgreSQL.

Outra funcionalidade do Frog é a sua capacidade de poder abrir mais conexões com diferentes SGBDs PostgreSQL, fazendo com que o administrador tenha uma real noção da magnitude de todos os SGBDs PostgreSQL que administra.

4.1.2 DIAGRAMA DE CASOS DE USO

Os casos de uso têm como propósito descrever os requisitos funcionais do sistema de maneira consensual entre usuários e desenvolvedores de sistemas visando fornecer uma descrição consistente e clara sobre as responsabilidades que devem ser cumpridas pelo sistema (FURLAN, 1998).

A Figura 6 representa os casos de uso que fazem parte do protótipo onde se definem as funções do sistema. Os diagramas de casos de uso foram especificados utilizando a ferramenta Rational Rose.

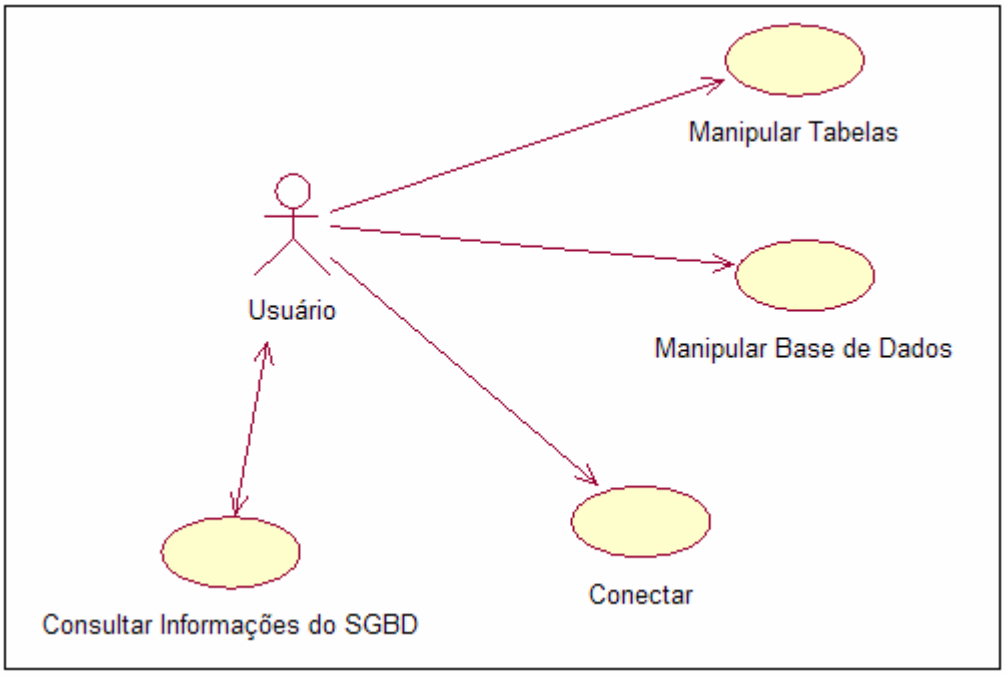

FIGURA 6 – Diagrama de casos de uso

A tabela 1 descreve cada caso de uso com o autor e nome dos casos.

TABELA 1 – Descrição do use case

Caso de Uso	Ator	Descrição
Manipular Tabela	Usuário	Usuário manipula a estrutura da tabela, podendo criar, alterar ou apagar tabelas
Manipular Base de Dados	Usuário	Usuário cria ou apaga as bases de dados existentes no SGBD
Conectar	Usuário	Usuário conecta-se em um servidor de PostgreSQL
Consultar Informações do SGBD	Usuário	Relatar as principais características do SGBD, como por exemplo, nome das base de dados, nome das tabelas ou nome dos campos das tabelas.

4.1.3 DIAGRAMA DE CLASSES

Segundo Furlan (1998), o diagrama de classes é representado por uma estrutura lógica e estática mostrando uma coleção de elementos declarativos de modelo, como classe,

tipos e seus respectivos conteúdos e relações. Estruturar atributos e operações em classes é fundamental para o trabalho de modelagem através do enfoque da orientação a objeto.

Na figura 7 pode-se observar através da representação no diagrama de classes a estrutura de classes do sistema, bem como seus eventos e atributos.

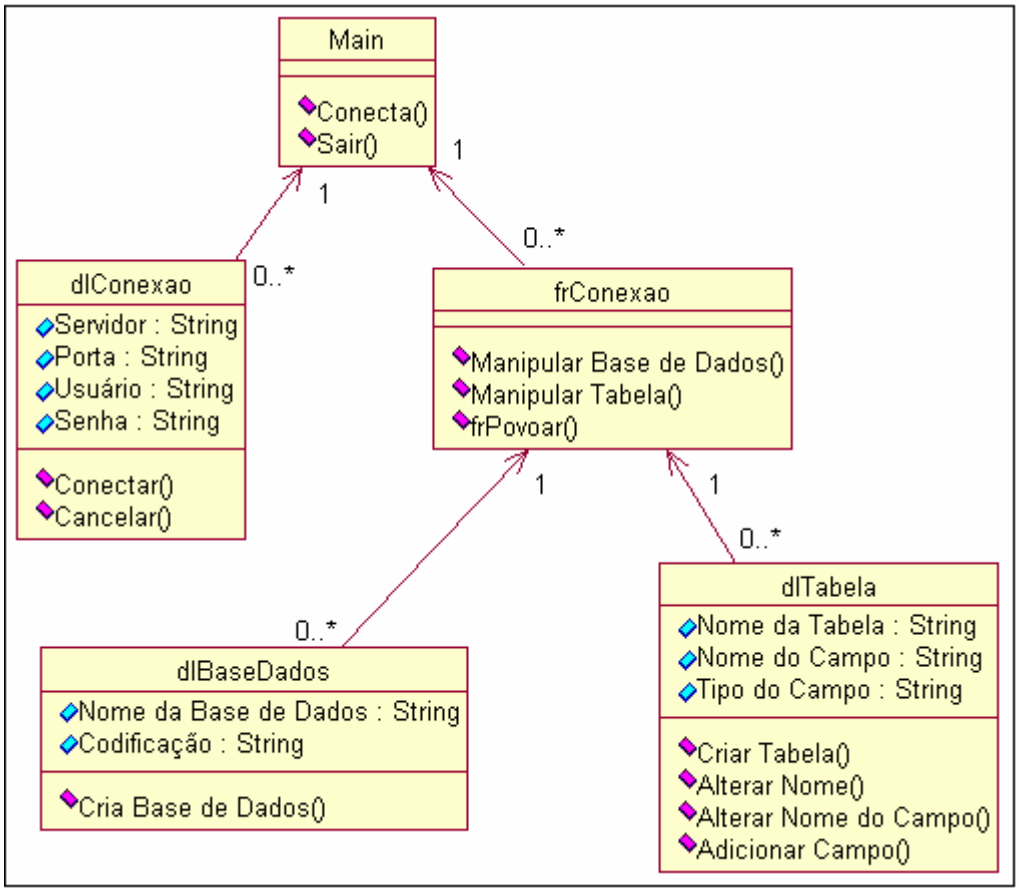

FIGURA 7 – Diagrama de classes

A classe *Main* é a classe principal do protótipo, ao se executar o protótipo é a classe *Main* que é iniciada. Para se fazer a conexão do protótipo com o SGBD é usada a classe *dlConexão*. A classe *frConexao* é a classe que mostra as características das bases de dados e tabelas. Na classe *frConexao* há o método *frPovoar* que é responsável pela atualização da árvore de visualização do banco de dados que será visto a seguir. As classes *dlBaseDados* e *dlTabelas* são responsáveis pela manipulação da estrutura das bases de dados e das tabelas respectivamente.

4.2 IMPLEMENTAÇÃO DO PROTÓTIPO DO SOFTWARE

O protótipo do software foi desenvolvido utilizando-se a ferramenta de desenvolvimento Borland JBuilder 9 e seus componentes para a interação com o banco de dados para a sua manipulação.

Para a comunicação do JBuilder com o banco de dados PostgreSQL foi utilizado o componente da Borland *Database* e o *QueryDataSet* para manipulação dos dados.

Ao iniciar o software, o usuário deve informar o *hostname* do banco de dados, a porta do banco de dados, o usuário e a senha. Estas informações são captadas pela classe *dlConexao* e passadas para a classe *frConexão* quer irá iniciar a conexão com o banco de dados.

Após a conexão estar estabelecida, o método *frPovoa* é executado para povoar a árvore de bases de dados e de tabelas.

Assim o software espera o usuário executar algum comando SQL para poder manipular os dados ou até mesmo criar, alterar ou apagar atributos de algum objeto no banco de dados.

Como o protótipo é um software de gerenciamento de um SGBD PostgreSQL, para seu funcionamento será necessário um SGBD PostgreSQL versão 7.3 instalado localmente num micro-computador ou instalado num micro-computador em estado remoto.

Na inicialização do software são instanciadas três classes chamadas *frMain*, *frConexao* e *dlConexao*. A classe *frMain* será a classe principal do programa. Ao ser instanciada a classe *dlConexão*, é passada como parâmetro a classe *frConexao* como é visto na figura 8.

```
void mnConectar_actionPerformed(ActionEvent e){
  frConexao frameConexao = new frConexao();
  frameConexao.setSize(new Dimension(700, 450));
  frameConexao.setOpaque(true);
  frameConexao.getContentPane().setLayout(null);
  frameConexao.setClosable(true);
  frameConexao.setIconifiable(true);
  frameConexao.setLayer(0);
  frameConexao.setMaximizable(true);
  frameConexao.setResizable(true);
  frameConexao.setEnabled(true);
  frameConexao.setAlignmentY((float) 0.5);
  frameConexao.setDoubleBuffered(true);
  frameConexao.setRequestFocusEnabled(false);
  frameConexao.setToolTipText("");
  frameConexao.setVerifyInputWhenFocusTarget(false);
  desktop.add(frameConexao);

  dlConexao dlg = new dlConexao(frameConexao);
  Dimension dlgSize = dlg.getPreferredSize();
  Dimension frmSize = getSize();
  Point loc = getLocation();
  dlg.setLocation( (frmSize.width - dlgSize.width) / 2 + loc.x,
                   (frmSize.height - dlgSize.height) / 2 + loc.y);
  dlg.setModal(true);
  dlg.show();
}
```

FIGURA 8 – Instanciação das classes frConexao e dlConexao

Na figura 8 pode-se notar a instanciação da classe *frConexao* pelo comando *frConexao frameConexao = new frConexao()*, os demais comandos são propriedades visuais da classe e o comando *desktop.add(frameConexao)* é responsável pela multi-conexão do software, onde *frames* são adicionados ao programa principal. *dlConexao dlg = new dlConexao(frameConexao)*, é a parte onde a classe *dlConexao* é instanciada e junto com a instanciação é mandada a classe *frConexão*.

A classe *dlConexao* é responsável pela conexão do protótipo com o SGBD e pela inicialização da classe *frConexao*. Para fazer essa inicialização, é passada classe *frConexao* no *constructor* da classe *dlConexao*, como é verificado na figura 9.

```
public dlConexao(frConexao frConexao) {
 this(null, "", false);
 frame = frConexao;
}
```

FIGURA 9 – Classe frConexao na classe dlConexao

Na classe *dlConexao* são recolhidos os dados do usuário, como *hostname*, porta do banco de dados, usuário e senha, como é visto na figura 10. Essas informações serão levadas para a classe *frConexão* e esta irá se conectar com o SGBD PostgreSQL descrito pelo usuário.

```
void btConectar_actionPerformed(ActionEvent e) {
  frame.Host = edServidor.getText();
  frame.Port = edPorta.getText();
  frame.setLocation(10, 10);
  frame.User = edUsuario.getText();
  frame.Password = edSenha.getText();
  try {
     frame.frPovoar();
    }
  catch (Exception ex1) {

  }
  frame.setVisible(true);
  cancel();
}
```

FIGURA 10 – Passagem das informações para frConexao

O atributo chamada frame, como foi visto anteriormente, referencia a classe *frConexao*. Nesta há 4 atributos importantes para a conexão software com o SGBD PostgreSQL que são Host, Port, User e Password. Na figura 10 é visto ainda o método *frPovoar* que será visto com maiores detalhes nas figuras seguintes.

O *frPovoar*, é na realidade um método para inicializar a árvore que mostra para o usurário as bases de dados e tabelas existentes no SGBD.

Esta árvore tem como raiz as palavras Banco de Dados; logo após, o primeiro filho será o conjunto do nome do usuário com o símbolo ‘@’ e o nome do host que o SBGD PostgreSQL está localizado; o filho deste será a base de dados existentes no SBGD. Os

filhos das bases de dados serão as tabelas existentes nelas. As tabelas serão o último ramo desta árvore.

Na figura 11 é mostrada a instanciação dos componentes *Database* e *QueryDataSet*, na inicialização da classe *frConexao*, que são responsáveis pela comunicação com o SGBD e manipulação dos dados, consecutivamente.

```
public class frConexao extends JInternalFrame {
  Database db = new Database();
  Database dbTabelas = new Database();
  QueryDataSet query = new QueryDataSet();
  QueryDataSet queryTabelas = new QueryDataSet();
```

FIGURA 11 – Instanciação dos componentes

São instanciados duas *Database* e duas *QueryDataSet* para poder manipular as bases de dados existentes e as tabelas nas bases de dados.

Na figura 12 é visto a instanciação da raiz da árvore chamada *top* e da árvore propriamente dita chamada *trInformacao*.

```
DefaultMutableTreeNode top = new DefaultMutableTreeNode("Banco de Dados");
JTree trInformacao = new JTree(top);
```

FIGURA 12 – Instanciação da árvore

Já falado anteriormente, o método chamado *frPovoar* é utilizado para inicializar a árvore chamada *trInforma*. A primeira parte do método, descrita na figura 13, tem como objetivo fazer a conexão do software com o SGBD.

```
public void frPovoar() throws Exception{
// povoar a tree
 this.setTitle(User+"@"+Host);
 aux = new DefaultMutableTreeNode(User+"@"+Host);
 // ADD NA TREEVIEW O USUARIO @ HOSTNAME
 if (top.getChildCount() == 0){
  top.add(aux);
 }
 dbTabelas.setConnection(new com.borland.dx.sql.dataset.
       ConnectionDescriptor("jdbc:postgresql://" + Host + ":" + Port + "/template1",
       User , Password, false,"org.postgresql.Driver"));
 dbTabelas.setDatabaseName("");

 db.setConnection(new com.borland.dx.sql.dataset.
     ConnectionDescriptor("jdbc:postgresql://" + Host + ":" + Port + "/template1",
     User , Password, false,"org.postgresql.Driver"));
 db.setDatabaseName("");
```

FIGURA 13 – Inicio do método frPovoar

Para se conectar no SGBD PostgreSQL, é necessário informar a base de dados que é desejado conectar. A base de dados "template1" é padrão em todos os SBGD´s PostgreSQL, por esse motivo a primeira conexão se dará sempre nela.

Além de fazer a conexão, na figura 13 é mostrado a adição do nodo chamado *aux* com o nome do usuário e o nome do host na raiz árvore, isto é feito pelos comandos *aux = new DefaultMutableTreeNode(User+"@"+Host)* e *top.add(aux)*.

```
// SELECIONA TODAS AS BASES DE DADOS
  query.closeStatement();
  query.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(dbTabelas,
     "SELECT d.datname as \"Name\", u.usename as \"Owner\" \nFROM pg_database " +
     "d LEFT JOIN pg_user u ON d.datdba = u.usesysid order by d.datname ", null, true,
     Load.ALL));
  query.executeQuery();
  DefaultMutableTreeNode node = new DefaultMutableTreeNode(query.getString("Name"));

  aux.add(node);
```

FIGURA 14 – Selecionando todas as bases de dados

Na figura 14 pode-se descrever o comando SQL para fazer a seleção do nome de todas as bases de dados existentes do SGBD. Cada nome de bases de dados selecionado é atribuído para a variável *node* do tipo *DefaultMutableTreeNode*. A variável *node* é

adicionada à variável *aux*, tornando-se assim filha desta. Isto é verificado no comando *aux.add(node)*. A variável *node* sempre vai estar relacionada a uma base de dados.

Na seqüência a variável *dbTabelas* é conectada em cada base de dados obtida pela variável *query*, vista na figura 18. Esta conexão pode ser vista na figura 15.

```
dbTabelas.setConnection(new com.borland.dx.sql.dataset.ConnectionDescriptor(
    "jdbc:postgresql://" + Host + ":" + Port + "/" +
    query.getString("Name"),
    db.getConnection().getUserName(), db.getConnection().getPassword(), false,
    "org.postgresql.Driver"));
```

FIGURA 15 – Conexão em cada base de dados obtida pela *query*

Com a variável *queryTabelas* são descobertas, por um comando SQL, as tabelas que estão em cada base de dados e estas tabelas são adicionadas a árvore, sendo filhas da variável *node*, como é visto na figura 16.

```
queryTabelas.closeStatement();
queryTabelas.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(
    dbTabelas, "select relname as \"Name\" from pg_class where relname not like \'pg_%\' and relkind " +
    "= \'r\' order by relname", null, true,
    Load.ALL));
queryTabelas.executeQuery();
if (!queryTabelas.isEmpty()) {
  DefaultMutableTreeNode child = new DefaultMutableTreeNode(queryTabelas.getString("Name"));
  node.add(child);
  while (!queryTabelas.atLast()) {
    queryTabelas.next();
    child = new DefaultMutableTreeNode(queryTabelas.getString("Name"));
    node.add(child);
  }
}
else {
  DefaultMutableTreeNode child = new DefaultMutableTreeNode(" ");
  node.add(child);
}
```

FIGURA 16 – Seleção das tabelas de cada base de dados

A variável *child* é responsável pela adição das tabelas na árvore. Se a base de dados não tiver nenhuma tabela criada por um usuário, como é o caso da base de dados “template1” e “template0”, é adicionada uma linha em branco na árvore, como é visto na figura 17.

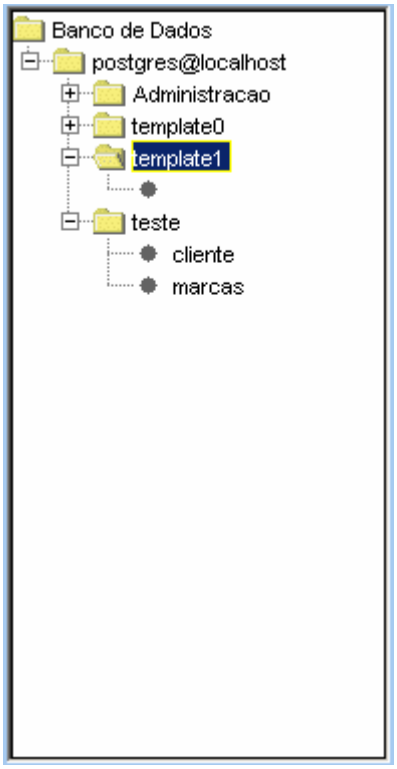

FIGURA 17 – Árvore de bases de dados e tabelas do SGBD PostgreSQL

Esse processo de conectar-se na base de dados selecionada por *query*, utizar *queryTabelas* para descobrir as tabelas relacionadas a esta base de dados e adicionar tudo a árvore, é feito tantas vezes quanto o total de bases de dados tiver no SGBD. Por exemplo, na figura 17 o *query* armazena todas as bases de dados, “Administração”, “template0”, “template1” e “teste”. O *queryTabelas* irá utilizar as bases de dados selecionadas por *query* para descobrir quais são as tabelas vinculadas a cada uma das bases, assim num determinado momento *queryTabelas* armazena “cliente”, “marcas” que são as tabelas da base de dados “teste”. Isto pode ser visto nas figuras 18 e 19.

```
do {
  query.next();
  queryTabelas.empty();

  node = new DefaultMutableTreeNode(query.getString("Name"));
  aux.add(node);

  if ( (!query.getString("Name").equalsIgnoreCase("template0")) &&
     (!query.getString("Name").equalsIgnoreCase("template1"))) {

    dbTabelas.closeConnection();
    dbTabelas.setConnection(new com.borland.dx.sql.dataset.
                        ConnectionDescriptor(
       "jdbc:postgresql://" + Host + ":" + Port + "/" +
       query.getString("Name"),
       db.getConnection().getUserName(), db.getConnection().getPassword(), false,
       "org.postgresql.Driver"));
    dbTabelas.setDatabaseName("");
    queryTabelas.closeStatement();
    queryTabelas.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(
       dbTabelas, "select relname as \"Name\" from pg_class where relname not like \'pg_%\' and relkind " +
       "= \'r\' order by relname", null, true,
       Load.ALL));
    queryTabelas.executeQuery();
  }
```

FIGURA 18 – Conexão na base de dados e seleção das tabelas da mesma

```
if (!queryTabelas.isEmpty()) {
       DefaultMutableTreeNode child = new DefaultMutableTreeNode(queryTabelas.getString("Name"));
       node.add(child);
       while (!queryTabelas.atLast()){
         queryTabelas.next();
         child = new DefaultMutableTreeNode(queryTabelas.getString("Name"));
         node.add(child);
       }

     }
     else {
       DefaultMutableTreeNode child = new DefaultMutableTreeNode(" ");
       node.add(child);
     }
   }while (!query.atLast());
```

FIGURA 19 – Comando para base de dados vazia

Ainda na classe *frConexao*, existem guias para a interface com o usuário, nelas se encontram as propriedades do nodo selecionado na árvore, a possibilidade do usuário escrever algum comando SQL, uma guia para o resultado deste comando se ele for um comando de seleção e uma guia de status para verificar se o comando SQL foi executado com sucesso, isto pode ser visto na figura 20.

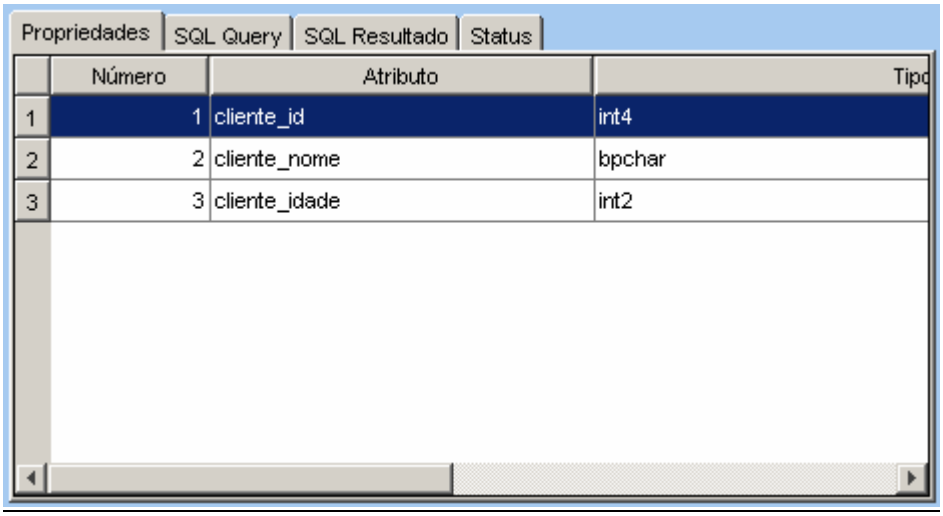

FIGURA 20 – Guias de interação do usuário com o software Frog

O comando para obter o nodo selecionado da árvore está descrito na figura 21.

```
this.nodo = (DefaultMutableTreeNode)this.trInformacao.getLastSelectedPathComponent();
```

FIGURA 21 – Obter o nome do nodo selecionado

Para poder mostrar as propriedades do nodo selecionado da árvore é preciso saber primeiramente qual o objeto que está sendo selecionado, por exemplo, se é uma base de dados ou uma tabela.

A solução foi encontrar a paternidade dos nodos. O nodo raiz sempre vai ter as palavras *Banco de Dados* associadas a ele. O seu filho será sempre o nome do usuário mais o nome do *host*. Os filhos deste serão as bases de dados e o resto serão as tabelas. Este raciocínio lógico é demonstrado nas figuras a seguir.

Se a raiz da árvore estiver selecionada, quer dizer, se o nome do *nodo* for *Banco de Dados*, na guia chamada propriedades aparecerá o nome do usuário e o nome do *host*, como está descrito na figura 22.

```
// BANCO DE DADOS (MOSTRA O HOST E O USER NAME)
    if (nodo.toString().equalsIgnoreCase("Banco de Dados")){
      query.closeStatement();;
      query.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(dbTabelas,
          "SELECT '"+User+"@"+Host+"' as \"              Servidor            \" ", null, true,
          Load.ALL));
      query.executeQuery();
      // (DEIXA A BASE DE DADOS EM BRANCO)
       textBd.setText("sem base de dados selecionada");
    }
```

FIGURA 22 – Se a raiz da árvore estiver selecionada

Se o nodo for filho direto da raiz, ou melhor, se o nome do nodo for o conjunto de nome do usuário, "@" e nome do *host*, será mostrado na guia Propriedades nome das bases de dados do SGBD, como é visto na figura 23.

```
else{
      // (MOSTRA AS BASES DE DADOS)
      if (nodo.toString().equalsIgnoreCase(User+"@"+Host)){
        query.closeStatement();;
        query.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(dbTabelas,
           "SELECT d.datname as \"Descrição da base de Dados\", u.usename as \"Owner\" \nFROM pg_database " +
           "d LEFT JOIN pg_user u ON d.datdba = u.usesysid order by d.datname ", null, true,
            Load.ALL));
        query.executeQuery();

        // (DEIXA A BASE DE DADOS EM BRANCO)
        textBd.setText("sem base de dados selecionada");
      }
```

FIGURA 23 – Se o filho da raiz estiver selecionado

Para descobrir se o nodo selecionado é uma base de dados, é preciso somente testar se o pai deste tem como texto o nome do usuário, a "@" e o nome do *host*. Como está descrito na figura 24.

```
else{
  //(MOSTRA AS TABELAS DA BASE DE DADOS SELECIONADA)
  if (nodo.getParent().toString().equalsIgnoreCase(User + "@" + Host)) {
```

FIGURA 24 – Se uma base de dados estiver selecionada

Se o nodo selecionado for uma base de dados serão mostradas na guia Propriedades as tabelas pertencentes a esta base de dados, como é visto na figura 25.

```
db.setConnection(new com.borland.dx.sql.dataset.
            ConnectionDescriptor("jdbc:postgresql://" + Host +
    ":" + Port + "/" +
    nodo.toString(),
    User, Password, false,
    "org.postgresql.Driver"));
db.setDatabaseName("");
query.closeStatement();
query.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(
    db, "select relname as \"Descrição das Tabelas\" from pg_class where relname" +
    "not like \'pg_%\' and relkind " +
    "= \'r\' order by relname", null, true, Load.ALL));
query.executeQuery();
if (nodo.toString().equalsIgnoreCase("template1")){
  query.closeStatement();
  query.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(dbTabelas,
       "SELECT 'Base de Dados do Sistema' as \"          Descrição          \" ", null, true,
       Load.ALL));
  query.executeQuery();
```

FIGURA 25 – Mostra as tabelas da base de dado selecionada na guia Propriedades

Sendo a base de dados selecionada do sistema, “template0” ou “template1”, aparecerá na guia de Propriedades a mensagem “Base de dados do Sistema”.

E se o nodo não for nenhuma das opções acima, este será uma tabela e na guia Propriedades deverão aparecer as principais características dos campos desta, como nome do campo, tipo, etc. Isto é visto na figura 26.

```
else{
 // (MOSTRA AS PROPRIEDADES DA TABELA SELECIONADA)
 db.closeConnection();
 db.setConnection(new com.borland.dx.sql.dataset.
             ConnectionDescriptor("jdbc:postgresql://" + Host +
                                  ":" + Port + "/" +
                                  nodo.getParent().toString(),
                                  User, Password, false,
                                  "org.postgresql.Driver"));
 db.setDatabaseName("");

 query.closeStatement();

 query.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(
    db, "SELECT a.attnum as \"Número\", a.attname as \"Atributo\", CASE WHEN t.typname = \'varchar\'"+
 " THEN t.typname || \'(\' || a.atttypmod - 4 || \')\' ELSE t.typname END as \"Tipo\","+
 " CASE WHEN a.attnotnull = \'t\' THEN \'not null \' ELSE \'\' END as \"Modificador\""+
 " FROM pg_class c, pg_attribute a, pg_type t WHERE c.relname = \'"+nodo.toString()+"\' AND" +
 " a.attnum > 0 and a.attrelid = c.oid and a.atttypid = t.oid ORDER BY a.attnum", null, true, Load.ALL));

 query.executeQuery();
```

FIGURA 26 – Características da tabela

Além das características da tabela, também são mostrados, na guia chamada SQL Resultado, os dados desta. O comando SQL referente a está ação é *select * from nome_da_tabela*, visto melhor na figura 27.

```
// (MOSTRA OS DADOS NA TABELA)
        queryTabelas.closeStatement();
        queryTabelas.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(
           db, "select * from \"" + nodo.toString()+"\" ", null, true, Load.ALL));
        queryTabelas.executeQuery();
```

FIGURA 27 – Dados da tabela selecionada na árvore

Na classe *frConexao* há um método chamado *textoSQL* que é utilizado para a execução de um comando SQL escrito pelo usuário. Este método obtém os dados nos componentes *jTextArea* chamado *textQuery* e *jTextField* chamado *textBd*. No *textQuery* está o comando SQL digitado pelo usuário e no *textBd* está o nome da base de dados selecionada na árvore.

Se o *textQuery* não estiver vazio e o *textBd* estiver com uma base de dados selecionada, os comandos SQL do usuário serão executados. Se o comando for uma seleção e não apresentar erro, o resultado aparecerá na guia SQL. Se for qualquer outro tipo de comando, o resultado aparecerá na guia Status. Isto é visto na figura 28.

```
public void textoSQL(){
  if (!textQuery.toString().equalsIgnoreCase("")){
    if (!textBd.toString().equalsIgnoreCase("sem base de dados selecionada")){
      db.closeConnection();
      db.setConnection(new com.borland.dx.sql.dataset.
            ConnectionDescriptor("jdbc:postgresql://" + Host +
                        ":" + Port + "/" +
                       textBd.getText(),
                        User, Password, false,
                        "org.postgresql.Driver"));
      db.setDatabaseName("");
      try {
        queryTabelas.closeStatement();
        queryTabelas.setQuery(new com.borland.dx.sql.dataset.QueryDescriptor(
           db, textQuery.getText(), null, true, Load.ALL));
        queryTabelas.executeQuery();
        pnDados.setSelectedIndex(2);
      }
      catch (DataSetException ex1) {
        if (!ex1.getExceptionChain().getException().getMessage().equalsIgnoreCase(
                    "No results were returned by the query."))
          textError.insert(erroString("\n>> "+
                      ex1.getExceptionChain().getException().getMessage()), 0);
        else
          textError.insert("\n>> OK!!!",0);
        pnDados.setSelectedIndex(3);
      }
    }
  }
```

FIGURA 28 – Método textoSQL

Para a criar uma base de dados é utilizada a classe *dlBaseDados*. Ao ser instanciada, a classe tem referência para a classe *frConexao* através da variável *frame*. Isto se deve ao fato de se ocorrer algum erro na criação da base de dados. Este erro é relatado na classe *frConexao* na guia chamada Status.

Todos os dados para a criação da base de dados são recolhidos na classe *dlBaseDados*, porém todos os comandos SQL são executados na classe *frConexao*.

A criação de uma base de dados tem um requisito principal, a base de dados corrente deve ser a “template1”. Por esse motivo deve-se primeiramente conectar-se a ela antes de executar o comando de criação de uma nova base de dados.

```
String auxBD = new String();
String auxQuery = new String();

auxBD = frame.textBd.getText();
auxQuery = frame.textQuery.getText();
frame.textBd.setText("template1");
frame.textQuery.setText("CREATE DATABASE \""+textNome.getText()+
                        "\" WITH ENCODING = '"+cbCod.getSelectedItem()+"' ");
try {
   frame.textoSQL();
}
catch (Exception ex) {
  ex.printStackTrace();
}
frame.textQuery.setText(auxQuery);
frame.textBd.setText(auxBD);
dispose();
```

FIGURA 29 – Método de criação da nova base de dados

Na figura 29 é representada a criação de uma base de dados que se dá na seguinte forma: são guardados os valores que estão nos componentes *textQuery* e *textBd* da classe *frQuery* e nesse componentes são inseridos o comando SQL para criação da base de dados *Create Database* e a base de dados “template1”, consecutivamente.

Ao terminar a execução do método *textSQL,* os valores salvos são postos de volta nos componentes, dando ao usuário a impressão de que tudo foi feito em *background*. Logo, a classe *dlBaseDados* é destruida voltando para a classe *frConexao*.

Basicamente, para apagar uma base de dados, o princípio é o mesmo da criação, com exceção de que não é necessária outra classe para isto, como descreve a figura 30.

```
void miBdApaga_actionPerformed(ActionEvent e) {
  String auxBD = new String();
  String auxQuery = new String();

  auxBD = textBd.getText();
  auxQuery = textQuery.getText();
  textBd.setText("template1");
  textQuery.setText("DROP DATABASE \"" +nodo.toString() + "\" ");
   try {
      textoSQL();
   }
    catch (Exception ex) {
       ex.printStackTrace();
   }
   textQuery.setText(auxQuery);
   textBd.setText(auxBD);
}
```

FIGURA 30 – Apagar base de dados

A ação de criar e alterar uma tabela assemelha-se a de criar uma base de dados. A classe *dlTabelas* faz a interface com o usuário e o método *textoSQL* da classe *frConexao* executa os comandos SQL como é visto nas figuras 31 e 32.

```
void btCriar_actionPerformed(ActionEvent e) {
  String auxBD = new String();
  String auxQuery = new String();
  String auxCol = new String();

  auxBD = frame.textBd.getText();
  auxQuery = frame.textQuery.getText();

  for ( int i = 0; i < cbCol.getItemCount(); i++){
    cbCol.setSelectedIndex(i);
    auxCol += colNome.get(cbCol.getSelectedItem().toString()) + " ";
    auxCol += colTipo.get(cbCol.getSelectedItem().toString());
    if (colTam.get(cbCol.getSelectedItem().toString())!=" ")
      auxCol += "(" + colTam.get(cbCol.getSelectedItem().toString()) + ") ";
    else
      auxCol += colTam.get(cbCol.getSelectedItem().toString()) + " ";
    auxCol += colVazio.get(cbCol.getSelectedItem().toString()) + " ";
    auxCol += colPK.get(cbCol.getSelectedItem().toString()) + " ";
    if (i != cbCol.getItemCount()-1)
      auxCol += ", \n";
    else
      auxCol += " ";
  }
```

FIGURA 31 – Primeira parte da criação da tabela

Na primeira parte da criação da tabela, figura 32, foi instanciada uma variável da classe *String* chamada *auxCol* para facilitar a captura dos campos. Estes estão guardados em *HashMap´s* chamados *colNome*, *colTipo*, *colTamanho*, *colVazio* e *colPK*, onde o índice é o nome do campo, visto isto na figura 32.

```
colNome.put(textColNome.getText(), textColNome.getText());
colTipo.put(textColNome.getText(), vlTipo(cbColTipo.getSelectedIndex()));
if (textColTam.isVisible()) colTam.put(textColNome.getText(), textColTam.getText());
else colTam.put(textColNome.getText(), " ");
colVazio.put(textColNome.getText(), " ");
if (cbColPK.isSelected())
  colPK.put(textColNome.getText(), "PRIMARY KEY");
else {
  colPK.put(textColNome.getText(), " ");
  if (cbColVazio.isSelected())
    colVazio.put(textColNome.getText(), "NOT NULL");
  else
    colVazio.put(textColNome.getText(), "NULL");
}
cbCol.setSelectedItem(makeObj(textColNome.getText()));
```

FIGURA 32 – Guardando os dados das colunas nos contêiners

A segunda parte da criação das tabelas, é muito similar a criação da base de dados, mudando somente o comando SQL, como é observado na figura 33.

```
frame.textQuery.setText("CREATE TABLE "+
      textNome.getText()+" ( " + auxCol + ")");
try {
    frame.textoSQL();
  }
catch (Exception ex) {
  ex.printStackTrace();
}
frame.textQuery.setText(auxQuery);
frame.textBd.setText(auxBD);
dispose();
}
```

FIGURA 33 – Segunda parte da criação da tabela

As alterações da tabela são feitas da mesma forma que a criação da mesma, como pode ser visto nas figuras seguintes.

```
void btGrNome_actionPerformed(ActionEvent e) {
  String auxBD = new String();
  String auxQuery = new String();

  auxBD = frame.textBd.getText();
  auxQuery = frame.textQuery.getText();

  frame.textQuery.setText("ALTER TABLE "+lbTitulo.getText()+
     " RENAME TO "+textNome.getText()+" ");
  try {
     frame.textoSQL();
   }
  catch (Exception ex) {
    ex.printStackTrace();
    this.dispose();
  }
  frame.textQuery.setText(auxQuery);
  frame.textBd.setText(auxBD);
  lbTitulo.setText(textNome.getText());
}
```

FIGURA 34 – Alteração do nome da tabela

Na figura 35 é vista a alteração do nome da coluna, onde *wAtrib* é uma variável da classe *String* e contém o nome do campo a ser alterado.

```
String auxBD = new String();
String auxQuery = new String();

if (!wNovo) {
  auxBD = frame.textBd.getText();
  auxQuery = frame.textQuery.getText();

  frame.textQuery.setText("ALTER TABLE " + lbTitulo.getText() +
                          " RENAME COLUMN " + wAtrib + " TO " +
                          textColNome.getText() + " ");
  try {
    frame.textoSQL();
  }
  catch (Exception ex) {
    ex.printStackTrace();
    this.dispose();
```

FIGURA 35 – Alteração do nome do campo

Na figura 36 é vista a adição de um campo na tabela. Se o campo for chave primária, não precisa ser descrito que o campo é não nulo, pois isso já está implícito.

```
auxBD = frame.textBd.getText();
auxQuery = frame.textQuery.getText();
String auxCol = new String();

auxCol += textColNome.getText() + " ";
auxCol += vlTipo(cbColTipo.getSelectedIndex()) + " ";
if (textColTam.isVisible())
  auxCol += "(" + textColTam.getText() + ") ";
if (!cbColPK.isSelected()){
  if (cbColVazio.isSelected())
    auxCol += "NOT NULL ";
  else
    auxCol += "NULL ";
}
else
  auxCol += "PRIMARY KEY ";

frame.textQuery.setText("ALTER TABLE " + lbTitulo.getText() +
                        " ADD COLUMN " + auxCol);

try {
  frame.textoSQL();
}
catch (Exception ex) {
  ex.printStackTrace();
  this.dispose();
```

FIGURA 36 – Adição de um campo na tabela

E finalmente para apagar uma tabela é utilizado o método descrito na figura 37, onde *nodo.toString* é o nome da tabela selecionada na árvore para ser apagada.

```
void miTbApaga_actionPerformed(ActionEvent e) {
  String auxQuery = new String();

  auxQuery = textQuery.getText();
  textQuery.setText("DROP TABLE \"" +nodo.toString() + "\" ");
   try {
      textoSQL();
   }
    catch (Exception ex) {
       ex.printStackTrace();
   }
   textQuery.setText(auxQuery);
}
```

FIGURA 37 – Apagar uma tabela

4.3 APRESENTAÇÃO DA FERRAMENTA

A demonstração do protótipo será feita através de ilustrações para a melhor visualização.

Ao inicilizar o protótipo farão-se necessárias as seguintes informações para acessar o SGBD PostgreSQL:

b) o nome do servidor (Hostname);

c) a porta do servidor (Port);

d) o nome do usuário (Username);

e) e a senha do usuário (Password).

Como é visto na figura 38.

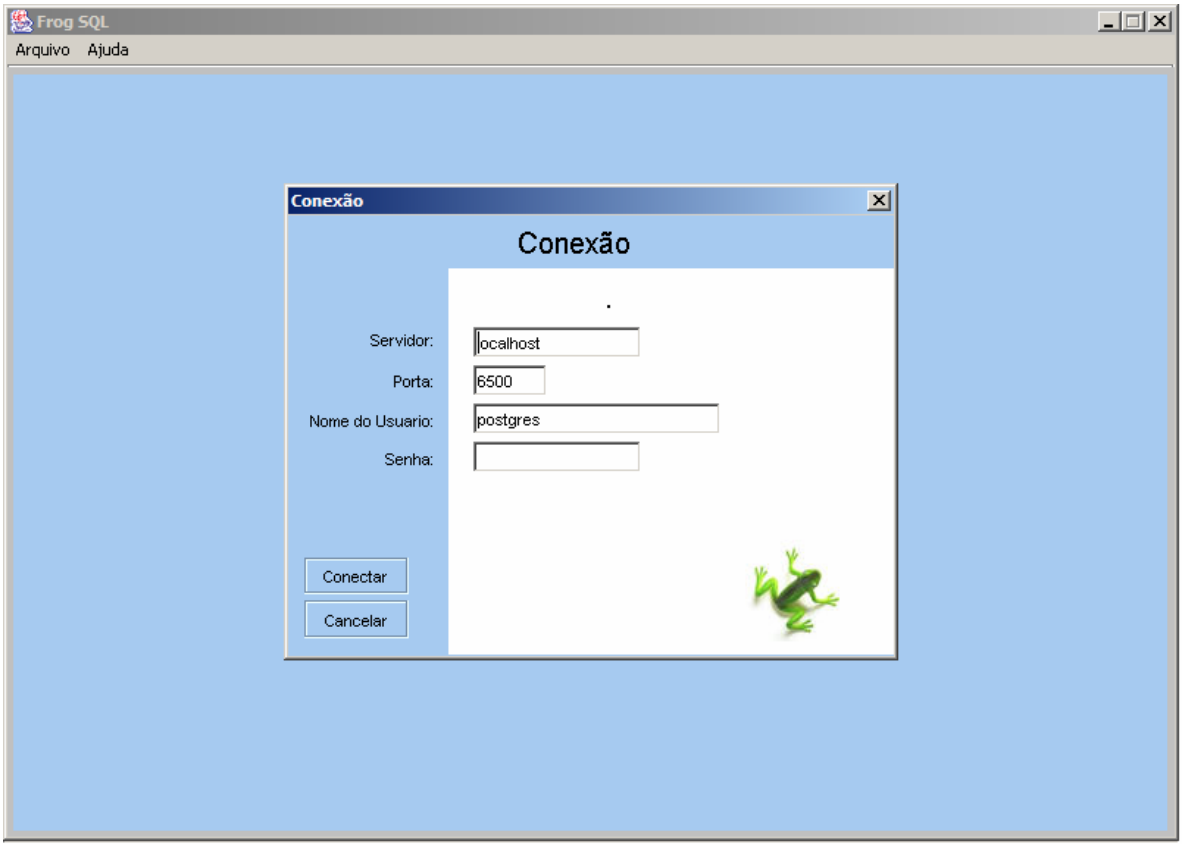

FIGURA 38 – Inicialização do protótipo

Logo após a conexão ao SGBD, o protótipo montará uma árvore para mostrar as bases de dados e as tabelas; aparecerão também guias para o usuário (desenvolvedor)

navegar. Na guia *Propriedades* pode-se ver as características dos itens da árvore. Na guia *SQL Query* pode-se montar um comando SQL. Na guia *SQL Resultado* pode-se ver o resultado de um comando de seleção. E na guia *Status* verificar se o comando foi bem sucedido ou se houve eventuais erros.

Na figura 39 pode-se ver a árvore e a guia chamada *Propriedades*, pode-se notar que acima das guias há um indicador da base de dados selecionada.

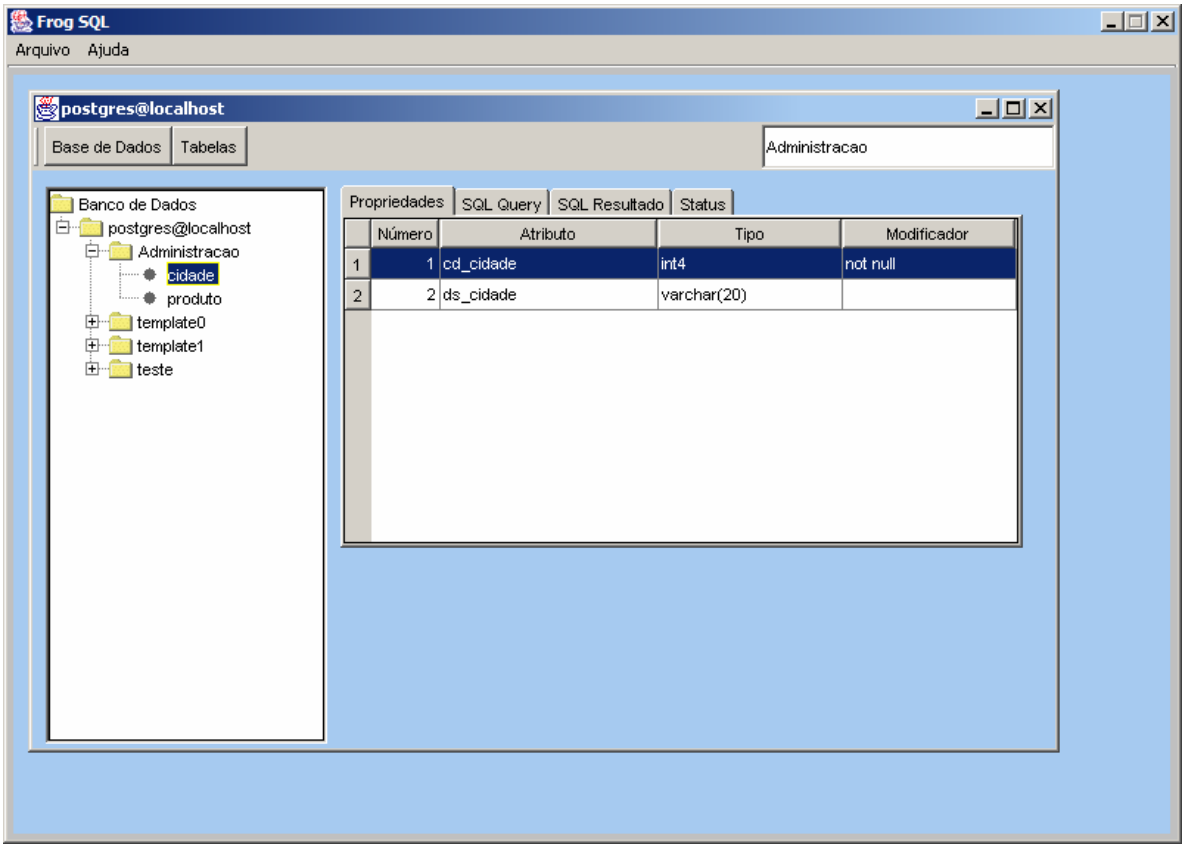

FIGURA 39 – Demonstração da árvore e das propriedades

A figura 40 ilustra a guia SQL Query, sendo inserido um comando SQL.

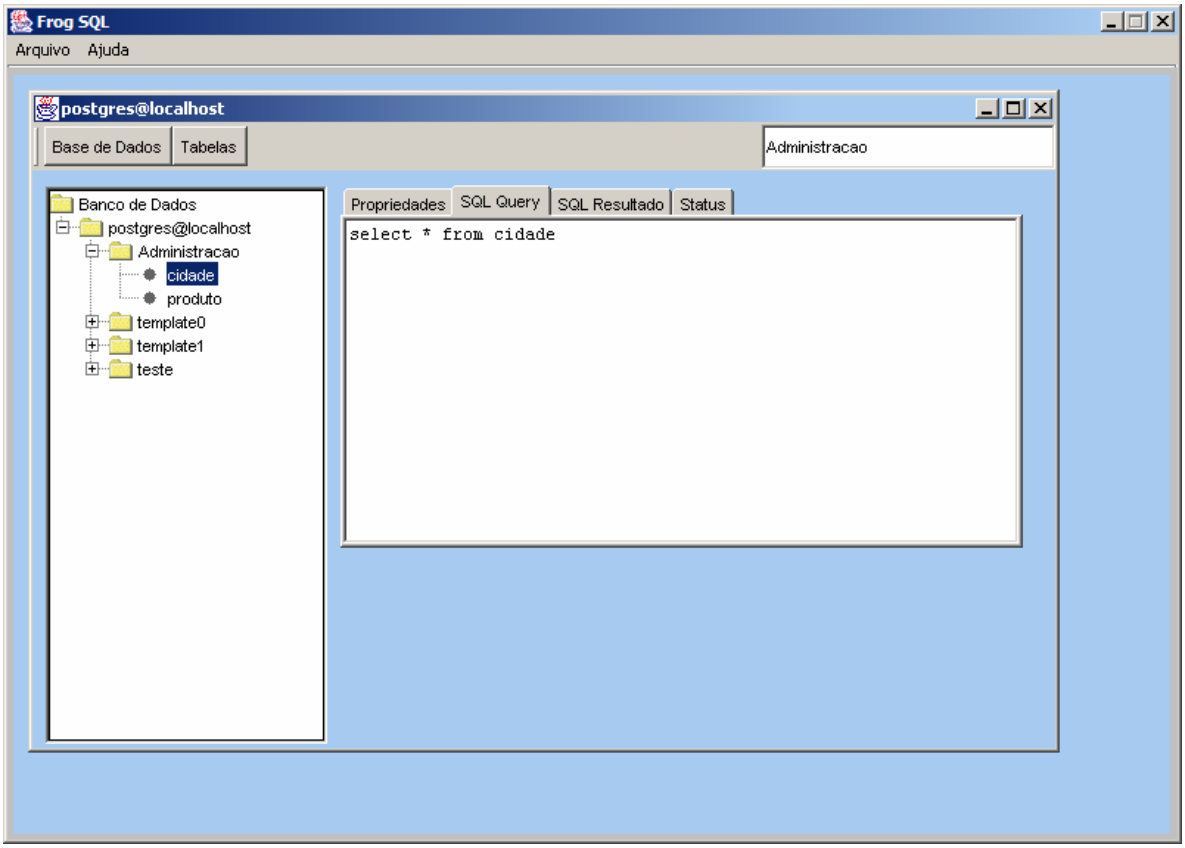

FIGURA 40 – Demonstração da guia *SQL Query*

Ao teclar o F9, o protótipo irá executar o comando escrito na guia *SQL Query* e passará para a guia *SQL Resultado*. Este mostrará o resultado se for uma seleção. Isto está descrito na figura 41.

O protótipo não tem o objetivo de analisar a parte léxica, sintática ou semântica da sentença SQL executado pelo usuário. Esta análise é feita pelo núcleo do SGBD PostgreSQL. O protótipo após o núcleo ter feito a análise, capta esta informação e expõe ao usuário.

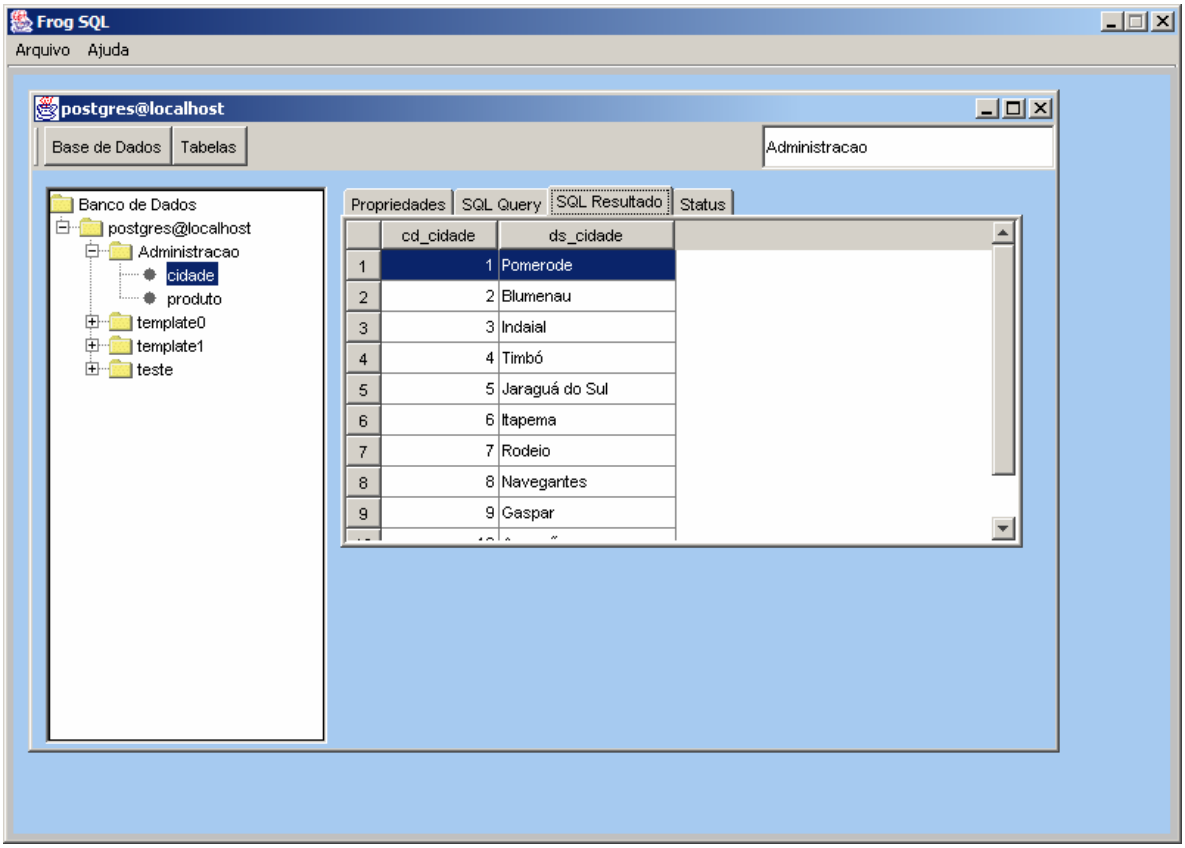

FIGURA 41 – Demonstração da guia SQL Resultado

A guia *Status* denuncia possíveis erros ocorridos na sentença SQL, como pode ser observado na figura 42.

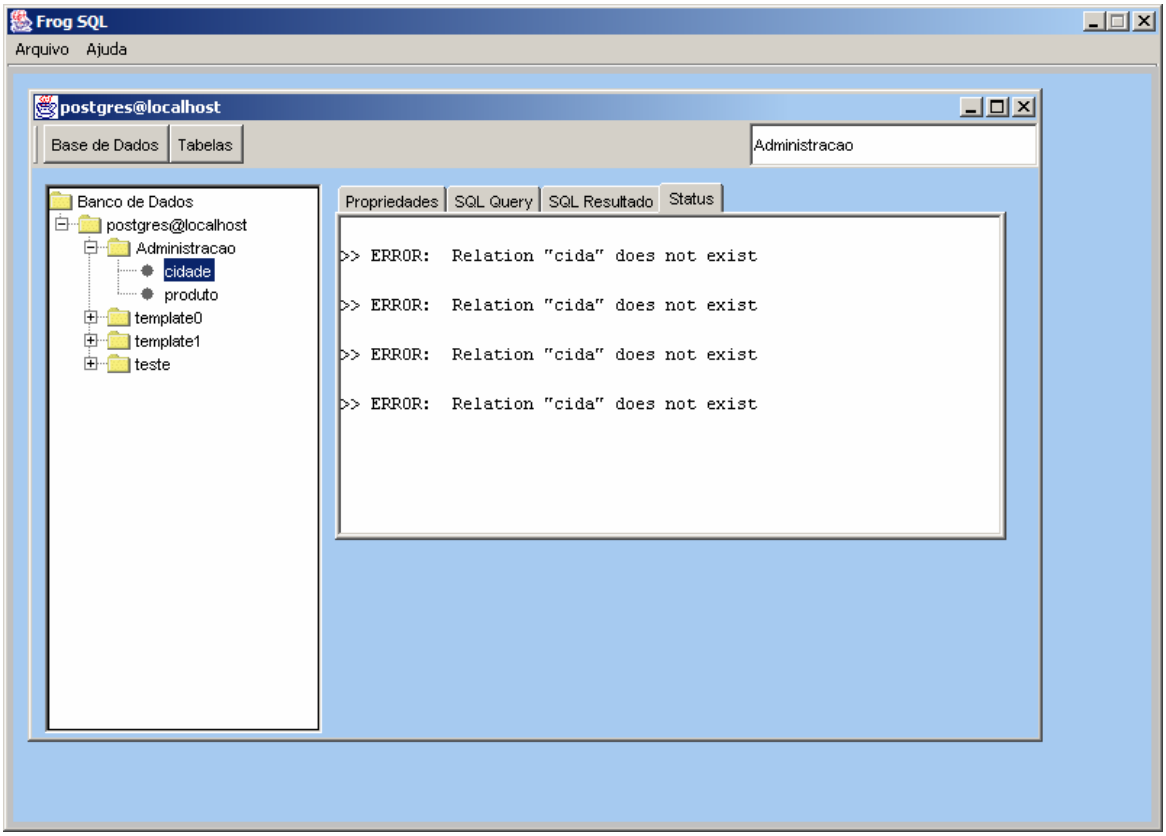

FIGURA 42 – Demonstração da guia Status com erro

Para criar uma base de dados é necessário preencher seu nome e sua codificação como mostra a figura 43.

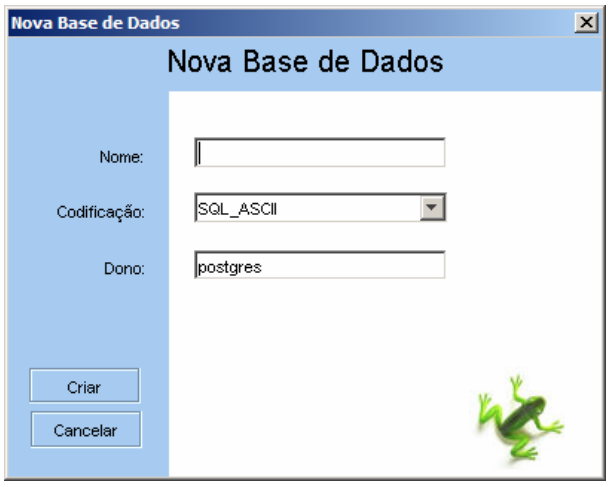

FIGURA 43 – Criação da base de dados

Para a criação da tabela, é necessário informar o nome da tabela e seus campos. Estes ficam guardados e podem ser alterados como é visto nas figuras 44 e 45.

FIGURA 44 – Nome da nova tabela

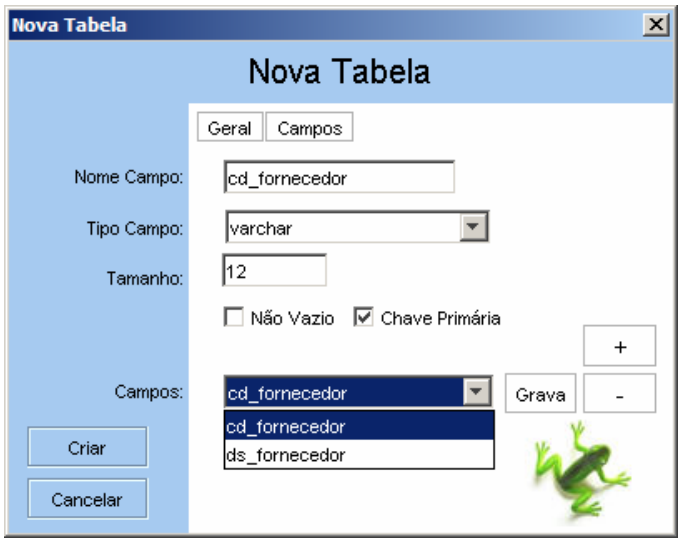

FIGURA 45 – Campos da nova tabela

A alteração de uma tabela é semelhante à criação da mesma, como é visto nas figuras 46 e 47.

Para se alterar o nome da tabela, é preciso somente digitar o nome e clicar em *gravar*, como é visto na figura 46.

FIGURA 46 – Alterar o nome da tabela cidade

Para alterar os campos é necessário selecioná-los, alterar suas propriedades e clicar em *Grava*, como é visto na figura 47.

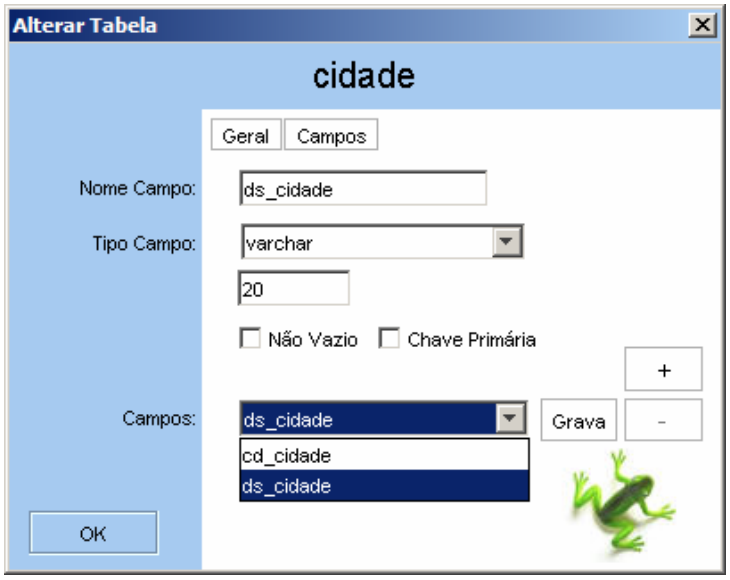

FIGURA 47 – Alteração dos campos da tabela cidade

Para apagar bases de dados e tabelas deve-se clicar com o botão direito do mouse sobre o objeto que é desejado apagar na árvore.

5 CONCLUSÃO

No decorrer do desenvolvimento do trabalho confirmou-se a grande gama de funcionalidades do SGBD PostgreSQL, tornando-o assim um dos melhores SGBD´s livre que existe na atualidade.

O PostgreSQL, criado em 1977 , ainda desperta o interesse dos desenvolvedores pela atual necessidade de se ter um SGBD robusto, estável, rápido e principalmente livre. Hoje em dia um dos fatores importantes no desenvolvimento de um software é a questão financeira.

Um dos fatores negativos do SGBD PostgreSQL é a falta de uma versão para o Microsoft Windows.

Para o desenvolvimento do protótipo foi utilizado o Borland Jbuilder. Como foi mencionado anteriormente, este software é uma ferramenta de desenvolvimento indicada principalmente para o uso comercial. A facilidade de aprendizado deve-se à similaridade ao Borland Delphi, principal ferramenta utilizada no ensino acadêmico. Este fato foi essencial na agilidade do desenvolvimento do trabalho, tornando assim possível focar a estética do software, ou seja, construir uma interface amistosa para o usuário.

Os objetivos propostos foram alcançados com sucesso. Foram implementados os principais comandos de manipulação de dados e de estrutura de dados do SBGD, tendo como principal prioridade à manipulação das bases de dados e das tabelas. As características de multi-plataforma e multi-conexão acrescentam e facilitam ainda mais a usabilidade do protótipo pelo usuário, mais especificamente, o desenvolvedor ou administrador.

O estudo e desenvolvimento do trabalho foram de grande importância devido a poucas referências sobre o tema, por ser um SGBD relativamente novo no mercado, mas que vem crescendo e acompanhando de perto a evolução da internet, já que muitas aplicações se voltam para este fim.

O fato do PostgreSQL ser um SGBD de muitas funcionalidades, implementá-las em pouco tempo torna-se complicado. O protótipo tem diversas limitações principalmente em relação à acessibilidade, não possuindo nenhuma gerência de usuários.

5.1 EXTENSÕES

Buscando aprimorar os resultados obtidos com o protótipo, sugere-se:

a) implementar um gerenciamento de usuário;
b) implementar um gerenciamento de funções;
c) implementar um gerenciamento de agregações;
d) implementar um gerenciamento de grupos;
e) implementar um gerenciamento de transformações;
f) implementar um gerenciamento de conversões;
g) implementar um gerenciamento de domínios;
h) implementar um gerenciamento de índices;
i) implementar um gerenciamento de linguagem procedural;
j) implementar um gerenciamento de operadores;
k) implementar um gerenciamento de regras;
l) implementar um gerenciamento de gatilhos;
m) implementar um gerenciamento de tipos de dados;
n) implementar um gerenciamento de visões.

REFERÊNCIAS BIBLIOGRÁFICAS

ARMSTRONG, Eric. **Jbuilder 2**. Foster City: IDG Books Wordwide, 1998.

DATE, J. C. **Introdução aos Sistemas de Bancos de Dados**. Trad.: Helio Auro Gouveia. Rio de Janeiro: Editor Campus, 1989.

FURLAN, José David. **Modelagem de objetos através de UML –** the unified modeling language. São Paulo: Makron Books, 1998.

GESCHWINDE, Ewald; SCHÖNIG, Hans-Jürgen. **PostgreSQL**: developer´s handbook. Indiana: Sams Publishing, 2002.

JEPSON, Brian. **Programando banco de dados em java.** Trad. Miguel Luis Cabrera. São Paulo: Makron Brooks, 1997.

MELO, Ana C. **Desenvolvendo aplicações com UML:** do conceitual à implementação. Rio de Janeiro: Brasport livros e multimidia, 2002.

NEWMAN, Alexander. **Usando Java.** Trad. Follow-Up Traduções e Assessoria de Informática. Rio de Janeiro: Campus, 1997.

PGADMIN III. Disponível em: <http://www.pgadmin.org/pgadmin3/index.php>. Acessado em: 01 out. 2003.

POSTGRESQL Disponível em: < http://www.postgresql.org/>. Acesso em: 17 agosto 2003.

QUATRANI, Terry. **Modelagem visual com Rational Rose 2000 e UML**.Trad. Savannah Hartmann. Rio de Janeiro: Ciência Moderna,2001.

WORSLEY, John C.; DRAKE, Joshua D. **Pratical PostgreSQL**. Sebastopol: O'Reilly & Associates Books, 2002.