



UNIVERSIDADE FEDERAL DE UBERLÂNDIA

INSTITUTO DE GEOGRAFIA

CURSO DE GEOGRAFIA

**A CONSTRUÇÃO DE UM SISTEMA GEOCODIFICADO PARA
CADASTRO DE ACIDENTES DE TRÂNSITO**

GEORGE RODRIGUES DA CUNHA SILVA

UBERLÂNDIA
2009

GEORGE RODRIGUES DA CUNHA SILVA

**A CONSTRUÇÃO DE UM SISTEMA GEOCODIFICADO PARA
CADASTRO DE ACIDENTES DE TRÂNSITO**

Trabalho de Conclusão de Curso apresentado ao
Instituto de Geografia de Universidade Federal de
Uberlândia, como requisito para obtenção do título de
Geógrafo

Orientadora: Profa. Dra. Denise Labrea Ferreira

**UBERLÂNDIA
2009**

GEORGE RODRIGUES DA CUNHA SILVA

**A CONSTRUÇÃO DE UM SISTEMA GEOCODIFICADO PARA CADASTRO DE
ACIDENTES DE TRÂNSITO**

Profa. Dra. Denise Labrea Ferreira

Prof. Dr. Roberto Rosa

Prof. Dr. William Rodrigues Ferreira

Data: __/__/____

Resultado:_____

À força de vontade.

AGRADECIMENTOS

Existem muitas pessoas e instituições merecedoras de agradecimentos. Tentarei não esquecer de todo mundo:

A David Bitner, David Fetter e outros do #postresql, pelo suporte e apoio.

Ao #openlayers, #javascript e #geoserver pelo suporte.

A Maria Cecília, pelo apoio, idéias e sugestões.

A orientadora Profa. Dra. Denise pelo imenso apoio, incentivo e puxões de orelha.

A empresa VERTRAN, pelos bons tempos que lá passei, pelo apoio e pela disponibilização dos dados de acidentes.

Aos amigos: Paulo Vitor, Marcos Augusto, Daniel Superbi, Gustavo Eugênio, Alfredo Arantes, Diogo Finotti, Januário, Leonardo Portilho, Daniel de Deus, Daniel Nascimento, Thiago Nogueira, Diogo Lemos, Maxsuel Barros, Rafael Franco, Pablo Ramos, Marcos Baioneta, Marcelão, Murilo, Árvore e muitos outros irmãos “adquiridos” ao longo de quatro anos de faculdade.

Aos meus pais: Érico e Betina, por tudo.

A Johnny Cash, Sublime, The Chemical Brothers e Megadeth, pela incrível trilha sonora.

LISTA DE FIGURAS

Figura 1: Mapa produzido por John Snow (UK 1854).....	35
Figura 2: Exemplo da representação de uma entidade.	39
Figura 3: Campo referenciado (“cod_acidente”, na tabela da esquerda) e campo com chave estrangeira (“cod_acidente_veiculo”, tabela da direita).	42
Figura 4: Violação da chave estrangeira na tabela veículos.....	43
Figura 5: Representação de um relacionamento de cardinalidade 1:1.	43
Figura 6: Representação de um relacionamento de cardinalidade 1:M.....	43
Figura 7: Representação de um relacionamento de cardinalidade M:N.....	44
Figura 8: Decomposição do relacionamento M:N em entidade.	44
Figura 9: Vista dos esquemas dentro do SGBD (PgAdmin III).	50
Figura 10: Diagrama E/R do esquema “geocode”.	51
Figura 11: Diagrama E/R relativo à entidade acidentes.	40
Figura 12: Diagrama E/R relativo à entidade veículos.....	41
Figura 13: Esquema E/R relativo à entidade condutores.....	41
Figura 14: Esquema E/R apresentando as relações entre as tabelas principais do sistema.	42
Figura 15: Organograma da função geradora de acidentes.	47
Figura 16: Linhas centrais de logradouros com vista da tabela de identificação.	51
Figura 17: código SQL para a criação da tabela logradouros.....	52
Figura 18 - Especificação da metodologia utilizada para assinalar valores iniciais e finais aos trechos que estes não foram identificados em campo.....	54
Figura 19: Visão da tabela finalizada de logradouros dentro do banco de dados.	55
Figura 20: Tabela do serviço geocodificador dentro do SGBD (PgAdmin III).	56
Figura 21: Modelo conceitual da geocodificação interpolada.....	57
Figura 22: Modelo conceitual da geocodificação por trechos ou entre logradouros.....	58
Figura 23: Modelo conceitual da geocodificação por cruzamento.....	58

Figura 24: Organograma da função geocodificadora.	60
Figura 25: Representação conceitual do modelo three-tier architecture.	64
Figura 26: Tela de Login.	65
Figura 27: Tela principal do aplicativo cliente.	65
Figura 28: Tela Configurações do Sistema.	66
Figura 29: Tela Cadastro de Acidentes, aba Acidentes.	68
Figura 30: Tela Cadastro de Acidentes, aba Veículos.	69
Figura 31: Tela Cadastro de Acidentes, aba Condutor.	70
Figura 32: Tela principal, acesso às variáveis de sistema.	71
Figura 33: Tela Cadastro de Variáveis de Sistema (Tipo Veículo).	72
Figura 34: Tela Cadastro de Usuários em modo administrador.	73
Figura 35: Tela Cadastro de usuários no modo usuário. Note que os botões e campos estão inacessíveis.	73
Figura 36: Tela do Terminal Interativo sem nenhuma consulta executada.	74
Figura 37: Terminal Interativo após a execução de uma consulta SQL.	75
Figura 38: Acesso aos relatórios gerenciais na tela principal.	76
Figura 39: Relatório Acidentes Totais por Trecho	77
Figura 40: Relatório Acidente Totais por Logradouros.	78
Figura 41: Relatório Severidade Total por Logradouro.	79
Figura 42: Tela de entrada do GeoServer.	85
Figura 43: Organização lógica de uma instância do GeoServer.	86
Figura 44 - Página Inicial do GeoServer.	86
Figura 45 - Página de configuração de dados do GeoServer.	87
Figura 46 - Editor de FeatureType no GeoServer.	88
Figura 47 - Editor de estilos.	89
Figura 48 - Criação de namespace.	90

Figura 49 - página de configuração de um datastore recém criado.	91
Figura 50 - Visualização de demonstração da <i>FeatureType</i> recém criado no <i>GeoServer</i> . A visualização de demonstração neste caso é gerada pela <i>API OpenLayers</i>	92
Figura 51 - Mapa gerado pelo código-fonte 11, visualizado no navegador <i>Firefox</i>	96

LISTA DE CÓDIGOS

Código 1: Exemplo de um comando SQL para selecionar dados de uma tabela.	46
Código 2: Exemplo de um comando SQL para inserir dados na tabela 'aux_acid_tipo'.....	46
Código 3: Exemplo de um comando SQL para atualizar o valor de código do logradouro dos registros com o nome 'JOÃO NAVES DE ÁVILA'	46
Código 4: Exemplo de um comando SQL para remover os registros da tabela acidentes quando a coluna 'the_geom' tiver valor nulo.	46
Código 5: Exemplo de um comando SQL para criar uma tabela 'aux_acid_tipo' com as colunas cod_tipo e desc_tipo dos tipos inteiro e texto, respectivamente.	46
Código 6 - Exemplo de um comando SQL para remover a tabela 'aux_acid_tipo' do banco de dados.....	46
Código 7: Exemplo de um comando SQL para alterar a tabela 'acidentes'. Neste caso adicionamos uma restrição à tabela, mas todo tipo de alteração é possível.	47
Código 8: Exemplo de um comando SQL SELECT utilizando algumas das cláusulas possíveis. Estas cláusulas podem ser utilizadas em qualquer tipo de comando DML.	48
Código 9: Exemplo de um comando SQL SELECT utilizando-se alguns operadores lógicos e relacionais. Neste exemplo queremos todos os acidentes da data 10/10/2001 entre 12 horas e 18 horas.	49
Código 10: Exemplo de código SQL solicitando a seleção de uma expansão do centróide de todas as quadras em 100 unidades de medida. Note que a função ST_Centroid foi embrulhada pela função st_buffer.	44
Código 11: Exemplo de um código simples para declaração de um mapa em OpenLayers....	95
Código 12: Função disparada após a entrada de um novo registro na tabela acidentes.	108
Código 13: Função geocode. Esta função executa a geocodificação interpolada e foi adaptada para se adequar ao formato de endereçamento brasileiro.....	110
Código 14: Função geocode_cruzamento. Esta função executa a localização dos endereços através dos cruzamentos fornecidos pelo usuário.....	110
Código 15: Função geocode_trecho. Esta função localiza os endereços passados pelo usuário de acordo com as interseções anterior e posterior.	111

Código 16: Função Wrapper Externa. Esta função é a única à qual o usuário comum tem acesso, administrando o acesso as outras conforme solicitado pelo usuário.....	111
Código 17: Função gerador. Esta função realiza as contagens necessárias para se inserir informações sem corrupção, assegurando que o número de veículos e condutores seja compatível com o gerado.....	112
Código 18: Função gera_acidente. Esta função gera aleatoriamente alguns parametros da tabela acidentes e os devolve para a função gerador, de modo a construir um comando SQL.	113
Código 19: Função gera_veiculo. Esta função gera aleatoriamente alguns parametros da tabela veiculos e os devolve para a função gerador, de modo a construir um comando SQL.....	114
Código 20: Função gera_condutor. Esta função gera aleatoriamente alguns parametros da tabela condutores e os devolve para a função gerador, de modo a construir um comando SQL.	115
Código 21: Função gera_severidade_veiculos. Esta função gera aleatoriamente a severidade a ser inserida na tabela veículos e retorna o resultado para a função gera_veiculos.....	116

LISTA DE QUADROS

Quadro 1: Alguns tipos de dados disponíveis	41
Quadro 2: Listagem das tabelas no esquema public.....	39
Quadro 3: Alguns números do PostgreSQL	44

LISTA DE ABREVIATURA E SIGLAS

1:1	RELACIONAMENTO 1 PARA 1
1:M	RELACIONAMENTO 1 PARA MUITOS
ANSI	AMERICAN NATIONAL STANDARDS INSTITUTE
API	APPLICATION PROGRAMMING INTERFACE
AT	ACIDENTE DE TRÂNSITO
CEP	CÓDIGO DE ENDEREÇAMENTO POSTAL
DER	DIAGRAMA ENTIDADE RELACIONAMENTO
ETL	EXTRACT/TRANSFORM/LOAD
FK	FOREIGN KEY
FSF	FREE SOFTWARE FOUNDATION
GEOS	GEOMETRY ENGINE - OPEN SOURCE
GIS	GEOGRAPHIC INFORMATION SYSTEM
GML	GEOGRAPHIC MARKUP LANGUAGE
HTML	HYPERTEXT MARKUP LANGUAGE
HTTP	HYPERTEXT TRANSFER PROTOCOL
ISO	INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
LAN	LOCAL AREA NETWORK
M:N	RELACIONAMENTO MUITOS PARA MUITOS
MVCC	MULTIVERSION CONCURRENCY CONTROL
OGC	OPEN GEOSPATIAL CONSORTIUM
OOP	OBJECT-ORIENTED PROGRAMMING
PITR	POINT IN TIME RECOVERY
PK	PRIMARY KEY
REGEX	REGULAR EXPRESSIONS
SFS	SIMPLE FEATURES FOR SQL
SGBD	SISTEMA GERENCIADOR DE BANCO DE DADOS
SIG	SISTEMA DE INFORMAÇÃO GEOGRÁFICA

SL	SOFTWARE LIVRE
SLD	STYLED LAYER DESCRIPTOR
SQL	STRUCTURED QUERY LANGAUGE
UFU	UNIVERSIDADE FEDERAL DE UBERLÂNDIA
VS	VISUAL STUDIO
WAN	WIDE AREA NETWORK
WCS	WEB COVERAGE SERVICE
WFS	WEB FEATURE SERVICE
WFS-T	WEB FEATURE SERVICE - TRANSACTIONAL
WMS	WEB MAP SERVICE
XML	EXTENSIBLE MARKUP LANGUAGE

SUMÁRIO

INTRODUÇÃO.....	18
CONSIDERAÇÕES INICIAIS	19
<i>Objetivos</i>	20
• <i>Objetivo Geral</i>	20
• <i>Objetivos Específicos</i>	20
<i>Referencial Teórico</i>	21
• <i>Caracterização dos Acidentes de Trânsito</i>	21
• <i>Coleta de Informações sobre acidentes de trânsito</i>	22
• <i>Banco de Dados</i>	23
• <i>Geocodificação</i>	23
• <i>WebGIS</i>	24
<i>Problemática</i>	25
<i>Metodologia</i>	25
• <i>Cartografia</i>	26
• <i>Modelagem de Dados</i>	26
• <i>Construção do Software Cliente</i>	26
• <i>Hardware</i>	27
CAPÍTULO 01	28
1. CONTEXTO DE DESENVOLVIMENTO.....	29
1.1. <i>O que é Software Livre?</i>	29
1.2. <i>A importância dos softwares livres</i>	30
1.3. <i>O que é Geoprocessamento?</i>	31
• 1.3.1. <i>Cartografia Digital</i>	32
• 1.3.2. <i>Sensoriamento Remoto</i>	32
• 1.3.3. <i>Banco de Dados</i>	33
• 1.3.4. <i>Análise Espacial</i>	34

CAPÍTULO 02	37
2. TEORIA DE BANCO DE DADOS E MODELAGEM DO BANCO DE DADOS	38
2.1. Teoria de Banco de Dados	38
• 2.1.1. Propriedades Transacionais	39
2.1.2. Classes de Atributos	40
2.1.3. Tipos de atributos	41
2.1.4. Relacionamentos	42
2.1.5. SQL (Structured Query Language)	45
2.1.6. Modelo E/R.....	49
2.1.7 - O Modelo de Dados utilizado no Sistema de Cadastro de Acidentes	49
2.1.8. PostgreSQL	42
2.1.9. PostGIS	44
2.1.10. Dados utilizados.....	45
CAPÍTULO 03	48
3. GEOCODIFICAÇÃO.....	49
3.1. Elementos de Endereço	50
3.2. Construção da Base de Referência	50
3.2.1. Criação da tabela de logradouros	52
3.2.2. Digitalização	53
3.2.3. Trabalho de Campo.....	53
3.3. Serviços de Geocodificação	56
3.3.1. Construção do Algoritmo Geocodificador	56
CAPÍTULO 04	61
4. CONSTRUÇÃO DO SOFTWARE CLIENTE	62
4.1. Arquitetura em três níveis	62
4.2. Apresentação do Software.....	64
4.2.1. Login	64

4.2.2. Tela Principal.....	65
4.2.3. Configurações	66
4.2.4. Cadastro de Acidentes.....	66
4.2.5. Cadastro de Variáveis de Sistema.....	71
4.2.6. Cadastro de Usuários.....	72
4.2.7. Terminal Interativo	73
4.2.8. Relatórios Gerenciais.....	75
CAPÍTULO 05	80
5. WEBGIS.....	81
5.1. OGC (Open Geospatial Consortium).....	82
5.1.2. WMS (Web Map Service)	82
5.1.3. WFS(Web Feature Server)	83
5.2. GeoServer.....	84
5.2.1. Disponibilização de Dados através do GeoServer.....	86
5.3. OpenLayers	92
5.3.1. Definições dos Mapas	93
CAPÍTULO 06	97
6. RESULTADOS FINAIS	98
REFERÊNCIAS	100
7. REFERÊNCIAS	101
APÊNDICES	106

INTRODUÇÃO

CONSIDERAÇÕES INICIAIS

A segurança viária é uma grande preocupação da administração pública, pois tem impacto significativo sobre a vida da população, além de acarretar imensos custos para a sociedade. Para que quaisquer medidas de segurança sejam tomadas, sendo elas educacionais, obras de engenharia ou aumento da fiscalização, é necessário um conhecimento profundo do problema, sugerindo a necessidade da presença um sistema de cadastro e análise de acidentes de trânsito eficiente.

Este trabalho propõe a criação de um sistema em meio digital capaz de armazenar diversas informações sobre estes acidentes, veículos e condutores envolvidos, com a capacidade de derivar mapas eficientes, análise espacial e consultas espaciais, que permita a identificação de possíveis correlações positivas com outros dados espaciais, como sinalização, condições das vias e severidade dos acidentes.

Nesta proposta de sistema para cadastro de acidentes, diversas técnicas e tecnologias heterogêneas foram utilizadas, reduzindo o custo para geração de mapas, análise espacial e do próprio cadastro dos acidentes em uma base de dados. Atualmente, existe um sistema em operação para cadastro dos acidentes, mas este não incorpora informação espacial propriamente dita, apenas dados referentes à localização dos mesmos. No escopo deste trabalho é proposto um sistema capaz de localizar e pontuar automaticamente as ocorrências, sem a interferência de um operador especializado em geoprocessamento para realizar tal trabalho. A tecnologia utilizada para automatizar a tarefa de transformar endereços (a referência primária para o local do acidente utilizada) foi a geocodificação, na qual este endereço é transformado em coordenadas e escrito em um banco de dados espacial, para geração de mapas e análises espaciais apuradas.

Além da criação de um método automatizado para realização deste processo, foi desenvolvida uma aplicação para inserção dos dados, geração de relatórios primários e mapas dinâmicos utilizando-se tecnologias WEB.

Dentre as etapas para a viabilização do sistema foram consideradas:

A criação de uma base de referência, passível de servir como comparação para o processo de geocodificação;

A coleta de dados e a metodologia utilizada pelos órgãos responsáveis atualmente;

O tratamento destas informações para posterior cadastro;

As deficiências do sistema de cadastro atual e, por fim, as melhorias a serem implantadas;

Todas as etapas para o desenvolvimento de um projeto piloto capaz de realizar o proposto são descritas nos capítulos subseqüentes, desde o modelo de dados necessário para a criação de uma base de logradouros à geração de mapas através de tecnologias WEB.

Objetivos

- *Objetivo Geral*

Desenvolver um sistema capaz de cadastrar acidentes de trânsito e suas diversas variáveis relacionadas e de um algoritmo compatível, capaz de geocodificar os endereços destes acidentes, transformando-os em pares de coordenadas X Y, facilmente representados em mapas;

- *Objetivos Específicos*

Desenvolver um algoritmo de geocodificação capaz de processar endereços incompletos, indicativos de cruzamentos ou sem o número viário.

Desenvolver um modelo de dados lógico e físico para o armazenamento dos dados relativos a acidentes e aplicação cliente para o cadastro dos mesmos.

Apresentar relatórios gerenciais capazes de informar as ocorrências de forma tabular.

Sugerir um sistema para geração de mapas dinâmicos, utilizando um banco de dados e servidores de mapas.

Referencial Teórico

Nesta seção, serão abordadas algumas considerações sobre os acidentes de trânsito e seu cadastro e análise. Também serão analisados conceitos pertinentes à fundamentação teórica das ferramentas, tecnologias e técnicas utilizadas para atingir o resultado final do trabalho.

- *Caracterização dos Acidentes de Trânsito*

Primeiramente é necessário definir o que é um acidente de trânsito. De acordo com Gold (1998) um acidente de trânsito (AT) é:

(...) um evento independente do desejo do homem, causado por uma força externa, alheia, que atua subitamente e deixa ferimentos no corpo e na mente. (...)

Assim, um acidente de trânsito pode ser definido como um evento do tipo descrito, que envolve ao menos um veículo que circula, normalmente por uma via para trânsito de veículos, podendo ser o veículo motorizado ou não. (GOLD 1998, p.09)

Existem diversos conceitos relacionados à geografia/distribuição dos acidentes no espaço.

Região Central: a região central de uma cidade ou município reflete a região em que há maior fluxo de veículos e eventualmente maior concentração de acidentes. A maior concentração de acidentes, contudo não configura condições de trânsito piores ou mais perigosas, apenas um fluxo significativamente maior de veículos. (GOLD, 1998, p. 10)

Interseções ou Cruzamentos: confluência de duas ou mais vias, configurando um cruzamento de vias. (GOLD, 1998, p. 10)

Pólos Geradores de Tráfego: são locais com taxa relativamente maior de acidentes que outros pontos da mesma área ou via, devido à localização de um empreendimento ou equipamento urbano gerador de tráfego. Exemplos de equipamentos urbanos e/ou empreendimentos são: supermercados, centros comerciais, terminais de passageiros, escolas, etc. Um exemplo bastante concreto é a região “baixa” do bairro Santa Mônica, em Uberlândia – MG, onde estão bastante próximos o centro administrativo municipal, o campus universitário Santa Mônica, o supermercado Carrefour e ainda o Center Shopping.(GOLD, 1998, p.10)

Pontos críticos: um ponto crítico por definição é um local que apresenta uma frequência de acidentes de trânsito muito acima do “normal”. É bom frisar que não existe uma definição concreta de ponto crítico, como um número mínimo de acidentes no local ou uma severidade mínima para se considerar aquele ponto como tal. Portanto existe uma separação da identificação de pontos críticos dentro de uma malha viária urbana, atendendo a diferentes critérios: pontos críticos de acidentes em geral, pontos críticos de acidentes com vítimas, pontos críticos de acidentes com vítima fatal, pontos críticos de atropelamentos, pontos críticos de acidentes com motocicletas e pontos críticos de acidentes com crianças. (GOLD, 1998, p. 10).

- *Coleta de Informações sobre acidentes de trânsito*

Existem dados relativos aos ATs de maior relevância que outros. Gold especifica a data e hora do acidente, local de ocorrência e a severidade de cada acidente dados de extrema importância para um cadastro e análise posterior efetiva. De posse destes quatro elementos podemos realizar análises espaço-temporais, considerando diversos intervalos de tempo, agrupamento de ATs e sua gravidade.

A identificação dos pontos críticos, ou seja, dos locais com elevados índices de acidentes, exige um cadastro de acidentes que contenha, pelo menos, as datas e os locais em que ocorreram os acidentes durante um ano ou mais. Igualmente imprescindível é a continuação deste cadastro, para poder avaliar as intervenções viárias implantadas, em função da frequência de acidentes. (GOLD, 1998, p.27)

A severidade dos acidentes é uma informação de extrema importância para um cadastro eficiente, registrando-se o número de pessoas feridas de forma leve, grave ou seu falecimento.

Em termos de relevância, a caracterização do acidente e como ele ocorreu vem em seguida. Através da análise dos boletins de ocorrência podemos identificar e realizar uma análise detalhada da situação, e cadastrar estes dados no sistema. O tipo de acidente, sua natureza e causa são essenciais para se identificar possíveis falhas de engenharia. Além disso, a trajetória dos veículos, tipo dos veículos envolvidos e fatores ambientais no momento do acidente são dados também importantes para uma melhor compreensão do ocorrido.

Gold ainda reitera a necessidade de um cadastro atualizado, explicitando a enorme ajuda que este pode fornecer para a tomada de medidas concretas, obtenção de recursos, etc.

- *Banco de Dados*

Os bancos de dados são sistemas capazes de armazenar e retornar dados de forma organizada sob uma determinada metodologia. Eles permitem diversas operações para leitura e consulta dos dados, retornando o resultado necessário.

Atualmente os bancos de dados mais utilizados são relacionais, isto é, utilizam um esquema relacional para organização dos dados armazenados. Através da modelagem dos dados em esquema relacional podemos inserir, consultar e manipular os dados dentro do banco.

Outra característica importante no conceito de banco de dados é a independência do esquema relacional/conceitual ao seu armazenamento em disco, sendo estes dois processos separados, no qual o usuário não tem controle (armazenamento em disco).

Os sistemas relacionais, também chamados de Sistemas Gerenciadores de Banco de Dados ou SGBDs, devem atender um conjunto de treze regras para serem considerados relacionais, regras definidas por Codd em 1985.

Uma definição formal de banco de dados utilizada neste trabalho é a de Elsmari e Navathe (2006, p.04)

Um sistema gerenciador de banco de dados (SGBD) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. O SGBD é, portanto, um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações. A definição de um banco de dados implica especificar os tipos de dados, as estruturas e as restrições para os dados.

- *Geocodificação*

A geocodificação é um procedimento computacional que transforma endereços ou referência a lugares em coordenadas pontuais de acordo com um sistema de referência utilizado. O resultado retornado ao usuário é uma estimativa de onde deveria estar localizado o endereço entregue para consulta.

De acordo com (ESRI, 2007, *Help Files*): “*Geocoding is the process of assigning a location, usually in the form of coordinate values (points), to an address by comparing the descriptive location elements in the address to those present in the reference material.*”

Através da construção de uma base de referência e um serviço geocodificador foi possível geocodificar os acidentes de trânsito no bairro Santa Mônica, em Uberlândia – MG.

- *WebGIS*

WebGIS é um conceito relativamente novo no mercado e no meio acadêmico. Em teoria um *WebGIS* é um conjunto de ferramentas capaz de realizar funções de um *GIS* (*Geographic Information System*) em ambiente *Web*, ou seja, distribuído pela internet sem a necessidade de instalação de pacotes de SIG/*GIS* no computador cliente, tanto para visualização como para edição dos dados.

Este tipo de distribuição de dados geográficos revolucionou e continua revolucionando o mercado de geotecnologias e de SIGs corporativos, sendo amplamente utilizados para gestão municipal. O principal trunfo desta tecnologia é a desmistificação de um software SIG para um usuário comum, sem a necessidade de treinamento intensivo e específico.

De acordo com Painho, Peixoto e Cabral (2001) um WebGIS é simplesmente a soma das funcionalidades de um GIS para web com o benefício de não necessitar de um software. É a democratização de tecnologias SIG e de dados geográficos para as massas não especializadas.

Todas as funcionalidades de um SIG comum estão disponíveis em um WebGIS, desde que sejam administrados os modos para se retornar os dados de um sistema de processamento. Isto é possível pois todas as operações realizadas (incluindo análises espaciais, disponíveis anteriormente apenas em grandes softwares especializados) não rodam na máquina cliente e sim em um servidor central, que apenas envia o resultado destas análises para o cliente.

Em suma, *WebGIS* é uma tecnologia e processo de disponibilização de dados geográficos e mapas interativos para consulta e manipulação através de um protocolo remoto, como o *http* (*Hypertext Transfer Protocol*) na internet ou intranet. Existem diversas maneiras de atingir este resultado, uma delas utilizada neste trabalho é através da estrutura cliente-servidor de mapas web (*GeoServer* + *OpenLayers*, respectivamente).

Problemática

O trânsito é um problema em todas as cidades de médio e grande porte no Brasil e no mundo. Somente no Brasil ocorrem, de acordo com Sousa (2008, p. vii), “cerca de 350 mil acidentes com vítimas, dos quais resultam mais de 33 mil óbitos e cerca de 400 mil feridos ou inválidos”, tornando o trânsito um assunto de segurança pública. O conhecimento aprofundado deste problema é a chave para melhorar as condições viárias em cada cidade e isto somente é possível através de um sistema que cadastre as informações relativas a cada acidente, veículo envolvido, condutores e suas variáveis. Sem este conhecimento, é impossível propor medidas concretas para a melhoria do trânsito.

De acordo com a Política Nacional de Trânsito, a estatística sobre acidentes de trânsito no Brasil é precária:

A estatística nacional de acidentes de trânsito no Brasil, que deveria representar a consolidação das informações de todos os órgãos e entidades de trânsito, mesmo após implantação, pelo DENATRAN, do Sistema Nacional de Estatísticas de Trânsito (SINET), ainda é imprecisa e incompleta, dada à precariedade e falta de padronização da coleta e tratamento das informações (BRASIL, 2004a, p. 14)

Este trabalho, portanto, propõe um sistema para o cadastro de acidentes de trânsito mais eficiente e apresenta um modelo de dados para este cadastro, de forma a iniciar uma discussão em torno da padronização de um cadastro nacional, facilitando a integração e análise dos dados em contexto nacional.

Metodologia

Para a realização deste trabalho foram utilizadas diversas ferramentas e materiais a fim de atingir os objetivos propostos. É possível separar etapas do trabalho e cada uma delas utilizou ferramentas e materiais específicos. Descreveremos, em termos gerais, cada uma delas e sua função principal.

As etapas principais foram: Cartografia, Modelagem de Dados e Construção do Software Cliente.

- *Cartografia*

Ortofotos em formato digital, escala 1:2000. Fonte: Prefeitura Municipal de Uberlândia;

gvSIG: O gvSIG é um software especializado de Sistemas de Informações Geográficas capaz de visualizar, criar, editar dados espaciais e mapas. Fonte: Generalitat Valenciana. Obtido em: www.gvsig.gva.es;

Guia SEI 2005: foi utilizado como fonte de referência para nome de logradouros, CEPs, números viários e planejamento inicial dos trabalhos de campo;

- *Modelagem de Dados*

PowerArchitect: este software é um software livre que executa a construção de diagramas E/R e projeto do modelo de dados. Este software foi utilizado para projetar todo o modelo de dados utilizado neste trabalho; (SQL Power 2009)

PgAdmin III: este software é o SGBD (Sistema Gerenciador de Banco de Dados) do PostgreSQL/PostGIS. É através dele que se tem acesso à execução de comandos SQL, construção do banco de dados e administração de suas características. (PostgreSQL 2009)

Planilha de Cadastro de Acidentes: esta planilha é utilizada pela empresa VERTRAN para interpretação dos boletins de ocorrência e posterior inserção das mesmas no banco de dados atual e guiou a modelagem do banco de dados. (VERTRAN, 2008)

- *Construção do Software Cliente*

Visual Studio 2005 Express: este pacote computacional é um ambiente de desenvolvimento utilizado em larga escala mundo afora. Suporta variadas linguagens de programação, de fácil de aprendizagem e uso, e é gratuito na versão *Express* (somente para projetos acadêmicos e para propósitos de estudo). (Microsoft 2009)

GeoServer: o *GeoServer* é um servidor de mapas capaz de ler dados de diversas fontes e apresentá-los em formato *GML*(*Geographic Markup Language*) a um cliente de mapas (como o *OpenLayers*). Ele foi utilizado como servidor de dados para os mapas dinâmicos gerados pelo software. (GeoServer 2009)

OpenLayers: esta *API* (*Application Programming Interface*) utilizada para renderizar feições geográficas apresentadas por um servidor em uma página *HTML* de forma dinâmica e estruturada. Esta *API* combinada com o servidor de dados geográficos *OpenLayers* permitiu representar os dados dentro do SGBD em tempo real aos usuários do sistema. (*OpenLayers* 2009)

- *Hardware*

Foram utilizados dois computadores para a realização deste trabalho:

Um computador portátil HP série dv5800 com processador Intel Core 2 Duo de 1.87GHz, 3Gb de memória RAM, 250Gb de disco rígido, placa de vídeo integrada de 512mb;

Um computador *desktop* com a seguinte configuração: processador Intel Core 2 Duo E4400 2.4Ghz, 6Gb de memória RAM, 250Gb de disco rígido, placa de vídeo dedicada GeForce 8600 DD2 512Mb;

CAPÍTULO 01

1. CONTEXTO DE DESENVOLVIMENTO

1.1. O que é Software Livre?

Um software é considerado livre quando o usuário tem a liberdade total de uso, estudo e modificação. Não basta ser apenas gratuito, ele deve ter código fonte aberto à comunidade de usuários e prevê a liberdade de seu uso.

De acordo com a *Free Software Foundation (FSF)* um software, para ser livre, deve atender as seguintes premissas (a numeração segue o especificado no site da *FSF*):

0 – Liberdade de execução/uso do software, para quaisquer propósitos;

1 – Liberdade para estudar como o programa funciona e modificá-lo à suas necessidades;

2 – Liberdade para distribuição de cópias;

3 – Liberdade para publicar atualizações e melhorias no código-fonte;

Nas palavras da própria *FSF*:

A program is free software if users have all of these freedoms. Thus, you should be free to redistribute copies, either with or without modifications, either gratis or charging a fee for distribution, to anyone anywhere. Being free to do these things means (among other things) that you do not have to ask or pay for permission. (Free Software Foundation 2009, <www.fsf.org>)

Em suma, um software, para ser livre, não depende de preço ou de revenda, mas sim de atendimento as quatro condições citadas acima. Um exemplo de software gratuito mas não livre é o SPRING (Sistema de Processamento de Informações Georreferenciadas – INPE), pois este não tem seu código-fonte publicado, é apenas distribuído de forma gratuita aos usuários.

Em geral um software livre é publicado sob os termos de alguma licença de uso (como um contrato), especificando os termos acima no caso da licença *GNU Public License (GPL)*.

Existem diversos tipos de licença, como a *Lesser GNU Public License*, BSD (*Berkeley Software Distribution*).

Cada licença se aplica a um tipo específico de software e quem utiliza produtos de uma licença ou outra tem a possibilidade de escolha sob qual licença seu software será publicado.

1.2. A importância dos softwares livres

Existem diversas vantagens em se utilizar esta classificação de programas, começando pela capacidade de visualizar e revisar seu código fonte, trabalho que é feito por milhares de usuários avançados mundo afora.

O software livre ou SL é revisado e atualizado constantemente já que não está preso a atualizações de grandes produtoras de software. Uma prova disso são as constantes versões intermediárias lançadas por produtoras de código livre. Enquanto grandes empresas lançam uma ou outra atualização de segurança durante o tempo de vida útil de um produto, as atualizações dos SLs são quase diárias. A utilização deste tipo de software traz um risco muito menor de segurança, com a erradicação de seus *bugs* (defeitos de programação) em tempo muito pequeno.

Outra vantagem do SL é seu custo. Ele é basicamente zero, gratuito para qualquer usuário fazer o *download* do programa e instalá-lo. Atualmente, quase todos os softwares livres criam versões para diferentes Sistemas Operacionais, inclusive Windows, facilitando sua distribuição em uma grande base de usuários. Como o software é livre, geralmente são cobrados apenas serviços relativos ao mesmo, como consultorias, modificações no código-fonte (caso não se consiga alterar o código sempre existe um bom programador a disposição), entre outros.

O suporte para software livre é gratuito e amplo, com documentação detalhada disponível na página dos projetos. Além de suporte em listas de discussão, a ampla base de usuários facilita a troca de experiências e código, configurando um software que cresce rapidamente em funcionalidade e qualidade.

Entretanto a maior vantagem do software livre é também um ponto de fraqueza. Como os softwares livres são essencialmente criados (programado de fato) por poucos

programadores, sem financiamento para manutenção do projeto, ele pode cair em desuso. É um caso complicado de se administrar. Se o suporte ou atualizações do software acabar, é impossível desenvolver novas funcionalidades e conserto de *bugs* anteriores. Anos atrás era um problema mais comum, mas hoje com a criação de diversas associações para auxiliar esta manutenção (no caso de softwares de geoprocessamento), a “morte” de projetos caiu significativamente. Este quadro é raro, mas pode acontecer com projetos novos fora da tutoria destas associações, como a *OSGeo* (*Open Source Geospatial Foundation* <<http://www.osgeo.org/>>) e a *OGC* (*Open Geospatial Consortium* <www.opengeospatial.org/>) que fazem pesadas exigências para que o projeto entre sob a responsabilidade da fundação.

1.3. O que é Geoprocessamento?

Em primeiro lugar, geoprocessamento é uma ferramenta utilizada em diversas ciências para resolver problemas de ordem espacial. Através do Geoprocessamento podemos representar a realidade do espaço geográfico em um ambiente computacional. São sistemas para armazenar, analisar, criar e modificar dados geográficos.

A ferramenta Geoprocessamento (SIG – Sistemas de Informações Geográficas / *GIS* – *Geographic Information Systems*) compreende quatro grandes áreas de conhecimentos específicos distintos e complementares, a saber: Cartografia Digital, Sensoriamento Remoto, Banco de Dados e Análise Espacial. Através destes quatro ramos podemos desenvolver metodologias e técnicas para resolver problemas de ordem espacial de maneira rápida, precisa e objetiva.

O geoprocessamento tem a capacidade de sobrepor diversos temas distintos sobre um mesmo plano, facilitando a análise conjunta destes elementos geográficos, o que, de outra maneira (em ambiente analógico), seria praticamente impossível/impraticável. Esta capacidade nos permite cruzar diversas informações e constitui o motor da análise espacial como conhecemos hoje.

De acordo com a *ESRI*, maior fabricante de softwares de SIG do mundo: “A *geographic information system (GIS) integrates hardware, software, and data for capturing,*

managing, analyzing, and displaying all forms of geographically referenced information.”
(ESRI,2009)

Uma definição de SIG, de acordo com CÂMARA et. al. (1996, p.33):

SIG são sistemas automatizados usados para armazenar, analisar e manipular dados geográficos, ou seja, dados que representam objetos e fenômenos em que a localização geográfica é uma característica inerente à informação e indispensável para analisá-la.

Abaixo serão discutidos os itens citados acima em maior profundidade:

- *1.3.1. Cartografia Digital*

Neste conjunto de ferramentas e técnicas se encontra a capacidade do Geoprocessamento de desenhar e construir mapas através de um computador, utilizando-se pontos, linhas, polígonos, sistema de referência, datum, combinando diversos temas tais como: estradas, hidrografia, hipsometria, localidades, cobertura vegetal, uso do solo, etc.

A cartografia digital é um sistema derivado dos *CADs* (*Computer Aided Design* – desenho auxiliado por computador), que evoluiu e se adaptou para atender as necessidades de um grupo específico de usuários, transformando a maneira pela qual trabalhamos e representamos o espaço.

De acordo com Filho (2000, p.04) a Cartografia Digital é:

Um sistema de Cartografia Digital (CD) pode ser compreendido como um conjunto de ferramentas, incluindo programas e equipamentos, orientado para a conversão para o meio digital, armazenamento e visualização de dados espaciais. Um sistema de Cartografia Digital tem como ênfase a produção final de mapas.

Existem diversos softwares hoje capazes de atender esta demanda, por exemplo: *AutoCAD MAP, MicroStation*, etc.

- *1.3.2. Sensoriamento Remoto*

O sensoriamento remoto é uma ciência que se dedica ao estudo de alvos de forma remota, sem contato direto com o objeto de estudo. É uma ciência bastante ampla e utiliza diversos instrumentos para estudar as diversas características dos objetos-alvo, como câmeras fotogramétricas, radares e uma variedade de sensores, cada qual com uma tarefa específica. De acordo com Rosa o sensoriamento remoto pode ser definido como:

(...) a forma de obter informações de um objeto ou alvo, sem que haja contato físico com o mesmo. As informações são obtidas utilizando-se a radiação eletromagnética, gerada por fontes naturais como o Sol e a Terra, ou por fontes artificiais como por exemplo o radar. (ROSA, 2003, p.02)

O Sensoriamento Remoto permite a obtenção de dados espaciais de forma relativamente barata, de confiança e de forma repetitiva viabilizando a observação contínua de qualquer área do planeta.

Os dados obtidos através de sensores remotos são apresentados em formato de grade regular, formado de diversas células, chamadas de pixels (a menor unidade que se pode distinguir em uma grade regular).

A junção do Sensoriamento Remoto com a Cartografia Digital permite o desenvolvimento de mapas precisos e de leitura ágil, diminuindo a necessidade de levantamentos em campo e facilitando o planejamento do mesmo.

• 1.3.3. Banco de Dados

Os sistemas gerenciadores de banco de dados (SGBD) são softwares específicos para guardar dezenas de milhares de informações em um esquema relacional (tabular), de fácil adaptação a diversas metodologias e propósitos. Todas as informações geográficas e alfanuméricas ficam armazenadas em um SGBD. A importância atual de um SGBD é sua capacidade de armazenar dados de forma a garantir sua integridade, facilitar sua organização e recuperação das informações de determinado projeto ou SIG.

De acordo com Elmasri e Navathe (2006, p. 04) um sistema gerenciador de banco de dados é:

Um sistema gerenciador de banco de dados (SGBD) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. O SGBD é, portanto, um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações. A definição de um banco de dados implica especificar os tipos de dados, as estruturas e as restrições para os dados.

Além de facilitar a maneira como os dados brutos são armazenados, o SGDB propicia o motor para desenho das feições geográficas. Controlando as informações dentro do banco pode-se alterar de forma rápida e precisa classes de feições geográficas e, conseqüentemente, o mapa gerado pelo software de *GIS*.

Detalha-se um arquivo *CAD* (*Computer Aided Design*) comum. Ele é composto por pontos, linhas, polígonos, textos, entre outros elementos. O desenho-objeto (ponto, linha ou polígono) não contém informações relacionadas a nenhum dos objetos reais representados naquele mapa. Quando é necessário mudar o mapa o que temos de fazer? Redesenhá-lo. *e.g.*: Um mapa de estradas, contendo estradas de terra, estradas pavimentadas pista simples e estradas pavimentadas pista dupla. Se o estado duplicar uma das estradas, é necessário redesenhar aquela linha correspondente como uma estrada duplicada.

Com a adição de Sistemas Gerenciadores de Banco de Dados (SGBD) à Cartografia Digital foi possível guardar diversas informações sobre os objetos ali representados de forma automatizada. Nota-se no caso descrito anteriormente: ao invés de redesenhar a linha, apenas uma alteração no atributo correspondente para “Pavimentada de Duas Pistas” e o sistema entenderia que esta estrada necessita ser representada de maneira distinta.

O papel deste ramo do geoprocessamento é facilitar o armazenamento de informações, permitir a aplicações de regras de negócio e simular atributos e funcionalidades dos objetos do mundo real em ambiente computacional. Existem algumas propriedades transacionais específicas dos SGBDs que merecem atenção, e são popularmente conhecidas por A.C.I.D. (Atomicidade, Consistência, Isolamento e Durabilidade) que garantem o funcionamento perfeito de qualquer banco de dados.

- *1.3.4. Análise Espacial*

A definição formal de Análise Espacial ou Estatística Espacial é o conjunto de técnicas que estuda entidades, suas relações: topológicas, geométricas ou geográficas, e propriedades espaciais. A análise espacial busca padrões, agrupamentos, tendências e previsões para resolver problemas complexos, com uma infinidade de variáveis.

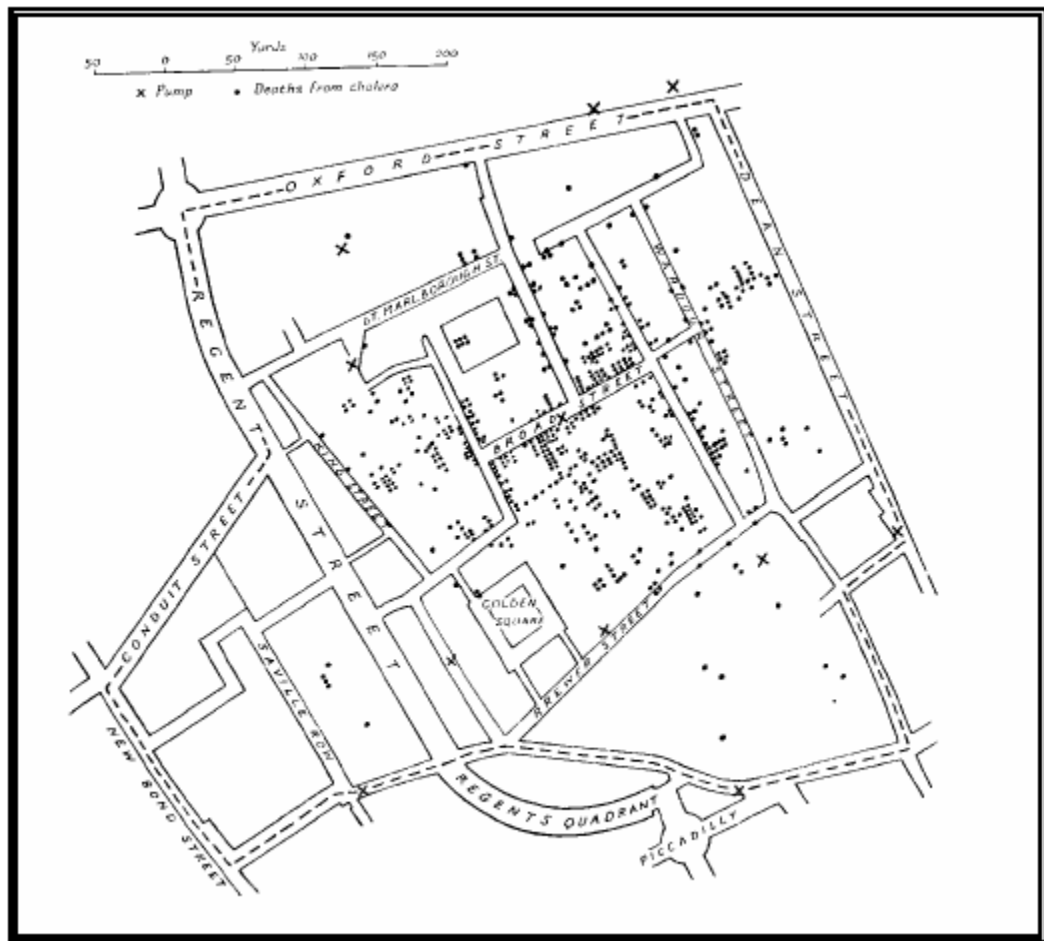


Figura 1: Mapa produzido por John Snow (UK 1854)

Fonte: Wikipedia <<http://en.wikipedia.org/wiki/File:Snow-cholera-map-1.jpg>>

Um exemplo pioneiro e comumente citado foi um estudo intuitivo do século 19, conduzido pelo inglês John Snow (figura 1). Em 1854, Londres sofria com diversas epidemias de cólera. Snow relacionou em um mapa a localização das bombas de água potável e as mortes por cólera. Este é um dos primeiros exemplos como a análise espacial contribuiu para o entendimento e compreensão de um problema, no caso, da corrente epidemia de cólera, através de correlação espacial positiva.

Análise Espacial é o motor que impulsiona o Geoprocessamento. É através dela que conseguimos prever e auxiliar na administração de empreendimentos (quando cita-se o termo empreendimento refere-se a qualquer projeto público, privado no qual podemos aplicar o geoprocessamento como ferramenta de apoio ao processo decisório).

Com a análise espacial, podemos aplicar pesos e trabalhar com superfícies de viabilidade, a fim de descobrir quais são as melhores alternativas para um problema de ordem espacial, por exemplo: onde instalar o próximo lixão da cidade? Sabemos que ele deve ter um

tamanho razoável, para funcionar por pelo menos 15 anos, deve fácil acesso, não deve ficar à montante da cidade, deve ficar a, pelo menos, 1000m de qualquer curso d'água e o solo não deve ser arenoso. Nota-se o cruzamento de informações de diversos temas para resolver esta situação específica, através de equações matemáticas.

No caso de nosso exemplo, combinando os temas estradas, hidrografia, perímetro urbano e tipos de solo da região, podemos “somar” (utiliza-se o termo somar, mas é possível utilizar diversos operadores matemáticos e relacionais para desenvolver estas equações) os pontos fracos e fortes de cada tema e descobrir o lugar ótimo para a instalação do lixão.

CAPÍTULO 02

2. TEORIA DE BANCO DE DADOS E MODELAGEM DO BANCO DE DADOS

Anteriormente, foi mencionado que um banco de dados era uma coleção de dados relacionados e um Sistema Gerenciador de Banco de Dados o software responsável por criar, manter, manipular e compartilhar as informações dentro de cada banco de dados.

Os SGBDs permitem projetar um banco de dados peça por peça, a partir do qual define-se um esquema de trabalho e regras de negócio. Este projeto ou representação é comumente denominado Modelo de Dados.

O Modelo de Dados tem uma importância enorme, pois é ele guiará o *DBA (Database Administrator – Administrador de banco de dados)* na construção do banco de dados de fato e os desenvolvedores na construção das interfaces com o usuário final.

Neste capítulo presta-se uma pequena introdução em Teoria de Banco de Dados (Modelo Relacional), Diagramas E/R e, por fim, serão apresentados os Diagramas utilizados na construção do banco de dados para o Cadastro de Acidentes.

Dentro deste trabalho serão descritos apenas alguns aspectos do Modelo E/R e da teoria de banco de dados, pois o assunto, muito extenso, foge do escopo final do trabalho. Após uma breve discussão sobre a teoria de banco de dados, define-se o Modelo E/R usado no presente trabalho, bem como algumas peculiaridades sobre seu funcionamento.

2.1. Teoria de Banco de Dados

Dentro do modelo de dados especificamos as entidades e os relacionamentos que as entidades têm entre si. As entidades são os objetos básicos (a partir do momento que as entidades entram no ambiente computacional passam a ser denominada de tabelas). Uma entidade simplesmente é “algo” do mundo real, e pode ser tanto um objeto concreto como uma pessoa, um carro, ou um objeto de existência conceitual como uma empresa, um curso

universitário, um órgão público, etc. As entidades são, em suma, tabelas em que armazenamos os dados de acordo com o modelo especificado.

Cada entidade tem propriedades que as descrevem em detalhes e as configuram como um objeto único dentro de nosso banco de dados. Os atributos de cada entidade são a maior parte dos dados armazenados em um banco de dados.

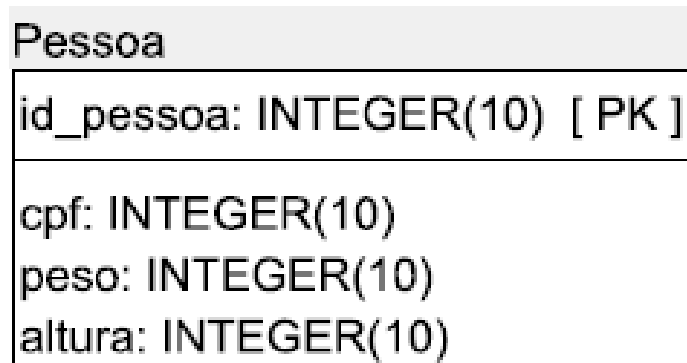


Figura 2: Exemplo da representação de uma entidade.

Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009

Na figura acima, constam quatro atributos: "id_pessoa", "cpf", "peso" e "altura", todos do tipo INTEIRO (isto significa que somente é possível armazenar números inteiros em cada campo ou coluna). Este processo de modelagem continua até que todas as entidades do sistema em questão estejam projetadas.

• 2.1.1. Propriedades Transacionais

As propriedades transacionais (A.C.I.D.) são muito importantes para os SGBDs, pois são elas que garantem o perfeito e íntegro funcionamento da base de dados. Estas propriedades garantem este funcionamento aplicando um conjunto de regras computacionais sobre os registros da base de dados, de modo que constantes atualizações e consultas simultâneas NÃO alterem seu estado verdadeiro (por verdadeiro entende-se coerente, e não-corrompido, íntegro).

Atomicidade: o conceito de atomicidade se refere à habilidade do SGBD de garantir o resultado de uma transação de dados de forma completa. Ou toda a alteração é processada ou nenhuma. Um exemplo no qual a atomicidade é crucial: uma transação bancária, na qual a transferência de R\$100,00 da conta bancária X para a conta Y não se perca no meio do caminho, na qual X moveria o valor de sua conta, mas devido a alguma pane, Y não os recebesse.

Consistência: o conceito de consistência garante que somente dados válidos serão escritos de forma definitiva no banco de dados. Esta propriedade é muito importante, pois evita o preenchimento de informações em campos errados, perdendo-se o valor da informação.

Isolamento: esta propriedade dos bancos de dados garante que ninguém possa acessar as informações em estados intermediários (lembramos aqui o exemplo de atomicidade, em que não é possível consultar o saldo de X nem Y antes que todas as transações sejam processadas – configurando um estado intermediário, não consistente para o banco de dados).

Durabilidade: este conceito garante aos usuários que após uma informação ter sido escrita (geralmente este momento é tratado como “*commit*” na literatura especializada) ela não será desfeita, nem por panes dentro do SGDB, nem por futuros *commits* no banco de dados.

2.1.2. Classes de Atributos

Existem algumas classes que cada atributo pode assumir. O entendimento destas classes é importante, pois um banco de dados eficiente depende de um bom projeto. Caso o usuário (ou DBA) não tenha entendimento da armazenagem de cada informação em seu nível mais primitivo pode ser extremamente complicado alterar a definição do banco de dados quando este estiver em um estágio de produção e um aumento na dificuldade de buscar informações de forma precisa.

2.1.2.1. Atributos Simples

Atributos Simples ou atômicos são atributos que não podem ser decompostos em partes menores. Um exemplo de um atributo atômico é o peso, a altura ou o CPF de uma pessoa, representados na entidade acima.

2.1.2.2. Atributos Compostos

Atributos Compostos podem ser decompostos em diversos atributos menores, ou em atributos atômicos. Um exemplo bem claro de um atributo composto é o Endereço de uma pessoa, que pode ser decomposto em: Tipo do Logradouro (Rua, Avenida, Alameda, etc.), Nome do Logradouro, Número Viário, CEP e Complemento, por exemplo.

2.1.2.3. Atributos monovalorados

Atributos monovalorados são atributos que somente podem assumir um único valor dentro da base de dados. Um exemplo de atributo monovalorado é a Idade de uma pessoa. Não é possível designar duas idades para a mesma pessoa simultaneamente.

2.1.2.4. Atributos Multivalorados

Atributos Multivalorados podem assumir mais de um valor em um ambiente não-normalizado. Um exemplo desta classe de atributo é o telefone ou o email de uma pessoa, já que não estamos limitados à apenas um número de telefone simultaneamente. É possível, neste caso, armazenar um ou mais valores dentro da base de dados para representar a multiplicidade que este atributo pode assumir.

2.1.2.5. Atributos Derivados

Atributos Derivados não são de fato armazenados dentro de um SGBD, eles são apenas calculados em momento de execução (ou leitura). Alguns exemplos são: Idade derivada de um atributo Data de Nascimento na entidade Pessoa; Área de um polígono, derivado de seu atributo Geometria. Note que o SGBD apenas apresenta o resultado de um atributo derivado, nunca armazena seu valor absoluto.

2.1.3. Tipos de atributos

Cada SGBD fornece ao usuário administrador a possibilidade de inserir atributos de diversos tipos. Os tipos de atributos vão definir o que pode ser armazenado em cada campo do banco de dados. Existem diversos tipos de atributos e com a possibilidade de se construir atributos compostos a partir de atributos simples, tornando o processo de modelagem de dados extremamente flexível. Foram listados apenas alguns tipos de atributos (Quadro 1), já que cada SGBD tem seus tipos específicos.

TIPO DE ATRIBUTO	DESCRIÇÃO
Inteiro	Números inteiros
Varchar	Strings de caracteres com limites definidos
Geometria	Geometria de um objeto
Time	Valores de tempo (formato hh:mm:ss:dd)
Date	Valores de data

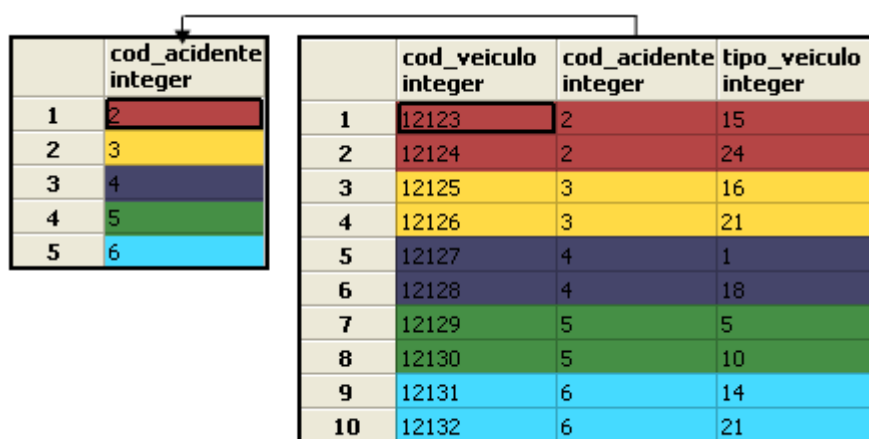
Quadro 1: Alguns tipos de dados disponíveis

2.1.4. Relacionamentos

Os relacionamentos denotam interações sobre uma ou mais entidades. O relacionamento ocorre quando um atributo de uma entidade referencia outro atributo de outra entidade. Quando isto ocorre chamamos estes campos de chaves estrangeiras.

A ligação de uma tabela com outra(s) se dá através de campos especiais chamados chaves estrangeiras. Um exemplo do uso de chaves estrangeiras utilizado neste trabalho é o de acidentes e veículos, com um código no registro do veículo representando o acidente do qual este participou.

Um acidente pode ter um ou mais veículos envolvidos. Neste caso a tabela veículos possui um campo específico para especificarmos o código do acidente do qual este veículo participou. Veja a figura a seguir:



	cod_acidente integer
1	2
2	3
3	4
4	5
5	6

	cod_veiculo integer	cod_acidente integer	tipo_veiculo integer
1	12123	2	15
2	12124	2	24
3	12125	3	16
4	12126	3	21
5	12127	4	1
6	12128	4	18
7	12129	5	5
8	12130	5	10
9	12131	6	14
10	12132	6	21

Figura 3: Campo referenciado (“cod_acidente”, na tabela da esquerda) e campo com chave estrangeira (“cod_acidente_veiculo”, tabela da direita).

Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009

Neste caso apresentado o campo referenciado é o “*cod_acidente*” na tabela acidentes (à esquerda na figura 3) e a chave estrangeira é o campo “*cod_acidente_veiculo*”¹ (à direita na figura 3). Este tipo de relacionamento permite cruzar informações entre as duas tabelas. Observa-se que além de um relacionamento, uma chave estrangeira configura uma restrição (*constraint*), já que o sistema não permite o cadastro de um veículo cujo “*cod_acidente_veiculo*” não exista na tabela acidentes. Na figura acima, seria impossível cadastrar um veículo e inserir o valor “10” no campo “*cod_acidente_veiculo*” já que não existe acidente com este código (figura 4).

¹ Dentro de um SGBD não é permitido o uso de acentos, caracteres especiais ou de espaços nos nomes de tabelas e nomes de colunas, sendo citadas neste trabalho com a mesma grafia utilizada pelo banco de dados.

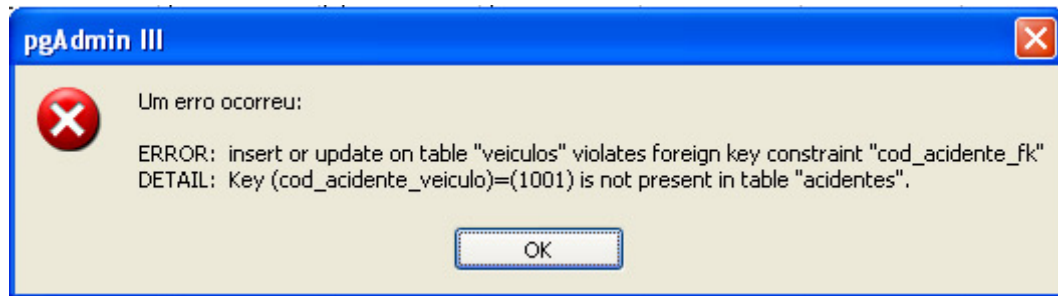


Figura 4: Violação da chave estrangeira na tabela veículos.

Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009

2.1.4.1. Cardinalidade de relacionamentos

Existem diversos tipos de relacionamentos disponíveis durante a modelagem de um banco de dados, de forma a facilitar o entendimento das relações entre os objetos de nosso dia-a-dia e replicá-la dentro de um ambiente computacional. Cada tipo tem sua aplicação específica.

1:1 – Um para Um: neste tipo de relacionamento existe somente um objeto de cada lado da relação, exemplo: um carro em movimento e seu motorista. Ninguém consegue dirigir dois carros ao mesmo tempo e nenhum carro (comum, pelo menos) possui mais de um motorista.



Figura 5: Representação de um relacionamento de cardinalidade 1:1.

Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009

1:M – Um para Muitos: nesta modalidade de relação temos um objeto do mundo real relacionado com diversos outros objetos ao mesmo tempo. Um exemplo é um carro e seus passageiros (lembre-se que o motorista está em outra relação), que pode não ter nenhum passageiro como pode ter vários, até o limite de espaço físico oferecido pelo veículo. A figura 6 abaixo representa um relacionamento 1:M.



Figura 6: Representação de um relacionamento de cardinalidade 1:M.

Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009

M:N – Muitos para Muitos: neste tipo de relacionamento temos um objeto do mundo real que se relaciona com diversos outros objetos representados na base de dados. Um exemplo é um Empregado que pode trabalhar em um ou mais projetos ao mesmo tempo. Todos os relacionamentos do tipo Muitos para Muitos exigem uma tabela intermediária (figura 8) para se relacionarem, pois seria impossível relacionar um empregado com diversos projetos de forma atômica (isto é, sem repetir o registro do empregado X em sua entidade). Desta forma é necessário criar uma tabela específica para o relacionamento.



Figura 7: Representação de um relacionamento de cardinalidade M:N.
Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009

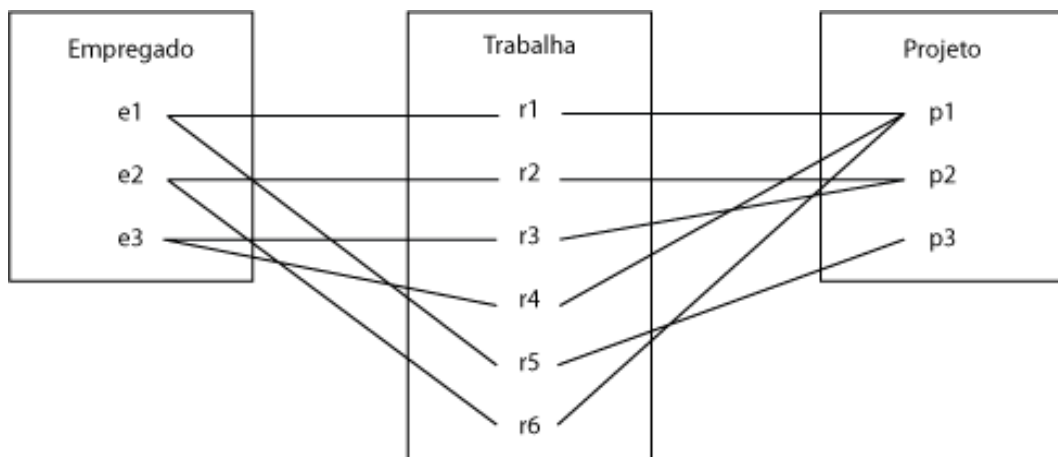


Figura 8: Decomposição do relacionamento M:N em entidade.
Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009

2.1.4.2. Atributos de Relacionamentos

Conforme a complexidade e as necessidades de um banco de dados crescem pode ser interessante criar atributos para determinado relacionamento. Os atributos de relacionamento funcionam da mesma maneira que os atributos de entidade, e guardam características específicas sobre o relacionamento e sua natureza. Observando o exemplo acima, o relacionamento “*trabalha*” poderia ter um atributo “*Horas Trabalhadas*”, para representar o número de horas semanais que cada trabalhador aplica em determinado projeto.

2.1.5. SQL (Structured Query Language)

A *Structured Query Language* ou Linguagem de Consulta Estruturada é uma linguagem de pesquisa declarativa entendida por todos SGBDs relacionais com pequenas variações de fabricante para fabricante.

A *SQL* é uma linguagem relativamente fácil de aprender, pois é declarativa e sua sintaxe bastante simples mas, apesar disso, bastante poderosa, possibilitando a execução de consultas complexas.

A linguagem *SQL* foi criada pela *IBM* na década de 1970, para retornar dados do primeiro modelo relacional proposto por E. F. CODD. Seu nome original era *SEQUEL* (*Structured English Query Language* – Linguagem de Consulta Estruturada em Inglês). Sua rápida expansão conduziu à necessidade de se estabelecer padrões, sendo criados pelo *ANSI* (*American National Standards Institute*) em 1986 e pelo *ISO* (*International Organization for Standardization*) em 1987. A linguagem passou por diversas revisões (1992, 1999 e 2003) e hoje suporta diversos recursos como expressões regulares (*RegEx* – Expressões Regulares), *XML* (*eXtensible Markup Language*), consultas recursivas, entre outros. Hoje a *SQL* é padrão para todos os SGBDs relacionais e foi através desta linguagem que se estruturou o banco de dados desta pesquisa, após a modelagem conceitual e a criação de um diagrama E/R.

Além disso, a maioria dos fabricantes de SGBDs acrescenta suporte a algumas linguagens adicionais, para desenvolvedores estenderem seus bancos de dados com funcionalidades customizadas. O *PostgreSQL* usado neste trabalho tem suporte à diversas linguagens como: *plpgsql*, *Tcl*, *C*, *Perl*, entre outras.

Existem diversas seções da linguagem *SQL*, que são classificadas por suas funções dentro do SGBD: DML, DDL, DCL, e DTL; Esta distinção somente ocorre para fins didáticos, pois todos os SGBDs suportam seus diversos subconjuntos, e os mesmos fazem parte de um padrão, conforme especificado acima. Serão apresentados alguns comandos principais da DML e DDL, que são os mais usados abaixo.

2.1.5.1. DML (Data Manipulation Language – Linguagem de Manipulação de Dados)

A DML suporta os comandos utilizados para manipular os dados dentro do banco de dados. Seus principais comandos são:

SELECT – utilizado para selecionar registros dentro do banco de dados com determinadas condições estabelecidas pelo usuário;

```
select * from acidentes;2
```

Código 1: Exemplo de um comando SQL para selecionar dados de uma tabela.

INSERT – utilizado para inserir registros dentro do banco de dados;

```
insert into aux_acid_tipo(cod_tipo,desc_tipo) values (100,'Trator');
```

Código 2: Exemplo de um comando SQL para inserir dados na tabela 'aux_acid_tipo'.

UPDATE – utilizado para atualizar e/ou modificar registros já existentes;

```
update logradouros set cod_logradouro = 5 where nome_logradouro = 'JOÃO NAVES DE ÁVILA';
```

Código 3: Exemplo de um comando SQL para atualizar o valor de código do logradouro dos registros com o nome 'JOÃO NAVES DE ÁVILA'

DELETE – utilizado para remover tuplas (linhas) de determinadas tabelas que atendem condições especificadas pelo usuário;

```
delete from acidentes where the_geom is null;
```

Código 4: Exemplo de um comando SQL para remover os registros da tabela acidentes quando a coluna 'the_geom' tiver valor nulo.

2.1.5.2. DDL (Data Definition Language – Linguagem de Definição de Dados)

A DDL suporta os comandos utilizados para definir o formato dos dados dentro do banco de dados e suas tabelas. Seus principais comandos são:

CREATE - cria um objeto dentro do banco de dados;

```
create table aux_acid_tipo(
cod_tipo integer,
desc_tipo Varchar(20),
constraint aux_acid_tipo_pk primary key(cod_tipo));
```

Código 5: Exemplo de um comando SQL para criar uma tabela 'aux_acid_tipo' com as colunas cod_tipo e desc_tipo dos tipos inteiro e texto, respectivamente.

DROP – deleta um objeto dentro do banco de dados;

```
drop table aux_acid_tipo;
```

Código 6 - Exemplo de um comando SQL para remover a tabela 'aux_acid_tipo' do banco de dados.

ALTER – altera um determinado objeto dentro do banco de dados

² O asterisco dentro da consulta representa “todas as colunas da tabela”.

```
alter table acidentes
add constraint foo primary key(the_geom);
```

Código 7: Exemplo de um comando SQL para alterar a tabela 'acidentes'. Neste caso adicionamos uma restrição à tabela, mas todo tipo de alteração é possível.

2.1.5.3. DCL (*Data Control Language – Linguagem de Controle de Dados*)

A DCL suporta os comandos utilizados para dar ou revogar permissões sobre os objetos dentro do banco de dados. Esta subseção da SQL é usada para controlar quem terá acesso aos dados e qual o nível de acesso.

2.1.5.4. DTL (*Data Transaction Language – Linguagem de Transação de Dados*)

A DTL suporta comandos relativos ao início e final de transações. Estes comandos são importantes pois controlam a propriedade do banco de dados Consistência. É possível, através destes comandos, confirmar ou não determinada transação, mantendo o estado consistente do banco de dados.

2.1.5.5. *Cláusulas de Consulta e Operadores*

Como especificado anteriormente a SQL dá suporte à realização de várias consultas nos dados contidos no SGBD. Apesar de ser possível consultar dados com um simples comando SELECT, este suporta diversas cláusulas e operadores. Uma linha específica do banco de dados é retornada para o usuário em um comando SELECT quando ela satisfaz todas as condições especificadas (quando uma proposição feita pelo usuário é verdadeira). Lembremos que através da SQL é possível retornar dados de diversas tabelas de uma só vez, através de *JOINS*³ relacionais mas, ainda assim, todos os dados devem atender às condições especificadas para serem considerados uma proposição verdadeira.

As cláusulas são instrumentos para se especificar premissas a serem avaliadas, sendo estas premissas compostas de valores e operadores.

2.1.5.5.1. *Cláusulas*

FROM (DE) – especifica de qual tabela queremos retirar os dados;

WHERE (ONDE) – utilizada para especificar as condições de seleção;

³ *JOINS* são cláusulas especiais que permitem a união de diversas tabelas, através da comparação de dois campos (um em cada tabela), geralmente chaves estrangeiras e campos referenciados. São unidos e retornados os dados que são iguais em ambas as colunas.

GROUP BY (agrupar por) – utilizada para agrupar registros semelhantes baseados em um atributo da tabela;

HAVING (tendo) – utilizada para especificar um segundo conjunto de condições que todos os registros devem satisfazer

ORDER BY (ordenar por) – ordena os dados retornados baseados em um campo específico;

```
select *, count(nome_logradouro) from acidentes where data =
'10/10/2001' group by nome_logradouro order by cod_logradouro;
```

Código 8: Exemplo de um comando SQL SELECT utilizando algumas das cláusulas possíveis. Estas cláusulas podem ser utilizadas em qualquer tipo de comando DML.

2.1.5.5.2. Operadores

Existem dois tipos de operadores: os lógicos e os relacionais. Combinando os dois tipos de operadores tem-se a capacidade de criar consultas extremamente complexas retornando resultados que dificilmente seriam obtidos se não fosse utilizada uma ferramenta SGBD.

2.1.5.5.3. Operadores Lógicos:

AND (E) – especifica um operador “E”, e retorna os dados caso as duas condições sejam verdadeiras;

OR (OU) – especifica um operador “OU”, e retorna os valores caso uma das condições for verdadeira;

NOT (NÃO) – negação de uma condição. A negação de uma condição retorna os dados que não a satisfazem;

2.1.5.5.4. Operadores Relacionais

“<” - Menor que

“<=” Menor ou Igual

“<>” Diferente de

“>” Maior que

“>=” Maior ou Igual que

“=” Igual a

```
select *, count(nome_logradouro) from acidentes where data =
'10/10/2001' and hora > '12:00:00' and hora < '18:00:00' group by
nome_logradouro, order by cod_logradouro;
```

Código 9: Exemplo de um comando SQL SELECT utilizando-se alguns operadores lógicos e relacionais. Neste exemplo queremos todos os acidentes da data 10/10/2001 entre 12 horas e 18 horas.

Existem outros operadores e a junção de um ou mais operadores permite uma flexibilidade superior para pesquisar os dados sob investigação. A linguagem SQL é extremamente robusta e podemos “embrulhar” comandos em outros comandos, facilitando a manipulação e retorno de dados. Todos estes operadores listados aqui podem ser utilizados em qualquer parte da SQL, não somente na seleção dos dados, mas também na inserção, deleção e atualização dos mesmos.

2.1.6. Modelo E/R

O modelo E/R é uma maneira abstrata e conceitual de representar a organização dos dados. É um método utilizado para se projetar e construir bancos de dados, seguindo uma notação de desenho específica, na qual cada símbolo possui um significado. Este tipo de metodologia produz diagramas fáceis de serem “lidos” e interpretados possibilitando uma compreensão maior do objeto trabalhado. É equivalente à uma planta baixa de um imóvel.

O projeto de um banco de dados, comumente chamado por Diagrama Entidade/Relacionamento, possibilita definir a forma como os dados serão organizados conceitualmente, ou seja, o Modelo de Dados. Dizemos que o Diagrama E/R é um modelo conceitual, pois não especifica o modo com os dados serão gravados fisicamente, dentro do disco rígido (a parte responsável pela administração das escritas diretas no disco rígido é o SGBD).

Através de um diagrama E/R é possível representar as diversas características imagináveis apresentadas no tópico Teoria de Banco de Dados, sendo bastante flexível e utilizado mundo afora.

2.1.7 - O Modelo de Dados utilizado no Sistema de Cadastro de Acidentes

Conforme explicitado acima, o modelo de dados é fundamental para a construção de um banco de dados funcional, ágil e que, de fato, ajude no processo de análise e decisão.

Foram utilizadas neste trabalho tabelas contendo um rol de valores possíveis para alguns campos de cada entidade principal. As tabelas auxiliares cumprem exatamente este papel. Todas as tabelas auxiliares, com exceção da tabela “*aux_endereco*”, têm somente dois atributos: “*cod_atributo*” (código atributo) e “*desc_atributo*” (descrição atributo). Os códigos armazenam números inteiros representando a descrição correspondente. Nas tabelas principais são armazenados os códigos de atributos evitando o armazenamento repetido de valores descritivos e previne erros de digitação que possam ocorrer durante o cadastro.

O processo de modelagem dos dados foi dividido em três esquemas distintos (figura 9), de modo a facilitar a compreensão e a manutenção das diversas tabelas presentes no banco de dados. Os esquemas criados foram: “*customfunctions*”, “*geocode*” e “*public*”.



Figura 9: Vista dos esquemas dentro do SGBD (PgAdmin III).
Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009

2.1.7.1. Esquema “*customfunctions*”

O esquema “*customfunctions*” foi criado para atender a necessidade de abrigar algumas funções de propósitos gerais. Este esquema não abriga nenhuma tabela e, portanto, não possui diagrama E/R.

2.1.7.2. Esquema “*geocode*”

Este esquema é de extrema importância para o banco de dados em geral. Ele abriga todas as informações necessárias para que o *PostGIS* geocodifique os endereços passados pelos usuários em tempo de execução.

O esquema “*geocode*” (figura 10) abriga apenas três tabelas, que são usadas por diversas funções internas para construir a base geocodificada, índices e visões de consultas. As tabelas são:

“*geocoders*”: esta tabela contém as informações sobre os serviços geocodificadores que podemos construir. É desta tabela que o software tem referência de ONDE procurar cada informação necessária ao processo de geocodificação.

“*dirs*”: nesta tabela são armazenadas informações sobre direções usuais que podem aparecer no sistema de endereçamento. Esta tabela foi concebida para o uso de pontos cardeais (N, NW, NE, S, etc.) comum em outros sistemas de endereçamento (como o dos Estados Unidos). O Brasil, em específico, não usa este tipo de informação para identificar seus logradouros, portanto foi adaptada e transformada em um repositório de logradouros e suas interseções anteriores e posteriores.

“*types*”: nesta tabela registram-se informações sobre os tipos de logradouros existentes e a melhor forma de comparação com a base geocodificada. Os tipos básicos de logradouros são: Avenida, Rua, Alameda, Praça, Travessa, Rodovia. Existem diversos tipos de logradouros registrados nos Correios, alguns bastante incomuns, como “Morro”, portanto limitou-se esta tabela aos logradouros mais comuns, citados acima. Fonte: (CORREIOS, 2009)

Além destas tabelas o esquema “*geocode*” possui vinte e duas funções que não serão relatadas nesta seção devido ao seu tamanho. Estas funções ou métodos permitem ao sistema localizar o ponto exato dentro de um logradouro e abrangem as diversas etapas do processo de geocodificação.

dirs	geocoders	types	view_geocode_udi
corrected: varchar(10485760) regex: varchar(10485760)	geocoder_name: varchar(10485760) table_name: varchar(10485760) left_from_add: varchar(10485760) right_from_add: varchar(10485760) left_to_add: varchar(10485760) right_to_add: varchar(10485760) zip5_left: varchar(10485760) zip5_right: varchar(10485760) city_left: varchar(10485760) city_right: varchar(10485760) prefix_direction: varchar(10485760) suffix_direction: varchar(10485760) street_type: varchar(10485760) streetname: varchar(10485760) geom: varchar(10485760) directions_table: varchar(10485760) types_table: varchar(10485760)	corrected: varchar(10485760) regex: varchar(10485760)	left_from_add: int4 right_from_add: int4 left_to_add: int4 right_to_add: int4 zip5_left: int4 zip5_right: int4 city_left: varchar(50) city_right: varchar(50) prefix_direction: varchar(50) suffix_direction: varchar(50) street_type: varchar(30) streetname: text(10485760) geom: geometry minadd: int4 maxadd: int4 searchadd: varchar(10485760) standardadd: varchar(10485760)

Figura 10: Diagrama E/R do esquema “*geocode*”.

Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009

2.1.7.3. Esquema “public”

O esquema lógico “*public*” é o local onde todas as operações do banco de dados ocorrem e são armazenadas. Este esquema é o mais importante para o sistema, pois é nele que se armazenam todas as informações relativas aos acidentes, veículos, condutores, eixos de logradouros, índices e consultas pré-fabricadas (*views*), destinadas à geração de relatórios.

Dentro deste esquema segmenta-se seu uso conforme a entidade principal utilizada. Existem três tabelas principais: acidentes, veículos e condutores. Nestas três tabelas armazenam-se todas as informações para cadastro e análise. Temos um total de trinta e duas tabelas, sendo duas de sistema (“*geometry_columns*” e “*spatial_ref_sys*”), necessárias para o funcionamento da extensão espacial *PostGIS* e suas funções.

TABELA	DESCRIÇÃO
acidentes	Informações sobre o acidente
veiculos	Informações sobre os veículos envolvidos
condutores	Informações sobre os condutores envolvidos
aux_acid_caracteristica	Característica do Acidente
aux_acid_caracteristicavia	Característica da Via onde o acidente ocorreu
aux_acid_causa	Causa do Acidente
aux_acid_clima	Clima no momento do Acidente
aux_acid_condicaovia	Estado atual da Via onde o acidente ocorreu
aux_acid_controletrafego	Forma de Controle de Tráfego no local do acidente
aux_acid_orgaoresp	Órgão Responsável pelo atendimento no acidente
aux_acid_pavimentacao	Tipo de Pavimentação na Via do Acidente
aux_acid_socorro	Tipo de Socorro prestado aos envolvidos
aux_acid_tipo	Tipo de Acidente
aux_bairros	Bairros
aux_cidades	Cidades
aux_cond_comportamento	Comportamento do Condutor
aux_cond_condicao	Condição Física do Condutor
aux_cond_escolaridade	Escolaridade do Condutor
aux_cond_habilitacao	Situação da Habilitação do Condutor
aux_cond_profissao	Profissão do Condutor
aux_condsexo	Sexo do condutor
aux_enderecos	Endereços completos dos logradouros
aux_estados	Estados da Federação
aux_tipo_logradouro	Tipo de Logradouro
aux_veic_apreendido	Situação do Veículo
aux_veic_equipamentoseg	Indica se o veículo possuía equipamentos de segurança
aux_veic_sentido	Sentido do Veículo na Via
aux_veic_tipo	Tipo do Veículo

geometry_columns	Tabela de sistema
<i>continuação</i>	
TABELA	DESCRIÇÃO
logradouros	Logradouros
spatial_ref_sys	Tabela de sistema
quadras_poligono	Tabela espacial com as quadras da cidade de Uberlândia

Quadro 2: Listagem das tabelas no esquema public.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

Todas as tabelas auxiliares apresentam o prefixo “aux_” para denotar seu uso. Foi utilizada a seguinte metodologia para nomeá-las:

“aux_”+ abreviação da entidade à qual a tabela se relaciona + atributo relacionando

As únicas exceções a esta metodologia foram tabelas de propósito mais amplo, como as tabelas “aux_enderecos”, “aux_estados”, “aux_cidades”, “aux_tipo_logradouro”, “aux_bairros”. A descrição detalhada das principais tabelas do esquema “public” segue abaixo:

“*acidentes*”: esta tabela contém todas as informações relativas a entidade conceitual acidente, suas características, causas, fatores externos (condição e tipo das vias, clima), endereço da ocorrência, número do boletim de ocorrência para o acidente e a geometria de cada acidente (produto da geocodificação).

“*veículos*”: esta tabela contém as informações relativas aos veículos envolvidos em determinado acidente. Tem uma relação M:1 com a tabela acidentes e uma relação 1:1 com a tabela condutores (somente um condutor por veículo). Nesta tabela armazena-se uma informação de extrema importância para o órgão gestor: a severidade total de cada veículo.

“*condutores*”: esta tabela contém as informações relativas aos condutores envolvidos no acidente, assim como a severidade individual de cada condutor, suas condições físicas, situação de sua habilitação e um campo para eventual entrada de número de prontuário do condutor.

Apresenta-se a seguir o diagrama E/R do esquema “public”, (figuras 11, 12, 13 e 14) dividido em três partes, para facilitar sua compreensão e visualização do projeto do banco de dados.

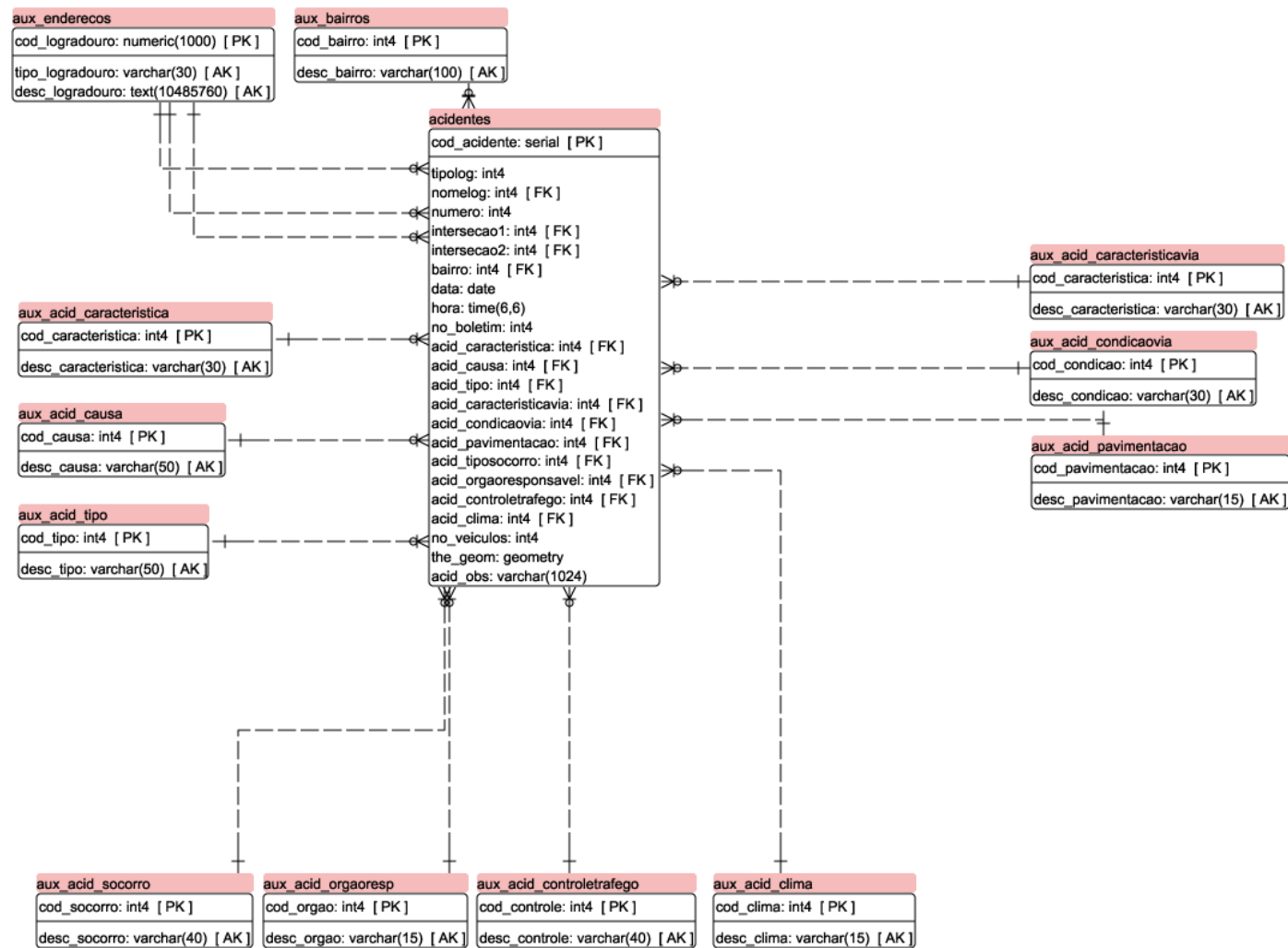


Figura 11: Diagrama E/R relativo à entidade acidentes.
 Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

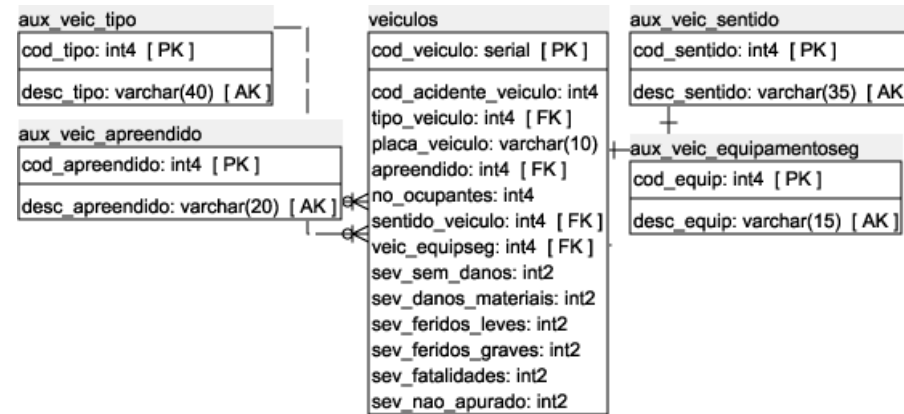


Figura 12: Diagrama E/R relativo à entidade veículos.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

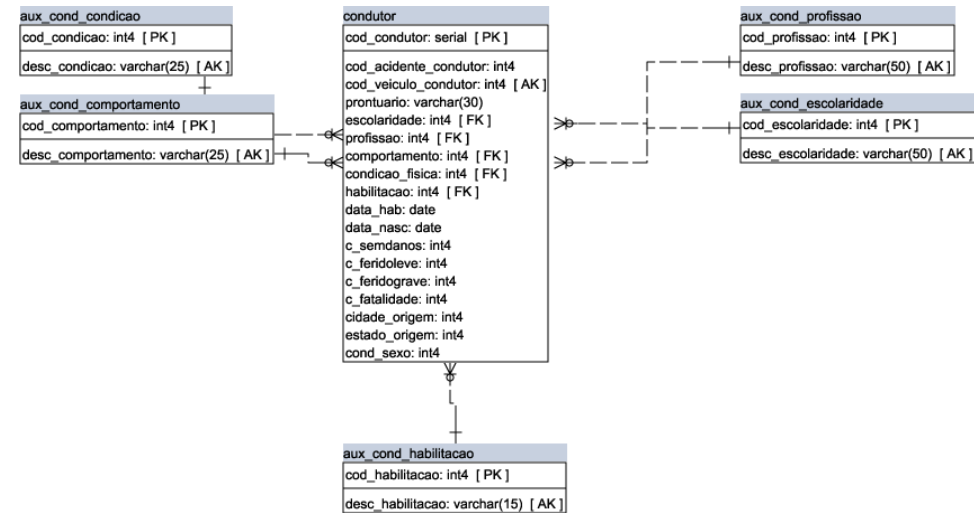


Figura 13: Esquema E/R relativo à entidade condutores.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

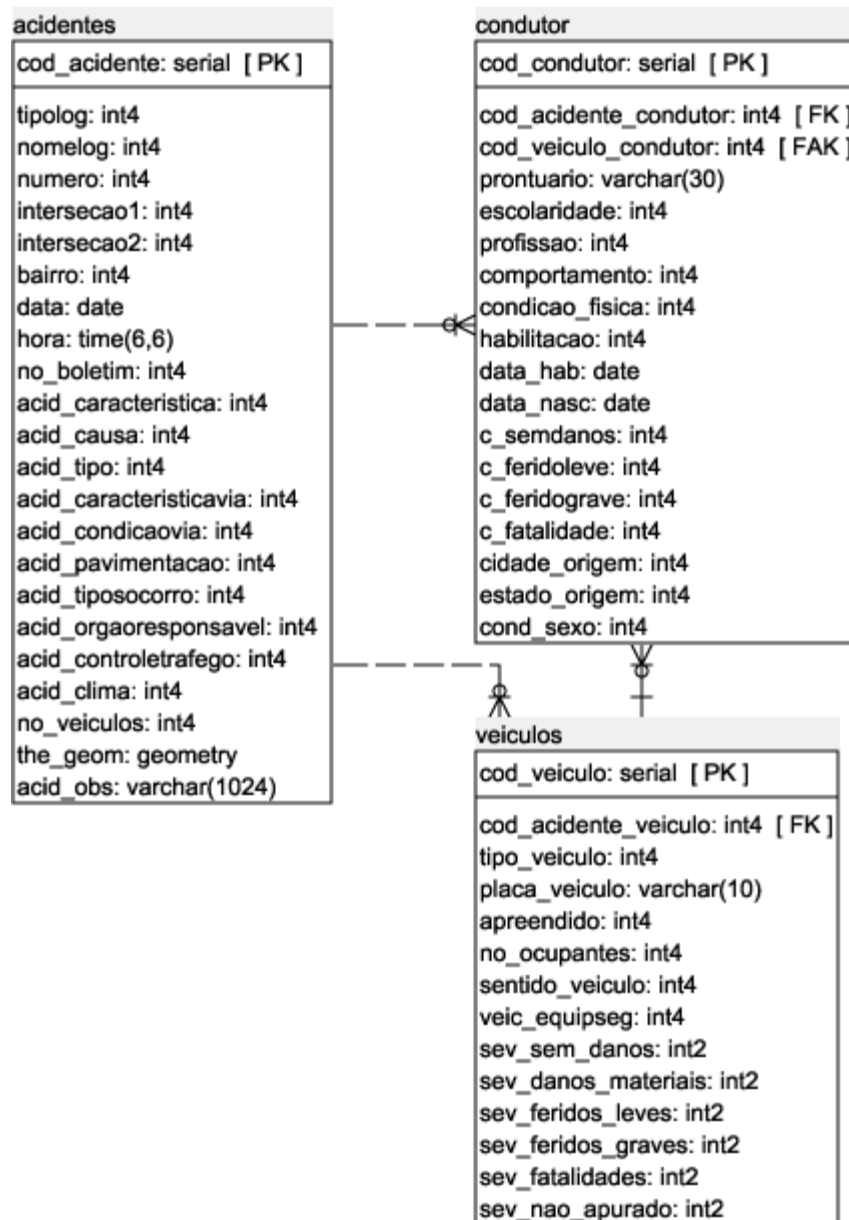


Figura 14: Esquema E/R apresentando as relações entre as tabelas principais do sistema.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

2.1.8. PostgreSQL

Existem no mercado diversos SGBDs que suportam dados espaciais em seus diversos formatos: *Oracle*, *MSSQL SERVER*, *PostgreSQL* e até mesmo o *MySQL*. A escolha do SGBD se baseou em um critério simples: custo x benefício. O *PostgreSQL* foi o primeiro banco de dados a suportar dados espaciais e sua extensão espacial é a de maior maturidade entre todos os fabricantes e seu custo é zero, sendo uma ferramenta livre de trabalho.

O *PostgreSQL* é um banco de dados robusto de primeira linha, oferecendo maiores funcionalidades avançadas à um custo virtualmente nulo em comparação com os outros SGBDs.

Conforme apresentado acima, a extensão espacial do *PostgreSQL*, o *PostGIS* é um projeto bastante maduro e é muito simples carregar seus dados em um sistema de *WebMaps*, como o utilizado neste trabalho (*GeoServer* + *OpenLayers*, futuramente discutidos). A integração com pacotes de *WebMaps* facilita de forma significativa o trabalho de renderizar e publicar os mapas propostos no trabalho, tudo isto de uma maneira “*seamless*” (integração perfeita, sem a necessidade de conversão de dados ou adaptação nos mesmos).

O *PostgreSQL* é um banco de dados com mais de quinze anos de desenvolvimento contínuo, com forte reputação tanto no meio acadêmico quanto com as indústrias de TI. Apresenta funcionalidades avançadas como *MVCC* (*Multi Version Concurrency Control*), *PITR* (*Point in Time Recovery*), replicação assíncrona, diversos idiomas e implementa completamente as regras ACID (discutidas no início deste capítulo).

2.1.8.1. Histórico e Características

Existem três ciclos de desenvolvimento do SGBD *PostgreSQL*:

Ciclo de desenvolvimento pela *Computer Associates*

Ciclo de desenvolvimento por Michael Stonebraker e estudantes da UCB

Ciclo de desenvolvimento *Open Source*

Inicialmente, ele se chamava *Ingres* e era desenvolvido pela *Computer Associates* (1977-1985). Após este período um professor de ciências da computação chamado Michael Stonebraker assumiu o controle do projeto, e, junto com diversos alunos, desenvolveu o software *Postgres* (“pós – *Ingres*”) de 1985 a 1994.

Em 1995, o *Postgres* assumiu, de forma definitiva, o padrão da indústria – a linguagem SQL, e foi renomeado para *Postgres95*. Após este período, o software foi levado para o mundo *Open Source*, onde teve seu desenvolvimento continuado e tendo alterado novamente seu nome para *PostgreSQL*.

LIMITE	VALOR
Tamanho máximo de banco de dados	Ilimitado
Tamanho máximo de tabela	32 TB
Tamanho máximo de linha	1.6 TB
Tamanho máximo de um campo	1 GB
Número máximo de linhas por tabela	Ilimitado
Número máximo de colunas por tabela	250 a 160, dependendo do tipo das colunas
Número máximo de índices por tabela	Ilimitado

Quadro 3: Alguns números do PostgreSQL

O *PostgreSQL*, além disto, é totalmente customizável, suportando linguagens de programação mais complexas, que ampliam de forma significativa o leque de opções para os desenvolvedores/usuários. Uma matriz de funcionalidades completa pode ser encontrada em <http://www.postgresql.org/about/featurematrix>.

2.1.9. PostGIS

O *PostGIS* é a extensão de dados espaciais para o SGBD *PostgreSQL*. Através desta extensão é possível armazenar, manipular e analisar dados espaciais, tornando o *PostgreSQL* um excelente meio de trabalho com os Sistemas de Informações Geográficas.

O *PostGIS* implementa a capacidade de criação de atributos do tipo geometria. Este campo pode armazenar dados espaciais do tipo PONTO, MULTIPONTO, LINHA, MULTILINHA, POLÍGONO, MULTIPOLÍGONO, CURVA, ARCO, e todas estas estruturas em até quatro dimensões (nativas dentro do campo geometria: x, y, z e m).

Além disso, a análise de dados dentro do sistema *PostGIS* é bastante simplificada, pois complementa a especificação a linguagem SQL, sendo possível “embrulhar” diversas funções de natureza espacial, eliminando etapas de trabalho e a necessidade de trabalhar com diversos arquivos.

```
select *, st_buffer(st_centroid(the_geom),100) from quadras;
```

Código 10: Exemplo de código SQL solicitando a seleção de uma expansão do centróide de todas as quadras em 100 unidades de medida. Note que a função ST_Centroid foi embrulhada pela função st_buffer.

2.1.9.1. Histórico e Características

O *PostGIS* nasceu de fato em 2001, utilizando-se de um padrão chamado “*SFS for SQL*” (Feições Simples para SQL em inglês) estabelecido pela *OGC* (*Open Geospatial Consortium* – descrito no capítulo Mapas Dinâmicos). O padrão estabelecido descreve as

formas escrita e leitura computacional de dados espaciais, funções espaciais padrão e um formato de comunicação padrão com outros sistemas.

Neste primeiro momento desenvolveu-se um tipo de atributo composto chamado *geometry*, atendendo as primeiras necessidades dos usuários, acessando tabelas com mais de um milhão de feições geográficas com tempos na casa dos milissegundos. (Refractons Research Inc. 2008).

O sistema continuou a evoluir a lentos passos até a versão 0.8, quando foi lançada uma biblioteca chamada *GEOS (Geometry Engine Open Source)* e incorporada às fontes do *PostGIS*. As novas funcionalidades trazidas pela *GEOS* ajudaram a difundir e consagrar a extensão espacial.

Diversas funções foram adicionadas a cada nova versão, trazendo-nos à atual, utilizada neste trabalho: *PostGIS 1.3*; O desenvolvimento do software continua, hoje, suportando funcionalidades (geocodificação, roteamento, armazenamento e análise de arquivos raster) antes somente disponíveis em SIGs de grande porte e alto valor de mercado (como o *ArcGIS* da *ESRI* e o *MapInfo* da *PitneyBowes*).

2.1.10. Dados utilizados

A empresa VERTRAN – Gerenciamento e Controle de Tráfego forneceu uma base de acidentes do ano de 2005 para ser testada no sistema. Houve dificuldades iniciais para se trabalhar com a base de dados fornecida, principalmente porque o modelo de dados utilizado na época é bastante diferente do proposto nesta pesquisa. Além disso, foi possível identificar algumas falhas básicas de desenho no modelo de dados de 2005, como: tabelas repetidas com nomenclaturas parecidas, funções e métodos de manipulação dos dados obscura e difícil de ser interpretada.

É bom frisar a importância do esquema conceitual do banco de dados, pois um bom esquema possibilita o crescimento “sadio” do banco de dados e sua expansão conceitual/funcional. Outro ponto importante é a dependência das funções que executam a geocodificação do esquema conceitual, sendo difícil e pouco prático converter um para o outro.

A partir deste ponto, existiam duas possibilidades: a criação de uma ferramenta complexa de *ETL (Extract, Transform e Load – Extração, Transformação e Carga)* para

popular os dados antigos no esquema atual e geocodificar os endereços dos acidentes ocorridos no ano de 2005 ou então construir uma ferramenta capaz de gerar acidentes completos aleatoriamente. A opção escolhida foi a segunda, por ser mais simples e de possibilitar o teste exaustivo do algoritmo geocodificador, quantas vezes necessário e avaliar seus índices de acerto/erro, apesar de gerar dados fictícios.

2.1.10.1. Ferramenta Geradora de Acidentes

A ferramenta construída foi escrita em linguagem nativa do *PostgreSQL*: a *plpgsql*, que roda dentro do próprio banco e com propósitos únicos, garantindo o acesso apenas à administradores do sistema.

O trabalho foi dividido em quatro etapas, assim delineado:

- 1 - Construção do gerador de acidentes primitivo;
- 2 - Construção do gerador de veículos primitivo;
- 3 - Construção do gerador de condutores primitivo;
- 4 - Construção de uma função capaz de concatenar todos os dados aleatórios gerados e inserir no banco de dados, n vezes, conforme especificado em seus parâmetros de execução.

A maior dificuldade da construção de tal ferramenta foi limitar os parâmetros de acordo com uma realidade “possível”, eliminando a existência de acidentes que não poderiam ou ocorreriam com muita raridade (ex: um acidente com 50 veículos juntos). Para melhor compreensão desta ferramenta determina-se um organograma de seu funcionamento (figura 15).

Esta ferramenta somente pode ser executada com um parâmetro definido do tipo inteiro, que representa o número total de acidentes a serem gerados. Após a inserção de um acidente na tabela, o próprio sistema dispara um gatilho que ativa a geocodificação, conforme explicitado no capítulo 3, Geocodificação.

O desempenho desta ferramenta é importante para a geração de dados e a performance testada foi aceitável. O sistema consegue em torno de vinte minutos inserir até um milhão de acidentes (lembrando que sempre são inseridos mais de um veículo e condutor para CADA acidente).

Para uma análise de dados e geração de relatórios foi utilizado um número muito menor de acidentes, de forma a não congestionar o mapa e impedir sua visualização completa.

Nos testes realizados com os protocolos *WEB* (*WMS* e *WFS* – especificados no capítulo Mapas Dinâmicos) um número aceitável de acidentes gerados foi entre 500 e 1000 ocorrências.

As funções completas estão descritas no APÊNDICE I.

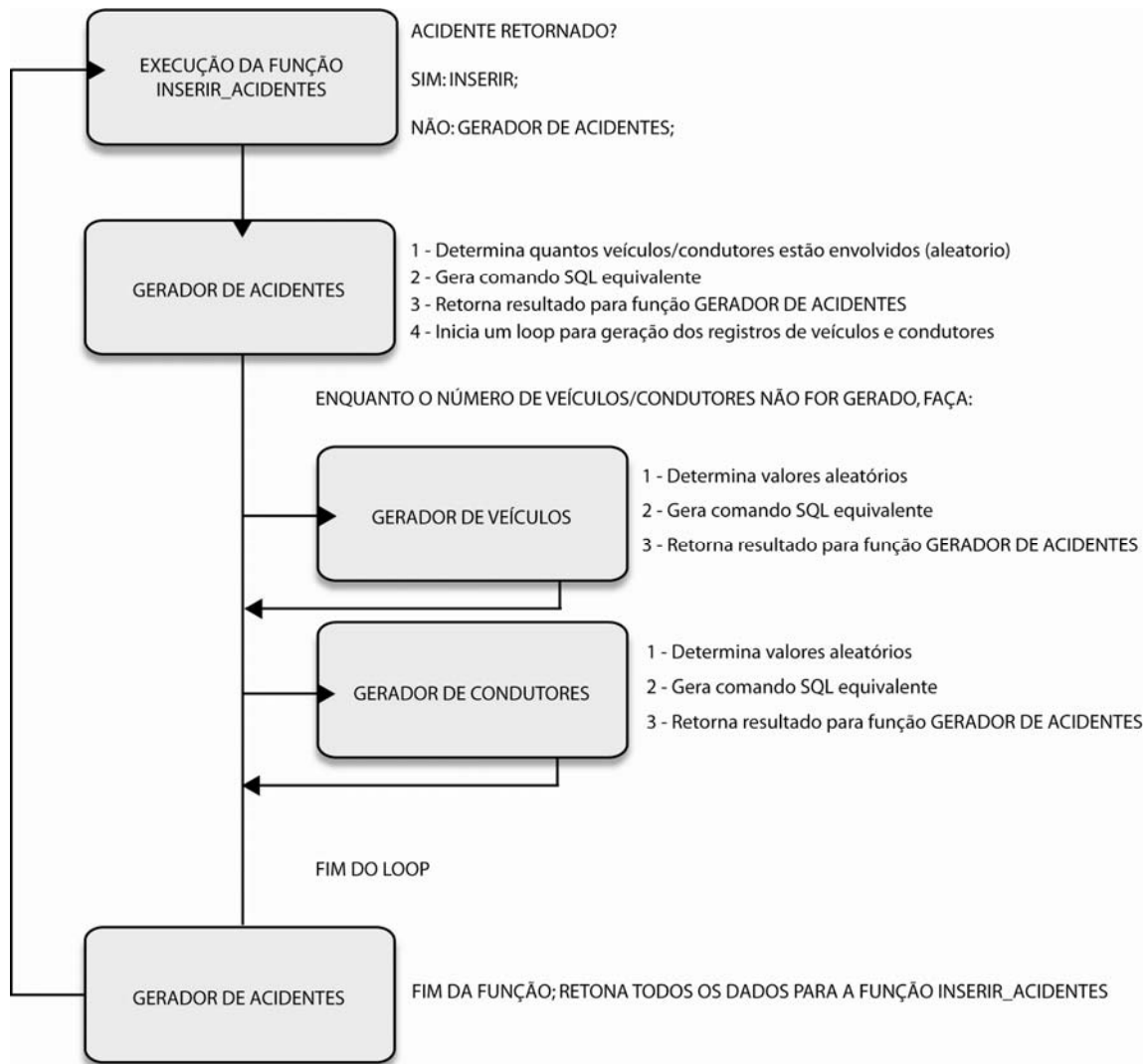


Figura 15: Organograma da função geradora de acidentes.
Fonte: SILVA, G. R. C 2009.

Futuramente, com maiores estudos e técnicas de programação será possível realizar funções geradoras nestes moldes, com propósitos preditivos, utilizando porcentagens e taxas de sistemas de análise de acidentes já estabelecidos Brasil afora para guiar a ferramenta de forma a construir um cenário fidedigno e com condições específicas (como tipo de pavimentação da pista e condições de tráfego).

CAPÍTULO 03

3. GEOCODIFICAÇÃO

A geocodificação é um sistema que permite localizar com relativa exatidão uma ou mais feições em um plano cartográfico através do uso de algoritmos e funções específicas. O fundamento que dirige a geocodificação é a transformação dados alfanuméricos (um endereço) em dados geográficos (um ponto específico com coordenadas).

De acordo com (ESRI, 2007, *Help Files*): “*Geocoding is the process of assigning a location, usually in the form of coordinate values (points), to an address by comparing the descriptive location elements in the address to those present in the reference material*”.

A geocodificação permite comparar informações alfanuméricas (endereços) com uma base de referência, dotada de parâmetros e geometria específicos, possibilitando a conversão dos dados alfanuméricos em informações geográficas.

Para tanto, a base de referência pode ser composta de diversas maneiras, suportando diversos tipos de endereçamento, desde sistemas de endereço métricos compostos por geometrias lineares (como o brasileiro e outros), como sistemas de endereçamento regionais compostos por polígonos (*ZIP codes* nos Estados Unidos).

O sistema de nível nacional Código de Endereçamento Postal (CEP) é a forma melhor estruturada de endereçamento, mas um CEP pode compreender partes de logradouros, áreas e até municípios inteiros, não sendo possível a localização precisa de determinado endereço através deste sistema.

No Brasil, o sistema de endereçamento é o sistema métrico, muito utilizado no mundo todo, em que cada logradouro possuiu um nome e numeração (do lado esquerdo e direito), sendo esta proporcional à distância percorrida desde seu ponto de início. Este sistema de endereçamento é possível de ser geocodificado e foi utilizado nesta pesquisa.

3.1. Elementos de Endereço

Todo endereço pode ser decomposto em partes menores para facilitar a localização e busca de determinado tipo de logradouro, logradouro, entre outros. Quando um endereço chega ao algoritmo geocodificador ele identifica as partes que o compõem e as quebra em pequenos pedaços, então chamados de elementos de endereço.

Os elementos de endereços mais comuns são: Tipo de Logradouro, Nome do Logradouro, CEP, Zona (pode ser um bairro, uma cidade, ou até mesmo localidades determinadas por outros métodos) e no caso dos serviços de geocodificação em outros países Direção da via (tanto em prefixos como sufixos).

No caso do presente trabalho foram empregados os seguintes elementos principais: Tipo do Logradouro, Nome do Logradouro e CEPs. Os outros elementos, apesar de estarem presentes na base de referência, são pré-requisitos para o serviço geocodificador funcionar, servindo apenas para desempate entre duas ou mais possíveis comparações positivas.

3.2. Construção da Base de Referência

Conhecendo as particularidades de cada sistema de endereçamento é possível elaborar uma base de logradouros ampla e confiável, permitindo de contemplar qualquer endereço que possa aparecer para o serviço geocodificador e este converter tal informação alfanumérica em informação geográfica.

A base de logradouros é composta pela linha central de cada logradouro (figura 16) do bairro Santa Mônica, em Uberlândia - MG, contendo informações suficientes de modo que o serviço geocodificador possa compará-la com os endereços entregues pelo usuário.

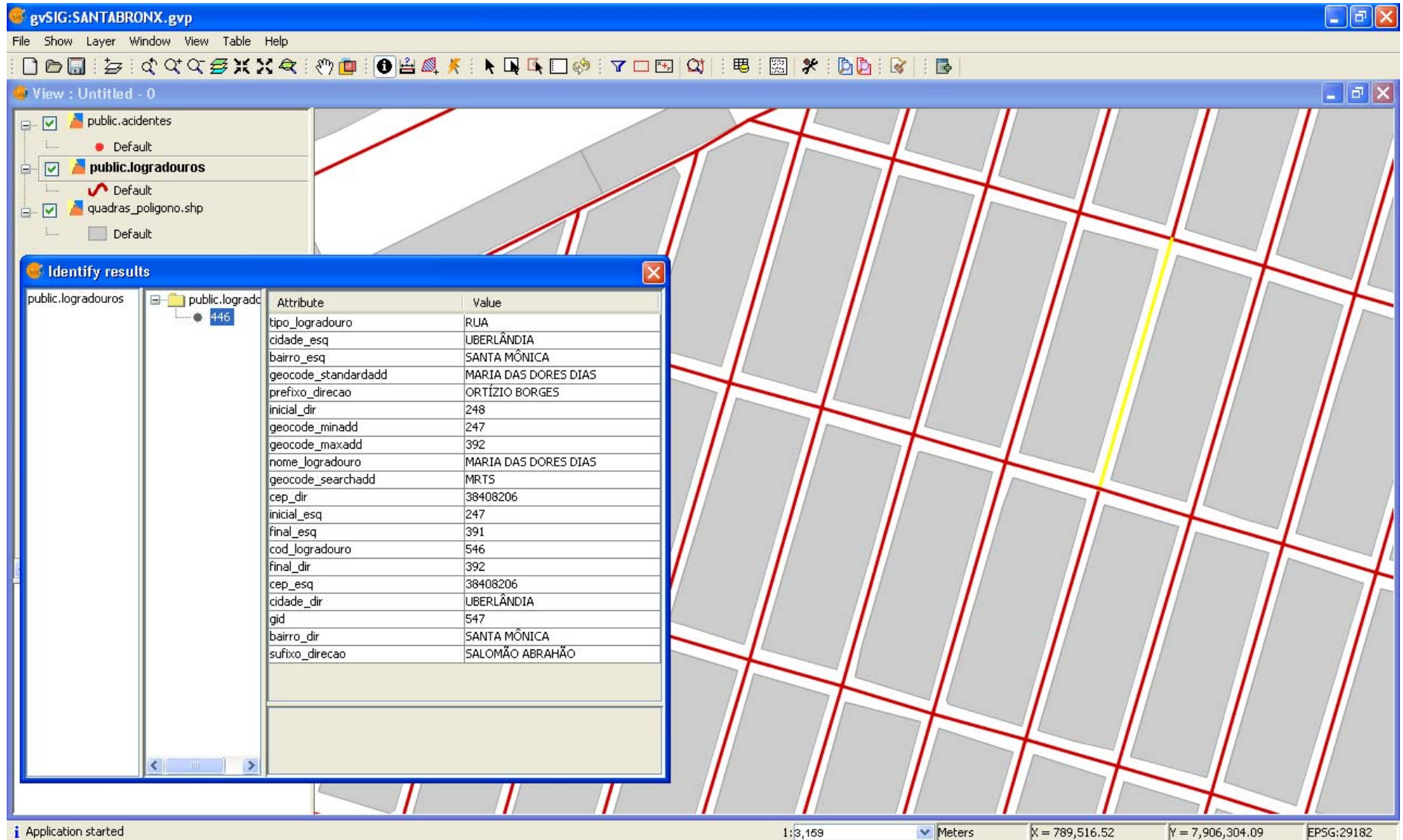


Figura 16: Linhas centrais de logradouros com vista da tabela de identificação.
 Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Pode-se notar que a base de referência requer um bom número de informações e isto se faz necessário para a criação de sistemas abrangentes, pois com mais informações relativas a cada logradouro pode-se aumentar a área do serviço, como diversas cidades, estados ou até mesmo um país inteiro, eliminando ambigüidades e endereços aparentemente similares.

Outra exigência na construção da base é a separação do logradouro por trechos, visando aumentar a precisão do serviço geocodificador, já que este funciona por interpolação, considerando os espaços formados por cruzamentos. É por esta exigência que o atributo “*cod_logradouro*” (código do logradouro) é incluído no sistema: um logradouro pode conter diversos trechos, e cada um destes trechos deve ser referenciado somente a um logradouro.

Outra diferença da maioria dos sistemas convencionais de geocodificação foi a inclusão dos atributos “*prefixo_direcao*” e “*sufixo_direcao*” permitindo uma alternativa na forma de endereçar um evento no caso de o serviço geocodificador receber um endereço incompleto, como: “Av. João Naves de Ávila, entre Av. Segismundo Pereira e Belarmino Cotta Pacheco” (neste caso sem o número viário).

3.2.1. Criação da tabela de logradouros

Dentro do SGBD PostgreSQL foi executado o seguinte comando para criar a tabela logradouros:

```
CREATE TABLE logradouros
(
gid serial NOT NULL,
nome_logradouro text,
the_geom geometry,
tipo_logradouro character varying(30),
inicial_esq integer,
inicial_dir integer,
final_esq integer,
final_dir integer,
cep_esq integer,
cep_dir integer,
cidade_esq character varying(50),
cidade_dir character varying(50),
bairro_esq character varying(50),
bairro_dir character varying(50),
prefixo_direcao character varying(50),
sufixo_direcao character varying(50),
cod_logradouro integer,
geocode_minadd integer,
geocode_maxadd integer,
geocode_searchadd character varying,
geocode_standardadd character varying,
CONSTRAINT logradouros2_pkey PRIMARY KEY (gid),
CONSTRAINT enforce_dims_the_geom CHECK (ndims(the_geom) = 2),
CONSTRAINT enforce_geotype_the_geom CHECK (geometrytype(the_geom) = 'MULTILINESTRING':text OR the_geom IS NULL),
CONSTRAINT enforce_srid_the_geom CHECK (srid(the_geom) = 29182)
);
```

Figura 17: código SQL para a criação da tabela logradouros
Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

3.2.2. Digitalização

Para compor este projeto piloto foi escolhido o bairro Santa Mônica, por dois motivos: primeiramente este bairro tem grande importância para a cidade de Uberlândia, concentrando serviços, comércio e o campus Santa Mônica da UFU, pólo gerador de tráfego e com uma concentração grande de pedestres. Além disso, o bairro contém ou está próximo de equipamentos urbanos importantes, como a Prefeitura Municipal de Uberlândia, o Center Shopping e o supermercado Carrefour.

O segundo motivo pela escolha do Bairro Santa Mônica foi pelo seu traçado regular e retilíneo de logradouros, facilitando a construção da base em escritório e a etapa de campo.

Os trechos de eixos de ruas foram digitalizados na escala 1:2000 utilizando-se ortofotos georreferenciadas (Prefeitura Municipal de Uberlândia, 2004) dentro do software gvSIG. Após a criação do projeto, conectou-se com o banco de dados PostgreSQL e teve início a digitalização.

Todos os trechos foram digitalizados com a ajuda de um Guia Sei 2005 e foram preenchidas algumas informações iniciais para identificar cada trecho de logradouro (“*tipo_logradouro*”, “*nome_logradouro*”, “*bairro_esq*”, “*bairro_dir*”, “*prefixo_dir*”, “*sufixo_dir*”, “*cep_esq*”, “*cep_dir*”, “*cidade_esq*”⁴, “*cidade_dir*”).

Foram gastos em média 30 horas para digitalizar todo o bairro e inserir as informações relacionadas acima. Ao todo foram digitalizados 999 trechos de 93 logradouros.

3.2.3. Trabalho de Campo

Após a digitalização de todos os eixos de logradouros foi realizado um trabalho de campo por todos os logradouros do bairro, coletando informações relativas aos números iniciais e finais, esquerdos e direitos.

Utilizou-se uma planilha formatada em Excel com os devidos locais para preenchimento destes dados e mapas para auxílio em campo.

⁴ Os atributos “*cidade_esq*” e “*cidade_dir*” foram incluídos no sistema de forma a possibilitar a criação de uma base ampla, abrangendo até diversos municípios. Através de atributo é possível identificar o lado da via em que determinado acidente ocorreu, caso esta via divida dois municípios. Além disso, é padrão de todos os algoritmos geocodificadores utilizarem uma referência à cidade em que determinado endereço está localizado, evitando confundir logradouros em municípios diferentes que tenham o mesmo nome.

O trabalho em campo durou duas semanas, nas quais todo o traçado viário do bairro foi percorrido, utilizando-se planilhas e mapas de referência, sendo possível identificar 80% dos números iniciais e finais de cada trecho de logradouro. Em algumas situações não era possível identificar os números pela existência de lotes ou construções sem número.

Para contornar este problema utilizou-se uma metodologia (figura 18) para interpolar os dados presentes, considerando o intervalo médio dos números viários em cada logradouro e assinalando este valor no banco de dados.

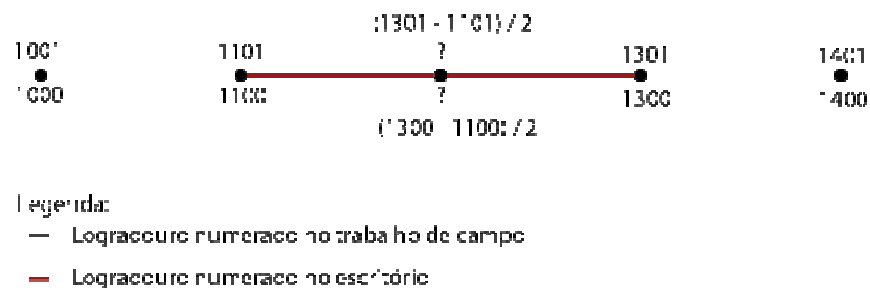


Figura 18 - Especificação da metodologia utilizada para assinalar valores iniciais e finais aos trechos que estes não foram identificados em campo.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

Com os dados em mãos, foram preenchidos os valores faltantes na tabela logradouros em escritório (“*inicial_esq*”⁵, “*inicial_dir*”, “*final_esq*”, “*final_dir*”)

⁵ Os atributos “*inicial_esq*”, “*inicial_dir*”, “*final_esq*” e “*final_dir*” representam, respectivamente, o número inicial esquerdo do trecho, o número inicial direito do trecho, o número final esquerdo do trecho e o número final direito do trecho.

Query - dummy em postgres@localhost:5432 *

Arquivo Editar Consulta Favoritos Macros Visualizar Ajuda

dummy em postgres@localhost:5432

```
select * from logradouros order by gid
```

Painel de saída

Saída de Dados Explain Mensagens Histórico

	gid integer	nome_logradouro text	the_geom geometry	tipo_logradouro character varying(30)	inicial_esq integer	inicial_dir integer	final_esq integer	final_dir integer	cep_esq integer	cep_dir integer	cidade_esq character var	cidade_dir character var	bairro_esq character var	bairr chara
1	2	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	21	20	85	86	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
2	3	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	91	92	189	190	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
3	4	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	195	196	269	270	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
4	5	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	285	286	313	314	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
5	6	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	319	320	399	400	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
6	7	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	403	404	499	498	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
7	8	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	501	500	563	564	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
8	9	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	585	586	621	622	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
9	10	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	645	646	687	688	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
10	11	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	709	710	739	740	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
11	12	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	771	770	807	808	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
12	13	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	817	818	871	872	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
13	14	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	875	876	963	964	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
14	15	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	967	968	1039	1040	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
15	16	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1123	1124	1159	1160	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
16	17	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1177	1178	1219	1219	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
17	18	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1225	1226	1269	1270	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
18	19	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1301	1302	1335	1336	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
19	20	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1369	1370	1395	1396	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
20	21	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1441	1442	1545	1546	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
21	22	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1549	1550	1599	1598	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
22	23	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1601	1600	1655	1654	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
23	24	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1657	1658	1681	1682	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
24	25	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1781	1780	1787	1788	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
25	26	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1835	1836	1851	1852	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA
26	27	BELARMINO COTTA PACHECO	0105000020FE7	AVENIDA	1865	1866	1867	1868	38408168	38408168	UBERLÂNDIA	UBERLÂNDIA	SANTA MÔNICA	SANTA

OK. Unix Lin 1 Col 39 Ch 39 999 registros. 313 ms

Figura 19: Visão da tabela finalizada de logradouros dentro do banco de dados.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

3.3. Serviços de Geocodificação

Um serviço de geocodificação é um conjunto de regras para as quais o algoritmo geocodificador utilizará para comparar com a base de referência, e está armazenado no esquema conceitual “*geocode*” da base de dados.

Antes que o algoritmo geocodificador possa, de fato, geocodificar um endereço ele precisa comparar o endereço a uma base de referência e é o serviço de geocodificação que realiza esta interface. O algoritmo não consegue “enxergar” a base, ele é apenas orientado pelo serviço, e este, de fato, tem conhecimento específico de onde procurar cada elemento de endereço (figura 20).

	geocoder_name	table_name	left_from_address	right_from_address	left_to_add	right_to_add	zip5_left	zip5_right	city_left	city_right	prefix_direct
1	ludi	logradouros	inicial_esq	inicial_dir	final_esq	final_dir	cep_esq	cep_dir	cidade_esq	cidade_dir	prefixo_direcao

Figura 20: Tabela do serviço geocodificador dentro do SGBD (PgAdmin III).

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

3.3.1. Construção do Algoritmo Geocodificador

A função usada neste estudo de caso foi desenvolvida nos Estados Unidos por Bitner, D.. A função foi desenvolvida sobre a licença pública e cedida através de um bate-papo informal, realizado em um sistema de comunicação chamado *IRC (Internet Relay Chat)*. Esta função foi escrita para trabalhar especificamente sobre a base de logradouros *TIGER (Topologically Integrated Geographic Encoding and Referencing System)*, disponível em < <http://www.census.gov/geo/www/tiger/>> e foi adaptada para processar endereços no sistema brasileiro e atender ao modelo de dados proposto.

Existem outros algoritmos capazes de realizar a geocodificação para o sistema *TIGER*, como o desenvolvido pela *Refractions Research Inc.*, mas o algoritmo utilizado tem uma instalação relativamente simples e alta taxa de sucesso.

Este algoritmo compreende todas as etapas citadas acima, sendo suas etapas de instalação bastante simples. O mesmo é o motor por trás do serviço de geocodificação usado no trabalho.

Deve se ressaltar que as duas funções que não a original (Interpolada) foram desenvolvidas especificamente para este trabalho, assim também como o gatilho que as disparam, automatizando o processo de cadastro de acidentes e sua localização.

3.3.1.1. Detalhe das funções geocodificadoras

No sistema do presente trabalho foram usadas três maneiras para converter informação textual em coordenadas geográficas, que serão apresentadas a seguir. Esta seção é uma breve descrição sobre o funcionamento de cada função, sem entrar em detalhes de suas mecânicas e algoritmos, por não compor o escopo do trabalho.

3.3.1.1.1. Interpolada

Este modo de operação do serviço funciona designando como total (100%) a extensão do trecho de logradouro baseado em sua numeração inicial e final e realiza uma interpolação linear para determinar em qual ponto do trecho de logradouro o endereço está localizado (figura 21). É o modo mais preciso de se determinar a posição de um evento qualquer através de um sistema geocodificador. Esta função será realizada sempre que o endereço passado esteja completo.

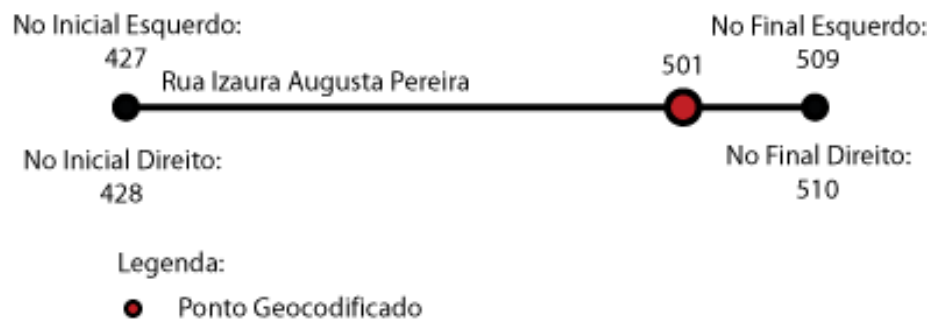


Figura 21: Modelo conceitual da geocodificação interpolada.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

3.3.1.1.2. Entre Logradouros

Esta função localiza os endereços sobre os quais não existem informações suficientes disponíveis para se localizar um evento em um ponto exato da extensão do logradouro ou são passados de forma incompleta para o serviço geocodificador. O resultado é então localizado no ponto médio do trecho de logradouro (figura 22), indicando sua posição de maneira aproximada.



Figura 22: Modelo conceitual da geocodificação por trechos ou entre logradouros.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

3.3.1.1.3. Cruzamentos

São localizados assim os endereços passados para o serviço geocodificador no formato “Rua X com Rua Y” ou parcialmente incompletos. O resultado desta função é a intersecção entre duas linhas, indicando o ponto exato do cruzamento (figura 23).



Figura 23: Modelo conceitual da geocodificação por cruzamento.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

As três funções servem a diferentes propósitos e a diferentes modelos de dados, e todas foram incorporadas neste modelo para garantir uma alta margem de sucesso no processo de geocodificação.

3.3.1.2. Fluxograma da geocodificação

Apesar de o conceito de geocodificação ser extremamente simples e direto, é necessário tratar as informações da base de referência e as informações passadas pelo usuário de modo a eliminar erros, sistemáticos (como o algoritmo confundir dois logradouros de nomes semelhantes) ou humanos (como a inserção de um logradouro inexistente ou digitado de forma incorreta, ex: “**Av. João Naves de Ávila**”). As etapas do algoritmo são descritas a seguir e na figura 24:

1 - Inserção do endereço (Avenida João Naves de Ávila, 2121, Bairro Santa Mônica);

2 - Passagem dos Elementos de Endereço ao Algoritmo (Avenida | João Naves de Ávila | 2121 | Santa Mônica);

Quando o endereço é passado ao algoritmo geocodificador ele é dissecado em elementos determinados pelo modelo de dados e base de referência para que seja efetuada a padronização de endereços.

3 - Abreviação dos Elementos e padronização (Av. | JNAVL | STMON);

Este passo se faz necessário para a padronização de certos elementos de endereço que podem tomar diversos formatos, para que não ocorram erros posteriores devido a esta gama de grafias, por exemplo, no caso do tipo do logradouro Avenida, que pode ser escrito como Av., Ave. ou até mesmo em sua forma completa avenida (o mesmo se aplica para todos os tipos de logradouros, como R.; Rua, Trv.; Travessa, entre outros).

A padronização de endereços também tem uma justificativa computacional, sendo muitas vezes mais rápido comparar um único formato de escrita com a base de referência ao invés de uma diversidade de formatos.

4 - Comparação dos Elementos de Endereço com a base de referência;

Este passo é de extrema importância e é crítico para a entrega de um bom resultado. Caso a base de referência esteja com nomes de logradouros errados, ou apenas parcialmente corretos, existe uma grande chance de a comparação falhar.

Outra característica fundamental desta etapa é o seu consumo de recursos computacionais - não é exatamente um processo expressivo, mas conforme o tamanho da base de referência aumenta, este tempo pode aumentar de forma significativa, sendo imprescindível

uma manutenção regular das tabelas que contém os dados de logradouros e a garantia absoluta de sua integridade relacional.

5 - Cálculo de valores índice;

6 - Estabelecimento de ranking de mais prováveis;

7 - Retorno dos resultados;

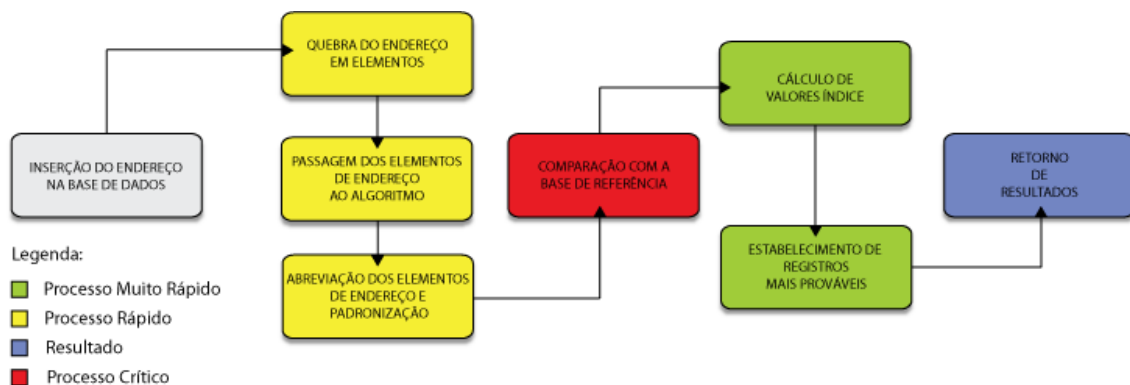


Figura 24: Organograma da função geocodificadora.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Após o retorno dos dados e dos valores de coordenadas, o sistema captura automaticamente este valor e o escreve na tabela “*acidentes*” em campo apropriado, transformando o par de coordenadas em um ponto. A escrita deste par de coordenadas em campo apropriado funciona de forma independente do cadastro do acidente em si, já que o usuário do sistema pode passar um endereço incompleto, ou impossível de geocodificar (com numeração não-existente na base de referência, por exemplo). O motivo desta separação não impede, portanto o registro do acidente que será processado e incluído em relatórios gerenciais, apenas indica a falha do processo de geocodificação.

Toda a metodologia deste trabalho visou à eliminação deste tipo de ocorrência (acidentes sem geometria pontual e falhas no processo de geocodificação), mas é impossível cobrir todas as situações sendo esta a melhor maneira encontrada para não se perder informações em uma base cadastral.

CAPÍTULO 04

4. CONSTRUÇÃO DO SOFTWARE CLIENTE

Em um projeto que envolve dados e cadastro, é fundamental apresentar ao usuário final do sistema uma interface agradável e que não exija uma formação específica do mesmo em Banco de Dados e ciências correlatas (incluindo SIG) para um melhor aproveitamento de pessoal e alocação de recursos. Neste sentido, foi desenvolvido um programa denominado de cliente para acessar as bases de dados.

O software cliente foi desenvolvido pelo autor utilizando a tecnologia .NET, bastante flexível, com diversas extensões e de uso gratuito (na versão Express) para projetos acadêmicos e estudo.

A plataforma .NET é uma iniciativa da Microsoft na criação de um ambiente único de desenvolvimento, conjugando diversas linguagens de programação (VB.NET, C#, J#,C++, ASP.NET) sob um único interpretador e integrando diversas outras através de extensões.

Todas as linguagens disponíveis na tecnologia .NET são Orientadas à Objetos (OOP – *Object Oriented Programming*), possibilitando maior dinamismo e reaproveitamento de código através do uso de classes, interfaces e módulos. Outra característica interessante da tecnologia em questão é a possibilidade de trabalhar com diferentes paradigmas de desenvolvimento, como os baseados em *Unit Tests* (testes padrões), metodologias ágeis, entre outros.

A linguagem utilizada para desenvolver o software em questão é o VB.NET por sua facilidade de aprendizado, ampla documentação e facilidade de se encontrar suporte técnico gratuito na rede.

4.1. Arquitetura em três níveis

Foi utilizada uma metodologia simples e consagrada para o acesso ao banco de dados e as informações: uma arquitetura em três níveis.

O modelo em três níveis geralmente é chamando também de *three-tier architecture* ou até de *n-tier architecture*, por não haver limites reais sobre o número de níveis em que se pode desenvolver.

Este modelo implica na separação lógica de cada nível: o que ocorre em um nível, não afeta os processos alojados em outro, eliminando a necessidade de repensar todo um sistema para alocar pequenas mudanças no banco de dados.

Os níveis em questão são: *Data Tier* (nível de dados), *Logic Tier* (nível lógico), *Presentation Tier* (nível de apresentação), representados na figura 25. Resumidamente, o *Data Tier* é compreendido por todos os processos que ocorrem dentro do banco de dados, enquanto o *Logic Tier* trabalha com as regras de negócio e constrói a ponte entre os outros dois níveis e o *Presentation Tier* controla como as informações são apresentadas para os usuários finais.

Data Tier: este nível, conforme especificado anteriormente é o responsável pelos processos dentro do banco de dados. Ele é, na verdade, o próprio banco de dados, com seu modelo, entidades, relações, visões e processos internos de busca e entrega de informações. Todos os processos deste nível são executados em um servidor, e nunca na máquina cliente.

Logic Tier: este nível é o que exige maior compreensão do modelo de dados utilizado e das regras de negócio de determinado banco. Neste ponto teremos objetos (esta definição de objeto é diferente da utilizada no tópico Teoria de Banco de Dados), classes e módulos acessando as informações requisitadas pelo *Presentation Tier* no *Data Tier* e entregando-as novamente ao *Presentation Tier*. Este nível é desenvolvido fora do banco de dados, utilizando algum ambiente de desenvolvimento compatível (.NET por exemplo). Este nível é executado tanto na máquina cliente como em servidores, já que é a ponte para os dois “conversarem”.

Presentation Tier: este nível é o que exige menor conhecimento dos processos internos do banco de dados e das regras de negócio e é destinado quase que exclusivamente aos usuários finais do sistema. É nele que apresenta a capacidade de cadastrar dados no banco de dados, gerar relatórios e visualizar os mapas gerados dinamicamente. É inteiramente executado na máquina cliente, desafogando os servidores com pedidos excessivos de dados.

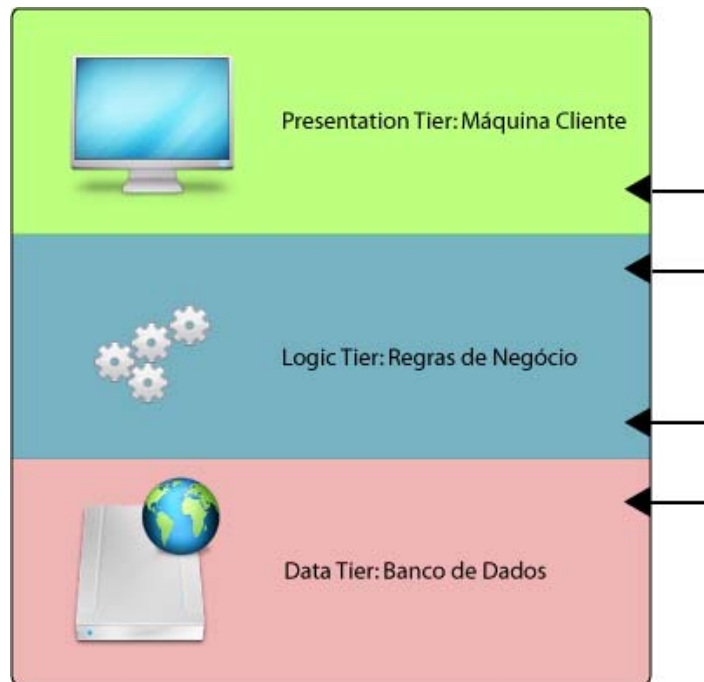


Figura 25: Representação conceitual do modelo three-tier architecture.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

4.2. Apresentação do Software

Neste tópico apresenta-se brevemente o software e algumas de suas funcionalidades. O software cliente foi separado em algumas telas que o usuário pode acessar e estas são explicadas em detalhes. São apresentadas as seguintes telas: *Login*, Tela Principal, Configurações, Cadastro de Acidentes, Cadastro de Variáveis do Sistema (estas variáveis de sistema são as apresentadas nas tabelas auxiliares, que controlam o que o usuário final pode ou não inserir em determinados campos), Cadastro de Usuários, Terminal Interativo, Relatórios e Mapas.

4.2.1. Login

A tela de Login (figura 26) é a primeira a aparecer para o usuário final. Ela tem a função de controlar o acesso aos dados do sistema. Somente usuários cadastrados podem ter acesso a outras telas do sistema. Ela é composta de um campo para o nome de usuário e outro para sua senha, bem como as opções de conectar e sair do software.

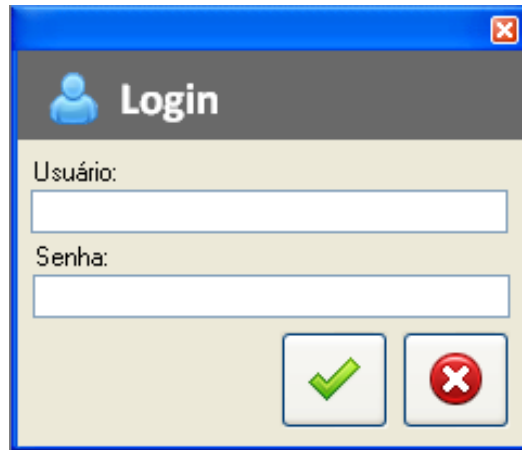


Figura 26: Tela de Login.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

4.2.2. Tela Principal

A partir desta tela (figura 27) podemos acessar todas as funcionalidades do software. Ela tem função de organizar e facilitar o acesso as informações, sem confundir o usuário final. No topo esta tela abriga o Menu de Opções, à direita ela mostra algumas estatísticas gerais do sistema, e na parte inferior o grupo de permissões em que o usuário se encontra, qual usuário está conectado e horário de sua entrada no sistema.

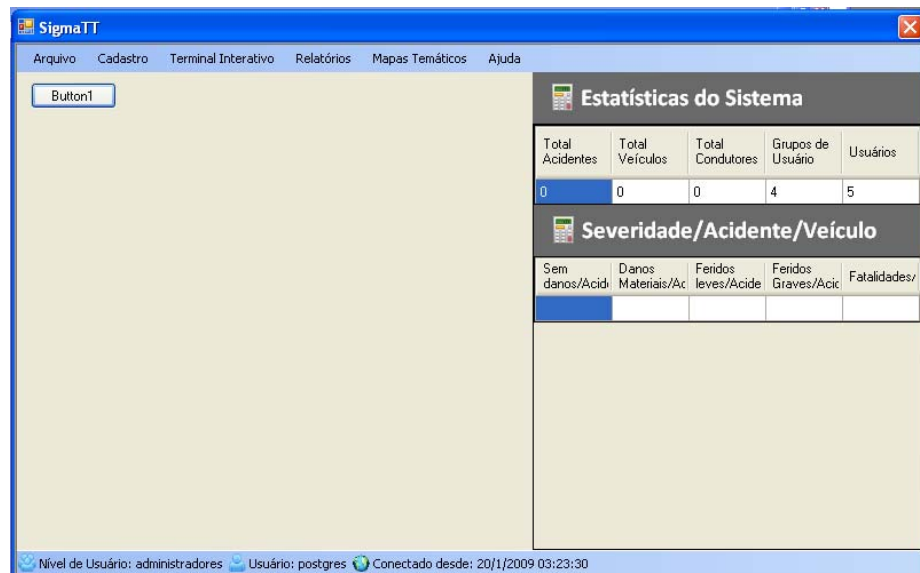


Figura 27: Tela principal do aplicativo cliente.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

4.2.3. Configurações

Esta tela (figura 28) apresenta algumas configurações disponíveis aos usuários finais, de forma a facilitar o uso do programa. Entre as opções disponíveis estão o tratamento automático de interseções, o aviso do resultado, e a opção do Terminal Interativo de funcionamento dos botões de termos de busca.

Tratamento de Interseções: Esta função tem efeito sobre a tela de cadastro de acidentes e, quando estiver ativada, assim que o usuário especificar o logradouro principal e um número, o sistema já busca as interseções, anterior e posterior, daquele trecho de logradouro. O usuário tem também a opção de ativar esta função e pedir para o sistema avisá-lo se foram encontradas (ou não, indicando um número viário não existente) as interseções anterior/posterior.

Terminal Interativo: esta opção controla os botões do Terminal Interativo que, quando clicados, inserem o valor SQL dos mesmos na linha de comando do terminal.

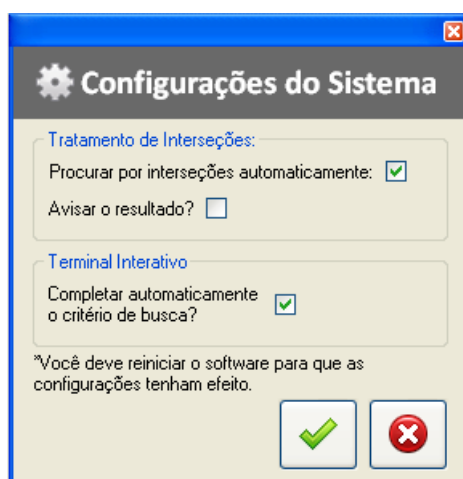


Figura 28: Tela Configurações do Sistema.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

4.2.4. Cadastro de Acidentes

Esta tela é a responsável pelo cadastro de acidentes na máquina cliente. É a tela mais complexa do sistema, já que lida com diversos acessos ao banco de dados e tráfego intenso entre os níveis da arquitetura.

Nesta tela pode-se notar um mapa interativo à direita, que o usuário pode utilizar para navegar na base de logradouros. Temos a opção de controlar o *Zoom* do mapa, navegar pelo

mesmo e resolver eventuais dúvidas sobre a localização de um acidente a ser cadastrado. Este mapa dinâmico foi criado utilizando a *API OpenLayers* e o servidor de mapas *Geoserver*, descritos em profundidade no capítulo WEBGIS.

Na parte superior esquerda existem três abas, separando o cadastro de Acidentes, Veículos e Condutores. Dentro das regras de negócio foram criados mecanismos para manter a integridade da base de dados, por exemplo: um usuário cadastra um acidente com três veículos envolvidos, mas somente cadastra dois veículos e três motoristas. Todas as informações relativas aos veículos esquecidos são valiosas e quebrariam a integridade da base. Esta foi uma das tarefas mais complexas para a construção do software, já que as possibilidades no cadastro deste tipo de evento são imensas.

No momento em que esta tela é carregada pela primeira vez, o sistema acessa a base de dados e traz todas as tabelas auxiliares para seus devidos campos, de forma que o usuário nunca poderá especificar uma característica que não foi pré-determinada.

Outra característica importante é a existência de uma metodologia para cadastro dos dados. É impossível cadastrar um veículo (ou condutor) sem que se cadastre um acidente primeiro, e impossível de se cadastrar um veículo e em seguida dois condutores. A ordem de cadastro deve ser seguida à risca: primeiramente o acidente, depois um veículo, o condutor do veículo em questão, outro veículo e assim por diante. O usuário ainda tem a liberdade de navegar pelas abas conforme for desejado, mas o cadastro está bloqueado para seguir esta ordem em particular.

Todas as três abas (Acidente – figura 29, Veículo – figura 30 e Condutores – figura 31) funcionam do mesmo modo: o usuário preenche as informações e adiciona o cadastro a uma área temporária, acessando o botão Cadastrar, na parte inferior do formulário. Neste momento o sistema acessa o banco de dados, transforma todas as descrições em código (conforme especificado no tópico Modelo de Dados) para serem posteriormente inseridos na tabela.

Após o término do cadastro de todas as informações (acidente, n veículos e n condutores) o sistema informa ao usuário que as informações estão prontas para o cadastro definitivo, que pode ser acesso no botão Inserir Acidente, na barra superior.

Cadastro de Acidentes Versão 0.1

Acidente Veículo Condutor

Informações Gerais:
 Código Veículo: Código do Acidente:

Veículo:
 Tipo de Veículo: Placa: Apreendido:
 Equipamento Segurança: Sentido Veículo: No Ocupantes:

Severidade:
 Sem Danos: D. Materiais: F. Leves: F. Graves: Fatalidades: N/A:

Código Veículo	Código Acidente	Tipo de Veículo	Placa	Apreendido?	No Ocupantes	Sentido Veículo	Eq. Segurança	Sem danos	Danos Materiais	Feridos leves

789536.54295, 7905447.37558

Escala = 1 : 13K

Figura 30: Tela Cadastro de Acidentes, aba Veículos.
 Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

Cadastro de Acidentes Versão 0.1

Acidente | Veículo | **Condutor**

Informações Gerais

Código Condutor: Código Veículo: Código Acidente:

Origem

Estado: Cidade: Habilitação: Data da Habilitação:

Condutor

Escolaridade: Comportamento: Data de Nascimento:

Profissão: Condição Física: Sexo:

Número do Prontuário:

Severidade:

☐ Sem Danos ☐ Ferido Leve ☐ Ferido Grave ☐ Fatalidade

Código Condutor	Código Veículo	Código Acidente	Estado de Origem	Cidade de Origem	Escolaridade	Profissão

789082.08208, 7905712.78073

Escala = 1 : 13K

Figura 31: Tela Cadastro de Acidentes, aba Condutor.
 Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

4.2.5. Cadastro de Variáveis de Sistema

Estas telas (figura 32 e 33) possibilitam ao usuário final a cadastrar outras variáveis que sejam de seu interesse nas tabelas auxiliares. Somente usuários com nível intermediário e administrador podem cadastrar estas variáveis, excluindo o acesso de usuários ao cadastro, mas permitindo a visualização de seu conteúdo. O propósito desta restrição é deixar o cadastro de variáveis de sistema na responsabilidade de um gerente pela operação do software.

Este cadastro é composto unicamente de dois campos: código e descrição, conforme a modelagem de dados descrita anteriormente.

Lembramos também que todo o sistema funciona em tempo real, não importando de onde um usuário intermediário cadastre a nova variável, automaticamente ela irá aparecer para todas as outras máquinas clientes como uma opção disponível.

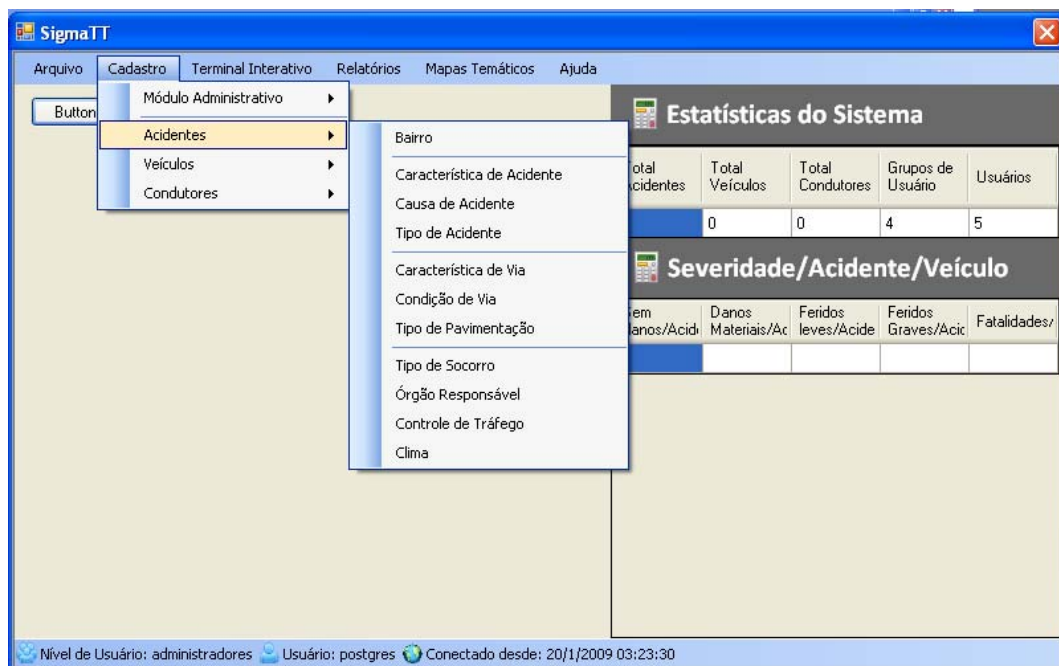


Figura 32: Tela principal, acesso às variáveis de sistema.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Cadastro de Variáveis de Sistema
Tipo de Veículo

Código:

Descrição:

cod_tipo	desc_tipo
1	PEDESTRE
3	BICICLETA
4	CICLOMOTO
5	MOTONETA
6	MOTOCICLETA
7	TRICICLO
8	QUADRICICLO
9	AUTOMÓVEL
10	MICRO-ÔNIBUS
11	ÔNIBUS
12	REBOQUE
13	SEMI-REBOQUE
14	CHARRETE/CARROÇA
15	CAMIONETA
16	CAMINHONETE
17	CAMINHÃO

Figura 33: Tela Cadastro de Variáveis de Sistema (Tipo Veículo).
Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

4.2.6. Cadastro de Usuários

O cadastro de novos usuários também foi previsto neste sistema. Apenas usuários de nível intermediário e administrador podem cadastrar, excluir usuários e determinar seu papel dentro do sistema. Usuários cadastrados com a opção Administrador cairão no nível intermediário e terão as habilidades equivalentes a de moderador (figura 34).

O nível Administrador real deve ficar a cargo de uma pessoa com conhecimentos avançados de Banco de Dados, Modelos Relacionais e SQL, pois ele pode virtualmente apagar todo o sistema, não sendo possível cadastrar um usuário neste grupo.

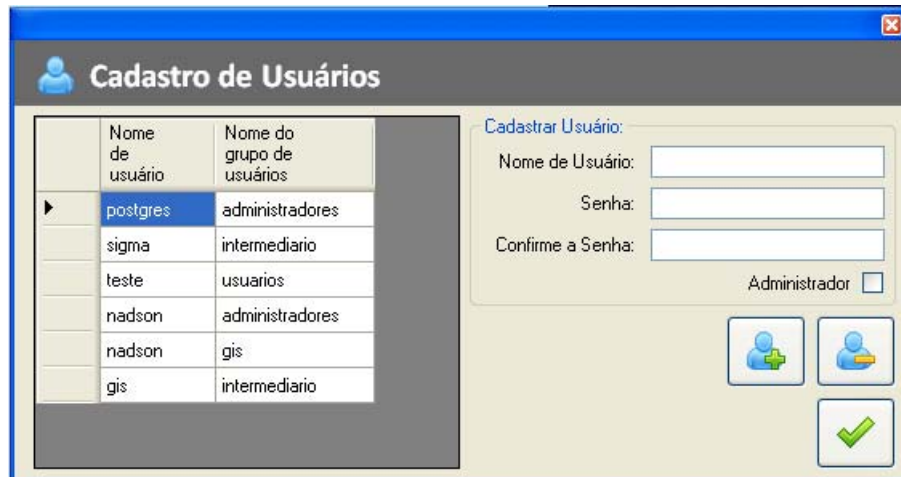


Figura 34: Tela Cadastro de Usuários em modo administrador.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

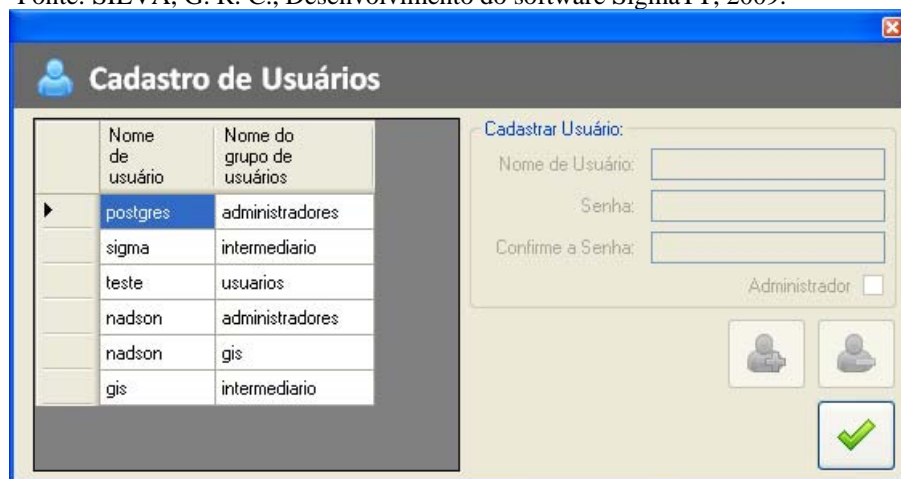


Figura 35: Tela Cadastro de usuários no modo usuário. Note que os botões e campos estão inacessíveis.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

4.2.7. Terminal Interativo

A tela do Terminal Interativo (figura 36) permite a um usuário acessar toda a base de dados especificando consultas SQL diretamente para o Banco de Dados. Mas, apesar disto, apresenta uma interface amigável e diversos botões para facilitar o acesso a cláusulas e operadores SQL, bem como às tabelas SQL do esquema *public*. Na barra de ferramentas superior se encontram algumas cláusulas e os campos para acessar suas tabelas e campos, sem a necessidade de que o usuário as digite manualmente. A funcionalidade desta barra de ferramentas é controlada pela opção apresentada na tela Configurações. A única função que

não é desabilitada com a alteração das configurações é a opção de exportar os resultados em formato tabular para o formato .xls, à direita.

Abaixo desta barra de ferramentas está o console SQL, no qual o usuário tem total liberdade para consultar a base de dados, seja digitando comandos SQL (simples ou complexos, a resolução deste comando é processada no *Data Tier*, portanto pode ser tão complexa quanto o usuário deseje) ou utilizado os botões da barra de ferramentas. Ao lado temos um botão que envia o comando para o banco de dados e retorna o resultado na forma tabular na parte inferior do formulário (figura 37).

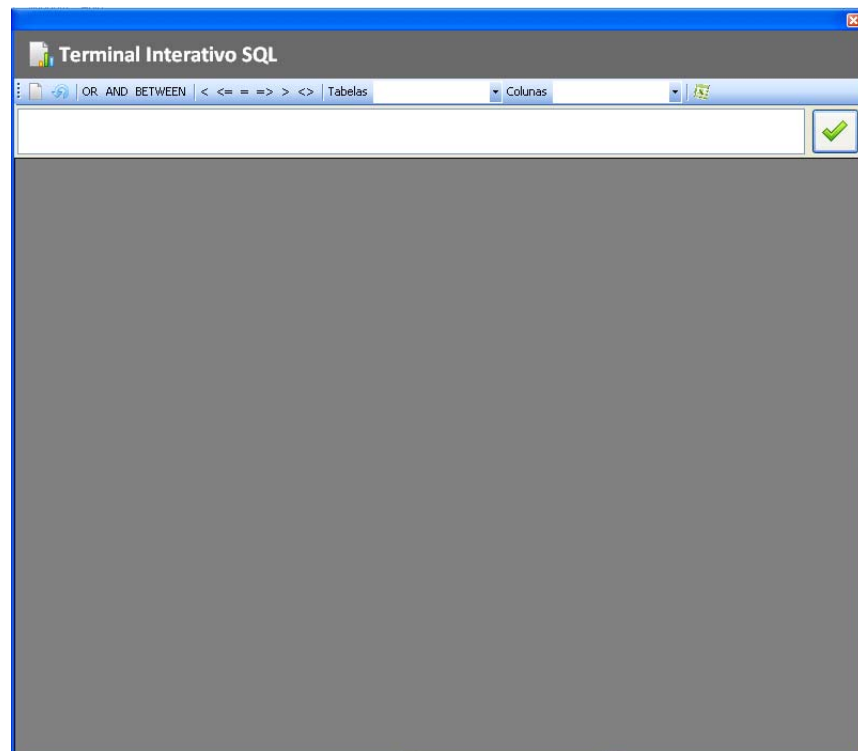


Figura 36: Tela do Terminal Interativo sem nenhuma consulta executada.
Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

tipo_logradouro	nome_logradouro
RUA	ALBERTO ALVES CABRAL
RUA	ALBERTO ALVES CABRAL
RUA	ALBERTO ALVES CABRAL
RUA	ALBERTO ALVES CABRAL
RUA	ALBERTO ALVES CABRAL
RUA	ALBERTO ALVES CABRAL
RUA	ALBERTO ALVES CABRAL
RUA	ALBERTO ALVES CABRAL
RUA	ALBERTO ALVES CABRAL
RUA	ALBERTO ALVES CABRAL
RUA	ALEXANDRINO SANTOS LIMA
RUA	ALEXANDRINO SANTOS LIMA
RUA	ALEXANDRINO SANTOS LIMA
RUA	ALEXANDRINO SANTOS LIMA
RUA	ALEXANDRINO SANTOS LIMA
RUA	ALEXANDRINO SANTOS LIMA
RUA	ALEXANDRINO SANTOS LIMA
RUA	ALFREDO TORMIM
RUA	ALFREDO TORMIM
RUA	ALFREDO TORMIM
RUA	ALFREDO TORMIM
AVENIDA	ANA GODOY DE SOUZA
AVENIDA	ANA GODOY DE SOUZA

Figura 37: Terminal Interativo após a execução de uma consulta SQL.
 Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

4.2.8. Relatórios Gerenciais

Os relatórios gerenciais são de extrema importância para o órgão gestor e para o administrador. Através destes relatórios pode-se ter uma idéia geral da situação dos acidentes de trânsito. Os relatórios trazem uma ampla gama de variáveis sumarizadas, facilitando o entendimento e trabalhando com as estatísticas gerais do cadastro, evitando a necessidade de se analisar acidente por acidente da base de dados.

Os relatórios são produtos de consultas SQL executadas no banco de dados e trazidas para a aplicação cliente de forma silenciosa, sem a necessidade de o usuário especificar ou ter de montar suas próprias consultas. Estas consultas pré-definidas são construídas no *Data Tier*, isto é, no Banco de Dados e são chamadas de *views* (visões).

As visões criadas ficam em espera no banco de dados e assim que são requisitadas, executam de forma silenciosa. A grande vantagem das visões é que consultas complexas são armazenadas, não é necessário redigitá-las e diminuem o tempo de execução da mesma, já que o banco de dados conhece a melhor maneira de retornar os dados.

Neste trabalho foram criados apenas cinco relatórios gerenciais de modo a exemplificar o funcionamento da tecnologia. Os relatórios desenvolvidos foram:

- 1 - Total de Acidentes por Trecho de Logradouro;
- 2 – Total de Acidentes por Logradouro;
- 3 – Severidade Total dos Acidentes por Logradouro;
- 4 – Relação detalhada do Logradouro;
- 5 – Relação detalhada do Acidente

O usuário acessa dentro da aplicação uma tela específica que lhe gera os relatórios solicitados⁶.

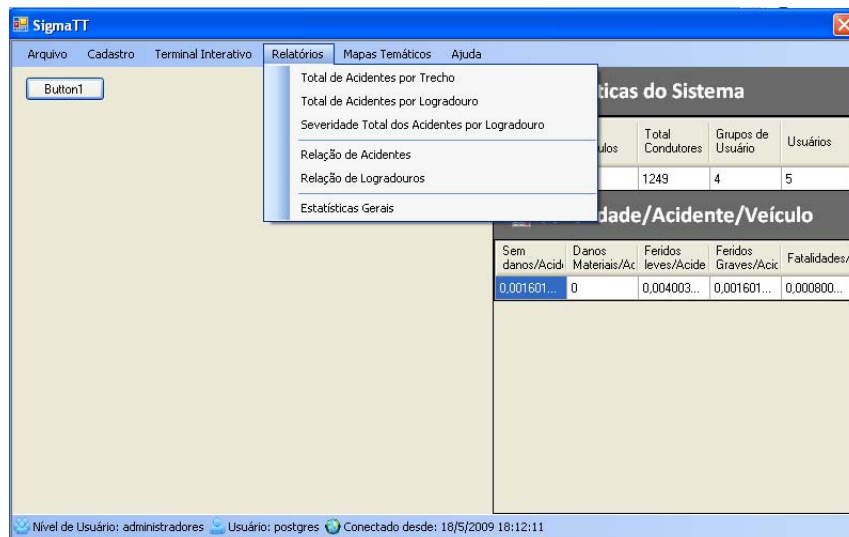


Figura 38: Acesso aos relatórios gerenciais na tela principal

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

⁶ Lembrando que os relatórios apresentados foram gerados com base em dados fictícios, gerados pela ferramenta construída para criar acidentes com dados aleatórios.

Relatório: Acidentes Totais por Trecho

Nome Logradouro	Prefixo Direção	Sufixo Direção	Total de Acidentes por Trecho	%
NELSON DE OLIVEIRA	DR. LAERTE VIEIRA GONÇALVES	ANA GODOY DE SOUZA	6	0.0140845070422535
ANTÔNIO DIAS	INEXISTENTE	JOAQUINA PORELO DIAS	6	0.0140845070422535
ANTÔNIO DIAS	JOÃO NAVES DE ÁVILA	JOAQUINA PORELO DIAS	5	0.0117370892018779
TOMAZ FALBO	FRANCISCO VICENTE FERREIRA	PLANALTO	5	0.0117370892018779
FRANCISCO VICENTE FERREIRA	JOÃO NAVES DE ÁVILA	ANTÔNIA SALTÃO DE ALMEIDA	5	0.0117370892018779
ARLINDO SOUZA MONTEIRO	JOSÉ PAES ALMEIDA	CÉSAR FINOTTI	4	0.00938967136150235
MARCIANO SANTOS	JOÃO NAVES DE ÁVILA	UBERABA	4	0.00938967136150235
JOÃO FURLANETO	UBERABA	IZAÚ RANGEL DE MENDONÇA	4	0.00938967136150235
JOAQUIM FERNADES VELOSO	JOÃO NAVES DE ÁVILA	JOSÉ PAES ALMEIDA	4	0.00938967136150235
OROZIMBO RIBEIRO	ORTÍZIO BORGES	BELARMINO COTTA PACHECO	4	0.00938967136150235
IZAAC ANTÔNIO SILVA	JOÃO JOSÉ DA SILVA	MARCIANO SANTOS	4	0.00938967136150235
MARIA DORA CUNHA	CÉSAR FINOTTI	IZAÚ RANGEL DE MENDONÇA	4	0.00938967136150235
JOÃO JOSÉ DA SILVA	FRANCISCO VICENTE FERREIRA	ISAAC ANTÔNIO SILVA	4	0.00938967136150235
MARIA DAS DORES DIAS	SEGISMUNDO PEREIRA	BELARMINO COTTA PACHECO	3	0.00704225352112676

Figura 39: Relatório Acidentes Totais por Trecho

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Total de Acidentes por Logradouro		
Relatório: Acidentes Totais por Logradouro		
Nome Logradouro	Total de Acidentes por Logradouro	%
PEDRO JOSÉ SAMORA	12	2.8169014084507
ANTÔNIO DIAS	10	2.34741784037559
JOÃO VELASCO ANDRADE	10	2.34741784037559
NELSON DE OLIVEIRA	10	2.34741784037559
DR. JAIME RIBEIRO DA LUZ	9	2.11267605633803
FRANCISCO ANTÔNIO DE OLIVEIRA	9	2.11267605633803
ARMANDO TUCCI	8	1.87793427230047
ATÍLIO VALENTINI	8	1.87793427230047
IZAAC ANTÔNIO SILVA	8	1.87793427230047
JOÃO FURLANETO	8	1.87793427230047
JORNALISTA JOÃO DE OLIVEIRA	8	1.87793427230047
JOSÉ CARRIJO	8	1.87793427230047
JOSÉ MIGUEL SARAMAGO	8	1.87793427230047
MARIA DAS DORES DIAS	8	1.87793427230047

Figura 40: Relatório Acidente Totais por Logradouros.
 Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Severidade Total por Logradouro

Relatório: Severidade e UPS total por Logradouro

Nome Logradouro	Sem Danos	Danos Materiais	Feridos Leves	Feridos Graves	Fatalidades	No de Acidentes	No de Veículos	UPS Veículo	UPS Acidente
ALBERTO ALVES CABRAL	9	3	20	9	4	5	12	18	43
ALEXANDRINO SANTOS LIMA	10	1	19	13	3	3	11	20	75
ALFREDO TORMIM	14	2	33	10	10	5	16	22	73
ANA GODOY DE SOUZA	9	8	35	9	6	6	17	19	54
ANTÔNIA SALTÃO DE ALMEIDA	11	1	25	10	5	3	12	21	87
ANTÔNIO DIAS	15	10	37	29	16	10	25	24	60
ANTÔNIO FORTUNATO DA SILVA	10	5	38	10	10	6	15	26	65
ANTÔNIO J.S. FRANQUEIRO	20	0	36	14	7	6	20	18	61
ANTÔNIO MARCIANO DE ÁVILA	11	1	19	15	5	4	12	22	66
ANTÔNIO REZENDE CHAVES	4	1	3	5	4	1	5	20	103
ANTÔNIO SALVIANO REZENDE	14	2	29	11	8	7	16	20	46
ARLINDO SOUZA MONTEIRO	13	1	20	16	8	5	14	22	63
ARMANDO TUCCI	15	7	40	13	9	8	22	18	51
ATÍLIO VALENTINI	16	8	43	16	14	8	24	21	64

Figura 41: Relatório Severidade Total por Logradouro.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

CAPÍTULO 05

5. WEBGIS

É possível identificar uma revolução na maneira em que trabalhamos com geoprocessamento e geotecnologias em geral. Na década de 1990, com a queda dos preços dos computadores e popularização natural do uso das geotecnologias nota-se um crescimento expressivo das aplicações chamadas *desktop* ou “topo de mesa”. Cada analista em sua mesa, trabalhando com um conjunto de dados, que depois era disponibilizado para outras áreas de uma empresa ou repartição pública. O modelo de trabalho era basicamente descentralizado, tornado uma tarefa de atualização cartográfica (em qualquer escala) morosa e dispendiosa, tanto em recursos humanos como financeiros.

Com a revolução da internet e a queda vertiginosa dos custos de transmissão de dados, as aplicações SIG começaram a se adaptar e passaram para um modelo centralizado, no qual temos um servidor (com um banco de dados) acessado por diversas aplicações clientes. Este modelo é tipicamente conhecido como geoprocessamento corporativo (*Corporate GIS*), tendo amplitude muito maior que seu modelo descentralizado, em que apenas o departamento responsável pelo SIG/GIS conhecia seu funcionamento e poderia tirar máximo proveito das análises e armazenamento de dados espaciais. Este foi o estágio conhecido como o período de maturação da tecnologia e técnicas GIS, em que seus principais atores presenciariam outra revolução: a transmissão de quantidades massivas de dados geográficos pela internet, em diversas interfaces, como *Google Earth*, *Virtual Earth*, *WorldWind*, entre outros. Entramos na era do WebGIS, ou do Geoprocessamento via internet.

Neste sentido as empresas líderes no mercado efetuaram diversos esforços separados criando seus protocolos e interfaces para a publicação de mapas e dados geográficos online. Após diversos anos de discussão a *OGC* (*Open Geospatial Consortium* – Consórcio Geoespacial Livre) estabeleceu alguns padrões para a disseminação de dados geográficos pela internet ou *LAN* (*local area network* / rede local) em protocolo *HTTP* sendo descritos abaixo. Para conhecimento do leitor estes padrões utilizados atualmente são: *Web Map Service* e *Web Feature Service*. Existem outros protocolos *HTTP* para manipulação e visualização de feições geográficas: *WFS-T* (*Web Feature Service-Transactional*), *WCS* (*Web Coverage Service* –

utilizado para servir arquivos em formato matricial/raster) que não serão descritos neste trabalho por não comporem parte do objetivo.

5.1. OGC (*Open Geospatial Consortium*)

O *Open Geospatial Consortium* (OGC ou Consórcio Geoespacial Livre) é uma empresa sem fins lucrativos composta por mais de 300 empresas, órgãos públicos e universidades perseguindo um mesmo objetivo: a definição de padrões para dados espaciais. A conformidade com estes padrões habilita softwares e sistemas complexos a interagir com outros softwares e sistemas de forma integrada, sem a necessidade de conversão de dados e habilita a publicação de dados via internet em formatos *agnósticos* (independente de plataforma).

Este consórcio já desenvolveu mais de vinte padrões para diversos tipos de dados espaciais, como *SFS* (*Simple features for SQL*, mencionado no capítulo de Banco de Dados), *SLD* (*Styled Layer Descriptor* – Descritor de camadas estilizadas), *WMS*, *WFS*, *WFS-T* (serviços de transmissão e publicação de dados via protocolo *HTTP*), entre outros.

Todos os padrões do consórcio geoespacial podem ser acessados em: <http://www.opengeospatial.org/standards>, contendo as especificações detalhadas para cada formato e serviço.

5.1.2. WMS (*Web Map Service*)

O protocolo *WMS* é simplesmente um pedido *HTTP* de um cliente (usuário acessando página da *web*) de temas georreferenciados com especificações de simbologia, extensão espacial, entre outras opções, processado pelo servidor que retorna uma grade de imagens (*tiles*) para o navegador de internet. Através desta opção podemos gerar somente imagens estáticas, possibilitando a navegação completa pelo mapa (*Zoom In*, *Zoom Out*, *Pan*), impressão, definição de escala, projeções e Data.

Os pedidos podem ser dinâmicos, de acordo com uma *query* (consulta) no banco de dados, sendo o mapa automaticamente gerado pelo navegador.

O protocolo *WMS* é interessante para visualização rápida de mapas online, já que o tempo necessário para sua configuração é menor que de um *WFS* e quando não existe a necessidade de alteração dos dados via *WEB*.

Este protocolo atualmente se encontra na versão 1.3, e é relativamente mais maduro que o protocolo *WFS*, embora os dois sejam amplamente usados, até mesmo em conjunto em uma mesma aplicação.

5.1.3. *WFS(Web Feature Server)*

A *OGC* definiu uma interface para especificar pedidos de feições geográficas através da internet utilizando chamadas *agnósticas* (independente de plataforma). O padrão *WFS* define as interfaces e funções para o acesso dos dados e sua manipulação que entre elas incluem: busca de feições específicas de acordo com *queries* (consultas) espaciais ou não espaciais, criação de novas feições ou objetos, recebimento de dados sobre cada objeto, deleção de feição, atualização de feição e travamento de feição (semelhante ao princípio integridade descrito no capítulo de banco de dados).

O formato padrão para transmissão e recebimento de novos dados é o *GML* (*Geographic Markup Language*) especificado também pelo Consórcio Geoespacial Livre. A grande diferença entre um *WMS* e *WFS* é o que é transmitido via internet para o navegador. No caso do *WMS* são transmitidos *tiles*, imagens, enquanto no caso do *WFS* são transmitidos vetores, as feições de fato.

Um *WFS* funciona da seguinte maneira: o cliente (no caso usuário acessando uma página da web) faz um pedido de dados ao servidor. O servidor responde ao cliente com um arquivo *GML* (implicitamente, o usuário não vê a transação) contendo a descrição das feições requisitadas e elas são desenhadas em um navegador de internet por outra aplicação (neste caso o *OpenLayers*).

A vantagem de um *WFS* sobre um *WMS* é a possibilidade de se alterar as feições retornadas em *GML*, transformando (com um pouco de esforço e programação de interfaces) o navegador de internet padrão em um aplicativo de SIG completo, permitindo usuários leigos editarem e realizar atualizações cartográficas sem o uso de aplicativos desktop clientes (*GvSig*, *uDIG*, entre outros) via *internet*. Este protocolo se encontra atualmente na versão 1.1.

5.2. *GeoServer*

Como especificado em capítulos anteriores, todas nossas informações geográficas e alfanuméricas em um sistema de banco de dados, o *PostgreSQL*.

Para a criação de mapas dinâmicos é necessário um sistema ou servidor de mapas, que leia os dados espaciais e alfanuméricos do banco de dados e apresente-os para um software cliente, da mesma maneira que um software SIG desktop faz com arquivos localizados em uma máquina local.

O software escolhido para esta tarefa foi o *GeoServer*, um servidor de mapas de licença livre, podendo ser utilizado gratuitamente e modificado conforme as necessidades de cada projeto.

Depois de instalado, o *GeoServer* permite ao usuário em interface web agradável a administração de seu sistema, possibilitando a configuração de quais arquivos (loais) ou tabelas (de um banco de dados) podem ser servidas, estilos para simbolização cartográfica, projeções cartográficas (oferecidas pela biblioteca *Proj4*, de licença livre) e até mapas complexos para serem servidos através do protocolo *WMS*. O *GeoServer* pode ser obtido em: <http://geoserver.org> .

O *GeoServer* também contém módulos de autenticação, impedindo os usuários comuns de alterarem quaisquer configurações sensíveis ao bom funcionamento do serviço (figura 42).

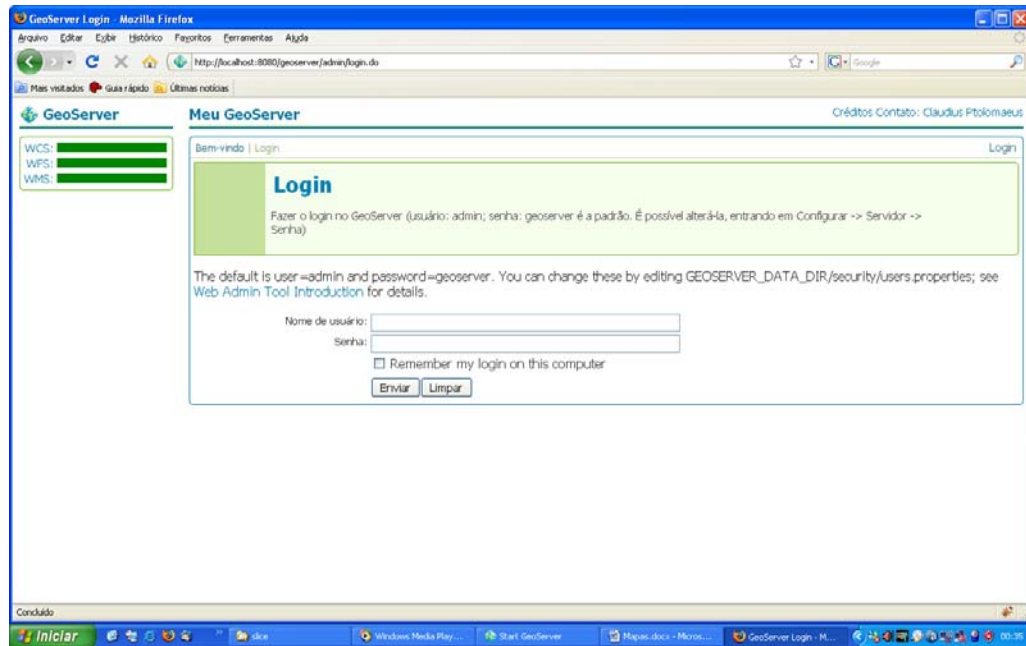


Figura 42: Tela de entrada do GeoServer.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Conforme especificado, o *GeoServer* apenas disponibiliza as informações espaciais para um leitor, sendo necessário uma biblioteca específica para desenhá-las em uma página da web. Existem diversas bibliotecas que conseguem realizar este trabalho na web, de licenças livres, mas o *OpenLayers* foi escolhido pela facilidade de se construir os mapas necessários e pela flexibilidade oferecida em sua *API* (*Application Programming Interface*) escrita em *JavaScript*, linguagem relativamente fácil de se aprender e amplamente suportada (*agnóstica*, funciona independentemente do sistema operacional ou navegador utilizado).

Esta solução em conjunto foi escolhida (*PostgreSQL/PostGIS* + *GeoServer* + *OpenLayers*) primeiramente pelo seu custo, que é zero, contando com amplo suporte oferecido pelas comunidades que constroem os softwares e documentação disponível na internet. Outra vantagem desta solução é que podemos trabalhar livremente, sem prender o usuário final a um sistema (como o *ArcObjects* da *ESRI*) ou licenças proprietárias.

Seguindo neste capítulo serão apresentados os procedimentos utilizados para cadastrar as informações a serem disponibilizadas pelo *GeoServer* ao *OpenLayers* e a um navegador de internet comum.

5.2.1. Disponibilização de Dados através do GeoServer

O processo de disponibilização de dados espaciais (figura 43) pelo *GeoServer* é bastante simples e será descrito a seguir em detalhes. Podemos disponibilizar dois tipos de dados através do *GeoServer*: vetores e dados matriciais ou raster. Também serão descritas as opções configuráveis do *GeoServer* para melhor acompanhamento do procedimento.

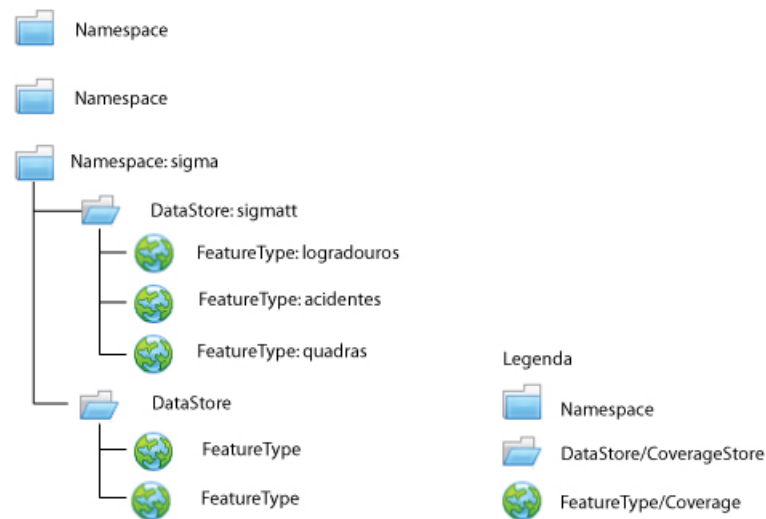


Figura 43: Organização lógica de uma instância do GeoServer.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

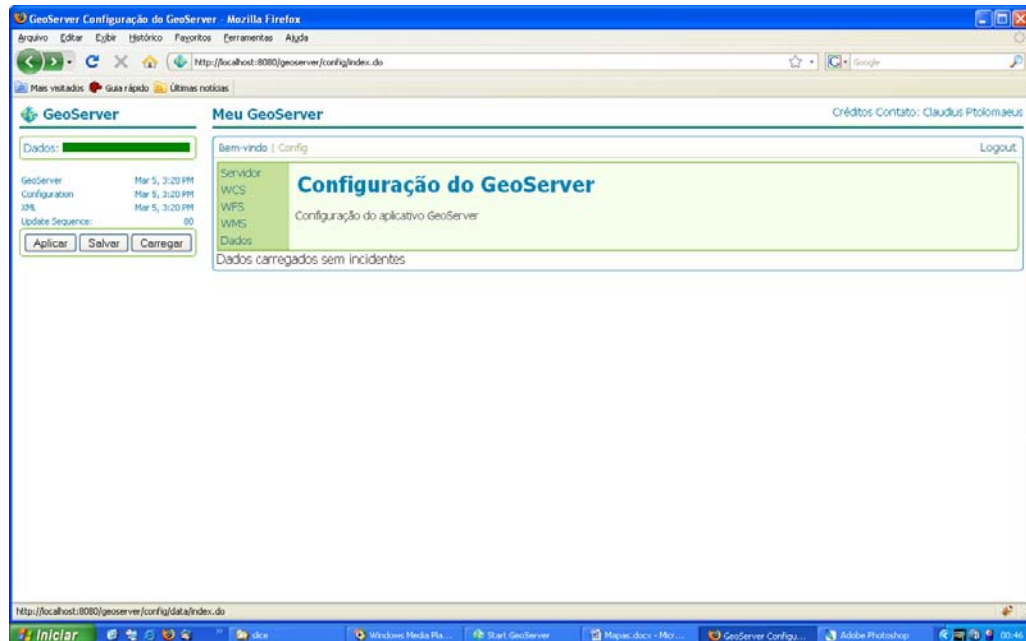


Figura 44 - Página Inicial do GeoServer.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Neste momento, podemos configurar seis tipos de parâmetros em nosso servidor de mapas: *Namespaces*, *CoverageStores*, *DataStores*, *Estilos*, *FeatureTypes* e *Coverages*.

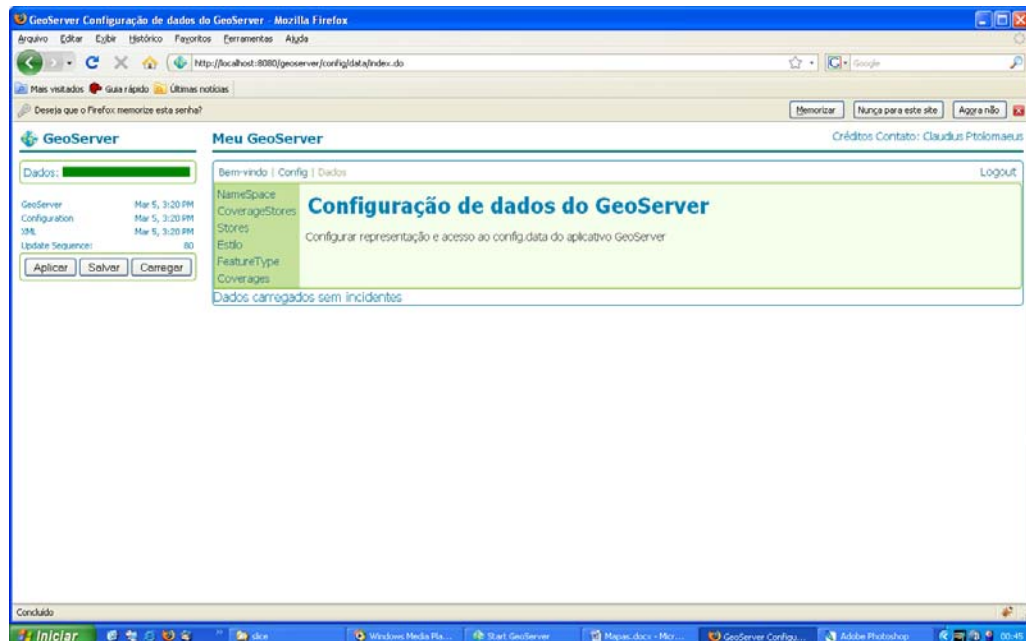


Figura 45 - Página de configuração de dados do GeoServer.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Os *Namespaces* são espaços de trabalho e tem a função de organizar em “diretórios” separados as *Stores* e *CoverageStores* (repositório de dados espaciais no formato vetorial);

As *CoverageStores* têm a função de separar logicamente as “lojas” de dados matriciais, em “diretórios”, podendo ser configuradas separadamente. Entre suas opções estão fonte (arquivos locais ou administrados por um banco de dados) e estado (ativada/desativada);

Assim como as *CoverageStores*, as *DataStores* separam logicamente as “lojas” de dados, mas desta vez, dados vetoriais, habilitando sua configuração de forma independente *Store* de outra. Entre opções estão fonte (arquivos locais ou administrados por um banco de dados), estado (ativado/desativado), criação de índices espaciais (para melhor performance via internet), e qual conjunto de caracteres utilizar (arábico, chinês, europeu, universal, etc);

As *FeatureType* são uma referência de fato às tabelas ou arquivos a serem servidos pelo GeoServer. Cada *FeatureType* pertence à uma *DataStore* criada previamente. Dentro das opções que são disponibilizadas estão: datum, projeção, estilo, resumo sobre a *FeatureType*, e diversas opções relacionadas à performance da *FeatureType* (figura 46) em uma transmissão via web (ou LAN) como: *Cachê*, *Tiles* (*tiles* são formados por uma grade que divide toda a

FeatureType em diversos blocos, sendo servidos somente aqueles necessários à visualização selecionada pelo usuário), número máximo de feições, entre outras.

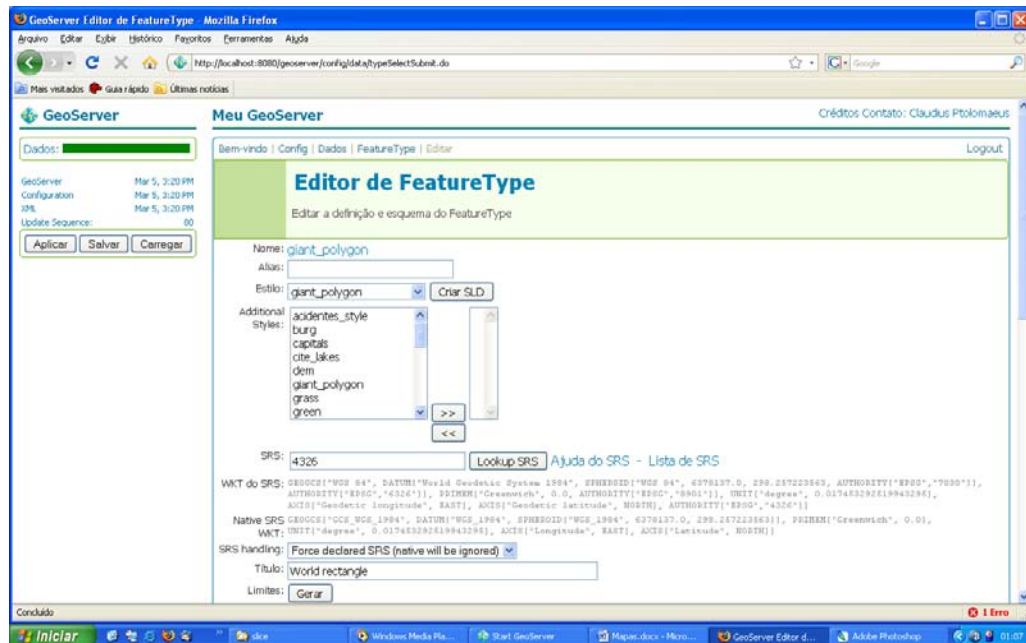


Figura 46 - Editor de FeatureType no GeoServer.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Finalmente, os Estilos são arquivos no formato .SLD (*Styled Layer Descriptor* – padrão estabelecido também pela *OGC – Open Geospatial Consortium*) que direcionam a forma como as *FeatureTypes* devem ser simbolizadas pelo navegador. Existem softwares que criam estes arquivos de forma amigável, sem a necessidade da construção de seu código manualmente, como o *uDIG* (também livre e gratuito, disponível em: <<http://udig.refrains.net/>>) sendo importados pelo *GeoServer* posteriormente (figura 47). O Estilo de cada *FeatureType* pode ser informado no momento de sua criação ou selecionado posteriormente na página de configuração da mesma.

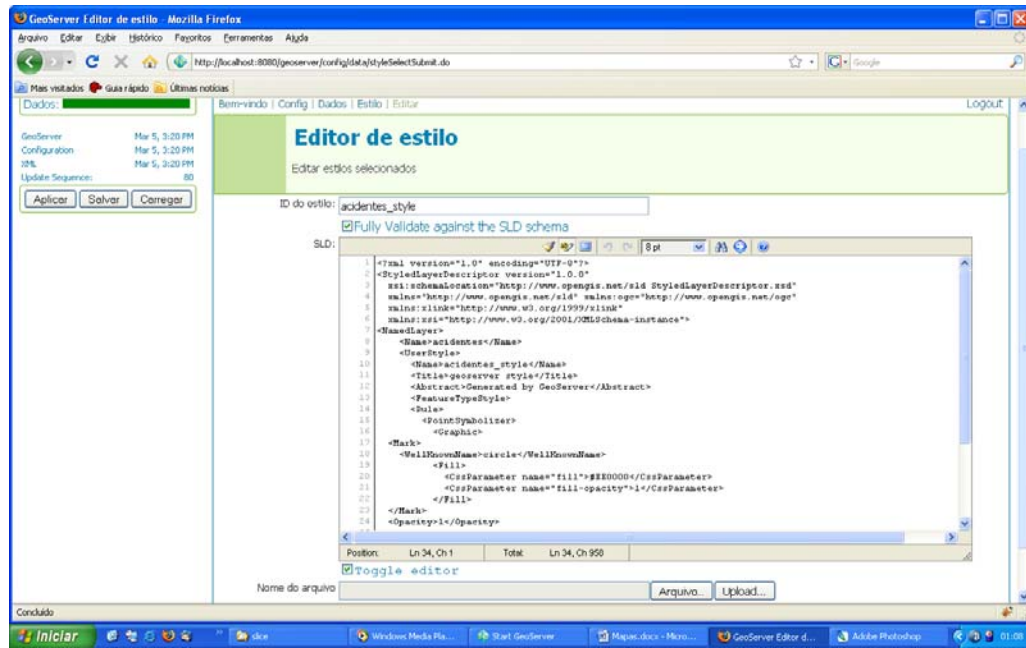


Figura 47 - Editor de estilos.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

5.2.1.1: Configuração de um Namespace, DataStore e FeatureType

Primeiramente, para se disponibilizar os dados via *GeoServer*, necessita-se de configurar um *Namespace* e um *DataStore* afim de configurar uma *FeatureType*.

Para criação de um *Namespace* simplesmente clica-se na página de configuração de dados e em *Namespace* (figura 48). O navegador encaminhará o usuário a uma página na qual podemos editar os *NameSpaces* existentes ou criar um novo. Basta clicar em NOVO para prosseguir. As únicas opções necessárias para a criação de um namespace são *URI* e Prefixo. A *URI* é a página sobre a qual iremos servir os dados (no nosso caso, “localhost”) e o prefixo pode ser qualquer um que descreva o serviço conforme agrade o usuário. Em nosso caso foi escolhido “sigma”. Basta clicar em criar que o namespace será criado.

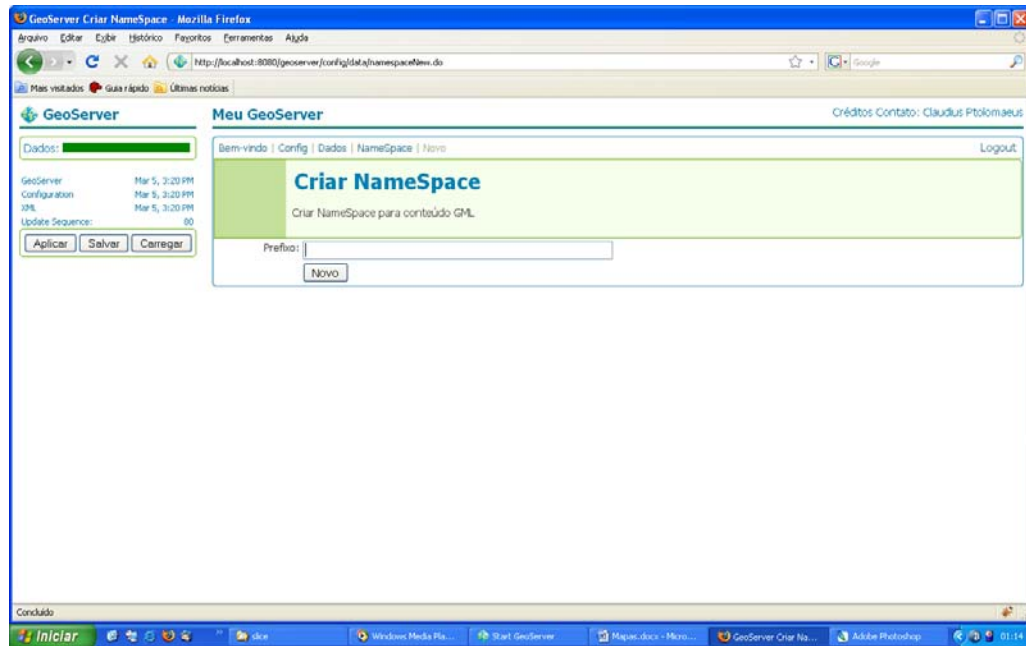


Figura 48 - Criação de namespace.

Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Após a criação do namespace deve-se criar uma *DataStore*, que conterá as informações necessárias para a conexão com o banco de dados e armazenará informações sobre as tabelas disponíveis.

Voltando para a página de configuração de dados do *GeoServer*, basta clicar em *Stores* (*Stores* e *DataStores* são termos equivalentes) e após o direcionamento do navegador clicar em NOVO (figura 49). Inicialmente, o *GeoServer* irá pedir ao usuário duas informações: tipo da base de dados e ID do *DataStore*. Os tipos suportados pelo *GeoServer* são: *PostGIS*, *Shapefile* e *WFS* externos. O ID do *DataStore* é um identificador alfanumérico para fácil identificação do usuário. Após preencher estas informações o *GeoServer* criará a *DataStore* e lhe permitirá configurá-la.

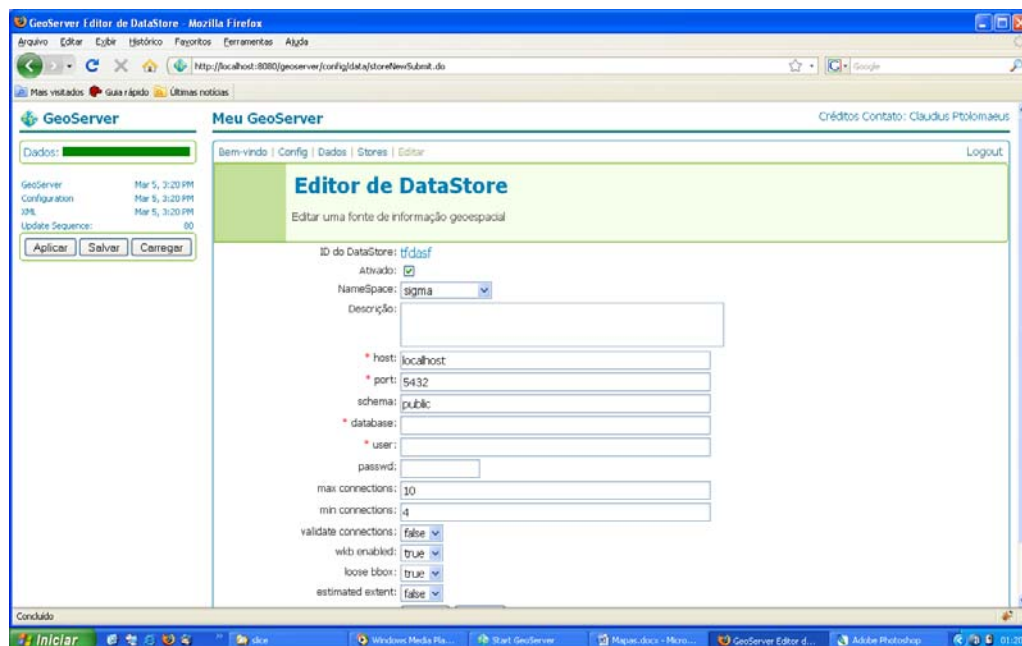


Figura 49 - página de configuração de um datastore recém criado.
Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009.

Estas configurações são muito importantes e variam de acordo com o tipo de datastore. Neste caso, foi escolhido o tipo *PostGIS*, em que preenche-se o nome de um namespace (criado anteriormente), nome do banco de dados, servidor, porta, senha e outras variáveis relacionadas à performance da datastore (número mínimo e máximo de conexões simultâneas, extensões geográficas estimadas, etc.).

Por último, iremos configurar uma *FeatureType*. Voltando a página de configuração de dados, clica-se em *FeatureType* e novamente em NOVO. Aparecerá uma caixa com diversos prefixos e sufixos. É nesta caixa que aparece a relação das tabelas espaciais que temos à disposição em nosso banco de dados. Basta escolher a tabela desejada (logradouros, por exemplo) e clicar em NOVO.

O sistema do *GeoServer* nos levará para configurar neste momento a tabela e suas opções, conforme mostrado na figura acima. Após clicar em enviar poderemos já servir nossos dados espaciais pela internet. Para um rápido teste, clique em Demo na página inicial do *GeoServer* e escolha “Pré-visualização do mapa”. A *FeatureType* recém criada estará à disposição e basta clicar em seu nome para ver uma demonstração criada automaticamente pelo *GeoServer* (figura 50).

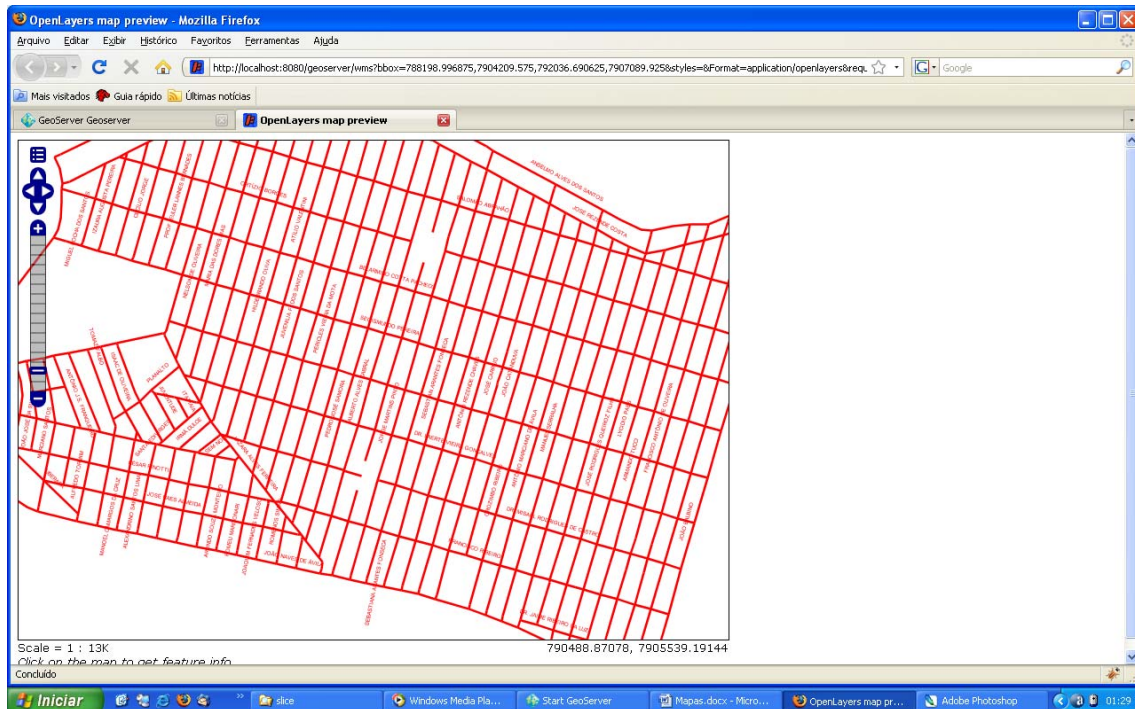


Figura 50 - Visualização de demonstração da *FeatureType* recém criado no *GeoServer*. A visualização de demonstração neste caso é gerada pela *API OpenLayers*.

Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009

5.3. OpenLayers

OpenLayers é uma *API (application programming interface)* utilizada para processar e disponibilizar pedidos de dados espaciais de forma gráfica em um navegador para internet. Esta *API* possibilita o desenvolvimento de aplicações geográficas dentro de um navegador de forma simples e com licença livre e gratuita.

O maior trunfo do *OpenLayers* é a separação de ferramentas do mapa dos dados que o compõem. Desta forma, o *OpenLayers* pode se concentrar em fornecer as ferramentas para visualizar, criar e editar feições, independente das informações geográficas, desde que elas sigam os padrões *OGC* (como é o caso do *GeoServer*, *PostGIS* e do próprio *OpenLayers*). No caso deste trabalho teremos o *GeoServer* entregando informações geográficas ao *OpenLayers*, e este as desenhando em ambiente web conforme as especificações do usuário.

A *API* do *OpenLayers* pode ser obtida gratuitamente em <http://www.openlayers.org/> bem como toda a documentação pertinente.

5.3.1. Definições dos Mapas

Nesta *API* utilizamos a linguagem de programação *JavaScript* para declarar funções e camadas de nosso mapa. A sintaxe *OpenLayers* (derivada de *JavaScript*) é bastante simples mas exige cuidado do desenvolvedor, já que ela é *case sensitive* (“OpenLayers” não é igual a “openlayers”) e fracamente tipada (podemos criar e designar variáveis em qualquer momento do código, sem preocupação com seus tipos nativos).

Dentro de um código *HTML* comum podemos construir controles, interfaces e botões para manipulação dos dados visualizados como mapa de referência, botões para criação, edição e seleção de feições, *layer switcher* (selecionador de camadas) entre outros.

Este tipo de solução conjugada facilita o desenvolvimento utilizando o mesmo princípio visto no capítulo 4 – Construção do Software cliente, *three tier architecture* separando a visualização (feita pela *API* do *OpenLayers*) das regras de negócio (administrada pelo *GeoServer*) dos dados em si (banco de dados *PostgreSQL*). Novamente, este tipo de separação permite ao desenvolver e usuário avançado um controle muito maior sobre o projeto e a implementação de seus requisitos, sendo fácil de instalar e configurar como um todo.

Existem diversas maneiras de se integrar um mapa gerado em *OpenLayers* à uma página da internet em formato *HTML*. Uma delas é a inserção de um código *JavaScript* no cabeçalho da página definindo sua execução no momento da visualização da página.

Veremos um exemplo de um código de mapa simples e análise posterior:

```
<script type="text/javascript">

//constantes
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";
OpenLayers.IMAGE_RELOAD_ATTEMPTS = 5;
OpenLayers.DOTS_PER_INCH = 25.4 / 0.28;

//declaracao de variaveis
var mapa, infoControls;

//extensao inicial
var bounds = new OpenLayers.Bounds(
788000, 7904000,
793000, 7908000
);

//extensao maxima do mapa
var maxBounds = new OpenLayers.Bounds(
774024.75625,7895424.4,798514.74375,7916007
);
```

```
//opcoes de mapa
var options = {
  controls: [],
  maxExtent: maxBounds,
  maxResolution: 100.79424330488837,
  projection: "EPSG:29192",
  units: 'm'
};

function init(){
  //declaracao da lingua mae
  OpenLayers.Lang.setCode("pt-BR");

  //declaracao do formato a ser gerado
  format ="image/png";

  //instanciando o mapa declarado acima
  mapa = new OpenLayers.Map("mapa",options)

  //instaciando logradouros
  logradouros = new OpenLayers.Layer.WMS("Logradouros",
    "http://localhost:8080/geoserver/wms",
    {layers: "sigma:logradouros",
    srs: "EPSG:29192",
    format: format,
    transparent: true},
    {isBaseLayer: false});

  //instaciando quadras
  quadras = new OpenLayers.Layer.WMS("Quadras",
    "http://localhost:8080/geoserver/wms",
    {layers: "sigma:quadras_poligono",
    srs: "EPSG:29192",
    format: format,
    transparent: true},
    {isBaseLayer: true});

  //instanciando acidentes
  acidentes = new OpenLayers.Layer.WMS("Acidentes",
    "http://localhost:8080/geoserver/wms",
    {layers: "sigma:acidentes",
    srs: "EPSG:29192",
    format: format,
    transparent: true},
    {isBaseLayer: false});

  //adiciona camadas ao mapa
  mapa.addLayers([quadras, logradouros, acidentes]);

  //cria os controles
  mapa.addControl(new OpenLayers.Control.PanZoomBar({
    position: new OpenLayers.Pixel(5, 5)
  }));

  mapa.addControl(new OpenLayers.Control.Navigation());

  mapa.addControl(new OpenLayers.Control.MousePosition({element: $('location')}));
```

```

mapa.addControl(new OpenLayers.Control.Scale($('scale')));

mapa.addControl(new OpenLayers.Control.LayerSwitcher());

mapa.addControl(new OpenLayers.Control.ScaleLine());

//zoom na extensao desejada
mapa.zoomToExtent(bounds);

}
</script>

```

Código 11: Exemplo de um código simples para declaração de um mapa em OpenLayers.

Fonte: SILVA, G. R. C, Desenvolvimento do software SigmaTT, 2009.

Como exemplificado o código em *OpenLayers* é bastante simples e direto sendo uma ferramenta fácil de aprender e construir mapas dinâmicos. Apesar de simples, a API do *OpenLayers* é extensa e contém diversos parâmetros, suportando conversão de projeções e data (no exemplo dado o código *EPSG 29192* representa a projeção *South American Datum-69 Zona UTM 22S*)⁷, consultas à feições específicas, desenho de feições e registro das mesmas no banco de dados e outras funções que o desenvolvedor possa desenvolver em cima de sua biblioteca.

Foi desenvolvido somente um mapa dinâmico (Figura 51) para demonstrar a tecnologia. Futuramente pretende-se desenvolver diversos mapas dinâmicos, com cláusulas de consultas semelhantes aos relatórios gerenciais, de forma a representar graficamente o relatório.

⁷ Uma lista completa de projeções suportadas e seus parâmetros podem ser encontrados no site Proj4.org e em Spatial References.org, ambos listados nas referências.



Acidentes no Bairro Santa Mônica - Uberlândia - MG



789986.57932, 7906056.69676

Escala = 1 : 11K

Concluído

Figura 51 - Mapa gerado pelo código-fonte 11, visualizado no navegador *Firefox*.
Fonte: SILVA, G. R. C., Desenvolvimento do software SigmaTT, 2009

CAPÍTULO 06

6. RESULTADOS FINAIS

Conforme a problemática apresentada, este sistema para cadastro de acidentes de trânsito atendeu as proposições iniciais de desenvolvimento.

Os resultados atingidos foram melhores que o esperado em termos de desempenho e abrangência. A abrangência do modelo de dados é satisfatória e permite os próprios usuários a adaptar o modelo à suas necessidades e realidades urbanas.

Os processos de geocodificação também tiveram sucesso maior do que o esperado inicialmente. Apesar da pequena área de abrangência deste serviço geocodificador inicial, com um pequeno investimento é possível abranger áreas muito maiores, tornando a existência de um sistema de cadastro de acidentes georreferenciados uma realidade próxima para Uberlândia – MG. Além disso, o algoritmo de geocodificação pode ser adaptado para localizar qualquer tipo de feições, incluindo crimes, equipamentos urbanos (como postes, sinalização viária, etc.), focos de dengue (e outras doenças de proporções epidêmicas), entre outros.

A maior dificuldade para construção do sistema foi de fato a base de referência devido a uma diversidade de informações, como grafias de logradouros distintas, dificuldade na localização dos números em campo e tempo gasto na digitalização da numeração viária. Esta dificuldade foi determinante para o tempo gasto neste trabalho, mas de posse de uma equipe de campo e escritório treinada é possível construir bases de referência precisas e em pouco tempo. Com maiores recursos e uma equipe este tempo pode cair drasticamente com o uso de computadores de mão e inserção dos números viários simultânea à coleta.

O uso de softwares livres para este projeto de pesquisa foi extremamente satisfatório, exibindo como produto final um software completo, com diversas funcionalidades avançadas e uma solução integrada para o cadastro de acidentes de trânsito.

Futuramente têm-se como prioridade um refinamento do modelo de dados e a conversão da arquitetura para uma plataforma *WEB*, *e.g. PHP e HTML*, permitindo o usuário

acessar a o cadastro e a base de dados pela internet e sem a necessidade de softwares instalados no seu computador.

REFERÊNCIAS

7. REFERÊNCIAS

- BESTEBREURTJE, Hans. *Gis Project Management*. Dissertação de Mestrado, 145 páginas. Free University of Amsterdam. Lieren: UNIGIS, 1997.
- BITNER, David. *dbSpatial*. 2009. Disponível em: <<http://dbspatial.com/>>. Acesso em jan, fev, mar de 2008).
- BITNER, David. *Geocoding Function in pl/perl*. Minneapolis: 2007.
- BORGES, Karla Albuquerque de Vasconcelos. *Modelagem de Dados Geográficos*. Belo Horizonte: UFMG, 2002.
- CÂMARA, Gilberto, M. CASANOVA, A. HEMERLY, G. MAGALHÃES, e C. MEDEIROS. *Anatomia de Sistemas de Informação Geográfica*. Campinas: UNICAMP, 1996.
- CARDOSO, G., e C. RUSCHEL. Desenvolvimento de um sistema integrado de cadastro e análise espacial de acidentes de trânsito em Porto Alegre - RS. In: 15º Congresso Brasileiro de Transporte e Trânsito, 2005, Goiânia, 2005.
- DAVIS, Clodoveu, e Frederico FONSECA. *Introdução aos Sistemas de Informação Geográficos*. Belo Horizonte: UFMG, 2001.
- DevGoiás. *Fórum DevGoiás*. 2009. Disponível em: <<http://www.devgoias.net>> Acesso em jan, fev e mar de 2009.
- DINIZ, Alexandre. *Estatística Espacial*. Vol. I e II. 2 vols. Belo Horizonte: UFMG, 2000.
- EMPRESA BRASILEIRA DE CORREIOS E TELÉGRAFOS. *Correios: Busca CEP*. 2009. <<http://www.buscacep.correios.com.br/servicos/dnec/menuAction.do?Metodo=menuLogradouro>> (acesso em abr de 2009).
- ELSMARI, RAMEZ, e Shamkant NAVATHE. *Sistemas de Banco de Dados*. São Paulo: Pearson, 2006.

EQUIPE SPATIAL REFERENCE. *SpatialReference.Org*. 2009. Disponível em: <<http://spatialreference.org/>> . Acesso em 29 de abr de 2009.

ESRI. *ArcGIS 9: Geocode Rule Base Developer Guide*. Redlands: ESRI Press, 2003.

_____. *ArcGIS 9: Geocoding in ArcGIS*. Redlands: ESRI Press, 2006.

_____. *ArcGis 9: Geoprocessing Quick Commands Reference Guide*. Redlands: ESRI Press, 2008.

_____. *ESRI Developer Network*. 2009. Disponível em <<http://www.edn.esri.com>>. Acesso em jan à dez de 2008.

_____. *ESRI User Forums*. 2009. Disponível em <<http://www.esri.com>> . Acesso em jan de 2009.

FILHO, Britaldo Silveira Soares. *Cartografia Assistida por Computador: Conceitos e Métodos*. Belo Horizonte: UFMG, 2000.

FILHO, Britaldo Silveira Soares. *Modelagem de Dados Espaciais*. Belo Horizonte: UFMG 2002.

FREE SOFTWARE FOUNDATION. *Free Software Foundation*. 2009. Disponível em <<http://www.fsf.org>> . Acesso em 28 de abr de 2009.

GEOLIVRE. *Portal GeoLivre*. Disponível em <<http://www.geolivre.org.br/>> . Acesso em jan, fev de 2009.

GEOSERVER. *GeoServer*. Disponível em <<http://www.geoserver.org/>> . Acesso em jan de 2009.

GOLD, Philip Anthony. *Segurança de Trânsito: Aplicações de Engenharia para Reduzir Acidentes*. BIRD: CET, 1998.

GRASS. *Grass History*. Disponível em <<http://grass.itc.it/devel/grasshist.html>> . Acesso em março de 2009.

MACORATTI, José Carlos. *Macoratti.NET*. Disponível em <<http://www.macoratti.net>>. Acesso em ago à dez de 2008.

MICROSOFT. *Microsoft Developer Network*. Disponível em <www.microsoft.com/brasil/msdn> . Acesso em jan, fev, mar de 2009.

_____. *Visual Studio Express 2008 Editions*. Disponível em <<http://www.microsoft.com/Express/>> . Acesso em jan à dez de 2008.

MOURA, Ana Clara Mourão. *A importância dos metadados no uso das Geotecnologias e na difusão da Cartografia Digital*. Belo Horizonte: UFMG, 2001.

NEVES, Marcela Lopes. *Tratamento de Dados Geográficos e Consultas Espaciais em Bancos de Dados Objeto-Relacionais*. Trabalho para obtenção do título de especialista, Belo Horizonte: UFMG, 2005.

OPEN GEOSPATIAL CONSORTIUM. “Web Feature Service Implementation Specification.” *Open Geospatial Consortium*. Disponível em. <<http://www.ogc.org>> . Acesso em março de 2009.

OPENGEO. *Introduction to Openlayers Workshop*. Disponível em <<http://workshops.opengео.org/openlayers/intro/doc/en/>>. Acesso em 2009.

OPENGEO. *Plano Diretor de Geoprocessamento da Prefeitura Municipal de Fortaleza*. Fortaleza, out de 2007.

OPENLAYERS ARCHITECT TEAM. *OpenLayers Architect*. Disponível em <<http://www.olarchitect.com/>> . Acesso em 2009.

OPENLAYERS. *OpenLayers*. Disponível em <<http://openlayers.org>> . Acesso em jan, fev, mar, abr, mai de 2009.

PAINHO, Marco, Miguel PEIXOTO, e Pedro CABRAL. *Advances in GIS Teaching: Towards WebGIS-based E-Learning*. Disponível em <http://www.isegi.unl.pt/unigis/papers/EMEA_2001.pdf> . Acesso em 28 de abr de 2009.

POSTGRESQL 8.3.3 DOCUMENTATION. *PostgreSQL*. Disponível em <<http://www.postgresql.org>> . Acesso em dez de 2008.

POSTGRESQL. *PostgreSQL Home*. 2009. Disponível em <<http://www.postgresql.org/>> . Acesso em jan à dez de 2008.

PREFEITURA MUNICIPAL DE UBERLÂNDIA. *Imagens de Satélite QuickBird*. Uberlândia, 2007.

PREFEITURA MUNICIPAL DE UBERLÂNDIA. *Ortofotos 1:2000*. Uberlândia, 2004.

QUADRO, Fernando Silveira. *Blog Fernando Quadro*. Disponível em <www.fernandoquadro.com.br> . Acesso em jan, fev, mar, abr de 2009.

_____. *Introdução ao OpenLayers*. Disponível em <www.fernandoquadro.com.br> . Acesso em jan de 2009.

_____. *Introdução Geoserver*. Disponível em <<http://www.fernandoquadro.com.br>>. Acesso em jan de 2009.

REFRACTIONS RESEARCH INC. *PostGIS*. Disponível em <<http://postgis.refrations.net>> . Acesso em jan, fev e mar de 2009.

_____. “PostGIS Documentation.” *PostGIS Refrations*. Refrations Research Inc. Disponível em <<http://postgis.refrations.net>> . Acesso em jul, ago, set, out, nov de 2008.

_____. *PostGIS History*. Disponível em <<http://www.refrations.net/products/postgis/history/>> . Acesso em 2009.

ROSA, Roberto. *Introdução ao Sensoriamento Remoto*. Uberlândia: EDUFU, 2003.

SILVA, Angela Maria, PINHEIRO, Maria Salete de Freitas e FRANÇA, Maria Nani. *Guia Para Normalização de Trabalhos Técnico-Científicos*. Uberlândia: EDUFU, 2006.

SILVA, George Rodrigues da Cunha, e TINOCO, Alex. *Fórum Geo.NET*. 2009. Disponível em <www.geoprocessamento.net> . Acesso em jan, fev de 2009.

SOUSA, Maria Cecília. “Fiscalização Eletrônica: Análise da Eficácia na Redução dos Acidentes de Trânsito - Uberlândia 2004 a 2006.” Dissertação de Mestrado. 213 páginas. Universidade Federal de Uberlândia. Uberlândia, 2008.

SQL POWER. *Power Architect*. 2009. Disponível em <<http://www.sqlpower.ca/page/architect>>. Acesso em out de 2008.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL 8.3.3 Documentation*. 2006.

U.S. CENSUS BUREAU. *TIGER*. 2009. Disponível em
<<http://www.census.gov/geo/www/tiger/>> . Acesso em 27 de abr de 2009.

ZEILER, Michael. *Modelling Our World*. Redlands: ESRI Press, 1999.

APÊNDICES

APÊNDICE I – FUNÇÕES PLPGSQL

Apresentaremos inicialmente as funções utilizadas para geocodificar os logradouros e posteriormente as funções utilizadas para a criação de um conjunto de dados aleatórios de acidentes.

1. Gatilho *geocodificar*

```
CREATE OR REPLACE FUNCTION geocodificar()
  RETURNS trigger AS
$BODY$
declare
endereco varchar(100);
tipo_logradouro varchar(30);
via_principal varchar(50);
intersecao1 varchar(50);
intersecao2 varchar(50);
geocode_minadd integer;
geocode_maxadd integer;
outofrange boolean;
numero_viario integer;

-- necessario testar cada variavel, para depois mandar o trigger para a funcao correta
begin
tipo_logradouro:= (select desc_tipolog from aux_tipo_logradouro where cod_tipolog = new.tipolog);
via_principal:= (select desc_logradouro from aux_enderecos where cod_logradouro = new.nomelog);
intersecao1:= (select desc_logradouro from aux_enderecos where cod_logradouro = new.intersecao1);
intersecao2:= (select desc_logradouro from aux_enderecos where cod_logradouro = new.intersecao2);
geocode_minadd:= (select geocode_minadd from logradouros where new.numero between geocode_minadd and
geocode_maxadd and nome_logradouro = via_principal);
geocode_maxadd:= (select geocode_maxadd from logradouros where new.numero between geocode_minadd and
geocode_maxadd and nome_logradouro = via_principal);

if new.numero not between geocode_minadd and geocode_maxadd then
    outofrange:= true;
else
    outofrange:= false;
end if;

numero_viario:=new.numero;

    if numero_viario is null or numero_viario = 0 or outofrange = true then
        if intersecao1 is not null and intersecao2 is not null then
            -- FUNCAO TRECHO
            raise notice 'Trecho';
            update acidentes set the_geom =
geocode.geocode_trecho(via_principal,intersecao1,intersecao2) where cod_acidente = new.cod_acidente;
            return null;
        else
            if intersecao2 is not null and intersecao1 is null then --or intersecao1 = 0
                -- FUNCAO CRUZAMENTO
                raise notice 'Cruzamento';
                update acidentes set the_geom =
geocode.geocode_cruzamento(via_principal,intersecao2) where cod_acidente = new.cod_acidente;
                return null;
            else
                -- nenhuma geocodificação executada
                end if;
            end if;
        else
            --execute a funcao geocoder
            --montar endereco
            endereco:= numero_viario::text || ' ' || via_principal || ' ' || tipo_logradouro;
            raise notice 'Interpolada';
            raise notice 'endereco: %', endereco;
```



```

        update acidentes set the_geom = (select point_geom from geocode.geocode(endereco,'udi'8) order
by rank limit 1)where cod_acidente = new.cod_acidente;
        return null;
    end if;
    return null;
end;
$BODY$
    LANGUAGE 'plpgsql' VOLATILE
    COST 100;
ALTER FUNCTION geocodificar() OWNER TO postgres;
GRANT EXECUTE ON FUNCTION geocodificar() TO public;
GRANT EXECUTE ON FUNCTION geocodificar() TO postgres;

```

Código 12: Função disparada após a entrada de um novo registro na tabela acidentes.

Fonte: SILVA, G. R. C, 2009.

2. Funções Utilizadas pelo Gatilho *geocodificar*

2.1 – Geocodificação Interpolada

```

CREATE OR REPLACE FUNCTION geocode.geocode(character varying, character varying)
RETURNS SETOF geocode.geocode_result AS
$BODY$declare
geocoder geocode.geocoders%ROWTYPE;
sr geocode.standardize_result;
nr geocode.normalize_result;
gr geocode.geocode_result;
percent float;
address varchar;
gn varchar;
sql varchar;
rds record;
nrname varchar;
nrnum int4;
rank int4;

begin
address:=$1;
gn:=$2;

select into geocoder * from geocode.geocoders where geocoder_name=gn;
select into sr * from geocode.standardize(address,geocoder.directions_table,geocoder.types_table);
select into nr * from geocode.normalize(sr.address,sr.dirs,sr.types);

sql:='select * from geocode.view_geocode_'
|| geocoder.geocoder_name ||
' where searchadd=geocode.fuzzyaddress('' || nr.streetname || '' )
and '
|| nr.numberint ||
' between minadd and maxadd;';

raise notice 'sql:%',sql;

for rds in execute sql
loop
raise notice 'nr.number: % right from:% to:%',nr.number, rds.right_from_add, rds.right_to_add;
raise notice 'nr.number: % left from:% to:%',nr.number, rds.left_from_add, rds.left_to_add;
if ((nr.numberint between geocode.min2(rds.left_from_add,rds.left_to_add) and
geocode.max2(rds.left_from_add,rds.left_to_add)) or
(nr.numberint between geocode.min2(rds.right_from_add,rds.right_to_add) and
geocode.max2(rds.right_from_add,rds.right_to_add))) then
raise notice 'good address range';
rank:=0;
if (rds.prefix_direction=nr.predir and rds.suffix_direction=nr.postdir) then rank=rank+4;
elseif (rds.prefix_direction=nr.predir or rds.suffix_direction=nr.postdir) then rank=rank+3;
elseif (rds.suffix_direction=nr.predir or rds.prefix_direction=nr.postdir) then rank=rank+2;

```

⁸ O valor especificado entre aspas simples é a designação do serviço geocodificador.

```

end if;
raise notice 'Score after pre/suffix: %',rank;
if (geocode.standardized_address(rds.streetname,'geocode.dirs','geocode.types')=nr.streetname) then
rank=rank+5; end if;
if (rds.zip5_right=nr.zip5 or rds.zip5_left=nr.zip5) then rank=rank+2; end if;
if (rds.street_type=nr.type) then rank=rank+5; end if;
raise notice 'Score after zip/type: %',rank;
if ((geocode.iseven(geocode.max2(rds.left_from_add,rds.left_to_add))=geocode.iseven(nr.numberint)
and nr.numberint between geocode.min2(rds.left_from_add,rds.left_to_add) and
geocode.max2(rds.left_from_add,rds.left_to_add)) or
not (nr.numberint between geocode.min2(rds.right_from_add,rds.right_to_add) and
geocode.max2(rds.right_from_add,rds.right_to_add))) then
gr.city=rds.city_left;
gr.zip=rds.zip5_left;
if rds.zip5_left=nr.zip5 then rank=rank+2; end if;
raise notice 'Score after left: %',rank;

percent:=(nr.number::float - geocode.min2(rds.left_from_add,rds.left_to_add)::float)/
(geocode.max2(rds.left_from_add,rds.left_to_add)::float -
geocode.min2(rds.left_from_add,rds.left_to_add)::float +.0001);
raise notice 'nr.number: % % %',nr.number, rds.left_from_add, rds.left_to_add;
raise notice 'percent: %',percent;
gr.point_geom=line_interpolate_point(st_geometryn(rds.geom,0),percent);
gr.fulladdress:=geocode.unnull(nr.numberint::varchar) || ' ' ||
geocode.unnull(rds.prefix_direction::varchar) || ' ' ||
geocode.unnull(rds.streetname::varchar) || ' ' || geocode.unnull(rds.street_type::varchar) || '
' || geocode.unnull(rds.suffix_direction::varchar) ||
' ' || geocode.unnull(rds.zip5_left::varchar);
raise notice 'fulladdress: %',gr.fulladdress;
gr.fulladdwcity:=geocode.unnull(nr.numberint::varchar) || ' ' ||
geocode.unnull(rds.prefix_direction) || ' ' ||
geocode.unnull(rds.streetname) || ' ' || geocode.unnull(rds.street_type) || ' ' ||
geocode.unnull(rds.suffix_direction) ||
' ' || geocode.unnull(rds.city_left) || ' ' || geocode.unnull(rds.zip5_left::varchar);
if not (nr.numberint between geocode.min2(rds.left_from_add,rds.left_to_add) and
geocode.max2(rds.left_from_add,rds.left_to_add)) then rank=rank-2; end if;
else
gr.city=rds.city_right;
gr.zip=rds.zip5_right;
if rds.zip5_left=nr.zip5 then rank=rank+2; end if;
raise notice 'Score after right: %',rank;
percent:=(nr.number::float - geocode.min2(rds.right_from_add,rds.right_to_add)::float)/
(geocode.max2(rds.right_from_add,rds.right_to_add)::float -
geocode.min2(rds.right_from_add,rds.right_to_add)::float +.0001);
raise notice 'nr.number: % % %',nr.number, rds.right_from_add, rds.right_to_add;
raise notice 'percent: %',percent;
gr.point_geom=st_line_interpolate_point(st_geometryn(rds.geom,1),percent);
gr.fulladdress:=geocode.unnull(nr.numberint::varchar) || ' ' ||
geocode.unnull(rds.prefix_direction::varchar) || ' ' || geocode.unnull(rds.streetname::varchar) || ' '
|| geocode.unnull(rds.street_type::varchar) || ' ' || geocode.unnull(rds.suffix_direction::varchar) ||
' ' || geocode.unnull(rds.zip5_right::varchar);
raise notice 'fulladdress: %',gr.fulladdress;
gr.fulladdwcity:=geocode.unnull(nr.numberint::varchar) || ' ' ||
geocode.unnull(rds.prefix_direction) || ' ' || geocode.unnull(rds.streetname) || ' ' ||
geocode.unnull(rds.street_type) || ' ' || geocode.unnull(rds.suffix_direction) || ' ' ||
geocode.unnull(rds.city_right) || ' ' || geocode.unnull(rds.zip5_right::varchar);
if not (nr.numberint between geocode.min2(rds.right_from_add,rds.right_to_add) and
geocode.max2(rds.right_from_add,rds.right_to_add)) then rank=rank-2; end if;
end if;

gr.rank=rank;
gr.streetname=rds.streetname;
gr.geom=rds.geom;
gr.predir:=rds.prefix_direction;
gr.sufdir:=rds.suffix_direction;
gr.streetnum:=nr.numberint;
gr.fulladdress:=geocode.strip_double_spaces(gr.fulladdress);
gr.fulladdwcity:=geocode.strip_double_spaces(gr.fulladdwcity);

return next gr;
end if;
end loop;
return;

```

```

end;
$BODY$
    LANGUAGE 'plpgsql' VOLATILE
    COST 100
    ROWS 1000;
ALTER FUNCTION geocode.geocode(character varying, character varying) OWNER TO postgres;

```

Código 13: Função geocode. Esta função executa a geocodificação interpolada e foi adaptada para se adequar ao formato de endereçamento brasileiro.

Fonte: BITNER, D., SILVA, G. R. C, 2009.

2.3. Função Geocodificar por Cruzamento

```

CREATE OR REPLACE FUNCTION geocode.geocode_cruzamento(character varying, character varying)
    RETURNS geometry AS
$BODY$

declare
geo_ponto geometry;
geometria geometry;

begin

geometria:=(select the_geom from logradouros where nome_logradouro=$1 and sufixo_direcao=$2);

    if geometria is not null then
        geo_ponto := st_line_interpolate_point(st_geometryn(geometria,1),.99);
        raise notice 'Ponto Encontrado';
        return geo_ponto;
    else
        raise notice 'Ponto Não Encontrado';
        return null;
    end if;
end;
$BODY$
    LANGUAGE 'plpgsql' VOLATILE
    COST 100;
ALTER FUNCTION geocode.geocode_cruzamento(character varying, character varying) OWNER TO postgres;

```

Código 14: Função geocode_cruzamento. Esta função executa a localização dos endereços através dos cruzamentos fornecidos pelo usuário.

Fonte: SILVA, G. R. C, 2009.

2.4. Função Geocodificar por Trecho

```

CREATE OR REPLACE FUNCTION geocode.geocode_trecho(text, text, text)
    RETURNS geometry AS
$BODY$

declare
geo_ponto geometry;
geometria geometry;
begin

--seleciona o trecho de logradouro correto
geometria:= (select the_geom from logradouros where nome_logradouro=$1 and prefixo_direcao=$2 and
sufixo_direcao=$3) ;

    if geometria is not null then
        --interpola o logradouro e localiza o ponto no ponto médio
        geo_ponto := st_line_interpolate_point(st_geometryn(geometria,1),.5);
        raise notice 'Ponto Encontrado';
        return geo_ponto;
    else
        raise notice 'Ponto Não-Encontrado';
        Return null;
    end if;
end;

```

```

        end if;
    end;
$BODY$
    LANGUAGE 'plpgsql' VOLATILE
    COST 100;
ALTER FUNCTION geocode.geocode_trecho(text, text, text) OWNER TO postgres;

```

Código 15: Função geocode_trecho. Esta função localiza os endereços passados pelo usuário de acordo com as interseções anterior e posterior.

Fonte: SILVA, G. R. C, 2009.

3. Funções utilizadas pelo gerador de acidentes

3.1. Função Wrapper Externa

```

CREATE OR REPLACE FUNCTION customfunctions.inserir_acidentes(limite integer)
    RETURNS void AS
$BODY$
    declare

    contador integer;
    dummiesql text;

    begin
    contador :=1;

        while contador<=limite loop
            dummiesql:=customfunctions.gerador(contador);
            if dummiesql is null then
                contador:=contador+1;
                dummiesql:='';
            else
                execute dummiesql;
                dummiesql:='';
                contador:=contador+1;
            end if;
        end loop; --saida loop

    end;
$BODY$
    LANGUAGE 'plpgsql' VOLATILE
    COST 100;
ALTER FUNCTION customfunctions.inserir_acidentes(integer) OWNER TO postgres;

```

Código 16: Função Wrapper Externa. Esta função é a única à qual o usuário comum tem acesso, administrando o acesso as outras conforme solicitado pelo usuário.

Fonte: SILVA, G. R. C, 2009.

3.2. Função Gerador

```

CREATE OR REPLACE FUNCTION customfunctions.gerador(contador_geral integer)
    RETURNS text AS
$BODY$
    declare
    contador integer;
    no_veiculos integer;

    sqlacidente text;
    sqlveiculo text;
    sqlcondutor text;

    dummyveiculo text;
    dummycondutor text;

    sqlgeral text;

```

```

begin
contador:=1;
no_veiculos:= myrandom(2,5);

raise notice 'noveiculos: %', no_veiculos;

--define sqlacidente
sqlacidente:=customfunctions.gera_acidente(contador_geral,no_veiculos);
sqlveiculo:='';
sqlcondutor:='';

    while contador <= no_veiculos - 1 loop --entrada loop

        dummyveiculo:= customfunctions.gera_veiculo(contador_geral);--esta variavel passa o
valor do codigo do acidente

        sqlveiculo:= sqlveiculo || dummyveiculo;

        dummycondutor:= customfunctions.gera_condutor(contador_geral,contador);--variaveis
passam o valor do codigo do acidente e veiculo

        sqlcondutor:=sqlcondutor || dummycondutor;

        contador:=contador+1;

--limpa variaveis

        dummyveiculo:='';
        dummycondutor:='';

    end loop; --saida loop

    sqlgeral:=sqlacidente || sqlveiculo || sqlcondutor;

    return sqlgeral;

end;
$BODY$
LANGUAGE 'plpgsql' VOLATILE
COST 100;
ALTER FUNCTION customfunctions.gerador(integer) OWNER TO postgres;

```

Código 17:Função gerador. Esta função realiza as contagens necessárias para se inserir informações sem corrupção, assegurando que o número de veículos e condutores seja compatível com o gerado.

Fonte: SILVA, G. R. C, 2009.

3.2. Função gera_acidente

```

CREATE OR REPLACE FUNCTION customfunctions.gera_acidente(cd_acidente integer, n_veiculos integer)
RETURNS text AS
$BODY$
declare
--declaracao das variaveis
tp_logradouro integer;
desc_tp_logradouro text;
nome_logradouro integer;
no_viario integer;
caracteristica_acidente integer;
causa_acidente integer;
tipo_acidente integer;
caracteristica_via integer;
condicao_via integer;
pavimentacao_via integer;
tipo_socorro integer;
orgao_responsavel integer;
controle_trafego integer;
clima integer;
veiculos_envolvidos integer;

num_max integer;
num_min integer;

```

```

sqlbase text;

begin

sqlbase:='insert into
acidentes(cod_acidente,tipolog,nomelog,numero,acid_caracteristica,acid_causa,acid_tipo,acid_caracterist
icavia,acid_condicaoavia,acid_pavimentacao,acid_tiposocorro,acid_orgaoresponsavel,acid_controletrafego,a
cid_clima,no_veiculos) values (';

--assinalando variaveis
--procurando um endereco compativel ao tipo do logradouro escolhido aleatoriamente
nome_logradouro:= (select cod_logradouro from aux_enderecos order by random() limit 1);
num_max:= (select max(geocode_maxadd) from logradouros where cod_logradouro=nome_logradouro);
num_min:= (select min(geocode_minadd) from logradouros where cod_logradouro=nome_logradouro);
desc_tp_logradouro:= (select tipo_logradouro from aux_enderecos where cod_logradouro =
nome_logradouro);
tp_logradouro:= (select cod_tipolog from aux_tipo_logradouro where desc_tipolog = desc_tp_logradouro);
--fim procura pelo nome_logradouro;

no_viario:=myrandom(num_min,num_max);

caracteristica_acidente:= (select cod_caracteristica from aux_acid_caracteristica order by random()
limit 1);
causa_acidente:= (select cod_causa from aux_acid_causa order by random() limit 1);
tipo_acidente:= (select cod_tipo from aux_acid_tipo order by random() limit 1);
caracteristica_via:= (select cod_caracteristica from aux_acid_caracteristicavia order by random() limit
1);
condicao_via:= (select cod_condicao from aux_acid_condicaoavia order by random() limit 1);
pavimentacao_via:= (select cod_pavimentacao from aux_acid_pavimentacao order by random() limit 1);
tipo_socorro:= (select cod_socorro from aux_acid_socorro order by random() limit 1);
orgao_responsavel:= (select cod_orgao from aux_acid_orgaoresp order by random() limit 1);
controle_trafego:= (select cod_controle from aux_acid_controletrafego order by random() limit 1);
clima:= (select cod_clima from aux_acid_clima order by random() limit 1);

veiculos_envolvidos:=n_veiculos;

sqlbase:= sqlbase ||
cd_acidente::text || ',' ||
tp_logradouro::text || ',' ||
nome_logradouro::text || ',' ||
no_viario::text || ',' ||
caracteristica_acidente::text || ',' ||
causa_acidente::text || ',' ||
tipo_acidente::text || ',' ||
caracteristica_via::text || ',' ||
condicao_via::text || ',' ||
pavimentacao_via::text || ',' ||
tipo_socorro::text || ',' ||
orgao_responsavel::text || ',' ||
controle_trafego::text || ',' ||
clima::text || ',' ||
veiculos_envolvidos::text || ');';

return sqlbase::text;

end;
$BODY$
LANGUAGE 'plpgsql' VOLATILE
COST 100;
ALTER FUNCTION customfunctions.gera_acidente(integer, integer) OWNER TO postgres;

```

Código 18: Função gera_acidente. Esta função gera aleatoriamente alguns parametros da tabela acidentes e os devolve para a função gerador, de modo a cosntruir um comando SQL.

Fonte: SILVA, G. R. C, 2009.

3.3. Função gera_veiculo

```

CREATE OR REPLACE FUNCTION customfunctions.gera_veiculo(cd_acidente integer)
RETURNS text AS
$BODY$

```

```

declare
tp_veiculo integer;
veiculo_apreendido integer;
no_ocupantes integer;
equipseguranca integer;
sentido_veic integer;

severidades text;

sqlbase text;

begin

sqlbase:='insert into
veiculos(cod_acidente_veiculo,tipo_veiculo,apreendido,no_ocupantes,sentido_veiculo,veic_equipseg,sev_se
m_danos,sev_danos_materiais,sev_feridos_leves,sev_feridos_graves,sev_fatalidades) values (';

tp_veiculo:= (select cod_tipo from aux_veic_tipo order by random() limit 1);

--definicao de quantos passageiros de acordo com o tipo de veiculo
if tp_veiculo = 1 then
no_ocupantes:= 1;
else
    if tp_veiculo <= 8 AND tp_veiculo >= 3 then
        no_ocupantes:=2;
    else
        no_ocupantes := myrandom(1,4);
    end if;
end if;
--fim passageiros

veiculo_apreendido := (select cod_apreendido from aux_veic_apreendido order by random() limit 1);
sentido_veic:= (select cod_sentido from aux_veic_sentido order by random() limit 1);

equipseguranca:= (select cod_equip from aux_veic_equipamentoseg order by random() limit 1);

--iniciar selecao de severidades
severidades:= customfunctions.gera_severidade_veiculos(no_ocupantes);

sqlbase:= sqlbase ||
cd_acidente::text || ',' ||
tp_veiculo::text || ',' ||
veiculo_apreendido::text || ',' ||
no_ocupantes::text || ',' ||
sentido_veic::text || ',' ||
equipseguranca::text || ',' ||
severidades::text ||
');';

return sqlbase;

end;
$BODY$
LANGUAGE 'plpgsql' VOLATILE
COST 100;
ALTER FUNCTION customfunctions.gera_veiculo(integer) OWNER TO postgres;

```

Código 19: Função gera_veiculo. Esta função gera aleatoriamente alguns parametros da tabela veiculos e os devolve para a função gerador, de modo a construir um comando SQL.

Fonte: SILVA, G. R. C, 2009.

3.4. Função gera_condutor

```

CREATE OR REPLACE FUNCTION customfunctions.gera_condutor(cd_acidente integer, cd_veiculo integer)
RETURNS text AS
$BODY$
declare
--declaracao de variaveis
--o codigo do acidente e condutor vem dos contadores na funcao

```

```

escolaridade_condutor integer;
profissao_condutor integer;
comportamento_condutor integer;
condicao_condutor integer;
habilitacao_condutor integer;
sexo_condutor integer;

sqlbase text;

begin

sqlbase:= 'insert into condutor(cod_acidente_condutor, cod_veiculo_condutor, escolaridade, profissao,
comportamento, condicao_fisica, habilitacao, cond_sexo) values (';

escolaridade_condutor:= (select cod_escolaridade from aux_cond_escolaridade order by random() limit 1);
profissao_condutor:= (select cod_profissao from aux_cond_profissao order by random() limit 1);
comportamento_condutor:= (select cod_comportamento from aux_cond_comportamento order by random() limit
1);
condicao_condutor:= (select cod_condicao from aux_cond_condicao order by random() limit 1);
habilitacao_condutor:= (select cod_habilitacao from aux_cond_habilitacao order by random() limit 1);
sexo_condutor:= (select cod_sexo from aux_cond_sexo order by random() limit 1);

sqlbase:=sqlbase ||
cd_acidente::text || ',' ||
cd_veiculo::text || ',' ||
escolaridade_condutor::text || ',' ||
profissao_condutor::text || ',' ||
comportamento_condutor::text || ',' ||
condicao_condutor::text || ',' ||
habilitacao_condutor::text || ',' ||
sexo_condutor::text || ');';

return sqlbase;

end;
$BODY$
LANGUAGE 'plpgsql' VOLATILE
COST 100;
ALTER FUNCTION customfunctions.gera_condutor(integer, integer) OWNER TO postgres

```

Código 20: Função gera_condutor. Esta função gera aleatoriamente alguns parametros da tabela condutores e os devolve para a função gerador, de modo a construir um comando SQL.

Fonte: SILVA, G. R. C, 2009.

3.5. Função gera_severidade_veiculos

```

CREATE OR REPLACE FUNCTION customfunctions.gera_severidade_veiculos(no_passageiros integer)
RETURNS text AS
$BODY$
declare

sev_semdanos integer;
sev_danosmateriais integer;
sev_feridosleves integer;
sev_feridosgraves integer;
sev_fatalidades integer;
sev_naoapurado integer;

contador integer;

dummy_sql text;
begin

contador := no_passageiros;

sev_semdanos := myrandom(0,1);

--determinando se ocorreram danos materiais
if sev_semdanos >= 1 then
sev_semdanos:=1;
sev_danosmateriais := 0;
else

```



```

        sev_semdanos:=0;
        sev_danosmateriais := 1;
    end if;
    --fim danos materiais

--determinando feridos leves
sev_feridosleves:=myrandom(0,contador);
--fim feridos leves

--atualizando valor possivel para numero de passageiros
contador := contador - sev_feridosleves;

--determinando feridos graves
sev_feridosgraves := myrandom(0,contador);
--fim feridos graves

--atualizando valor numero passageiros
contador := contador- sev_feridosgraves;

--determinando mortalidade
sev_fatalidades := myrandom(0,contador);

if contador <> 0 then
sev_naoapurado := contador;
    if sev_naoapurado < 0 then
        sev_naoapurado = 0;
    else
        end if;
else
sev_naoapurado := 0;
end if;

dummy_sql := sev_semdanos::text || ',' || sev_danosmateriais::text || ',' || sev_feridosleves::text ||
',' || sev_feridosgraves::text || ',' || sev_fatalidades::text;

return dummy_sql;

end
$BODY$
    LANGUAGE 'plpgsql' VOLATILE
    COST 100;
ALTER FUNCTION customfunctions.gera_severidade_veiculos(integer) OWNER TO postgres;

```

Código 21: Função gera_severidade_veiculos. Esta função gera aleatoriamente a severidade a ser inserida na tabela veículos e retorna o resultado para a função gera_veiculos.

Fonte: SILVA, G.R.C., 2009.