UNIVERSIDADE ESTADUAL DO PIAUÍ
CAMPUS PROF. ALEXANDRE ALVES DE OLIVEIRA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

WASHINGTON COSTA SILVA

SGDBs LIVRES: um estudo comparativo entre os sistemas gerenciadores de banco de dados Firebird, MySQL e PostgreSQL

PARNAÍBA
2011

WASHINGTON COSTA SILVA

SGDBs LIVRES: um estudo comparativo entre os sistemas gerenciadores de banco de dados Firebird, MySQL e PostgreSQL

Monografia apresentada ao programa do Curso de Bacharelado em Ciências da Computação, como pré-requisito para obtenção do Título de Bacharel em Ciência da Computação, sob a orientação do Prof. MSc. José Flávio Gomes Barros.

PARNAÍBA

2011

FICHA CATALOGRÁFICA ELABORADA PELO BIBLIOTECÁRIA
CÁTIA REGINA FURTADO DA COSTA CRB-3/1109

S586s Silva, Washington Costa Silva.
SGDBs livres: um estudo comparativo entre os sistemas gerenciadores de banco de dados Firebird, MySQL e PostgreSQL. / Washington Costa Silva. – Parnaíba, 2011.
78 f.

Monografia de conclusão de curso de Bacharelado em Ciências da Computação, Universidade Estadual do Piauí, Parnaíba, 2011.
Orientador: Prof. Msc. José Flávio Gomes Barros.

1. Programação – Computadores. 2. Sistemas gerenciadores – SGDBs. 3. Banco de Dados – Firebird – MySQL. 4. Banco de Dados – PostgreSQL. I. Título.

CDD – 005.43

GOVERNO DO ESTADO DO PIAUÍ
UNIVERSIDADE ESTADUAL DO PIAUÍ – UESPI
CAMPUS PROF. ALEXANDRE ALVES DE OLIVEIRA
COORDENAÇÃO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Ata de Defesa de Trabalho de Conclusão de Curso

Aos dezenove dias do mês de março de dois mil e onze, às 14h00, no Laboratório de Informática do Campus Prof. Alexandre Alves Oliveira - UESPI realizou-se a sessão pública de defesa do Trabalho de Conclusão de Curso do aluno **Washington Costa Silva**, intitulado **SGBDs Libres: um estudo comparativo entre os sistemas gerenciadores de banco de dados Firebird, MySQL e PostgreSQL**, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação. A Banca Examinadora foi presidida pelo Prof. José Flávio Gomes Barros e composta pelos membros efetivos os professores Francisco das Chagas Rocha e Jean Carlo Galvão Mourão. Aberta a sessão pública, o candidato teve a oportunidade de expor seu trabalho. Após a exposição, o aluno foi arguido oralmente pelos membros da Banca Examinadora. Em seguida, em sessão reservada e nos termos do Regulamento Geral dos Trabalhos de Conclusão de Curso da UESPI, os membros da Banca realizaram a avaliação do candidato, que obteve nota **8,4 (oito ponto quatro)** sendo, portanto, **aprovado**. Nada mais havendo a tratar, eu, Prof. José Flávio Gomes Barros, lavrei a presente Ata que será lida e assinada por mim, pelos demais membros da Banca Examinadora e pelo candidato. Parnaíba (PI), 19 de março de 2011.

Banca Examinadora

Prof. M.Sc. José Flávio Gomes Barros
Orientador, UESPI

Prof. M.Sc. Francisco das Chagas Rocha
Avaliador, UESPI

Prof. Esp. Jean Carlo Galvão Mourão
Avaliador, UESPI

Candidato

Washington Costa Silva

Dedico a Deus, à minha família, à minha namorada, e amigos.

AGRADECIMENTOS

Por poder abrir meus olhos todas as manhãs
Agradeço a Deus por conceder este milagre
Fazendo da fé e esperança minhas guardiãs
Faça-me de amor, e com seu poder me sagre

Por eu existir e fazer-me aqui estar
Agradeço à minha família por seu seio
Fazer de mim menino e poder me educar
Para transformar-me hoje homem feito

Por aquecer e transbordar meu coração
Agradeço a ti anja na terra assim feita
A guiar-me e amar-me além da razão
A flor de meu campo és tu Romeilla

Por sempre alegrarem a minha vida
De já agradeço a todos meus amigos
Nos revoltos tempos me dão energia
Em dias felizes me dão seus sorrisos

Por ter me feito buscar o conhecimento
Agradeço à UESPI por todo seu suporte
Dos mestres aos colegas fica o alento
Para poder agora eu buscar meu norte

Comece fazendo o que é necessário, depois o que é possível, e de repente você estará fazendo o impossível.

São Francisco de Assis

RESUMO

Este trabalho apresenta uma investigação sobre o desempenho entre os SGDBs livres Firebird, MySQL e PostgreSQL. A partir deste objetivo, buscaram-se informações técnicas a respeito de software livre e bancos de dados; dos referidos SGDBs (Firebird, MySQL e PostgreSQL), compará-los por meio de tópicos qualitativos e analisar suas performances por meio de um aplicativo feito em JAVA. Tais metas foram alcançadas e descritas nesta monografia. Diante do exposto, espera-se que o referido trabalho monográfico possa ajudar os profissionais e estudantes da computação a refletir sobre o uso desses bancos de dados, formando suas opiniões e dá-los argumentos para a escolha do SGDB ideal para seu projeto.

PALAVRAS- CHAVE: Firebird. MySQL. PostgreSQL. JAVA. Performances. SGDBs livres.

ABSTRACT

This paper presents an investigation about the performance between the free DBMSes Firebird, MySQL and PostgreSQL. From this aim, sought to technical information about free software and databases; of those DBMSes (Firebird, MySQL and PostgreSQL), compare them by means of qualitative topics and analyze their performances by means of an application made in JAVA. These goals were achieved and described in this monograph. Given the above, it is hoped that this monograph can help computing professionals and students to reflect on the use of these databases, forming their opinions and arguments to give them the choice of DBMS ideal for your project.

KEY-WORDS: Firebird. MySQL. PostgreSQL. JAVA. Performances. Free DBMSes.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

SGDBs – Sistemas Gerenciadores de Banco de Dados
SGDB – Sistema Gerenciador de Banco de Dados
POLI/USP – Escola Politécnica da Universidade de São Paulo
BD – Banco de Dados
SQL – Structured Query Language
XML – Extensible Markup Language
TI – Tecnologia da Informação
IMS – Information Management System
DBTG – Data Base Task Group
CODASYL – Committee on Data Systems and Languages
E-R – Entidade-Relacional
OO – Orientado a Objetos
O/R – Objeto/Relacional
LOO – Linguagem Orientada a Objetos
DBA – Database Administrator
SGDBOO – Sistema Gerenciador de Banco de Dados Orientado a Objetos
ODMG – Object Data Management Group
OQL – Object Query Language
ANSI – American National Standards Institute
SEQUEL – Structured English Query Language
ISO – International Organization for Standardization
DDL – Data Definition Language
DML – Data Manipulation Language
UDFs – User Defined Functions
API – Application Programming Interface
EMBRAPA – Empresa Brasileira de Pesquisa Agropecuária
ESALQ-USP – Escola superior de Agricultura Luiz de Queiroz da Universidade de São Paulo
SPs – Stored Procedures
GIS – Geographic Information System
ODBC – Open Database Connectivity
OLAP – Online Analytical Processing
SGDBR – Sistema Gerenciador de Banco de Dados Relacional
WAL – Write Ahead Log
Md5 – Message-Digest Algorithm 5
SSL – Secure Sockets Layer
SSH – Secure Shell
RAM – Random Access Memory
FK – Foreign Key
MV – Máquina Virtual
HD – Hard Disk
SATA – Serial Advanced Technology Attachment
Gb – Giga Byte

SUMÁRIO

1 INTRODUÇÃO

A sociedade vem modificando rapidamente a maneira de agir e de como pensar o mundo, e este processo tem laços diretos ao modo pelo qual se lida com a informação, em que o domínio desta significa estar a frente a seu tempo. No entanto, nas mais diversas áreas, este saber necessita de um meio que o comportasse e o pudesse processar mais rapidamente, pois o mesmo no decorrer dos tempos tornou-se muito vasto.

Isso se tornou realidade graças ao advento da computação, onde com ela foi possível desenvolver *Software* capazes de armazenar, gerir e extrair informações de massas de dados gigantescas num espaço de tempo dantes inimaginável, existindo diversos tipos de programas de computador para as mais variadas atividades.

E os *Software* por sua vez podem ser divididos em diversas categorias, dentre as quais existe uma que nos últimos anos vem ganhando foco e despertando a curiosidade de usuários, desenvolvedores e empresas: a licença. Nesta modalidade existem basicamente duas categorias, a saber: *Software* livres e proprietários.

A diferença primordial entre ambos os tipos é justamente em relação ao acesso ao código-fonte de um programa, em que os livres há esta liberdade e nos proprietários não se goza desta particularidade. Além disso, nos primeiros também se tem totais poderes para estudá-lo, compartilhá-lo e modificá-lo. Já nos segundos, seus fabricantes limitam suas funcionalidades mediante versões, sendo ilegal modificá-lo ou redistribuir cópias dos mesmos.

Este primeiro paradigma se mostra muito interessante para a comunidade pelo intuito de se compartilhar o conhecimento. E, nesta leva de liberdade no meio da informática, para quase todas as demandas de programas, existe uma alternativa livre para cada uma. Assim, com o passar dos tempos, essas alternativas livres desenvolveram-se de tal maneira chegando, em certos casos, a serem tão competitivas, quanto aos mais bem sucedidos programas pagos presentes no mercado.

Não diferente desse movimento, o ramo dos sistemas gerenciadores de bancos de dados, em que estes são sistemas onde armazenam e gerenciam informações e as retorna quando solicitadas, possui seus eleitos, valendo destacar três deles que vem a ser: *Firebird*, *MySQL* e *PostgreSQL*.

Cada um dos servidores de bancos de dados mencionados anteriormente possuem suas peculiaridades, divergências e pontos em comum. Desta forma, para a decisão de qual deva se

escolher para se implementar uma solução computacional, deve-se investigar qual atende aos pré-requisitos necessários para se atingir à solução ótima. Portanto, para chegar a este objetivo é preciso estudá-los e confrontá-los por meio dos critérios exigidos pela aplicação a ser implementada.

Tendo em vista investigar a performance dos gerenciadores de bancos de dados *Firebird*, *MySQL* e *PostgreSQL*, nas respectivas versões 2.1 *Classic*, 5.1 e 8.4, foi feito este trabalho com os seguintes objetivos específicos:

- Obter informações técnicas a respeito de *Software* livres, banco de dados e das ferramentas de gerência e desenvolvimento de base de dados *Firebird*, *MySQL* e *PostgreSQL*;
- Comparar os três SGDBs por meio de tópicos qualitativos;
- Implementar um aplicativo na linguagem de programação JAVA (FMPSys), que se comunique com os referidos SGDBs;
- Analisar quantitativamente algumas das principais rotinas existentes e comuns nos SGDBs mencionados por via desta aplicação.

A iniciativa para a elaboração deste trabalho deu-se pela necessidade de se saber qual dos sistemas de bancos de dados, no caso *Firebird*, *MySQL* e *PostgreSQL*, seria o ideal para o desenvolvimento de um projeto. Chega-se a esta problemática a partir dos seguintes questionamentos:

- O que diferencia e/ou iguala cada gerenciador de banco de dados?
- Quais seriam as limitações de cada SGDB?
- Dadas as principais rotinas de um sistema gerenciador de banco de dados, qual seria o que possui melhores resultados quantitativos em cada uma das transações executadas no sistema FMPSys?

Como meio para realização deste estudo, inicialmente buscou-se informações mais detalhadas a respeito de *Software* livres e bancos de dados e em seguida pesquisou-se a respeito de cada gerenciador de banco de dados em questão. Logo depois, por meio de tópicos comparativos, confrontaram-se os referidos SGDBs, podendo assim ver em que pontos há paridades e/ou convergências entre ambos. Dando sequência, fez-se um aplicativo chamado FMPSys, feito na linguagem de programação JAVA na plataforma Netbeans 6.9.1 que se conecta aos três sistemas de bases de dados e realizando algumas das mais utilizadas rotinas de bancos de dados (*Create* - criação de banco ou tabela, *Insert* - inserir dados numa tabela, *Select* - retorna os dados de uma tabela, *Delete* - apaga os dados de uma tabela, *Drop* – exclui

um banco de dados ou tabela), com intuito de analisar o tempo gasto, a carga e integridade dos dados.

Para a realização desta atividade monográfica, contaram-se como principais suportes teóricos para se criar um entendimento a cerca do assunto e a partir daí formular argumentação suficiente para se afirmar os fatos aqui contidos: Silberschartz et.al. (2006), Stallman (2010), Takai et. al. (2005), Machado (1996), Cantu (2005), Kofler (2005), Scherer et. al. (2007), Neto (2007), Caelum (2010), entre outros.

Contudo, para a realização de qualquer atividade, deve haver algo que nos motive para tal realização. O que nos fez cumprir esta meta foi a afinidade com a área de banco de dados, a carência de materiais com este foco, pois acham-se mais conteúdos tratando de cada SGDB individualmente ou comparando-se dois deles apenas de cada vez. Outro motivo, sendo o preponderante, é formar as opiniões tanto dos profissionais da área de computação assim como de estudantes e curiosos, para que tenham uma base sobre cada sistema gerenciador de banco de dados tratado neste trabalho, e daí então buscar melhores soluções em desenvolvimento.

O presente trabalho está dividido da seguinte maneira: esta introdução, abordando aspectos introdutórios, objetivos e justificativa para realização do referido estudo.

No capítulo 2, serão mostradas definições à respeito de *Software* livres; banco de dados, no tocante aos conteúdos, definição e origem; alguns paradigmas de modelagem; além de projeto de banco de dados e SQL.

No capítulo 3, discorreu-se à respeito dos gerenciadores de banco de dados em foco: *Firebird*, *Mysql*, e *Postgresql*, levantando-se suas características, diferenciais e deficiências.

No capítulo 4, foram comparados os respectivos três SGDBs por meio de alguns dos principais tópicos pertinentes à gerenciadores de bancos de dados, através de uma tabela.

No capítulo 5, mostrou-se a implementação do sistema FMPSys na plataforma de programação JAVA, no qual falou-se sobre a referida linguagem; descreveu-se à aplicação com suas funcionalidades; e efetuaram-se testes para análise de performance nos dados gerenciadores banco de dados.

E, finalmente, o capítulo 6 expõe as conclusões a cerca do trabalho, dando ênfase em perspectivas futuras.

2 SOFTWARE E BANCO DE DADOS LIVRES

2.1 Software Livre

Atualmente ouve-se falar muito sobre *Software* livre, migração de programas pagos à livres e grandes empresas e governos aderindo a este movimento. Mas o que é realmente todo este fenômeno? Defini-se então *Software* livre, segundo Cassino (2003), como um programa de computador com código-fonte aberto, possibilitando que qualquer técnico possa estudá-lo, alterá-lo, adequá-lo às suas próprias necessidades e redistribuí-lo, sem restrições.

De acordo com Beline et.al. (2006), nos primórdios os aplicativos eram distribuídos juntamente com o hardware e isentos de pagamento, mas em certo instante as grandes empresas do ramo registraram patentes sobre os códigos-fonte, cobrando e limitando assim o acesso e modificação a estes. Assim nasce o *Software* proprietário, onde não se tem as liberdades mencionadas anteriormente e, segundo Bacic (2003), esta modalidade visa limitar seu uso via as licenças adquiridas, onde é necessário pagar por cada cópia instalada, além de ser ilegal copiá-lo, distribuí-lo, ter acesso e modificar seu código.

Apesar de hoje ser um assunto em voga, o *Software* livre não é tão recente quanto se pensa. As bases deste nos remete ao início dos anos 80, onde segundo Bacic (2003), Richard Stallman trabalhava em seu laboratório e sua impressora quebrou, sendo necessário adquirir uma nova. Para continuar a realizar seu estudo, ele precisava conhecer o funcionamento desta nova impressora, no entanto suas solicitações ao fabricante para que este liberasse os códigos-fonte do equipamento foram negadas. A partir deste momento, Richard Stallman passou a pensar em uma forma de tornar acessíveis os programas e seus códigos-fonte, surgindo a ideia de *Software* livre.

No entanto, para que um programa seja dito livre, é forçoso que o usuário venha a ter autonomia deste em relação a alguns tópicos e assim poder manipulá-lo. Em Stallman (2010) são enumeradas quatro liberdades aos usuários em relação aos *Software*, em que defini-se o mesmo é ou não de fato livre, os quais são:

- **Liberdade nº 0**: a liberdade de executar o programa, para qualquer propósito;
- **Liberdade nº 1**: a liberdade de estudar como o programa funciona, e adaptá-lo para as suas necessidades. O acesso ao código-fonte é um pré-requisito para esta liberdade;

- **Liberdade nº 2**: a liberdade de redistribuir cópias de modo a poder ajudar ao outro;
- **Liberdade nº 3**: a liberdade de aperfeiçoar o programa e liberar os seus aperfeiçoamentos de modo que toda a comunidade se beneficie. O acesso ao código-fonte também é um pré-requisito para esta liberdade.

Assim, mesmo nesta abordagem de desenvolvimento, no caso livre, há um dispêndio financeiro. E, por mais que sua intitulação sugira gratuidade, o mesmo não necessariamente é gratuito. Este equívoco é cometido por causa da tradução e entendimento errôneo do termo *Free Software* (*Software* Livre), onde Ferraz (2002) explica justamente esta situação, onde este vocábulo é associado à palavra "grátis". No entanto, esta palavra representa o caráter das liberdades mencionadas anteriormente e não ao preço:

> (...) deve-se ser livre para redistribuir cópias, seja com ou sem modificações, seja gratuitamente ou cobrando taxas pela distribuição, para qualquer pessoa, empresa ou grupo e em qualquer lugar. Ser livre para fazer essas coisas significa (entre outras coisas) que não há necessidade de pedir ou pagar pela permissão (STALLMAN, 2010, p.01).

Corroborando com o mencionado acima, o *Software* livre tem seus custos associados a insumos e pessoal para gerência, suporte e manutenção de tais recursos tecnológicos. Isto é explicitado por Beline et.al. (2006), o qual diz que o *Software* livre não é sinônimo de gratuidade ou de custo zero. O usuário terá que disponibilizar pessoal, tempo e outros recursos para administrar qualquer sistema baseado em *Software* livre.

Entretanto, o que faz desta possibilidade de se usar um programa de código-aberto ser real é devido à grande integração existente entre a comunidade de desenvolvedores. Segundo Beline et.al. (2006), John 'Maddog' Hall (Diretor-executivo da *Linux International*), numa palestra proferida na POLI/USP em novembro de 2003, abordou a existência de cerca de 300.000 programadores envolvidos com *Software* livre no mundo. Sendo este o maior projeto colaborativo/cooperativo já imaginado pela humanidade.

Porém em ambos modelos, proprietário ou livre, há uma dependência de recursos financeiros para sua manutenção, mas o grande diferencial do *Software* livre é a curiosidade e interesse pessoal dos desenvolvedores. Já as indústrias dos *Software* proprietários não divulgam os códigos fontes para que o conhecimento seja explicitado, mas sim divulga o código binário, o que não pode ser internalizado. Sendo assim, a proteção intelectual, neste tipo de *Software* proprietário, é utilizada como um mero instrumento de recompensa financeira, não beneficiando a sociedade e sim os próprios fabricantes (Ferraz, 2002).

Mas, no caso de sistemas proprietário, segundo Hexsel (2002), a manutenção após sua

compra recai em gastos muito mais dispendiosos, pois depende dos serviços monopolizados pelo fabricante. E, na pior das hipóteses, ainda segundo este teórico, os custos chegam a ser similares aos livres. Já, em relação aos *Software Open Source*, Ferraz (2002) relata que o preço para a obtenção de um *Software* livre é quase zero, sendo baixo também o custo para testes, ao contrário das versões proprietárias.

Logo, ainda segundo Ferraz (2002), é muito mais cômodo investir em treinamento do que em licenças. Ou seja, é mais proveitoso aprender a manusear um aplicativo livre do que gastar vultosas quantias em dinheiro para adquirir *Software* proprietários e estar atrelado ao suporte dado a mercê destas empresas.

Então, quem opta por usar *Software* livre garante independência de fornecedor pelo fato do código-fonte estar disponível. Desta forma, se por alguma eventualidade, houver uma troca de empresa fornecedora, isto não recairá em problema devido seu código ser conhecido. Portanto, um negócio não fica refém de um fornecedor.

Tendo em vista os argumentos discutidos, vê-se o *Software* livre como mais do que uma escolha economicamente viável, chegando a ir além do financeiro. Assim, Silveira (2003), diz que trata-se de um movimento que se permeia no compartilhamento do saber e no solidarismo coletivo como foco o conhecimento, onde estes laços vão além das conexões à rede mundial de computadores.

2.2 Definição e Origem de Banco de Dados

Neste mundo globalizado e competitivo, se faz necessário buscar formas de se otimizar rotinas e se inovar os produtos e/ou serviços prestados por qualquer entidade. E, para isso, o gerenciamento de qualquer negócio deve primar pela busca e uso racional da informação.

> A informação deve ser utilizada de maneira estratégica, para que se possa atender e atingir muito rapidamente os objetivos, metas e desafios traçados pela alta gerência de um negócio. Esta velocidade de mudança faz com que qualquer negócio possa aproveitar uma oportunidade de competição de mercado, sabendo que as informações estratégicas, táticas e operacionais estão disponíveis a qualquer momento para tomada de decisões (MACHADO, 1996, p. 12).

Porém, antes de tê-las, é preciso coletar os dados inerentes ao negócio em questão. Daí é gerado um conflito entre as definições de ambas as entidades, no caso dado e informação. O dado é uma representação; um registro de informação e este pode ser registrado fisicamente. Já a informação, acrescenta algo ao conhecimento da realidade a ser analisada.

De acordo com Heuser (1998), ao conjunto destes arquivos integrados que atendem a um conjunto de sistemas dá-se o nome de banco de dados (BD). Assim, o tamanho de um empreendimento é diretamente proporcional a gama de informações que este deve gerir. Logo, torna-se inviável tratá-las manualmente e em um tempo hábil.

Com o advento da computação, surge para guardar, gerir e processar dados, os sistemas de banco de dados, é o que diz Heuser (1998). Estes sistemas, mencionados por Silberschartz et.al. (2006), são projetados para gerir grandes volumes de informações.

Defini-se então, sistemas de banco de dados (SGDB) como:

> (...) 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 usuários e aplicações (ELMASRI, 2005, p.4).

Também, consoante com a definição anterior, tem-se:

> (...) um sistema de banco de dados é basicamente um sistema computadorizado de manutenção de registros; em outras palavras, é um sistema computadorizado cuja finalidade geral é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando as solicitar (DATE, 2003, p.6).

A figura 1 ilustra conceitualmente a estrutura de um sistema de banco de dados.

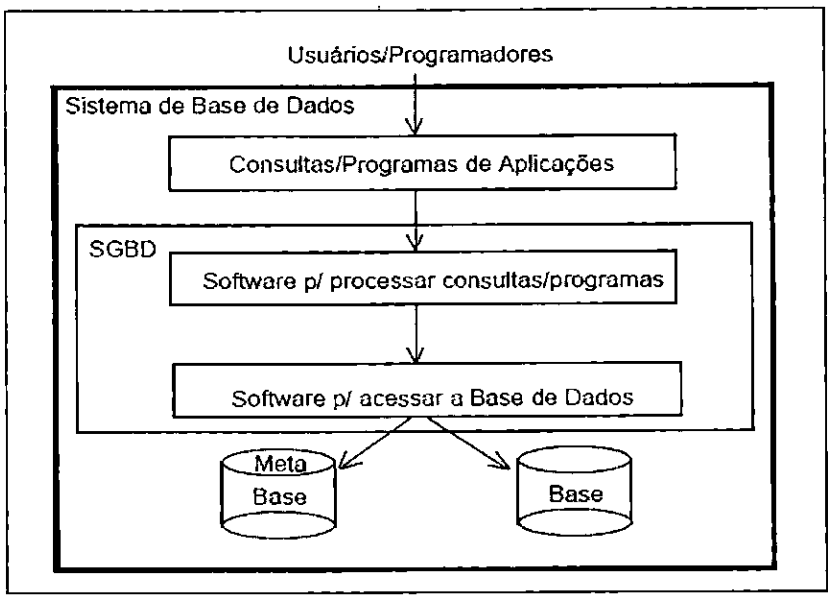


Figura 1 - Sistema de Banco de Dados

Fonte: (TAKAI ET AL, 2005, p. 15)

Implementado-se este modelo em organizações, de acordo com Silberschartz et.al. (2006), conclui-se que a informação tem determinado o desenvolvimento de técnicas mais eficazes de administração desses dados.

Antes dos SGDBs, existiram artifícios para se agilizar o tratamento dos dados que foram desenvolvidos ao longo dos tempos:

> Na verdade, a automação das tarefas de processamento de dados já existia antes mesmo dos computadores. Cartões perfurados, inventados por Herman Hollerith, foram usados no início do século XX para registrar dados do censo nos Estados Unidos, e sistemas mecânicos foram usados para processar os cartões e tabular os resultados. Mais tarde, os cartões perfurados passaram a ser amplamente usados como um meio de inserir dados em computadores (SILBERSCHARTZ ET.AL., 2006, p. 19).

Neste contexto, para o desenvolvimento de novos meios de se armazenar e tratar os dados foi a partir da década de 50. Abaixo, cronologicamente, segue de acordo com Silberschartz et.al. (2006) a evolução desta busca:

- **Década de 50 e 60**: fitas magnéticas foram desenvolvidas para armazenamento de dados e podiam ser lidas apenas sequencialmente, e os tamanhos de dados eram muito maiores do que a memória principal; portanto, os programas de processamento de dados eram obrigados a processar dados em uma ordem específica, lendo e mesclando dados de fitas e *Decks* de cartão.
- **Década de 70:** trocou-se as fitas magnéticas pelos discos rígidos, onde qualquer

local nele era indiferente, uma vez que qualquer local no disco podia ser acessado em apenas dezenas de milissegundos. Os dados, portanto, estavam livres da tirania da sequencialidade.

- **Final da década de 70**: um documento revolucionário de Codd [1970] definiu o modelo relacional e os métodos procedurais de consultar dados no modelo relacional, dando origem aos bancos de dados relacionais.
- **Década de 80**: os bancos de dados relacionais não podiam se igualar aos bancos de dados em rede e hierárquicos existentes. Isso mudou com o *System R*, um projeto inovador da IBM *Research* que desenvolvia técnicas para a construção de um sistema de banco de dados relacional eficiente.
- **Início da década de 90**: a linguagem SQL (Strutured English Query Language) foi projetada principalmente para aplicações de suporte a tomada de decisão, que são concentradas na consulta, enquanto a base dos dados na década de 1980 eram aplicações de processamento de transação, que são concentradas na atualização.
- **Final da década de 90**: o evento mais importante foi o crescimento explosivo da *Wolrd Wide Web*. Os bancos de dados eram utilizados muito mais extensivamente do que já mais foram.
- **Início dos anos 2000:** surge o XML (*Extensible Markup Language*) e a linguagem de consulta associada, como uma nova tecnologia de banco de dados. Nesse período de tempo, também foi observado o crescimento das técnicas de "computação autonômica/auto-administração" para reduzir os esforços de administração de sistema.

Entretanto, nos pioneiros dos SGDBs, os dados eram tratados computacionalmente por meio de sistemas de processamento de arquivos. Segundo Silberschartz et.al. (2006), se armazenava as informações no computador por meio de sistemas de arquivos permanentes. E, para cada transação que viesse a tratar esse conjunto de dados, seria necessário um apanhado de programas. Logo, mais *Software* e arquivos seriam demandados com o passar do tempo:

> Os sistemas de processamento de arquivos típico que acabamos de descrever pode ser aceito pelos sistemas operacionais convencionais. Registros permanentes são armazenados em vários arquivos e diversos programas de aplicação são escritos para extrair e gravar registros nos arquivos apropriados. Antes do advento dos SGDBs, as organizações usavam esses sistemas para armazenar as informações (SILBERSCHARTZ ET.AL., 2006, p. 2).

Ainda consoante com o teórico mencionado, esta metodologia recai em desvantagens que pode vir a deturpar a veracidade da informação, dentre elas podem-se citar:

- Inconsistência e redundância de dados;
- Dificuldade de acesso aos dados;
- Isolamento de dados;
- Problemas de Integridade;
- Problemas de Atomicidade;
- Anomalias no acesso concorrente;
- Problemas de segurança.

Takai et. al. (2005) coloca que o intuito dos sistemas gerenciadores de banco de dados é justamente minimizar estes e outros problemas. Somente não se aconselha o uso de SGDBs, nos seguintes casos:

- Bases de dados e aplicações simples, bem definidas sem expectativa de mudança;
- Existem restrições de tempo que não podem ser satisfeitas em SGBDs;
- Não há necessidade de acesso multiusuário.

Porém, há uma série de cuidados que devem ser levados em conta para a escolha e implantação de um sistema gerenciador de banco de dados. O primeiro aspecto a considerar-se é a escolha de um SGDB e o conjunto de ferramentas que compõem o projeto a se implementar. Segundo Mecenas (2006), esta escolha, como a de qualquer outra na área de tecnologia, requer muita avaliação. Dessa maneira:

> (...) a escolha de todas as ferramentas de um projeto de sistema (linguagens de programação, gerenciadores de banco de dados, compiladores e protocolos de comunicação) é essencial para que ele seja portável e tenha uma grande vida útil (NETO, 2007, p. 26).

Outro fator muito importante já até predito é a sua portabilidade. Santos (2007) diz que o ideal é que a base de dados possa acompanhar as mudanças e atualizações na área de tecnologia da informação (TI) permitindo transições bem sucedidas. Silberschartz et.al. (2006) enfatiza a ideia de que complementando as qualidades que um SGDB deve possuir, tem-se a segurança tanto dos dados e de acesso a estes, como ponto crítico a ser gerido. E, se a base é compartilhada entre vários usuários, o sistema deve evitar a concorrência de respostas inconsistentes, ainda referente a este teórico.

Corroborando com estes aspectos discutidos, Rebecca (2002, p. 39) ressalta:

> Um dos aspectos mais importantes em gerenciar um banco de dados é garantir a segurança dos dados. Você deve assegurar-se de que todos que precisam realmente acessar os dados possam fazê-lo, mas que o acesso seja adequado às reais necessidades e que ninguém tenha acesso não apropriado.

Em vista de seu conceito e histórico debatido, o sistema gerenciador de banco de dados nos remete a vários benefícios. Dentre estes, o mais palpável é justamente a visão abstrata dos dados, ou seja, o SGDB oculta detalhes das transações, armazenamento e manutenção da base.

Logo se vê o quão importante é o uso de sistemas gerenciadores de dados, tanto no que diz respeito ao armazenamento e segurança das informações, mas o rápido acesso a elas. Em suma, dependendo do porte de um negócio, o não uso desta ferramenta para gerir e perseverar seus dados pode contribuir à poda de seu crescimento e ser engolido pela forte maré do mercado.

2.2.1 Alguns Paradigmas de Modelagem

Com o evoluir da tecnologia, desde os simples sistemas de arquivos, buscou-se maneiras que melhor representassem os dados. Esses são ditos modelos de dados que descrevem a estrutura das informações contidas nos bancos de dados. Dentre as metodologias existentes, alguns foram ou ainda são muito utilizados.

> O primeiro Sistema Gerenciador de Banco de Dados (SGBD) comercial surgiu no final de 1960 com base nos primitivos sistemas de arquivos disponíveis na época, os quais não controlavam o acesso concorrente por vários usuários ou processos. Os SGBDs evoluíram desses sistemas de arquivos de armazenamento em disco, criando novas estruturas de dados com o objetivo de armazenar informações. Com o tempo, os SGBD's passaram a utilizar diferentes formas de representação, ou modelos de dados, para descrever a estrutura das informações contidas em seus bancos de dados. Atualmente, os seguintes modelos de dados são normalmente utilizados pelos SGBD's: modelo hierárquico, modelo em redes, modelo relacional (amplamente usado) e o modelo orientado a objetos (TAKAI et. al, 2005, p. 6).

Cronologicamente, o primeiro modelo de dados concebido, segundo Galante et. al.(2007), foi o hierárquico. Ainda é dito por esse teórico que sua estrutura é do tipo árvore e é formado por registros e *Links*, onde cada registro é uma coleção de dados e o *Link* é uma

associação entre dois registros, e os registros que precedem outros são ditos pais, o restante é chamado de registros filhos.

De acordo com Takai et. al. (2005), a criação deste paradigma só foi possível com a consolidação dos discos de armazenamento endereçáveis, na qual se possibilita explorar a estrutura de armazenamento físico. Isto nos propicia representar a informação hierarquicamente, sendo o sistema comercial mais divulgado no modelo hierárquico foi o *Information Management System* da IBM *Corp* (IMS).

O funcionamento do modelo hierárquico, consoante com Galante et. al. (2007, p. 4), segue como:

> Cada registro tem suas ligações, o registro pai pode ter vários filhos (1:N), o registro filho só pode ter um pai. Caso um determinado registro filho tenha a necessidade de ter dois pais é necessário replicar o registro filho.

De acordo com Takai et. al. (2005), os dados são percorridos segundo uma sequencia hierárquica com uma navegação do topo da árvore as suas folhas, sendo da esquerda para a direita. É dito ainda por este autor que em grande parte do controle de consistência e restrição era exercido pelas aplicações, sendo necessário criar *Software* na ordem para acessar o banco de dados.

No entanto, este modelo possui desvantagens, na qual a primeira se mostra na replicação. Segundo Galante et. al. (2007), isso pode recair em inconsistência dos dados quando houver atualização, fora o desperdício de espaço de armazenamento. O mesmo autor diz também:

> (...) desvantagens como: complexidade dos diagramas de estrutura de árvores, limitações das ligações entre registros - ex.: um filho só pode ter um pai, a navegação é feita através de ponteiros, complexidade na hora das consultas, ausência de facilidades de consultas declarativas, só trabalha com dados primitivos (GALANTE ET. AL, 2007, p. 4)

A figura 2 representa o paradigma hierárquico de um banco de dados.

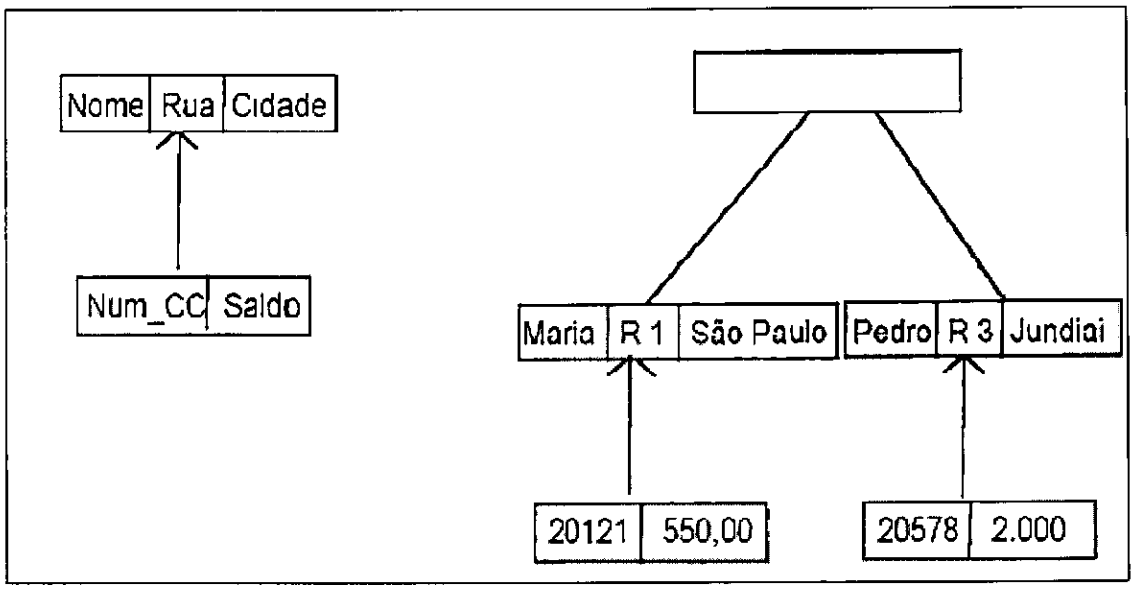


Figura 2 - Modelo de um Banco de Dados Hierárquico

Fonte: (TAKAI ET. AL., 2005, p. 7)

Assim, como extensão do modelo falado na figura 2, surge a metodologia em rede, cujo elimina a hierarquia, ou seja, permite que um registro possa ter várias associações. Neste paradigma, os registros são organizados em grafos, definindo um só tipo de relacionamento de 1 para N entre dois tipos de registros: proprietário e membro.

A definição e o funcionamento deste molde de dados é colocada da seguinte forma:

> O modelo em rede surgiu para suprir algumas deficiências do modelo hierárquico. O conceito de hierarquia foi abolido nesse novo modelo, o que permitiu que um registro estivesse envolvido em várias associações, ou seja, esse modelo aceitar relacionamentos (N:M), um filho pode ter mais de um pai, ele só trabalha com dados primitivos. Uma outra característica que difere esse modelo do hierárquico é que ele utiliza grafos ao invés de árvores (GALANTE ET. AL., 2007, p. 5).

No modelo em rede qualquer nó pode ser acessado sem precisar passar pelo nó raiz. Outro fato relevante a ser colocado a cerca deste modelo é sua segurança, na qual parte ou área do banco pode ser bloqueada para evitar acessos simultâneos quando preciso. Ainda a sintaxe de segurança associa uma senha a cada objeto incluso neste esquema (TAKAI ET. AL., 2005).

O gerenciador DBTG (*Data Base Task Group*) da CODASYL (*Committee on Data Systems and Languages*) criou um padrão para este tipo de banco (rede), tanto na linguagem de definição e manipulação de dados. Para Galante et. al. (2007), o gerenciador mais

conhecido nesta abordagem é o IDMS da *Computer Associates*, sendo formado por registros e *Links*.

A figura 3 denota o molde de um banco de dados em rede.

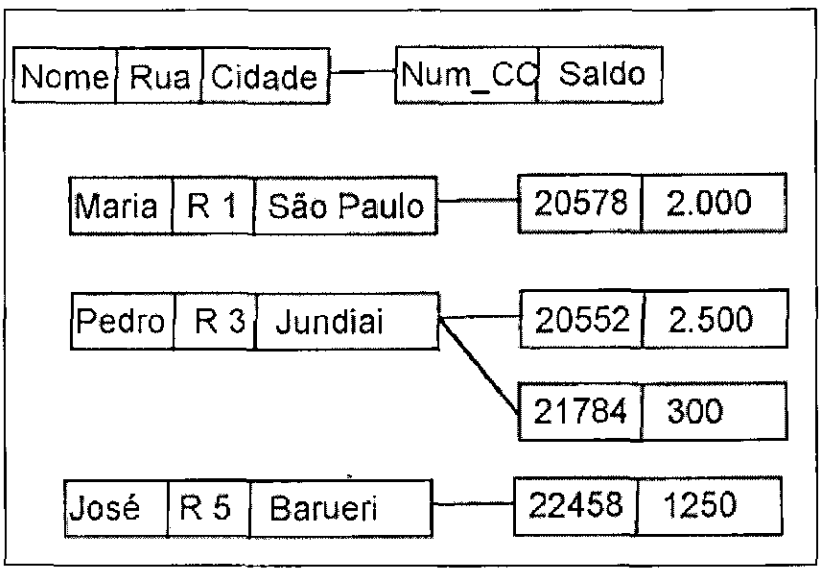


Figura 3 - Modelo de um Banco de Dados em Rede

Fonte: (TAKAI ET. AL., 2005, p. 7)

Entretanto, ambos modelos (hierárquico e em rede), são orientados a registros. Logo, qualquer transação (inserção, consulta, alteração ou remoção) é feito sequencialmente.

Silberschatz et. al. (2006) esclarece que entre os modelos de dados destaca-se o de entidade relacional (E-R), já até mencionado anteriormente, no qual tem por base a percepção de que a realidade é formada por um conjunto de objetos chamados de entidades e por uma gama de relacionamentos entre esses objetos. Muitas das ferramentas de projeto foram concebidas para este modelo, o que vem a influenciar na sua escolha como modelo de dados.

Estes objetos que desejamos conhecer e modelar para um sistema classificou-se em dois grupos: Entidades e Relacionamentos. Segundo Machado, (1996), entidade um objeto que existe no mundo real com uma identificação própria, já relacionamento, para Silberschartz et.al. (2006), seria uma associação entre uma ou várias entidades, sendo que este ainda diz que existem três noções básicas empregadas pelo modelo E-R: conjunto de entidades, conjunto de relacionamentos e os atributos.

Uma entidade pode ser concreta (um livro, por exemplo) ou abstrata (uma compra ou um empréstimo por exemplo), e esta é representada por um conjunto de atributos, os quais são as características de cada membro de um rol de entidades.

Os atributos podem ser de acordo com Silberschartz et.al. (2006), classificados como:

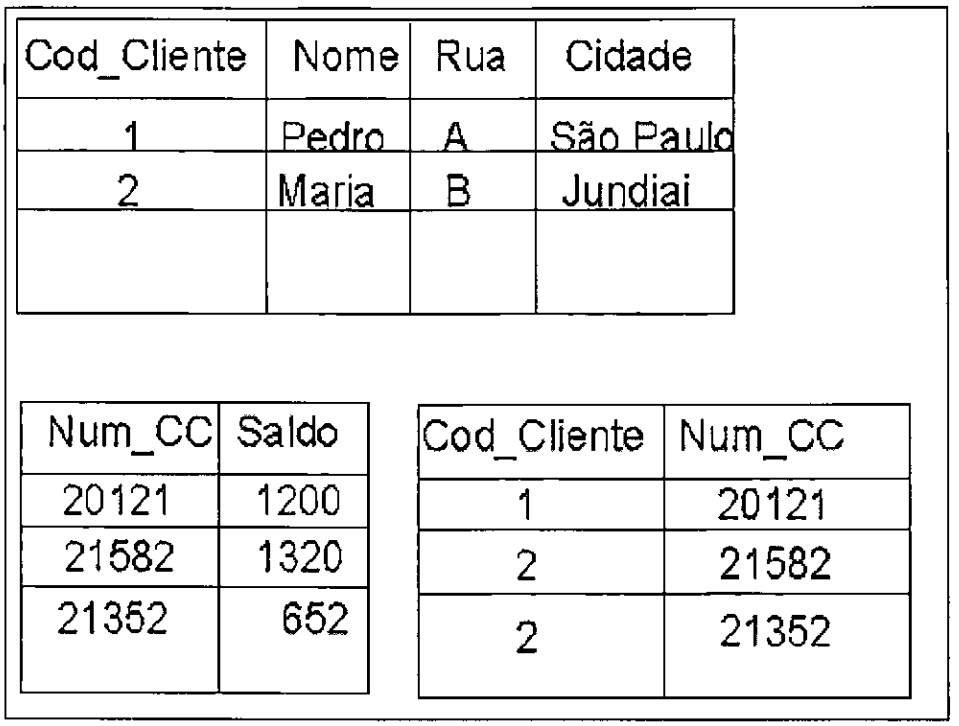
- **Simples ou compostos**: o primeiro não é dividido em partes, o contrário do segundo;
- **Monovalorados** ou multivalorados: o primeiro acontece quando o atributo refere-se a apenas uma entidade, o segundo pode referir-se a várias;
- **Nulo**: quando a entidade não possui valor para determinado valor;
- **Derivado**: o valor desse tipo de atributo pode ser derivado de outros atributos ou entidades a ele relacionados.

Um banco de dados representado neste paradigma pode ser representado por uma coleção de tabelas e para cada conjunto de entidades e para cada conjunto de relacionamentos do banco de dados há uma única tabela. Esta conversão é a base para a construção de um banco de dados relacional. Porém, este meio possui limitações que dentre as quais se destaca a incapacidade de abstrair relacionamentos entre relacionamento:

> Uma limitação do modelo E-R é sua incapacidade de expressar relacionamentos entre relacionamentos. A solução é o uso de agregação; uma abstração por meio de da qual os conjuntos de relacionamentos são tratados como conjunto de entidades de nível superior. Assim, um conjunto de relacionamentos e seu conjunto de entidades associadas podem ser vistos como um conjunto de entidades de alto nível que é tratado da mesma maneira que qualquer outro conjunto de entidades (SILBERSCHARTZ et. al., 2006, p.21).

Para complementar, o autor ainda diz que devido os vastos benefícios trazidos pelo modelo E-R, o projetista tem o privilégio de escolher a melhor representação do que se está modelando, ou seja, a modelagem não fica limitada em relação à infraestrutura física e, como já foi mencionado, este trabalha muito bem em sintonia com a linguagem SQL.

A figura 4 mostra o modelo entidade relacional de banco de dados.



Cod_Cliente	Nome	Rua	Cidade
1	Pedro	A	São Paulo
2	Maria	B	Jundiai

Num_CC	Saldo
20121	1200
21582	1320
21352	652

Cod_Cliente	Num_CC
1	20121
2	21582
2	21352

Figura 4 - Modelo Entidade Relacional de Banco de Dados.

Fonte: (Takai et. al., 2005, p. 8)

Outro paradigma de programação que merece destaque é o da OO. Ricarte (1998 p.12) define esta metodologia como:

> (...) desenvolvimento de um sistema em um processo de informação através de transformações de seu estado. A substância do processo é organizada como componentes do sistema, denominados objetos. Uma propriedade mensurável da substância é uma propriedade de um objeto. Transformações de estado são consideradas como ações por objetos.

Incorporando esta temática, a orientação a objetos surgiu como uma alternativa para a modelagem de banco de dados, podendo acrescentar muitos dos conceitos propostos pelas outras abordagens alternativas ao modelo relacional de dados.

De acordo com Galante et. al. (2007), o padrão (orientado a objetos) OO cresceu muito, principalmente no que diz respeito a integração entre o mundo relacional e orientado a objetos, onde ferramentas de mapeamento objeto / relacional (O/R) foram desenvolvidas para este fim. Neste cenário, este autor relata:

> O paradigma da linguagem orientado a objeto vem ganhando ao longo do tempo muita importância. Nos últimos anos começou a ser bastante empregada e mostra resultados satisfatórios. Passou a ser a linguagem mais utilizada para construção de software. A LOO nada mais é que uma extensão da linguagem modular, e teve seu início na década de 70, com a linguagem

> SIMULA (Simula Language) que foi desenvolvida na Noruega (Galante et. al., 2007, p.2).

No entanto, para que um modelo seja dito OO, ele deve obedecer certos critérios, perante Galante et. al. (2007), dentre os quais destacam-se: abstração, encapsulamento, herança, e polimorfismo. Cada um desses pré-requisitos são definidos:

- **Abstração:** é a capacidade de modelar coisas do mundo real, que o programador esteja tentando resolver por meio de classes. Uma classe descreve um grupo de objetos com propriedades (atributos), similares comportamentos (operações), relacionamentos comuns com outros objetos e uma semântica comum.
- **Encapsulamento:** o encapsulamento tem como principal benefício evitar que interferências externas interfiram na manipulação dos dados. Também protege o mundo externo de alterações na implementação da classe. Com esse princípio, toda ou qualquer transação feita com esses dados só pode ser feita através de procedimentos especificados dentro desse objeto, pelo envio de mensagens.
- **Herança**: é o mecanismo que permite a reutilização de códigos. É uma relação entre duas ou mais classes onde existe a Super Classe e a Sub Classe.
- **Polimorfismo**: O termo polimorfismo quer dizer etimologicamente várias formas. Muitas pessoas acham que sobrecarga e polimorfismo são a mesma coisa, o que não é verdade. Quando um método declarado em uma super classe é herdado para as sub classes e pode ser utilizado em todas as sub classes tendo o mesmo nome e pode atuar em cada sub classe de forma diferente, ou seja, Polimorfismo. Quando um método é herdado e reescrito para atender necessidades específicas de cada sub classe, se ele ganhar uma nova forma de implementação na sub classe, ou seja, reescrever um código que já tenha sido declarado e modifica-lo para atender a necessidade de cada classe , isto é caracterizado como Sobrecarga.

Dada a definição do referido paradigma e suas prerrogativas, formaliza-se banco de dados OO como um meio de se armazenar as informações na forma de objetos, e só podendo manipulá-los através de métodos pela classe a qûe este pertença.

Na utilização desta metodologia é possível retirar a fase de mapeamento objeto relacional e usar dos benefícios da orientação a objetos sem se prender ao banco de dados, construindo assim modelos mais ricos. E objetos são modulares, mudanças podem ser feitas internamente, sem afetar outras partes do programa. Outro benefício trazido por esta modalidade é justamente uma maior segurança, devido a ausência de um DBA (*Database*

Administrator) o que o torna uma solução mais viável economicamente. Ainda permite salvar objetos grandes e depois obter a recuperação facilmente desses grandes objetos como textos longos, imagens etc.

A figura 5 exibe o esquema orientado a objetos de banco de dados:

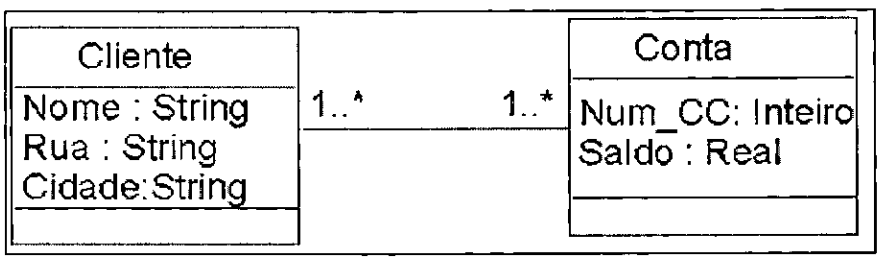


Figura 5 - Modelo Orientado a Objetos de Banco de Dados

Fonte: (TAKAI et al, 2005, p. 9)

Mais um ponto forte dos Sistemas de Banco de Dados Orientado a Objetos (SGDBOO) é ocultar informação e tipos abstratos de dados, sendo que é muito complicado aplicar esse modelo na prática. Este modelo ganhou espaço em várias áreas como banco de dados espaciais, telecomunicações, e nas áreas científicas como física de alta energia e biologia molecular.

Galante el. al. (2007) retrata que para a padronização de uma linguagem de consulta compatível para esse modelo, a ODMG (*Object Data Management Group*) criou a OQL (*Object Query Language)* tendo as sintaxes muito parecidas com o SQL.

Entretanto, um SGDBOO possui pontos críticos no que diz respeito a sua escolha para o gerenciamento dos dados. O fato de ser uma tecnologia relativamente nova, o torna uma alternativa arriscada, tendo em vista, soluções mais difundidas no mercado. Além disso, SGBDOO não é capaz de processar diretamente condições de seleções e outras operações desses objetos. É necessário que esses dados sejam passados para o BD para que ele possa saber tratar os objetos corretamente.

Neste contexto, Abreu (2006) relata que, diante do evoluir da computação no que diz respeito a hardware e *Software*, e os mesmos estarem mais acessíveis financeiramente, vê-se como uma alternativa para se gerir os dados de uma corporação distribuir e ou replicar as bases de dados para se ganhar velocidade de processamento e segurança. O mesmo autor afirma que a fragmentação dos dados permite que cada um pode ser tratado como uma unidade isolada, sendo possível execuções paralelas e concorrente das consultas, visto que os dados estão disponíveis na rede.

Esta metodologia para modelagem de dados é chamada de banco de dados distribuído, pois, segundo (Ozsu e Valduriez 1999), consiste em numa coleção de múltiplos bancos de dados logicamente interrelacionados sobre uma rede de computadores. Já Shibayama (2004) diz que um sistema de banco de dados distribuídos é definido como uma série de elementos de processamento interligados por um sistema de rede de computadores e que cooperam na realização de tarefas específicas. Este veio a surgir como uma intercalação de duas tecnologias: tecnologia de banco de dados e tecnologia de comunicação de dados e de rede (MATTOSO, 1998). Corroborando:

> Com as novas tecnologias de comunicação de dados, além da necessidade da descentralização, é possível se trabalhar com bancos de dados não mais centralizados em um servidor, mas sim distribuídos em vários servidores, que podem estar na mesma rede, assim como podem estar em países diferentes, compartilhando informações de uma maneira transparente para os usuários (SHIBAYAMA, 2004, p. 12).

A seguir, tem-se a ilustração do modelo distribuído de banco de dados:

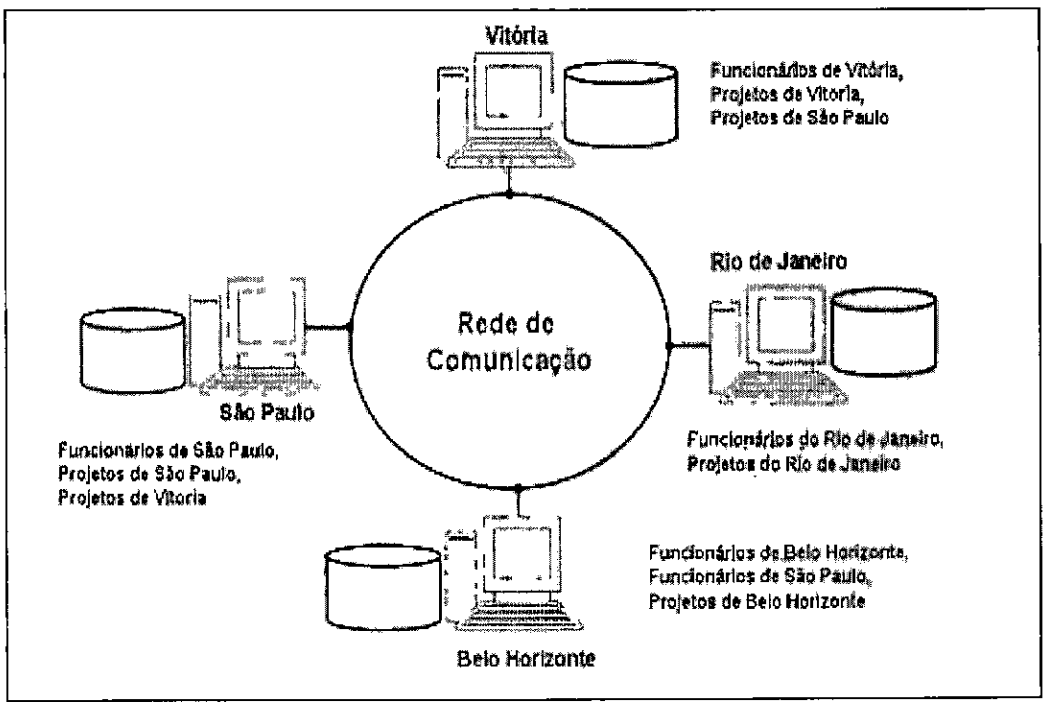


Figura 6 - Modelo Distribuído de Banco de Dados

Fonte: (MACAFERI, 2007, p. 7)

O principal objetivo deste modelo, diz Shibayama (2004), é repartir uma massa de dados ingerenciável em partes menores para serem processadas. Esse modelo funciona dá seguinte forma: a base de dados é armazenada em vários computadores conectados por

diversos meios, em que cada qual pode participar na execução de transações que acessam dados em um ou diversos nós. Ainda é dito que a principal diferença entre o modelo centralizado e o distribuído é que no primeiro os dados são acessados em um único lugar, já no segundo os dados estão armazenados em diversos lugares.

As principais vantagens de seu uso são elas: Independência de dados; Confiabilidade em transações distribuídas; Desempenho otimizado; Expansão de sistema mais fácil; Transparência de dados; Transparência de rede; Transparência de replicação; e Transparência de fragmentação.

Desta forma, um dos motivos para distribuir um banco de dados é que, um banco de dados distribuído além de ser mais confiável, tem a capacidade de resolver da melhor maneira os grandes e complexos problemas com que nos defrontamos hoje, usando uma variação da regra de dividir e conquistar, ou seja, problemas complicados podem ser resolvidos simplesmente dividindo-os em fragmentos menores e atribuindo-os a grupos de *Software* distintos, que funcionem em computadores diferentes e produzam um sistema que atue sobre vários elementos de processamento, mas que possam colaborar de forma eficaz para a execução de uma tarefa em comum (SHIBAYAMA, 2004, p.12).

2.2.2 Projeto de Banco de Dados

Para que se possa criar algo e este venha a ter um bom desempenho, antes de mais nada deve-se planejar sua construção. A construção de um planejamento para se atingir um objetivo pode definir o sucesso desta empreita. Não diferente, no campo computacional, mais especificamente no que diz respeito à banco de dados, é forçoso que se crie e siga um projeto para seu desenvolvimento.

Perante Silberscharte et. al. (2006), o primeiro passo para o projeto de um banco de dados é captar as necessidades por completo dos usuários do mesmo e traduzir estas especificações por meio de tabelas. Nessa especificação funcional, os agentes que manipularão o sistema devem descrever os tipos de operações que vão ser realizadas.

Como foi discutido anteriormente, o banco de dados tem como essência salvaguardar as informações de uma realidade em específico. E, para abstrair os requisitos a serem levantados, é necessária uma modelagem que descreva a rotina desse ambiente, para que possam ser registradas as informações, é o que diz Machado (1996). Segundo este, o objetivo

da modelagem de dados é transmitir e apresentar uma representação única, não redundante e resumida, dos dados de uma aplicação.

Na escolha da metodologia a ser seguida para realizar a modelagem, de acordo com Silberscharte et. al. (2006), o foco é descrever os dados e suas relações e não se preocupar nesta etapa com detalhes do armazenamento físico. E, após a escolha do modelo a ser seguido, de acordo com Machado (1996), o projeto envolve basicamente três níveis: conceitual, lógico e físico.

A figura 7 mostra as etapas de um projeto de banco de dados:

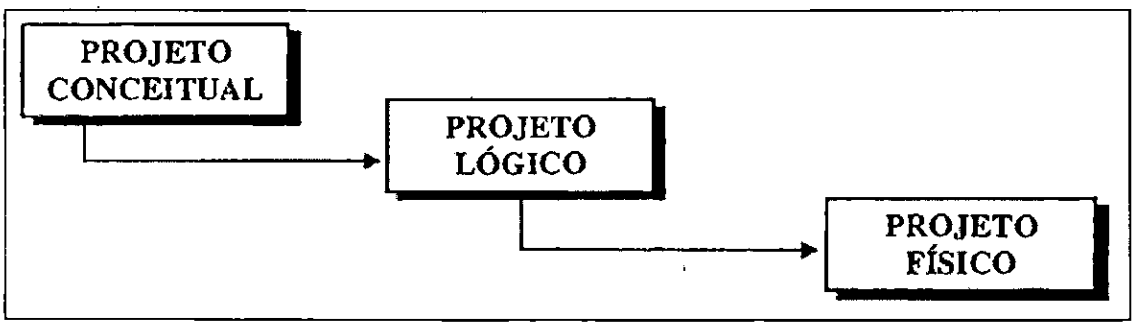


Figura 7 - Esquema de um Projeto de Banco de Dados.

Fonte: (MACHADO, 1996, p. 26)

A respeito do primeiro passo, tem-se a seguinte descrição:

> Representa e/ou descreve a realidade do ambiente do problema, constituindo-se em uma visão global dos principais dados e relacionamentos (estruturas de informação), independente das restrições de implementação (SILBERSCHARTZ et.al., 2006, p. 23).

Machado (1996) diz que nesta fase, o analista deve se concentrar nas rotinas mais importantes que ocorrem na realidade com o intuito de automatizar as necessidades de informação a ser atendidas. É dito ainda que o modelo conceitual não exibe os tópicos ligados à abordagem do banco de dados que irá ser utilizado e muito menos no que diz respeito ao acesso e as rotinas implementadas por um SGDB especifico.

Consoante com Silberschartz et. al. (2006), o processo de mutação do projeto abstrato para a implementação do banco de dados em si realiza-se nas fases subsequentes, as quais são: lógico e físico.

Findando a etapa conceitual, inicia-se o modelo lógico:

> O Modelo Lógico descreve as estruturas que estarão contidas no banco de dados, de acordo com as possibilidades permitidas pela abordagem, mas sem considerar, ainda, nenhuma característica específica de um Sistema Gerenciador de Dados (SGDB), resultando em um esquema lógico de dados sob a ótica de uma das abordagens citadas (MACHADO, 1996, p. 24).

Concluída a parte lógica tem-se inicio à física, onde ainda é dito pelo autor anteriormente mencionado, que é nesta modalidade a qual descreve as estruturas físicas de armazenamento dos dados, e deve-se planejar este ato de maneira que atenda aos requisitos levantados e economize recursos computacionais.

Com isso, chega-se à construção de um banco de dados ideal para um caso em específico. No entanto, existem situações nas quais modelos são quase equivalentes. Para isso, existem os chamados meta-modelos, em que a partir dele possa construir vários outras modelagens. Ou seja, o meta-modelo, é um tipo genérico do qual se faz modificações para que venha a corroborar com os requisitos estabelecidos. Em suma, "Devido a esta não similaridade entre ambientes de mesma natureza, será sempre necessária a criação de um modelo específico para cada nova realidade observada". Isto é, constrói-se um modelo genérico chamado meta-modelo. Este vem a ser utilizado para se adaptar às características de qualquer realidade, simplificando um trabalho e ilustrando-o graficamente. (MACHADO, 1996, p. 26).

2.3 SQL

Para Neto (2007), inicialmente, cada SGDB possuía suas especificidades no que diz respeito em como guardar e recuperar os dados. E dentre os modelos insurgentes, por volta de 1975 destaca-se o molde relacional de banco de dados. Em busca de um padrão para manipular as informações, o ANSI em meados da década de 80 inicia estudos para o desenvolvimento de uma linguagem que corresponda a estas expectativas.

Surge então o SQL, acrônimo de *Structured Query Language* (Linguagem Estruturada de Pesquisa). Suas bases veem do modelo relacional de Codd (1970), onde sua primeira versão foi chamada de SEQUEL ("*Structured English Query Language*"), desenvolvida por D. D. CHAMBERLIN em 1974, nos laboratórios da IBM. (Machado, 1996)

E, de acordo com Neto (2007), esta linguagem serviria para acessar dados em bases relacionais e esta foi formalizada por volta dos anos 80, na mesma empresa referida, IBM.

Logo esta nova forma de consulta e manipulação dos dados alcançou sucesso e muitos dos SGDBs existentes incluíram-na em seus ambientes de desenvolvimento. Ainda assim, o ANSI e o ISO juntaram esforços e trabalharam num conjunto maior de padronização para o SQL, criando o chamado SQL ANSI-92.

Para Machado (1996), a linguagem SQL possui vários enfoques, dentre os quais se destacam a seguir:

- Linguagem interativa de consulta (*Query* AdHoc): possibilidade de montar consultas poderosas sem a necessidade de criação de programas, utilizando ferramentas para geração de relatórios;
- Linguagem de programação para acesso a banco de dados: códigos SQL embutidos nas aplicações que acessam a base de dados;
- Linguagem de administração de banco de dados: o responsável pela administração do banco de dados pode utilizar comandos SQL para realizar sua tarefas;
- Linguagem cliente\servidor: programas clientes que se comuniquem com o servidor podem requerer consultas à base via códigos SQL;
- Linguagem para banco de dados distribuídos: SQL auxilia na distribuição dos dados por meio de vários nós conectados ao sistema computacional, chegando a ajudar na comunicação com outros sistemas;
- Definição de dados (DDL): permite ao usuário a definição da estrutura e organização dos dados armazenados, e as relações que existem entre eles;
- Maniulação de dados (DML): permite ao usuário ou *Software* a inclusão, remoção, seleção ou atualização de dados pré-armazenados no banco;
- Controle de acesso: protege os dados de manipulações não autorizadas;
- Compartilhamento de dados: administra a concorrência de acesso à base por vários usuários sem causar interferência entre eles;
- Integridade dos dados: ajuda no processo de definição da integridade dos dados, seja por inconsistências ou falhas do sistema computacional.

Entretanto, cada empresa que incorporou o SQL em seu SGDB desenvolveu extensões, que são um conjunto de características e funções proprietárias além do padrão SQL ANSI-92. Isso recai em um possível problema de incompatibilidade, em que o referido autor afirma que:

> (...) os desenvolvedores terão muitas dificuldades, usando essas facilidades, principalmente ligadas à incompatibilidade nos códigos-fonte, a não-

> implementação de algumas facilidades por um ou outro SGBD a usar, tanto no uso de SQL quanto nas Linguagens de Programação (NETO, 2007, p. 25).

Machado (1996, p. 199), cita as vantagens e desvantagens desta linguagem:

Tabela 1- Vantagens e desvantagens da linguagem SQL

Vantagens	Desvantagens
Independência de Fabricante	Padronização acarreta a estaticidade
Portabilidade	Falta de algumas funções
Inglês estruturado de alto nível	Discordância com linguagens hospedeiras
Consulta interativa	Falta de ortogonalidade nas expressões, funções embutidas, variáveis indicadoras, referência a dados concorrentes, constante NULL, conjuntos vazios e etc.
Múltiplas visões dos dados	Não dá suporte a alguns aspectos do modelo relacional (atribuição de relação, join explícito, domínios, etc)
Definição dinâmica dos dados	

No entanto, a linguagem SQL veio com o intuito de facilitar a manipulação dos dados e deixar o foco do programador na regra de negócio a ser abstraída em um sistema computacional, onde as informações são geridas e armazenadas por um SGDB e recuperadas via comandos SQL. Apesar de suas deficiências, esta facilita, para os usuários e analistas, o manuseio dos dados armazenados em uma base relacional (Machado, 1996, p. 200)

3 GERENCIADORES DE BANCO DE DADOS LIVRES FIREBIRD, MYSQL E POSTGRESQL

3.1 FIREBIRD

Este sistema gerenciador de banco de dados surgiu a partir da abertura do código-fonte de outro, chamado InterBase 6, que era licenciado pela empresa, denominada Borland[1], ou seja, o *Firebird* que é um SGDB livre nasceu de um *Software* proprietário.

O *Firebird* é definido, segundo Cantu (2005), como um banco de dados cliente\servidor livre, compatível com o padrão SQL-ANSI 92, rodando em mais de 10 plataformas (sistemas operacionais) e possui duas versões: Classic e SuperServer.

Entretanto, perante o autor acima enunciado, por mais que a Borland tenha aberto o código do InterBase 6, esta empresa anunciou que continuaria desenvolvendo uma versão nova e comercial do InterBase em que seu código seria fechado. Mas o *Firebird* ganhou novas versões e vários usuários que migraram do InterBase 6 com a garantia de que sendo livre, o Firebid está em constante aperfeiçoamento.

Santos (2007) coloca que o *Firebird* ainda possui várias funcionalidades interessantes, onde possui uma longa história e se destaca na facilidade de manutenção e configuração, tendo grande autonomia.

As principais características deste SGDB que o diferencia dos demais, segundo Cantu (2005), são elas:

- **Facilidade de instalação e manutenção:** o *Firebird* leva poucos minutos para ser instalado e não há a necessidade de conhecimentos específicos. Os bancos criados em sua plataforma não precisam de um espaço definido previamente, sendo seu crescimento proporcional a quantidade de dados inseridos no decorrer de seu uso. O servidor possui poucos parâmetros de configuração, onde os valores padrões já são

[1] Em julho de 2000, a Borland abriu o código fonte do InterBase 6 tornando-o Free e Open Source. No entanto, entre o anuncio da abertura do código e a consolidação do processo, a comunidade de usuários passou por um período de incertezas, o que acabou gerando um certo descontentamento pelo modo como a Borland estava conduzindo o processo de abertura do código. Quando a promessa finalmente se tornou realidade, um grupo de pessoas resolveu criar um novo banco de dados, baseado no código do InterBase 6, que foi chamado de Firebird. Entre essas pessoas estavam nomes como bastante influentes da comunidade InterBase, como por exemplo Ann Harrison, que é conhecida como "mãe do InterBase" (CANTU, 2005, p. 1).

suficientes. O mesmo dispensa seu monitoramento por um DBA, sendo considerado um servidor "*Light*" em questão de tamanho, devido seu instalador na versão Windows ter cerca de 4 MB.

- **Suporte a protocolos de rede**: o *Firebird* apresenta suporte aos protocolos TCP/IP, NetBEUI e IPX/SPX, onde este último foi descontinuado na versão 1.5 do *Firebird.*
- **Arquitetura Versioning**: Geralmente nos outros sistemas de banco de dados, quando um usuário faz uma requisição e chega a alterar algum dado da base, este segmento do banco é bloqueado contra acesso de outros usuários, isso para garantir a integridade do mesmo. O *Firebird* por sua vez utiliza um modelo inovador chamado de *Versioning*, baseado em um sistema otimista de concorrência. Esse mecanismo assume que a probabilidade de que mais de um usuário acesse o mesmo dado e o venha a alterar simultaneamente é muito baixa. O *Versioning* cria versões temporárias dos registros que estão sendo utilizados, e estes registros ficam a disposição o tempo que for preciso dependendo de vários fatores. Quando é percebido pelo *Firebird* que um registro temporário não está mais em uso, ele marca o espaço utilizado pro esse registro como "lixo", para ser reaproveitado pelo próprio banco com outras informações.
- **Transaction Log/Crash Recovey**: a maioria dos SGDBs utilizam *Log* de transação, para que em uma eventual pane, este log desfaz as últimas transações que não foram completadas. Dependendo da quantidade de usuários conectados e do número de transações, e se o servidor de banco de dados virem a entrar em pane, um tempo considerável será consumido para que seja feita a reversão das operações e deixe o banco íntegro. Já o *Firebird*, devido à arquitetura de *Versioning*, não precisa manter um arquivo de log, pois quando o servidor *Firebird* é reiniciado, ele altera automaticamente o "*Flag*" das transações pendentes para *Rollback*, fazendo com que o banco retorne a um estado consistente em um breve espaço de tempo.
- **Backups\Restores**: para se realizar o *Backup*, o *Firebird* não precisa de acesso exclusivo do banco de dados e o mesmo contém uma imagem do banco de dados no exato momento em que esse processo foi iniciado. Isso se dá mais uma vez pela arquitetura de *Versioning*. Estes arquivos de backup não incluem os dados dos índices definidos no banco de dados, diminuindo substancialmente em relação ao arquivo original. Durante o *Restore*, o banco é recriado assim como seus índices. O *Firebird* se utiliza do conceito de árvores BTREE na sua estrutura de indexação, ou

seja, no momento de recriação, essa árvore fica balanceada para a otimização de consultas SQL. Também durante esse processo, as informações dispensáveis são excluídas reduzindo ainda mais a base de dados.

- **Transações**: *Commits* em duas fases: apesar de o *Firebird* não permitir consultas (*Selects*) ou relacionamentos entre tabelas que estejam armazenadas em banco de dados diferentes, ele permite que uma transação esteja ligada a mais de um banco. Ou seja, podem-se alterar dados em dois ou mais bancos ligados numa mesma transação, onde em um *Commit/Rollback* os dados serão confirmados/descartados em todos os bancos em que esta transação estiver ativa.
- **Domínios**: o *Firebird* é flexível em relação ao feitio de domínios para tipos de dados cujos se tornam repetitivos (Ex.: *Numeric*(9,2) pode ser convencionado chamar de DINHEIRO). Todo e qualquer tipo de campo definido em uma tabela no *Firebird* possuí um domínio associado que pode ser definido pelo desenvolvedor ou automaticamente pelo banco de modo transparente.
- **Gerenciamento\Reaproveitamento de espaço**: a alocação dos dados no *Firebird* dinâmica, sendo que a base cresce em disco à medida que novos dados são inseridos. Entretanto, na remoção destes dados, o servidor não reduz o banco devido ser prejudicial à performance do mesmo devido a alocação de espaço em disco ser bastante onerosa. Simplesmente o *Firebird* reutiliza os espaços marcados como "lixo" para armazenar novas informações.
- **Multiplataforma**: o *Firebird* possui versões nativas para múltiplas plataformas, perdendo somente para algum outro desenvolvido em JAVA. O *Firebird* roda nas plataformas DARWIN, FreeBSD, HPUX, Linux, Sinixz, Sparc, Solaris, Win32, MacOS, AIX, WinCE 3.0 (beta).
- **Stored Procedure Selecionáveis**: um dos recursos poderosos do *Firebird*\InterBase as *Select Stored Procedures* ou *Stored Procedures* Selecionáveis, do qual pode-se utilizar *Stored Procedures* como fonte de dados em *Selects* como se fossem "tabelas virtuais".
- **Múltiplos Triggers em um mesmo evento**: o *Firebird* possui *Triggers* de *Before\After-Insert*, *Before\After-Delete* e *Before\After-Update*, dessa maneira pode-se criar mais de um *Trigger* para um mesmo evento. Na versão 1.5 do *Firebird* há a possibilidade de criar *Triggers* associados a múltiplos eventos.
- **Funções Definidas pelo Usuário (UDFs)**: UDFs são funções criadas em qualquer

linguagem que gere bibliotecas compartilhadas. Com funções em mãos, pode-se potencializar comandos SQL, *Stroed Procedures* e *Triggers*. As linguagens que o *Firebird* suporta preferencialmente, pelo fato de ser descendente do InterBase, são Delphi, Cbuilder, C, C++, Pascal e etc, as quais são linguagens originárias da Borland.

- **Ferramentas de administração**: o *Firebird* não dispõe de nenhuma ferramenta gráfica de administração nativa, sendo disponível uma versão interativa por linha de comando. Isso acontece pelo fato de não querer prejudicar os fabricantes das ferramentas já existentes e permitir que os usuários mais lhe agradar. Todavia, existem inúmeras interfaces administrativas, inclusive gratuitas.
- **Componentes de acesso**: a maneira mais eficiente de aproveitar os recursos de um banco de dados é a utilização de componentes de acesso, de preferência nativos para que haja uma sincronia com a API do servidor. Dentre os inúmeros pacotes disponíveis para tal destacam-se IBO e FIBPlus.

Já no quesito de segurança o *Firebird* apresenta algumas falhas, uma no que diz respeito ao usuário padrão SYSDBA que tem acesso a todos os bancos de dados, mas melhorias nesses sentidos são esperadas nas versões 2.5 ou 3.0. Mecenas (2006) elenca que outro fator ainda neste viés, vem a ser a criptografia, que só ocorre nas senhas dos usuários, não havendo para os dados.

Mas, o *Firebird* tem demonstrado robustez e está em constante evolução:

> Um bom exemplo foi mostrado na Softool 06, onde o Avarda (ERP russo) estava rodando com um servidor Firebird 2.0 Classic e um número médio de 100 conexões simultâneas, acessando um banco de dados de 120GB com 700 milhões de registros (BUBLITZ, 2007, p. 50).

Além disso, Cantu (2005) menciona que existem várias empresas que adotaram o *Firebird* em seus negócios, tais como a EMBRAPA (Empresa Brasileira de Pesquisa Agropecuária) e a Clínica do Leite/ESALQ-USP. Mas o que talvez limite a expansão deste SGDB seja a adoção de um *Software* livre. Cantu (apud Freitas e Paris, 2007, p.36) diz:

> Talvez o maior fator limitante na adoção do Firebird [...] frente aos bancos comerciais não seja a falta de um ou outro recurso, mas sim o medo que algumas empresas ainda tem na adoção de um software livre. Felizmente, com a popularização do Linux e de outros Software abertos em grandes empresas e corporações, assim como o investimento de gigantes como a IBM em alguns desses Software, esse medo tende a deixar de existir em um futuro próximo (CANTU, 2004, p.22).

Tendo em vista os valores deste sistema de banco de dados, vê-se que este é uma realidade, sendo competitivo, tanto no campo comercial como nas comunidades, que crescem e desenvolvem o seu código, onde cada vez mais tende a romper barreiras e se equiparar aos mais bem sucedidos.

3.2 MYSQL

O *MySQL* é mais uma alternativa livre de gerenciamento de bases de informação, e, segundo Kofler (2007), mundialmente conhecido por sua velocidade, tem suas origens em meados dos anos 90:

> O MySQL teve origem quando os desenvolvedores David Axmark, Allan Larsson e Michael "Monty" Widenius, na década de 90, precisaram de uma interface SQL compatível com as rotinas ISAM que utilizavam em suas aplicações e tabelas. Em um primeiro momento, tentaram utilizar a API mSQL, contudo a API não era tão rápida quanto eles precisavam, pois utilizavam rotinas de baixo nível (mais rápidas que rotinas normais). Utilizando a API do mSQL, escreveram em C e C++ uma nova API que deu origem ao MySQL (MANZIEIRO E SANCHES, 2009, p. 4)

Este SGDB foi idealizado para que venha a trabalhar no mercado on-line (internet), eficaz no meio corporativo (sistemas *Desktop*) e que seus custos fossem reduzidos. E, apesar de atualmente este ser multiplataforma, o *MySQL* também tinha como objetivo ser compatível com o Linux, acabando por difundi-lo ainda mais quando tanto esse sistema operacional e o movimento *Open Source* ganharam força no mercado mundial.

Com o sucesso desta nova API (*Application Programming Interface*), seus idealizadores fundaram uma empresa de consultoria (manutenção) chamada *MySQL* AB. Daí então, este banco de dados tornou-se extensamente conhecido pela sua velocidade, sendo um sistema emergente no concorrido mercado de gerenciadores de bancos de dados, onde novas versões vieram a ser lançadas implementando novas rotinas que se fizeram necessárias.

Mas a *MySQL AB*[2], desenvolvedora do sistema *MySQL*, nos últimos tempos foi

[2] No dia 16 de Janeiro de 2008, a MySQL AB, desenvolvedora do MySQL foi adquirida pela Sun Microsystems, por US$ 1 bilhão, um preço jamais visto no setor de licenças livres. No dia 20 de Abril de 2009 a Oracle compra a Sun Microsystems e todos o seu produtos, incluindo o MySQL (MANZIEIRO E SANCHES, 2009, p. 4)

adquirida por empresas, tais como Sun Microsystems e Oracle:

O *MySQL*, ao contrário do que muito pensam, não é totalmente livre, trata-se de um sistema de licença dupla, ou seja, existe uma versão livre e outra que pode-se comprar uma licença comercial.

Como principais características deste sistema de banco de dados, segundo Kofler (2007), destacam-se:

- **Sistema de banco de dados relacional**: como quase todos os sistemas de banco de dados no mercado, o *MySQL* é um sistema de banco de dados relacional.
- **Arquitetura Cliente/Servidor**: *MySQL* é um sistema cliente servidor. Existe um servidor de banco de dados (*MySQL*) e vários clientes arbitrários (programas de aplicação), que se comunicam com o servidor, isto é, consltuam dados, salvam alterações, etc. Os clientes podem executar no mesmo computador ou na internet. Quase todos os grandes sistemas de banco de dados (Oracle, Microsoft SQL Server, etc.) são sistemas cliente/servidor. Estes estão em contraste com os sistemas de servidores de arquivos, que incluem Microsoft Access, dBase, e FoxPro. A desvantagem decisiva para os sistemas servidores de arquivo é que, quando executados em rede, tornam-se extremamente ineficientes quando o número de usuários crescem.
- **Compatibilidade com SQL**: O *MySQL* suporta como sua linguagem de banco de dados - como o próprio nome sugere - SQL (Linguagem Estruturada de Consulta). SQL é uma linguagem padronizada para consulta e atualização de dados e para a administração de um banco de dados. Existem vários dialetos SQL (cerca de quantos sistemas de banco de dados existirem). *MySQL* adere ao padrão SQL atual (no momento em SQL: 2003), embora com restrições importantes e um grande número de extensões. Através da configuração *Sql-Mode* você pode fazer o servidor *MySQL* se comporta em sua maioria de forma compatível com os sistemas de banco de dados diversos. Entre eles estão a IBM DB / 2 e Oracle.
- **Subselects**: Desde a versão 4.1, o *MySQL* é capaz de processar uma consulta na forma: **SELECT** * FROM table1 WHERE x IN (SELECT y FROM table2)

Há também inúmeras variantes sintaxe para subconsultas:

- **Views**: Simplificando, views relacionam-se com uma consulta SQL que é vista como um objeto distinto do banco de dados e torna possível uma visão particular do

banco de dados. O *MySQL* tem suporte a views desde a versão 5.0.

- **Stored Procedures**: Aqui lidamos com o código SQL que é armazenado no sistema de banco de dados. *Stored Procedures* (SPs para ser curto) são geralmente usados para simplificar algumas etapas, tais como inserir ou excluir um registro de dados. Para programadores clientes esta é uma vantagem de que eles não precisam processar as tabelas diretamente, mas pode confiar em SPs. Como *Views*, SPs ajudam na administração de projetos de banco de dados grande. SPs também podem aumentar a eficiência. O *MySQL* tem suporte SPs desde a versão 5.0.
- **Triggers**: Triggers são comandos SQL que são executados automaticamente pelo servidor em certas operações de banco de dados (*INSERT*, *UPDATE* e *DELETE*). *MySQL* suporta triggers de forma limitada da versão 5.0, e funcionalidades adicionais são prometidas para a versão 5.1.
- **Unicode**: *MySQL* suporta todos os conjuntos de caracteres possíveis desde a versão 4.1, incluindo Latin-1, latino-2, e Unicode (UTF8 quer na variante ou UCS2).
- **Interface com o usuário**: Há uma série de interfaces convenientes para o usuário para administrar um servidor *MySQL*.
- **Full-text search**: Pesquisa completa de texto simplifica e acelera a busca de palavras que estão localizados dentro de um campo de texto. Se você utilizar o *MySQL* para armazenar texto (como em um grupo de discussão na internet), você pode usar a pesquisa completa de texto para implementar simplesmente uma função de pesquisa eficiente.
- **Replicação:** A replicação permite que o conteúdo de um banco de dados a seja copiado (replicado) para um número de computadores. Na prática, isso é feito por duas razões: para aumentar a proteção contra falhas do sistema (de modo que, se um cair, o outro pode ser colocado em serviço) e para melhorar a velocidade de consultas do banco de dados.
- **Transactions**: No contexto de um sistema de banco de dados, uma transaction é a execução de várias operações de banco de dados como um bloco. O sistema de banco de dados garante que todas as operações sejam corretamente executadas ou nenhuma delas. Este mantém-se mesmo no meio de uma operação que há uma falha de energia, o computador travar, ou algum outro desastre ocorre. Assim, por exemplo, não pode ocorrer que uma soma de dinheiro seja retirada da conta A mas

por falha não chega a ser depositada na conta B, devido a algum tipo de erro do sistema. As transactions também dão aos programadores a possibilidade de interromper uma série de comandos já executados (um tipo de anulação). Em muitas situações, isto leva a uma considerável simplificação do processo de programação. Apesar da opinião popular, o *MySQL* tem suporte a transações para um longo tempo. Deve-se observar aqui que o *MySQL* pode armazenar as tabelas em uma variedade de formatos. O formato de tabela padrão é chamado MyISAM, e esse formato não oferece suporte a transações. Mas há um número adicional de formatos que fazem suportam transações. O mais popular deles é InnoDB

- **Foreign key constraints**: são regras que asseguram que não há referências cruzadas em tabelas vinculadas que levam a lugar nenhum. O *MySQL* suporta restrições de chaves estrangeiras em tabelas InnoDB.
- **Funções GIS**: Desde a versão 4.1, o *MySQL* tem suporte a armazenamento e processamento de dados geográficos bidimensional. Assim, o *MySQL* é bem adequado para aplicações GIS (sistemas de informação geográfica).
- **Linguagens de programação**: Há um bom número de APIs e bibliotecas para o desenvolvimento de aplicações *MySQL*. Para a programação do cliente você pode usar, entre outros, as linguagens C, C + +, JAVA, Perl, PHP, Python e TCL.
- **ODBC**: *MySQL* suporta a interface ODBC Connector/ODBC. Isso permite que o *MySQL* possa ser usado por todas as linguagens de programação usuais que funcionam com o *Microsoft Windows* (Delphi, Visual Basic, etc.). A interface ODBC também pode ser implementado em *Unix*, mas que raramente é necessário. Programadores de Windows que migraram para o novo *Microsoft* NET. pode, se assim o desejarem, usar o provedor ODBC ou a interface. NET Connector / NET.
- **Independência de plataforma**: não são somente aplicações de cliente que podem executar em uma variedade de sistemas operacionais; o próprio *MySQL* (isto é, o servidor) pode ser executado no âmbito de vários sistemas operacionais. Os mais importantes são o *Apple Maçintosh OS X*, *Linux*, *Microsoft Windows*, e as inúmeras variantes do *Unix*, tais como AIX, BSDI, FreeBSD, HP-UX, OpenBSD, o BSD Net, Iris SGI, e Sun Solaris.
- **Velocidade**: O *MySQL* é considerado um programa de banco de dados muito rápido. Esta velocidade tem sido apoiada por um grande número de testes de *Benchmark* (embora esses testes - independentemente da origem - devem ser

considerados com uma boa dose de ceticismo).

Contudo, este SGDB possui limitações, onde, ainda consoante com o autor enunciado anteriormente, várias das deficiências que serão ditas logo abaixo ou já foram implementadas ou são metas da equipe de desenvolvedores do *MySQL*. Assim, os principais défcits desse banco de dados, são eles:

- Quando o *MySQL* é usado com tabelas padrão (tabela do tipo MyISAM), seguida de *bloqueio*, ou seja, o bloqueio temporário para acesso ou alteração da informação no banco de dados, fica em operação para tabelas por inteiro (*Table Locking*). Você pode contornar o problema de *table-locking* através da implementação de formatos transaction-capable de tabelas, como InnoDB, que suporta o bloqueio de linhas.
- Ao utilizar tabelas MyISAM, o *MySQL* não é capaz de executar *Hot Backups*, que são backups durante a operação sem travar as tabelas com os bloqueios. Aqui, novamente, a solução é InnoDB, embora aqui a função de hot backup está disponível apenas na forma de um suplemento comercial.
- Muitos sistemas de banco de dados oferecem a possibilidade de definir tipos de dados personalizados. O *MySQL* não suporta essa funcionalidade, nem se planeja atualmente.
- O *MySQL* tem até agora ignorado a tendência geral de XML. Não está claro quando o *MySQL* irá oferecer suporte a processamento direto de dados XML. Inúmeros sistemas de banco de dados comerciais oferecem funcionalidades muito mais nesta área, e até mesmo o padrão SQL:2003 prevê uma série de funções XML.
- O *MySQL* é de fato um sistema de banco de dados muito rápido, mas é muito limitado na sua usabilidade para aplicações em tempo real, e não oferece funções OLAP. OLAP significa processamento analítico *Online*, e se refere a métodos especiais para a gestão e análise de dados multidimensional. Sistemas de banco de dados OLAP com capacidade são freqüentemente chamados de *Data Warehouses*.
- O *MySQL* suporta, desde a versão 5.0, *Stored Procedures* e *Triggers*, mas estas funções ainda não são completamente maduras (isso se aplica especialmente aos *Triggers*) e ainda não têm a mesma estabilidade e plenitude de funções oferecidas pelos sistemas de banco de dados comerciais.
- Restrições semelhantes mantem-se, como para realizar as funções GIS introduzido

na versão 4.1. Sistemas de banco de dados comerciais oferecem em alguns casos, consideravelmente maior funcionalidade

Vistos qualidades, deficiências e o tempo em que o *MySQL* está na ativa, nota-se que o mesmo alcançou sucesso em sua trajetória, vindo a trazer inovações e se adaptar às necessidades do mercado, onde apesar de fazer parte do patrimônio de uma empresa proprietária, este banco de dados continua a ter licença livre.

Desta forma, mesmo que a companhia que o detém vier a depreciá-lo, não há como extingui-lo devido este ser livre, o seu código está a disposição da comunidade, onde tem-se a possibilidade de continuar a aprimorá-lo.

Pode-se dizer a respeito do *MySQL* que o mesmo foi desenvolvido para lhe dar com bases bastante povoadas, de maneira rápida e eficiente, onde o mesmo tem sido empregado em vários ambientes de produção, possuíndo uma vasta lista de funcionalidades, tornando-o ideal em moldes que necessitem de conectividade, velocidade e segurança. Isto é, "(...) altamente adaptável para acessar bancos de dados na Internet." (Manzieiro e Sanches, 2009, p.9)

3.3 POSTGRESQL

O *PostsgreSQL* é mais uma opção de gerenciamento de banco de dados e, perante Neto (2007), vem ganhando cada vez mais espaço no concorrido mercado de SGDBs.

> (...) é um SGDBR - Sistema Gerenciador de Base de Dados Relacional (ou, mais comunmente indicado, do inglês RDBMS - Relational Database Management System) que está baseado nos padrões SQL ANSI-92, 96 e 99, de alta performance, de fácil administração e utilização em projetos (por especialistas DBAs - Database Administrator e Projetistas de Sistemas) (NETO, 2007, p. 26).

Sua origem nos remete a 1986, na Universidade Norte-Americana da Califórnia, a partir de seu descendente, o ingres. Entretanto, esse sistema de banco de dados obteve seu interpretador SQL por volta de 1995, trabalho esse realizado por Andrew Yu e Jolly Chen e que daí em diante recebeu o nome de *Postgres95*. Esta mudança não só acarretou em um aumento de código, mas o *Postgres95* passou a ser um legítimo SGDB relacional, assim como todos os outros concorrentes. Este teórico ainda afirma que seu código só foi aberto em 1996,

onde o mesmo já era conhecido como *PostgreSQL*.

Além disso, várias novidades vem sido acrescentadas no *PostgreSQL* aos passar dos anos tornando-o competitivo, não deixando a desejar em relação a outros SGDBs famosos, e com uma vantagem muito importante: ser livre.

> (...) permite a utilização de SQL, Triggers (disparadores), e toda a gama de programação e construção pertinente aos mais famosos RDMSs do mercado (como Oracle, InterBase, SQL Server, DN/2, etc.), além de permitir "Embedded SQL" com pré-compilação (com linguagens C e C++) e possui drivers ODBC e JDBC para interface com ambientes e linguagens de programação, dentre elas, por exemplo: BDE, Borland Delphi, Borland C++ Builder, MS Visual C++, MS Visual Basic, Perl, XML, e Java (NETO, 2007, p. 27)

O *PostgreSQL* ainda suporta várias plataformas, dentre elas o *Windows,* o *Linux*, e o *Unix*. E, segundo Scherer et. al. (2007), o *PostgreSQL* é considerado o banco de dados baseado em *Software* livre mais avançado, chegando a se equiparar a SGDBs proprietários. Devido a isso, seu uso aumentou significativamente, sendo desenvolvido por comunidades voluntárias no mundo inteiro, sendo reconhecido pela sua robustez e segurança.

> (...) vem ganhando cada vez mais espaço no mercado e é utilizado tanto em organizações públicas quanto organizações privadas. Um dos fatores que gerou um aumento significativo de utilização foi a sua inclusão na maioria das distribuições do Linux.
> (...) suporta grandes volumes de dados, sendo que o tamanho do mesmo, o número de linhas e o número de índices são ilimitados. As tabelas suportam um valor aproximado de 32 TB cada uma, uma linha pode conter cerca de 1.6 TB e um campo cerca de 1 GB (SCHERER ET. AL., 2007, p. 04)

O *PostgreSQL*, de acordo com Neto (2007), possui algumas características de orientação a objetos (por exemplo, herança de tabelas) propostas no SQL ANSI-99. Implementa também, segundo Scherer et. al. (2007), um recurso chamado *Garbage Collection* que tem como papel apagar fisicamente os dados excluídos, aprimorando o desempenho do sistema. A maneira de armazenar os dados difere dos demais bancos de dados, tendo esse sistema baseado em cluster, cujo vem a ser um diretório que armazena todos os dados da base como os arquivos de configuração.

> (...) os dados são gravados no cluster[3]. Os dados são temporariamente armazenados em um diretório do cluster que contém um conjunto de

[3] Cluster: é um diretório onde todos os arquivos do banco estão armazenados.

> arquivos chamado WAL (Write Ahead Log). A cada modificação no conteúdo dos dados do banco, os novos dados são gravados temporariamente no WAL, que podem ter até 16 MB de tamanho. Os dados somente são transferidos do WAL para a base de dados original quando a função de checkpoint for chamada (SCHERER ET. AL., 2007, p.5)

O *Checkpoint* pode ser configurado para funcionar periodicamente ou via comandos SQL. Dessa maneira a performance do banco aumenta consideravelmente, pois reduz a quantidade de escritas nas tabelas originais.

Este SGDB funciona através de processos, dentre os quais o principal é o *Postmater*, cujo gerencia o cluster e os processos de conexão com os clientes. Os outros processos chamam-se *Postgres*, que iniciam após as requisições dos clientes ou quando o coletor de estatísticas estiver executando. O coletor de estatísticas por sua vez é um aplicativo que auxilia na administração do banco de dados, coletando informações a cerca do servidor. Com essas informações em mãos, o administrador faz ajustes para que o sistema trabalhe com mais eficiência.

A concorrência de atualização e visualização dos dados, em diferentes sessões simultâneas, são controlados com eficiência. Além disso, o *PostgreSQL* não tem um limite máximo de conexões concomitantes, sendo limitado apenas pelo hardware ao qual o servidor *PostgreSQL* se encontra.

Isso é implementado de duas formas: *o Read Commited* e *Serializable*. No primeiro, os dados visualizados são os que foram efetivados antes do comando iniciar ou os que foram efetivados por transações concorrentes, e nunca os dados que não foram efetivados. Já no segundo, os comandos enxergam os dados efetivados antes de a transação iniciar. "Esta é uma forma mais rígida de controle de concorrência e pode ser alterada através do comando *Set Transaction* utilizado dentro dos *Software* que gerenciam o banco" (SCHERER ET. AL., 2007, p. 05)

Por sua vez, o controle para se acessar os dados é baseado em direitos de acesso ou privilégios, donde se controlam os conteúdos que o usuário pode ou não visualizar. A identidade do usuário é que vai determinar o grau de acessibilidade que o mesmo terá sobre o banco de dados.

Ainda, Scherer et. al. (2007), descreve que o *PostgreSQL* dispõe de métodos eficientes de autenticação, relativo à segurança onde destacam-se:

- **Senha**: o principal método é o md5 (Message-Digest Algorithm 5) que suporta senhas criptografadas. As senhas, criptografadas ou não, ficam armazenadas em uma tabela do catálogo do sistema chamada *Pg_Shadow*.

- **Kerberos**: implementa um protocolo de transporte de dados em rede, assegurando a comunicação dos dados mesmo em uma rede insegura.
- **Sistema Operacional**: a autenticação pode ser feita pelo nome e senha do usuário do sistema operacional. Esta maneira não é aconselhada, pois dependerá da segurança oferecida pelo sistema operacional.

Para reforçar a segurança, os dados que trafegam na rede é possível criptografar as comunicações entre cliente e servidor por meio de conexões SSL (*Secure Sockets Layer*) através da instalação do *OpenSSL* no cliente e no servidor. Ainda é possível criar as conexões de clientes através de túneis SSH (*Secure Shell*).

Outro tópico importante, consoante com autor dantes referido, é a geração de cópias de segurança para o restabelecimento da base de dados quando houver falhas físicas ou por outro tipo. Essas cópias podem ser geradas de duas maneiras pelo *PostgreSQL*:

- **Dump**: esse recurso armazena comandos SQL em um arquivo texto, cujo reconstitui todos os dados da base. Pode ser usado para restaurar alguns registros ou todos (*Dumpall*) contidos no cluster.
- **Point-in-time recovery**: trata-se de reconfigurar os dados para a data e hora informada pelo DBA (*Data Base Administrator*), que se dá com a utilização do WAL como um sistema de armazenamento de arquivos. O WAL manipula os pontos de verificação que são gravados sempre que uma modificação é feita.

Ainda podem-se realizar cópias de segurança via sistema operacional, entretanto o *Software* do banco de dados (*PostgreSQL*) deve estar fechado, não havendo flexibilidade para se escolher quais registros se quer copiar, sendo copiado absolutamente tudo (dados e arquivos de configuração), além de ser extremamente demorado e não garantir a integridade dos arquivos.

O desempenho do *PostgreSQL* difere entre plataformas, em especial *Windows* e *Linux*, onde na primeira é necessário reescrever parte das funções do SGDB, diminuindo sua performance. Esta queda de rendimento é sentida somente em aplicações de grande porte, sendo imperceptível em pequenos projetos. Para aprimorar ainda mais o processamento, o *PostgreSQL* trabalha com memória exclusiva, assim como bancos de dados proprietários, em que uma fatia da memória RAM (*Random Access Memory*) é reservada para uso específico do banco.

De fato, nota-se o quão este SGDB é avançado, não só em meio aos de licença livre, mas também os proprietários, trazendo diversas inovações que o faz ser extremamente seguro

e eficaz, a um preço muito interessante: gratuito.

Com isso, Scherer et. al. (2007, p.9) conclui que o *PostgreSQL* é tanto eficiente e eficaz quanto seguro, dispondo de vastas opções de funcionalidades. Entretanto, o desenvolvedor deve ter um conhecimento mais refinado a cerca deste gerenciador e usufruir de todo o seu potencial, chegando, em certos casos, a superar os concorrentes proprietários.

4 MYSQL versus POSTGRESQL versus FIREBIRD

Para realizar-se a análise entre os SGDBs, buscaram-se tópicos para que se pudesse fazer as devidas comparações entre os servidores bancos de dados *Firebird*, *MySQL* e *PostgreSQL*, os quais são:

- **Interfaces com linguagens:** é a junção entre o SGDB com linguagens de desenvolvimento.
- **Stored Procedures:** são procedimentos ou funções de programação armazenados no banco de dados, executados no próprio SGDB.
- **Triggers:** são ações que o banco de dados deve tomar quando uma certa ação for executada.
- **Cursores:** encapsula uma consulta e permite então ler o resultados linha(s) por linha(s) e processá-las.
- **XML:** suporte à linguagem XML, usada para troca de dados.
- **Domínios:** é um tipo de dado com opcionais constraints.
- **Transactions:** mantem a integridade dos dados.
- **Bloqueios:** impede que duas transações simultâneas acessem o mesmo recurso ou dado ao mesmo tempo.
- **Ferramentas disponíveis:** ferramentas de suporte que vão desde a modelagem até a administração.
- **Portabilidade dos bancos:** capacidade de se executar em sistemas operacionais diferentes.
- **Sub-Selects:** são selects aninhados.
- **Visões:** visões relacionam-se com uma consulta *SQL* que é vista como um objeto distinto do banco de dados e torna possível uma visão particular do banco de

dados.

- **Relações de chave estrangeira (FK):** define o relacionamento entre duas tabelas num banco de dados.
- **FK Constraints:** são regras que asseguram que não há referências cruzadas em tabelas vinculadas que levam a lugar nenhum.
- **Indexar tipos não-triviais:** indexar tipos como texto.
- **Sequencias:** geração de campos numéricos em sequencia em tabelas.
- **OO:** alguns coneceitos de orientação a objetos, como herança de tabelas.
- **Notificações Async:** notificações assíncronas.
- **Constraints:** são restrições a uma coluna de uma tabela de banco de dados com o intuito de manter a integridade dos dados.
- **Select into:** seleciona dados de uma tabela do banco de dados e inseri-los numa tabela diferente ao mesmo tempo.
- **Nível de bloqueio de linha:** bloqueio de acesso simultâneo a uma mesma linha de uma tabela de um banco de dados.
- **Nível de bloqueio de tabela:** bloqueio de acesso simultâneo a uma mesma tabela de banco de dados.
- **Versão Multi Controle de Concorrência:** trata o acesso de vários usuários e transações simultâneos a uma mesma base.
- **UDF:** são funções criadas em qualquer linguagem que gere bibliotecas compartilhadas.
- **Unions:** unir tipos, que podem não ser semelhantes, para que se tornem um único conjunto de resultados.
- **Backup:** cópias de segurança para restauração da base caso haja algum problema.
- **Full-text search: em campos de tipo texto,** pesquisa completa textual para implementar uma função de pesquisa eficiente.
- **Insert Múltiplo:** salvar mais de um registro simultaneamente em uma tabela de banco de dados.
- **Replicação:** cópias de uma base de dados, sendo um backup contínuo e progressivo.

Outra medida a ser tomada para este estudo é a definição das versões utilizadas dos SGBDs, onde a preferência seja feita por versões mais estáveis, cujas escolhidas são: 2.x para o *Firebird*, 5.x para o *MySQL* e 8.x para o *PostgreSQL*.

Na tabela 2, encontra-se os pontos mais relevantes dantes elucidados de comparação entre os referidos SGDBs e suas respectivas versões.

Tabela 2- Tópicos Relevantes de Comparação entre os Servidores de Banco de Dados e suas Respectivas Versões

Critério	*Firebird* 2.x	*MySQL* 5.x	PostrgeSQL 8.x
Interfaces com linguagens	Sim. Ex.: JAVA, Delphi, C, C++, C# e etc.	Sim. Ex.: C, C + +, JAVA, Perl, PHP, Python e TCL	Sim. Ex.: BDE, Borland Delphi, Borland C++ Builder, MS Visual C++, MS Visual Basic, Perl, XML, e JAVA
Stored Procedures	Sim	Sim	Sim
Triggers	Sim	Sim	Sim
Cursores	Sim	Sim	Sim
XML	Não	Não, mas é planejado.	Sim
Domínios	Sim	Sim	Sim
Transactions	Sim	Sim	Sim
Bloqueios	Sim	Sim	Sim
Ferramentas disponiveis	Sim. Ex.: IBOConsole, IBExpert e etc.	Sim. Ex.: *MySQL* Query Browser, *MySQL* Work-bench e etc.	Sim. Ex.: PGAdmin III.
Portabilidade dos bancos	Sim. Ex.: Linux, Win32, MacOS, Solaris, Sparc e etc.	Sim.Ex.: linux, netWare, Unix (BSD,MacOS,Solarise outros) Windows	Sim. Ex.:Linix, Windows, MacOS e etc.
Sub-Selects	Sim	Sim	Sim
Visões	Sim	Sim	Sim
Relações de chave estrangeira (FK)	Sim	Sim	Sim
FK Constraints	Não	Sim	Sim
Indexar tipos não-triviais	Não	Não	Sim

Sequencias	Sim	Sim	Sim
OO	Não	Não	Sim
Notificações Async	Não	Não	Sim
Constraints	Sim	Sim	Sim
Select into	Sim	Sim	Sim
Nível de bloqueio de linha	Sim	Sim	Sim
Nível de bloqueio de tabela	Sim	Sim	Sim
Versão Multi Controle de Concorrência	Não	Não	Sim
UDF	Sim	Sim	Sim
Unions	Sim	Sim	Sim
Backup	Sim	Sim	Sim
Full-text search	Não	Sim	Sim
Insert Múltiplo	Não	Sim	Sim
Replicação	Somente por meio de ferramentas intermediárias	Sim	Sim

Notadamente se vê a superioridade do gerenciador de banco de dados *PostgreSQL* em relação aos demais, suportando todos os tópicos de comparação utilizados. Em seguida, tem-se o *MySQL* seguido do *Firebird* que satisfazem a maioria das características de comparação, onde o *MySQL* ainda leva uma pequena vantagem em relação aos tópicos abordados.

Entretanto, a escolha do SGDB fica a critério do desenvolvedor, dependendo dos requisitos necessários que devam estar presentes em um banco de dados para o desenvolvimento de uma aplicação, sem contar outros fatores abordados em um projeto de banco de dados.

5 IMPLEMENTAÇÃO E ESTUDO DE PERFOMANCE

5.1 JAVA

Na ciência da computação, um dos principais conceitos e ferramentas utilizadas e debatidas são as chamadas linguagens de programação. Definem-se as mesmas, segundo Marques (2000), como linguagens formais que descrevem mecanismos abstratos e comportam-se como diretivas intermediárias à instruções computacionais.

Existem vários paradigmas de programação, tais como estruturado, orientado a objetos e etc. E, dentre as inúmeras linguagens de programação orientadas a objetos existentes, destaca-se o JAVA.

> Em 1991, foi criada a linguagem Java a partir do Green Project, que tinha como mentores James Gosling, Patrick Naughton e Mike Sheridan. A priori, o objetivo desse projeto não era criar uma nova linguagem de programação, mas sim, proporcionar meios para a convergência de computadores com outros tipos de equipamentos. Em 1995, a empresa Sun MicroSystems introduziu a plataforma JAVA no mercado. Ela é constituída pela linguagem de programação Java, sua máquina virtual e diversas APIs (*Application Programming Interface*) (SILVA; SILVA, 2008, p. 02)

Sua ideia de se desenvolver uma linguagem para pequenos dispositivos, como videocassete, geladeiras e televisão, não vingou, sendo dispensado pelas grandes empresas deste ramo, devido conflitos de interesses e custos (CAELUM, 2010).

Entretanto, com a explosão da intemet, desviaram o foco do JAVA para o desenvolvimento de aplicações que rodassem nos navegadores *web*. Isto assemelhava-se com a proposição inicial da criação do JAVA para ser multiplataforma, ou seja, ser executada em diversos dispositivos. E, sabe-se que existem diversos sistemas operacionais, sendo muito mais vantajoso poder programar em uma única linguagem, independendo da plataforma.

Mas este caráter de ser multiplataforma do JAVA só foi possível graças à chamada máquina virtual.

> Sob o slogan "*write once, run everywhere*", que referencia sua portabilidade, indicando que um programa escrito em Java pode ser executado em qualquer plataforma desde que esta possua a máquina virtual Java, essa linguagem se consolidou como linguagem de programação de caráter multi-plataforma e

> tornou-se popular pelo seu amplo uso na internet. Sendo assim, um programa pode ser executado em um computador comum independente do sistema operacional, da arquitetura ou do processador deste (SILVA; SILVA, 2008, p. 2)

O conceito de máquina virtual (MV), segundo Caelum (2010), é de, entre a aplicação e o sistema operacional, existir outra camada para traduzir, respectivamente, suas instruções às correspondentes do sistema operacional no qual se está executando.

Silva e Silva (2008) ressalva que esta camada intermediária é a chamada linguagem bytecode, que nada mais são do que instruções em código de máquina para a máquina virtual JAVA, tornando um programa multiplataforma desde que haja suporte ao JAVA.

A figura 9 destaca o funcionamento da máquina virtual JAVA:

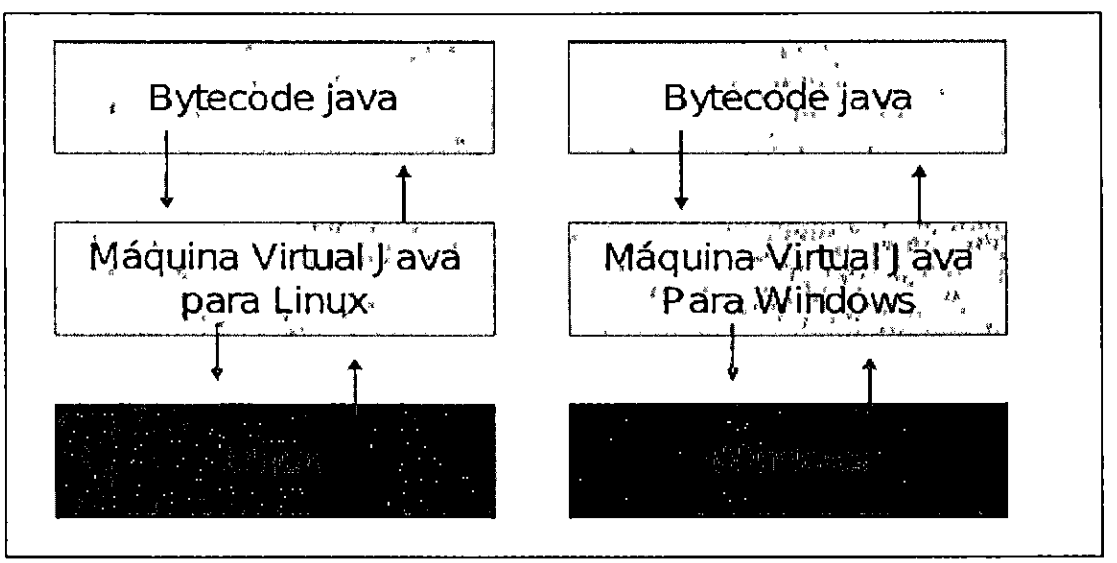


Figura 8 - Funcionamento da Máquina Virtual JAVA.

Fonte: (CAELUM, 2010, p. 5)

Com isso chega-se a concluir que o JAVA não é só mais uma linguagem dentre as várias existentes, mas também não se pode declará-la superior a todas as outras. Cabe ao desenvolvedor escolher qual a que se adapta melhor ao seu projeto.

5.2 Aplicação

5.2.1 Descrição

Tendo em vista as características e ou deficiências de cada SGDB relativo a nosso estudo, no caso o *Firebird*, o *MySQL* e o *PostgreSQL*, pelos meios em paralelo e por via de tópicos de comparação, é interessante ver este desempenho na prática por meio de um aplicativo.

Para satisfazer este tento, foi construído um *Software* chamado FMPSys (sendo cada letra, no caso F, M e P, as respectivas iniciais dos gerenciadores de banco de dados usados), desenvolvido na linguagem de programação JAVA e *desktop*, sob a IDE Netbeans 6.9.1, que comunica-se com o *Firebird*, *MySQL* e *PostgreSQL*, em suas respectivas versões: 2.1 Classic, 5.1 e 8.4. Como arquitetura de desenvolvimento, utilizou-se o sistema operacional Ubuntu 10.10 (*Maverick, Kernel Linux* 2.6.35-25 *Generic*, GNOME 2.32.0) o qual também é livre, e máquina de processador Intel(R) Pentium(R) D CPU 3.00GHz, HD SATA de 160 Gb e 2 Gb de memória RAM.

Utilizaram-se também ferramentas de administração de banco de dados para visualização da ocorrência das transações e controle. Para os respectivos SGDBs mencionados as ferramentas, que são *open source* inclusive, foram as seguintes:

- *Firebird*: *FlameRobin* 0.9.2;
- *MySQL*: *MySQL Query Browser* 1.2.12;
- *PostgreSQL*: *pgAdmin* III 1.10.5.

Este estudo consiste em analisar, no que diz respeito ao tempo gasto de processamento das principais rotinas SQL (*Create*, *Select*, *Insert*, *Delete* e *Drop*) dos sistemas gerenciadores de bancos de dados em questão, e que estas sejam comuns em ambos, quer seja individualmente como em transações simultâneas em ambos. Para mensurar este tempo, capturou-se do sistema o tempo em milissegundos imediatamente antes de se enviar o comando SQL ao SGDB, e o tempo em milissegundos imediatamente depois do retorno da transação SQL, assim podendo estimar o tempo transcorrido.

A tabela 3, discrimina como foram implementadas as consultas dantes mencionadas na aplicação:

Tabela 3 - Funcionalidades e Descrição das mesmas no Sistema FMPSys.

Consultas	Descrição
Create	Como opções de criação têm-se bancos e tabelas. Para se criar um banco pode-se fazê-lo isoladamente, relativo ao SGDB escolhido. Ou ainda pode-se construir uma base comum aos três gerenciadores ao mesmo tempo. Similar a este processo ocorre com a construção de tabelas, podendo se criar em bancos em específicos, ou em bancos similares criados. Os tipos de dados usados para os atributos das tabelas foram os seguintes: integer, char, varchar, float, date e numeric. Pelo fato de o *Firebird* possuir uma quantidade reduzida de tipos de dados em relação aos outros dois gerenciadores, e por estes serem os mais empregados, e pela razão de estes estarem presentes em ambos SGDBs, optou-se pelos referidos tipos para o feitio das tabelas.
Select	Esta transação recupera um ou todos os campos de uma tabela de um banco de dados específico contido em um dos SGDBs.
Insert	Insere dados em uma tabela existente em uma base de dados criada, onde esta base pode ser comum a ambos SGDBs, ou específica. Uma peculiaridade nesta rotina foi encontrada no que diz respeito à inserção dos dados nos SGDBs, onde no *Firebird*, diferente dos demais estudados, não suporta inserir múltiplos registros. Ou seja, só se pode popular um banco no *Firebird* registro a registro, ao contrário do *MySQL* e do *PostgreSQL* que suportam o chamado insert múltiplo. Para contornar esta deficiência, existe uma maneira de se inserir, em um comando só, 255 registros simultâneos no *Firebird*. Para inserir mais de 255 registros, a ação é feita de bloco em bloco até completar a operação.
Delete	Remover os dados contidos da tabela selecionada de um banco, englobando os que sejam comuns, de dados disponíveis.
Drop	Escolhido o SGDB, englobando os que sejam comuns a ambos gerenciadores, pode-se escolher tanto base(s) como tabela(s) para remoção.

Outro fato a ser dito para o funcionamento do sistema, é a criação em cada SGDB de um banco de dados chamado BANCOS, onde neste existe uma tabela chamada DBS que possui um campo chamado VNOME do tipo varchar(20) (tipo que guarda até 20 caracteres),

onde guarda o nome das bases criadas pela aplicação para que a mesma só venha a acessar bancos criados por ela própria.

Vistas as funcionalidades, vale ressaltar que o foco deste aplicativo é analisar critérios de algumas das mais importantes transações (criação, povoamento e consultas) quantitativamente (tempo) e a integridade dos dados relativo a cada transação.

5.2.2 Funcionalidades

A seguir, serão mostradas com mais detalhes cada funcionalidade, mencionadas anteriormente, presente no aplicativo FMPsys mencionadas anteriormente e suas particularidades.

Inicialmente temos o formulário principal: composto de abas, nas quais representam cada função que o *Software* desempenha (*Create*, *Select*, *Insert*, *Delete* e *Drop*); uma barra de menu composta de dois itens: Arquivo e Ajuda.

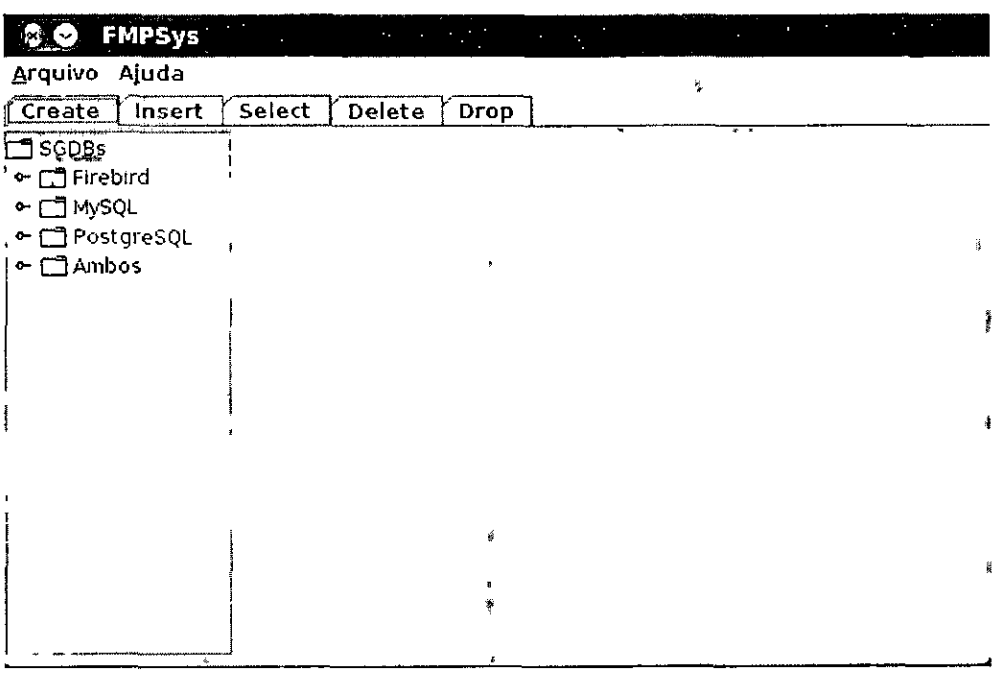


Figura 9 - FMPSys: Abas e Barra de Menu.

Em arquivo há a opção de encerrar o sistema, clicando no item Sair:

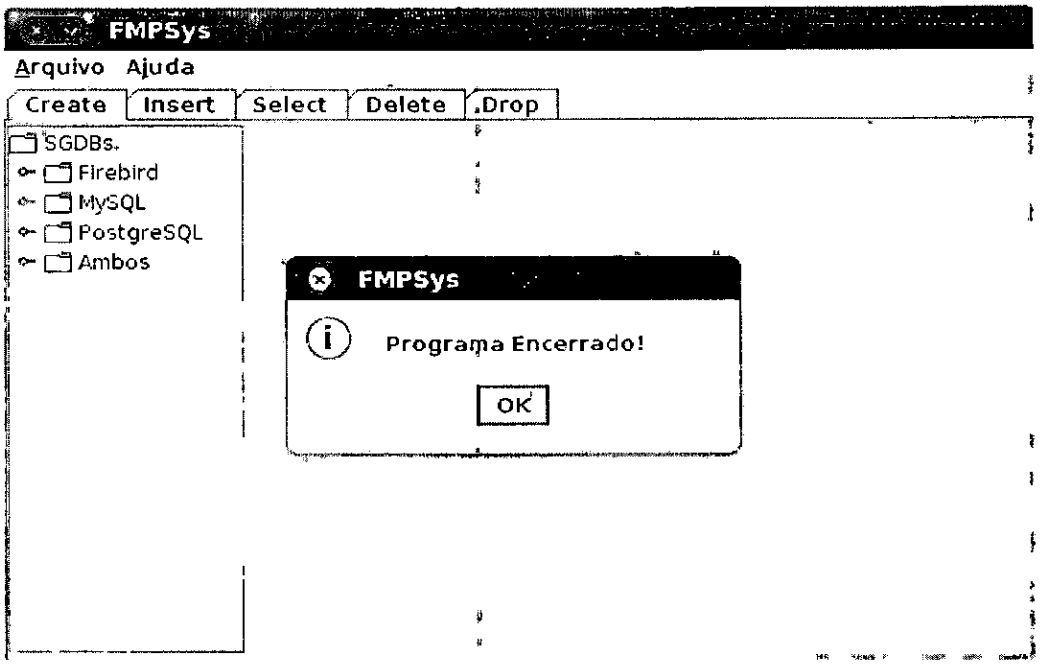


Figura 10 - FMPSys: Fechando.

Em Ajuda, tem-se a opção Sobre, que gera um formulário que sintetiza o *Software* FMPSys:

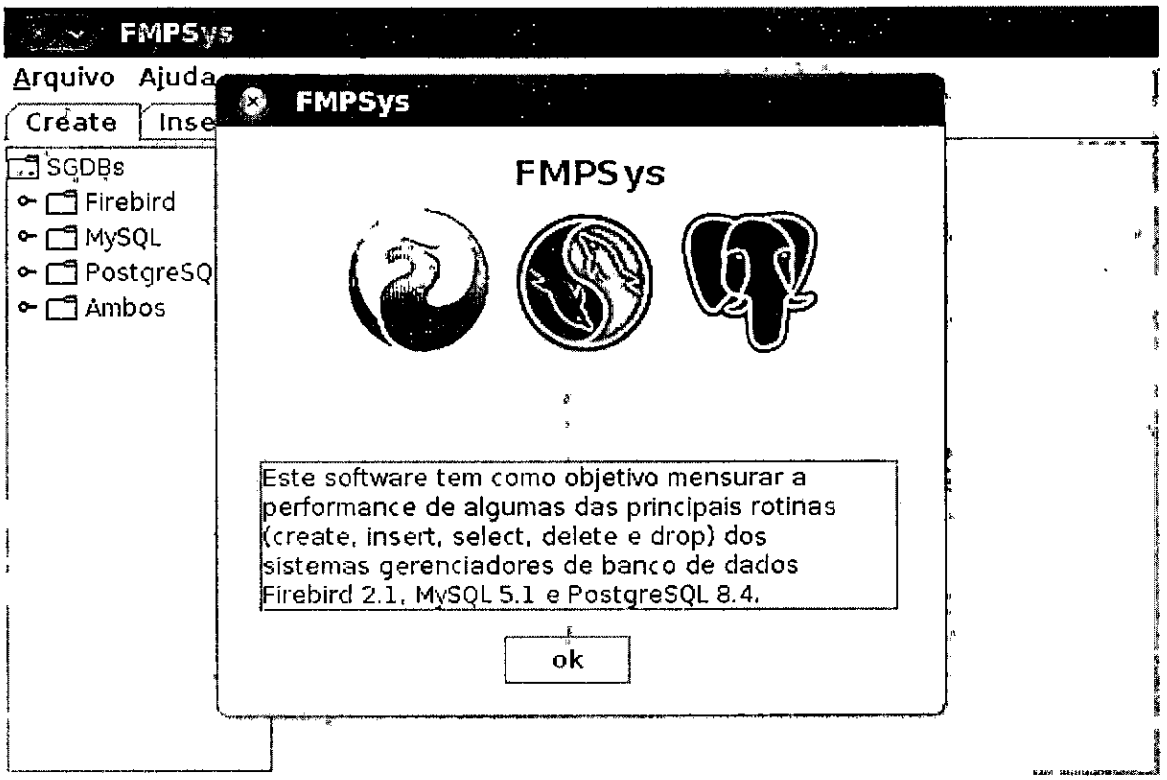


Figura 11 - FMPSys: Sobre.

Seguindo a ordem das aba temos:

1. **Create**: esta rotina cria bancos de dados ou tabelas. Nesta aba tem-se uma árvore com as opções de se criar bancos de dados ou tabelas em bancos de dados já criados em algum dos SGDBs à disposição, ou em todos simultaneamente. (*Vide Figura*)

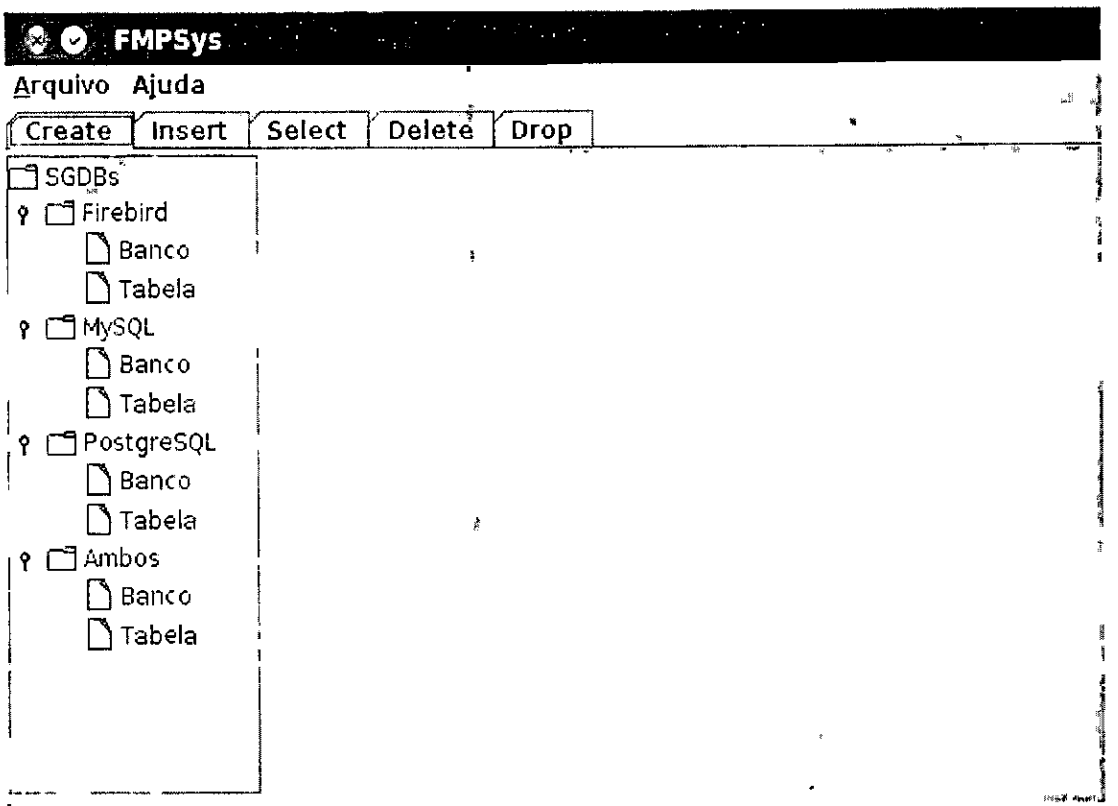


Figura 12 - Opções de Criação: Bancos ou Tabelas.

Criação de Banco: selecionado o(s) SGDB(s), gera-se uma pequena janela para a entrada do nome da(s) nova(s) base(s) criada(s), com as opções de confirmar ou cancelar a ação. (*Vide Figura*)

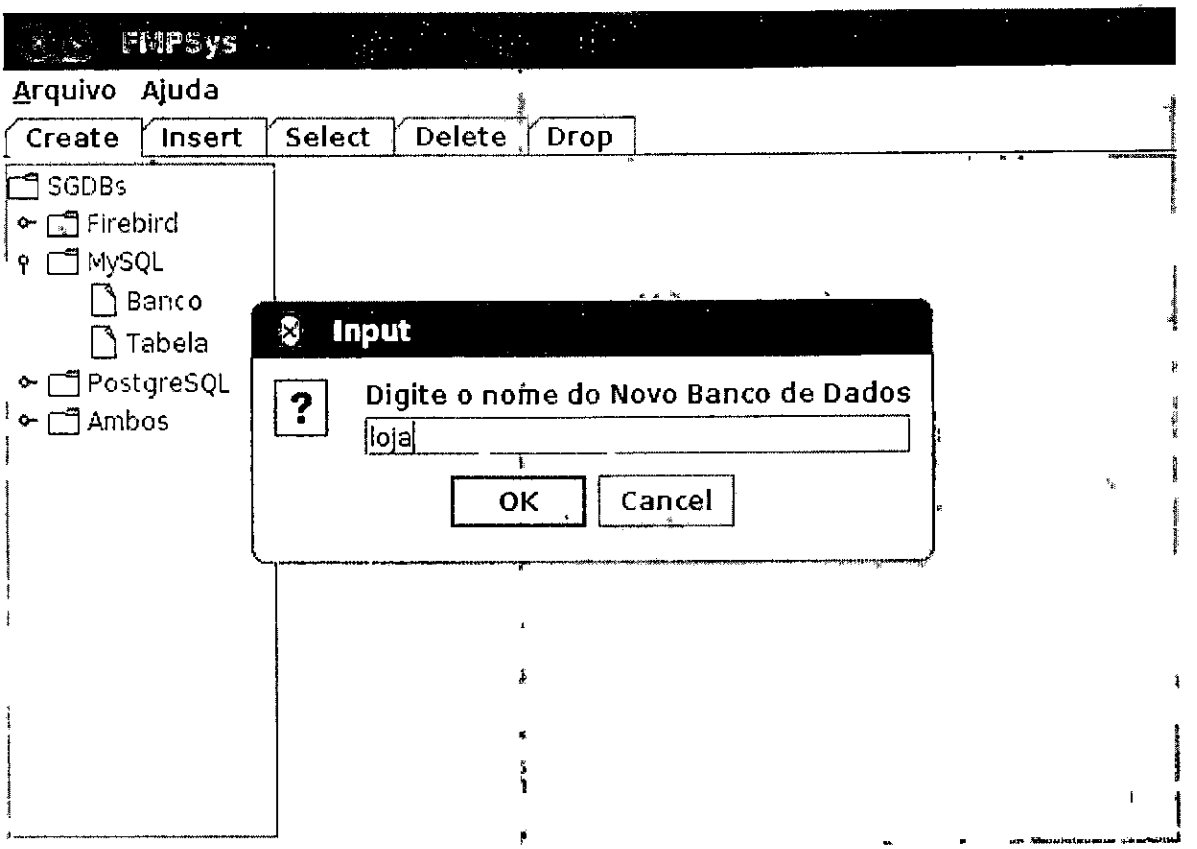


Figura 13 - Criação de um Banco de Dados.

O comando SQL desta ação seria: "CREATE DATABASE <NOME DO BANCO>;".

Criação de Tabela: similar à rotina anterior, escolhe(m)-se o(s) SGDB(s), e em seguida, se já houver algum banco de dados criado, é gerado no formulário uma *Combobox* contendo bases criadas relativas ao SGDB escolhido (no caso se for em Ambos, serão exibidas bases comuns aos SGDBs) para que se selecione uma, e em seguida informa-se o nome da tabela, onde esta ação pode ser cancelada ou invalidade, se já houver uma tabela com o mesmo nome. (*Vide Figura*)

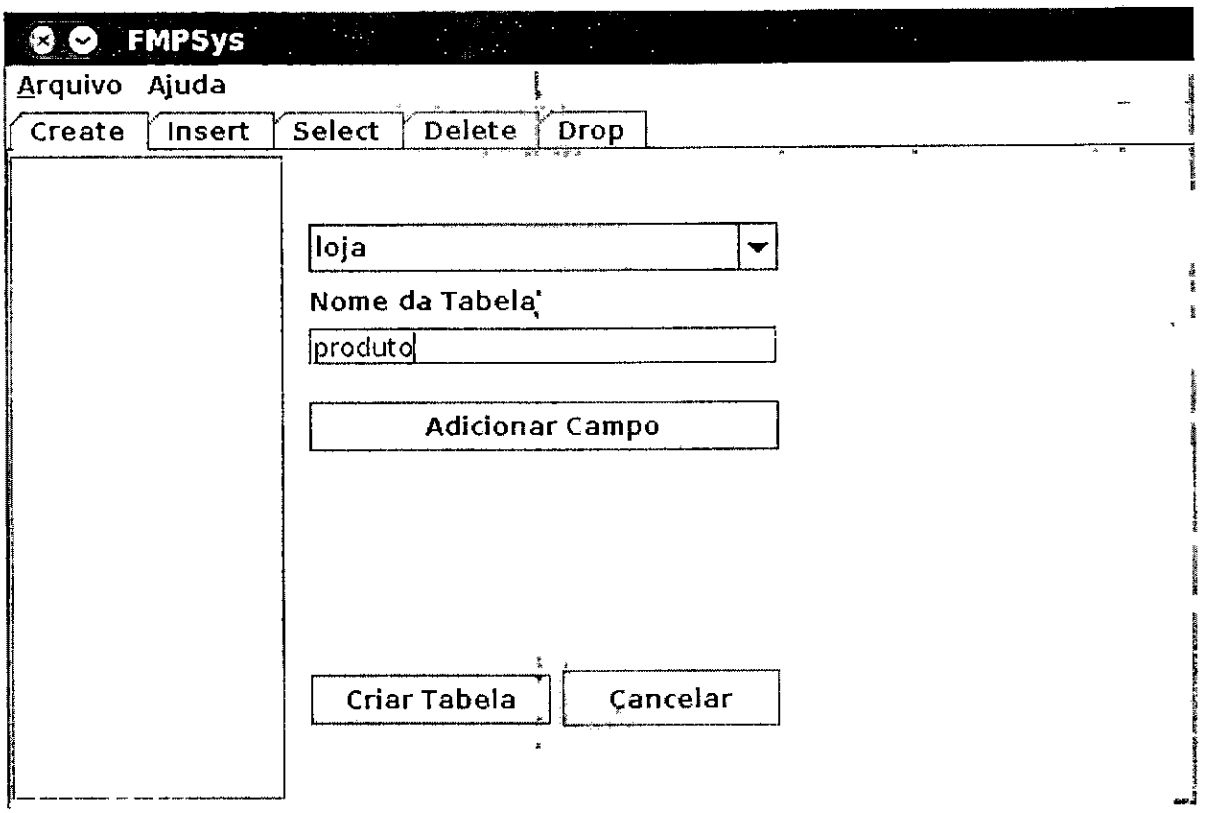


Figura 14 - Tabelas: Criação.

Caso confirmada a criação, para que os campos sejam adicionados, clica-se no botão Adicionar Campo, em que é carregado um novo formulário para a inserção dos campos, sendo nome, tipo, se é chave primária, se é não nulo e tamanho se for varchar. Inseridos os campos desejados, pode-se confirmar, remover ou cancelar a operação.

O comando correspondente em SQL seria:

"CREATE TABLE <NOME DA TABELA>

(<CAMPO(1)> <TIPO DE DADO> <OPÇÃO (CHAVE PRIMÁRIA E OU NÃO NULO)>, <CAMPO(2)> <TIPO DE DADO> <OPÇÃO (CHAVE PRIMÁRIA E OU NÃO NULO)>,

...

<CAMPO(N)> <TIPO DE DADO> <OPÇÃO (CHAVE PRIMÁRIA E OU NÃO NULO)>

);"

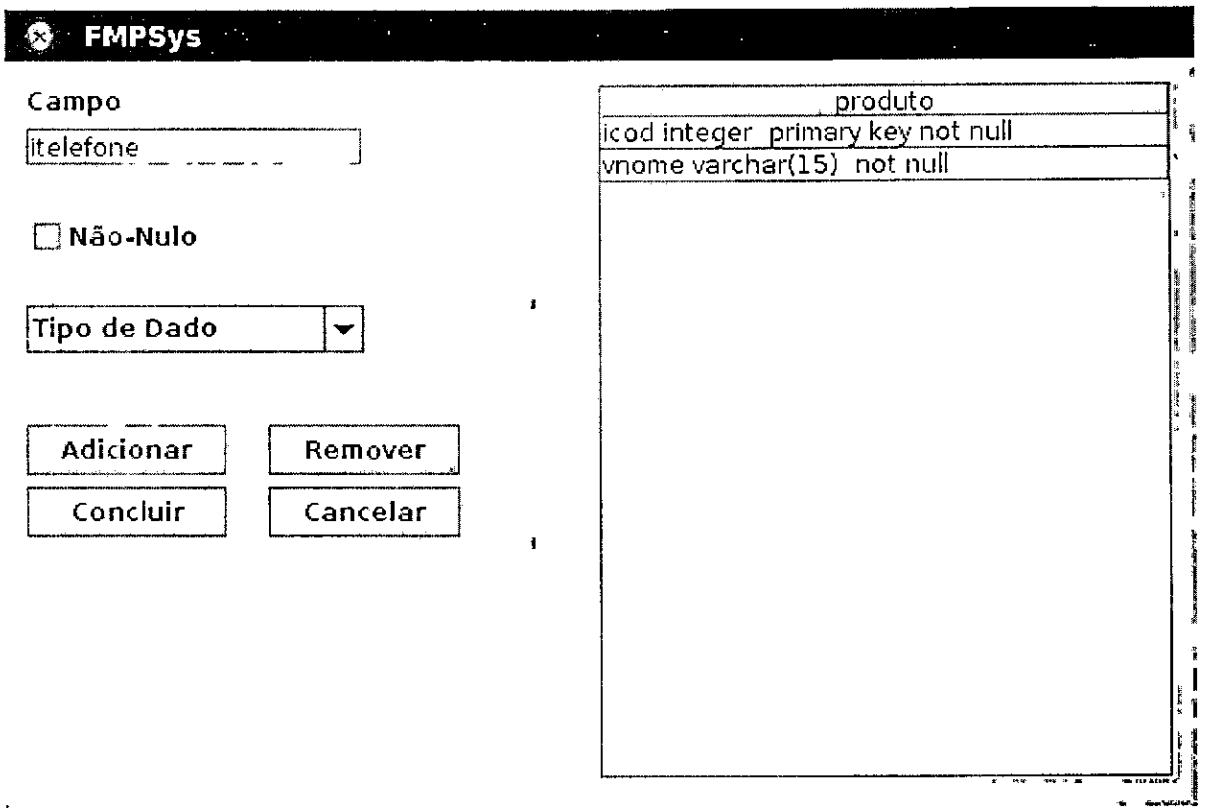


Figura 15 - Formulário de Criação de Tabela.

2. **Insert**: esta operação serve para inserir dados em uma tabela. Parecido com as transações anteriores, tem-se uma árvore com os respectivos SGDDs, onde tendo sido escolhido(s), habilita-se uma *Combobox* para a seleção de uma das bases de dados existente se houver. Caso não haja, uma mensagem é informada e cancela-se a operação. Havendo algum banco de dados, e escolhendo algum, clica-se no botão Banco para confirmar, habilitando assim outra *Combobox* que possui as tabelas existentes. Se houver tabela criada clica-se no botão Inserir, caso não haja, uma mensagem é informada e a operação finda. Ao clicar em Inserir, gera-se um novo formulário com uma tabela contendo colunas, onde respectivamente contém os campos da referida tabela, onde por padrão vem uma linha em branco e novas linha podem ser adicionadas e ou removidas. Inseridos os valores, pode-se cancelar ou concluir esta rotina. Deve-se ter o devido cuidado em saber os tios de cada campo, ou se já não existe uma chave primária criada (integridade), caso algumas dessas opções ocorra, somente até a linha que esteja correta será inserida. Seu comando na linguagem SQL seria:

"INSERT INTO <NOME DA TABELA> (<CAMPO(1)>, ..., <CAMPO(N)>) VALUES (<VALOR CAMPO(1)>, ..., <VALOR CAMPO(N)>);" (*MySQL* e *PostgreSQL*)

"EXECUTE BLOCK AS BEGIN

INSERT INTO <TABELA>

(<CAMPO(1)>,...,CAMPO(N))VALUES(<VALORCAMPO(1)>,...,<VALOR CAMPO(N)>);

...

(<CAMPO(1)>,...,CAMPO(N))VALUES(<VALORCAMPO(1)>,...,<VALOR CAMPO(N)>);

END" (*Firebird* - Suportado desde a versão 2.0)

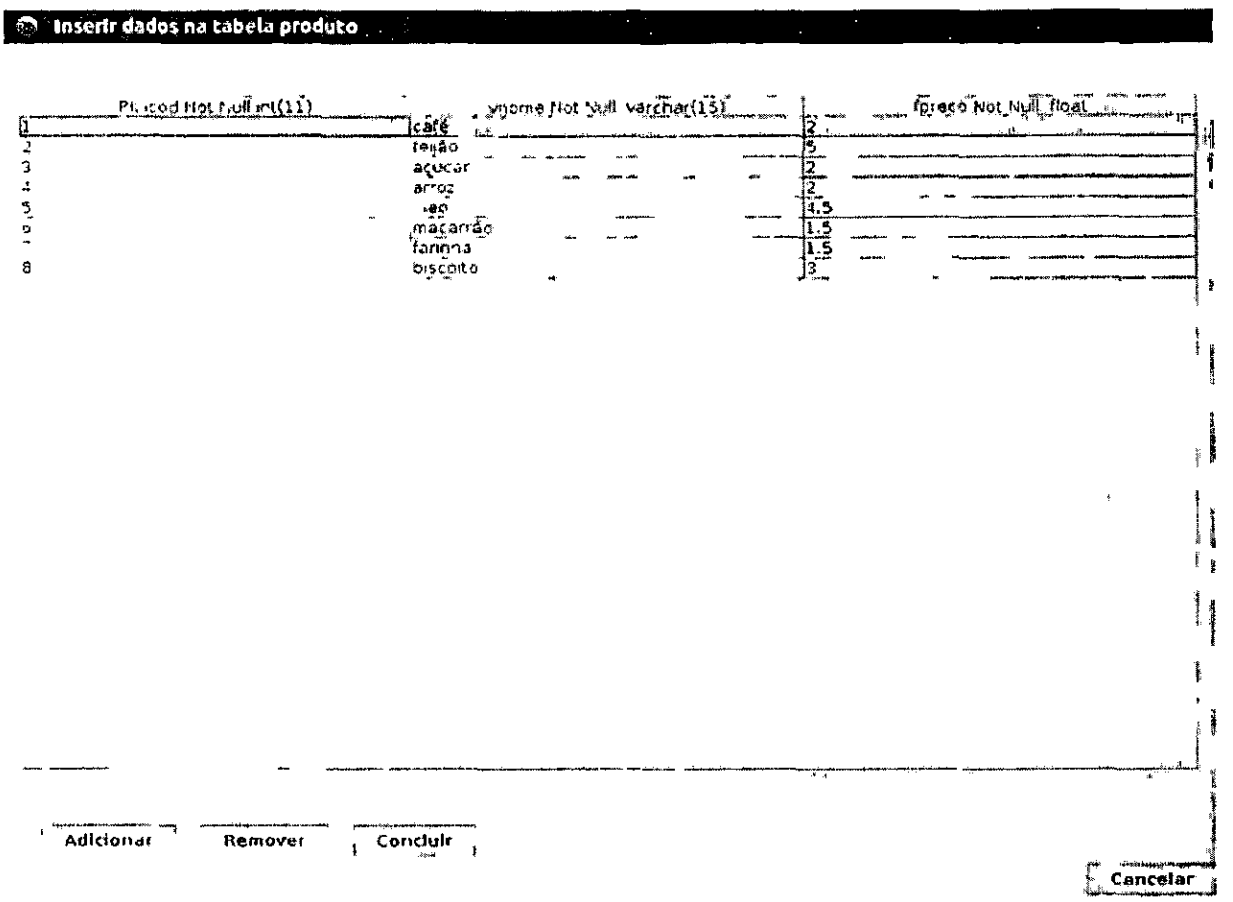


Figura 16 - Formulário de Inserção de Dados.

3. **Select**: esta rotina retorna os dados de uma tabela. Nesta aba também possui uma árvore contendo os SGDBs para escolha. Selecionado o servidor de banco de dados, gera-se uma *Combobox* para seleção de uma base de dados existente, se houver. Caso não exista, uma mensagem informa e a operação é cancelada. Existindo um banco e seja selecionado, clica-se no botão Banco e outra *Combobox* é habilitada com as opções das tabelas existentes na referida base, em que havendo alguma tabela criada, clica-se no botão Tabela e surge outra *Combobox* contendo os campos existentes para escolha. Em seguida, tendo escolhido algum ou todos os campos, surge outro formulário contendo uma tabela com os campos da mesma e seus dados inseridos. Caso não haja dado inserido, uma mensagem informa e a operação é encerrada. Seu comando SQL é dado por:

"SELECT <NOME DO CAMPO OU TODOS(*)> FROM <NOME DA TABELA>;"

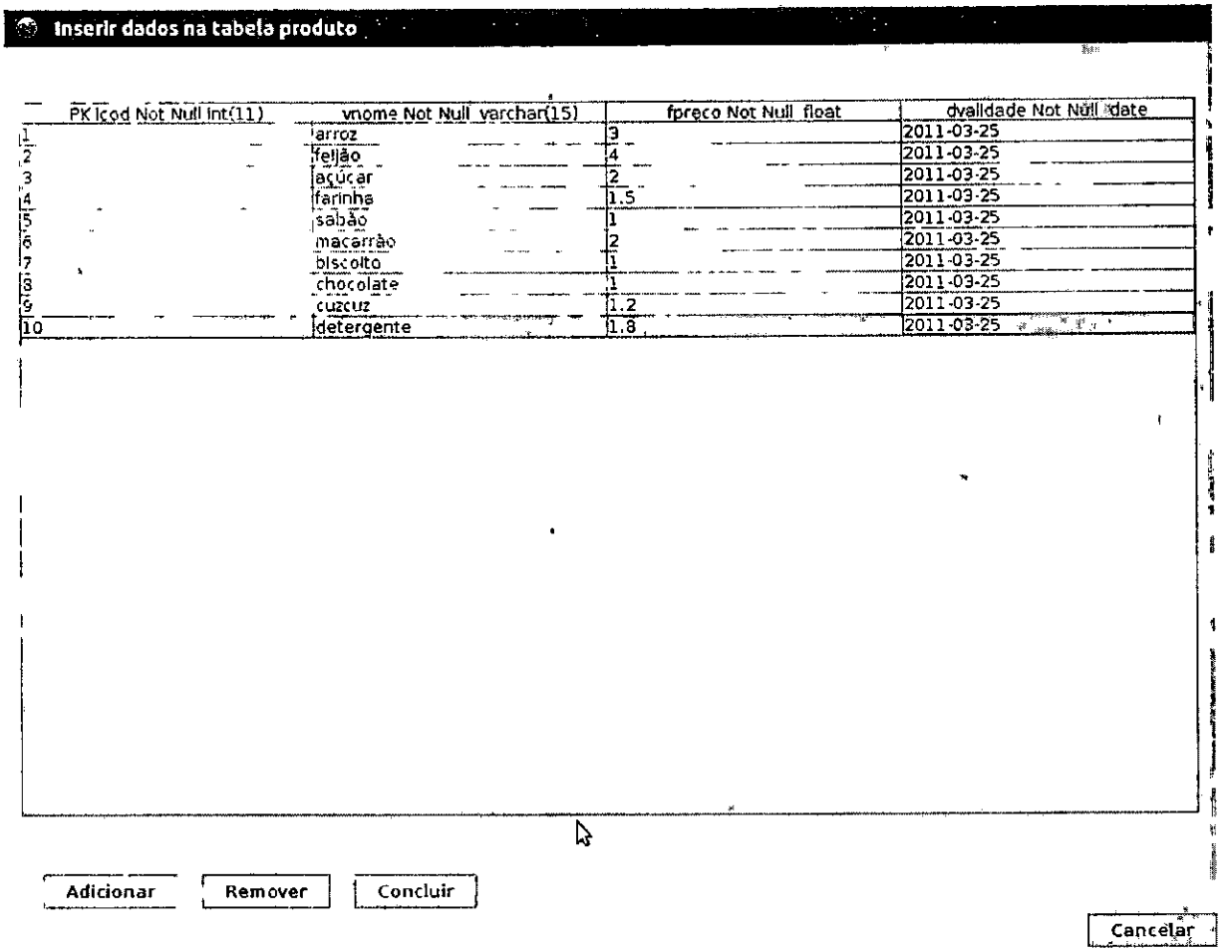


Figura 17 - Aba Select.

4. **Delete**: esta operação serve para apagar os dados de uma tabela. Nesta aba, tem-se habilitado inicialmente uma *Combobox* contendo os SGDBs disponíveis. Escolhendo um deles ou ambos, clica-se no botão Confirma. Em seguida, surge uma nova *combobox* com a(s) base(s) de dados criada(s), se houver(em). Caso não haja, uma mensagem informa ao usuario. Havendo base(s) de dados, escolhe-se uma e clica-se no botão Confirma DB e mais outra *Combobox* é gerada contendo as tabelas do referido banco, caso exista alguma tabela. Não havendo tabelas, uma mensagem é gerada e a transação finda. Se houver e após selecionar alguma tabela, clica-se no botão Delete e a operação é realizada. O correspondente ao comando SQL é dado por:

"DELETE FROM <NOME DA TABELA>;"

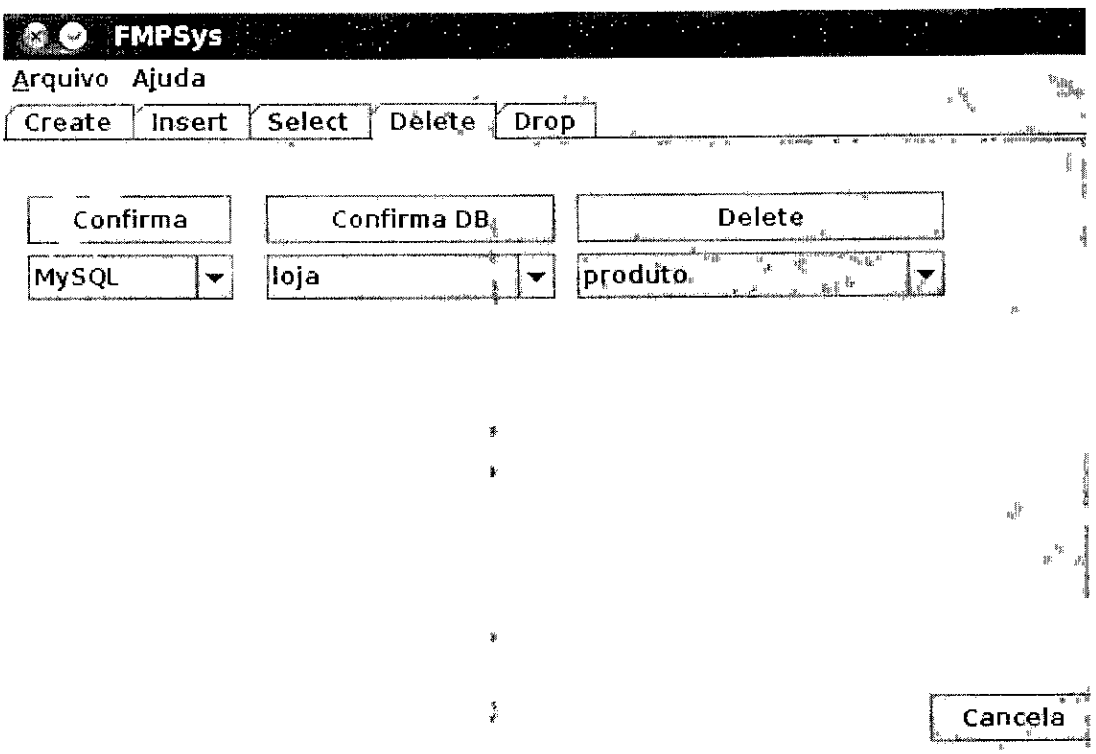


Figura 18 - Aba Delete.

5. **Drop**: esta transação serve para excluir uma tabela ou bancos de dados. Mais uma vez, tem-se uma árvore contendo os SGDBs disponíveis, em que, respectivamente em cada um, possuem a opção de banco ou tabela. Caso escolha banco, uma *Combobox* é gerada contendo as bases criadas, e, existindo alguma, clica-se no botão Drop apagando a base de dados. Não havendo base de dados criada, uma mensagem informa ao usuário. O comando SQL para tal ação é:

"DROP DATABASE <NOME DO BANCO>;"

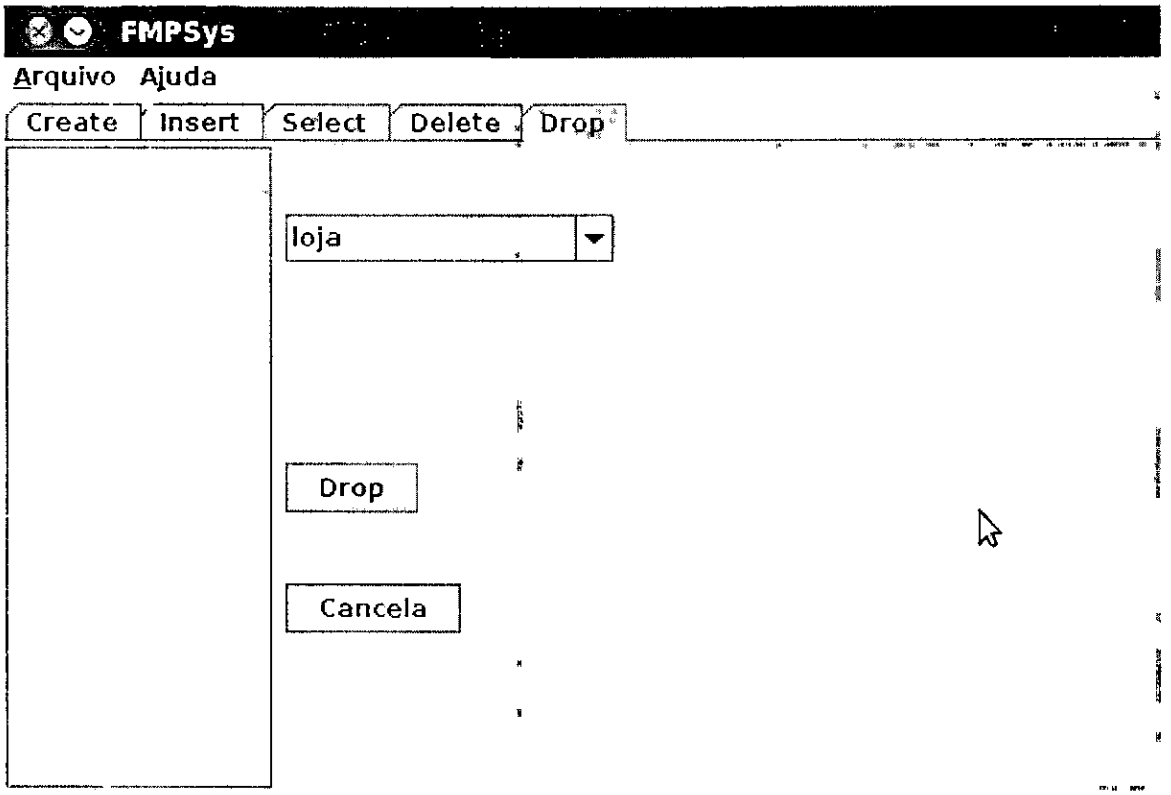


Figura 19 - Aba Drop: Banco.

Se escolher tabela, uma *Combobox* é gerada para a seleção de uma base de dados. Havendo uma base de dados, clica-se em Confirma Banco e outra *Combobox* é habilitada contendo as tabelas existentes. Existindo alguma tabela e a selecionando, clica-se no botão Drop e a mesma será excluída. O comando SQL correspondente é:

"DROP TABLE <NOME DA TABELA>;"

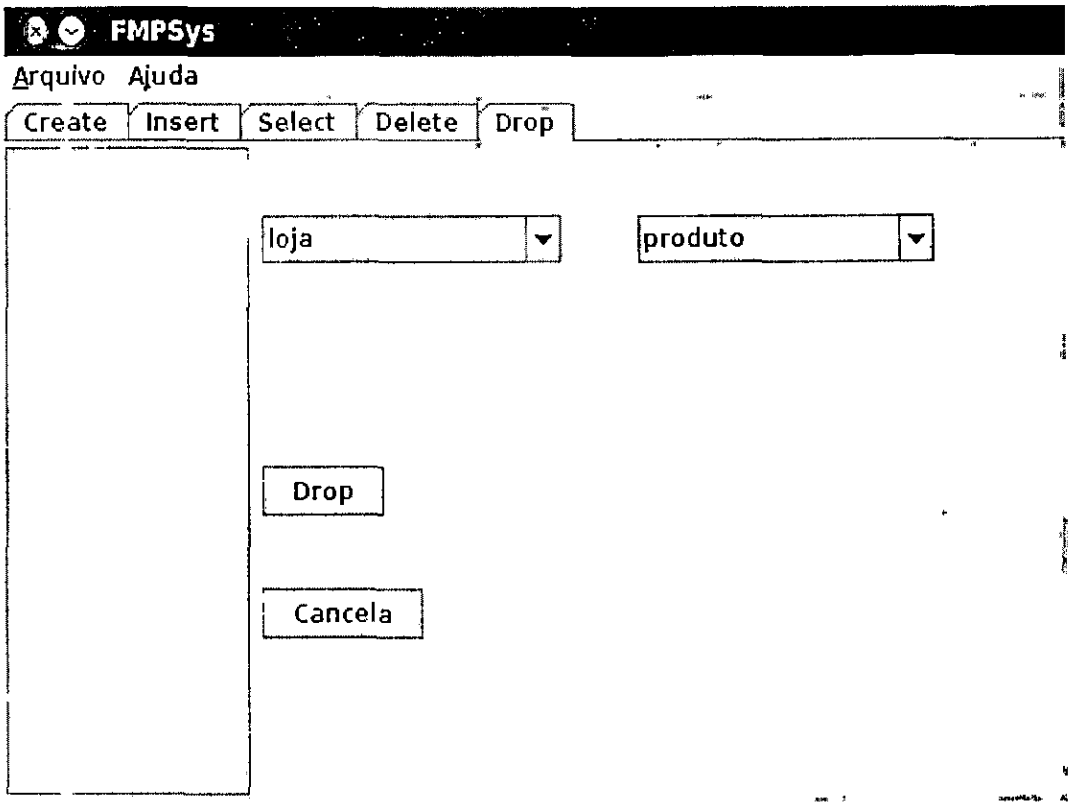


Figura 20- Aba Drop: Tabela

Estas são as funcionalidades presentes no sistema FMPSys que serviram de base para testes entre os bancos de dados *Firebird*, *MySQL* e *PostgreSQL*, onde cada uma foi cronometrada pela própria aplicação e no término de cada rotina, uma mensagem informa o tempo gasto pela mesma. Este tempo foi coletado para formular conclusões a respeito de cada SGDB em relação a cada rotina realizada.

5.2.3 Apresentação dos Dados

Para a referida análise, criou-se em cada SGDB um bancos de dados genérico, chamado "loja" e uma tabela em cada base (loja) dita "produto". Etapa a etapa, coletou-se o tempo gasto, na ordem das abas, das consultas e rotinas possíveis e feitas nos bancos e tabelas, presentes nestas bases. Os atributos da tabela criada são os seguintes com seus respectivos tipos e restrições: icod (Tipo: integer (nº inteiro). Restrições: primary key (chave primária), not null (não-nulo)), vnome (Tipo: varchar(15) (texto de até 15 caracteres). Restrição: not null), fpreco (Tipo: float (nº de ponto flutuante). Restrição: not null) e

dvalidade (Tipo: date (data). Restrição not null).

- Código SQL referente à criação dos bancos de dados: create database loja;
- Código SQL referente à criação da tabela: create table produto (icod integer primary key not null, vnome varchar(15) not null, fpreco float not null, dvalidade date not null).

Formalizados bases e tabelas, são necessários parâmetros para testes, onde estes vão influir diretamente na obtenção de resultados mais estatisticamente concisos. Como parâmetros, escolheu-se a tabela já até mencionada, e trabalhou-se com gamas de dados de 1000, 5000 e 30000 registros.

Dados os pré-requisitos, tem-se o resultado, em média, de cada rotina realizada no sistema FMPSys, na tabela a seguir:

Tabela 4 - Resultados obtidos, em média, das rotinas de criação de banco e tabela, nos gerenciadores de banco de dados utilizados

Criar Banco	*Firebird*	0 minutos, 0 segundos e 320 milésimos de segundo
	MySQL	0 minutos, 0 segundos e 1 milésimo de segundo
	PostgreSQL	0 minutos, 9 segundos e 459 milésimos de segundo
Criar Tabela	*Firebird*	0 minutos, 0 segundos e 24 milésimos de segundo
	MySQL	0 minutos, 0 segundos e 94 milésimos de segundo
	PostgreSQL	0 minutos, 0 segundos e 61 milésimos de segundo

Nas tabelas 5, 6 e 7, encontram-se os resultados obtidos, em média, nos respectivos gerenciadores de banco de dados estudados, em bases populadas sucessivamente com: 1000, 5000 e 30000 registros, nas rotinas existentes do sistema FMPSys.

Tabela 5- Resultados obtidos, em média, relativos a 1000 registros.

1000 Registros	*Firebird*	*Insert*	0 minutos, 0 segundos e 182 milésimos de segundo.
		Select	0 minutos, 0 segundos e 5 milésimos de segundo.
		Delete	0 minutos, 0 segundos e 10 milésimos de segundo.
		Drop Tabela	0 minutos, 0 segundos e 0 milésimos de segundo.
		Drop Banco	0 minutos, 0 segundos e 117 milésimos de segundo.
		Insert	0 minutos, 0 segundos e 12 milésimos

	MySQL		de segundo.
		Select	0 minutos, 0 segundos e 1 milésimos de segundo.
		Delete	0 minutos, 0 segundos e 0 milésimos de segundo.
		Drop Tabela	0 minutos, 0 segundos e 0 milésimos de segundo.
		Drop Banco	0 minutos, 0 segundos e 1 milésimos de segundo.
	PostgreSQL	*Insert*	0 minutos, 0 segundos e 69 milésimos de segundo.
		Select	0 minutos, 0 segundos e 10 milésimos de segundo.
		Delete	0 minutos, 0 segundos e 30 milésimos de segundo.
		Drop Tabela	0 minutos, 0 segundos e 14 milésimos de segundo.
		Drop Banco	0 minutos, 0 segundos e 221 milésimos de segundo.

Tabela 6- Resultados obtidos, em média, relativos a 5000 registros.

5000 Registros	*Firebird*	*Insert*	0 minutos, 0 segundos e 882 milésimos de segundo.
		Select	0 minutos, 0 segundos e 4 milésimos de segundo.
		Delete	0 minutos, 0 segundos e 28 milésimos de segundo.
		Drop Tabela	0 minutos, 0 segundos e 0 milésimos de segundo.
		Drop Banco	0 minutos, 0 segundos e 117 milésimos de segundo.
	MySQL	*Insert*	0 minutos, 0 segundos e 77 milésimos de segundo.
		Select	0 minutos, 0 segundos e 6 milésimos de segundo.
		Delete	0 minutos, 0 segundos e 1 milésimos de segundo.
		Drop Tabela	0 minutos, 0 segundos e 0 milésimos de segundo.
		Drop Banco	0 minutos, 0 segundos e 1 milésimos de segundo.
	PostgreSQL	*Insert*	0 minutos, 0 segundos e 222 milésimos de segundo.
		Select	0 minutos, 0 segundos e 31 milésimos de segundo.
		Delete	0 minutos, 0 segundos e 29 milésimos de segundo.
		Drop Tabela	0 minutos, 0 segundos e 14

			milésimos de segundo.
		Drop Banco	0 minutos, 0 segundos e 228 milésimos de segundo.

Tabela 7- Resultados obtidos, em média, relativos a 30000 registros.

30000 Registros	*Firebird*	*Insert*	0 minutos, 5 segundos e 492 milésimos de segundo.
		Select	0 minutos, 0 segundos e 5 milésimos de segundo.
		Delete	0 minutos, 0 segundos e 124 milésimos de segundo.
		Drop Tabela	0 minutos, segundos e 0 milésimos de segundo.
		Drop Banco	0 minutos, segundos e 119 milésimos de segundo.
	MySQL	*Insert*	0 minutos, 0 segundos e 413 milésimos de segundo.
		Select	0 minutos, 0 segundos e 28 milésimos de segundo.
		Delete	0 minutos, 0 segundos e 3 milésimos de segundo.
		Drop Tabela	0 minutos, 0 segundos e 1 milésimos de segundo.
		Drop Banco	0 minutos, 0 segundos e 48 milésimos de segundo.
	PostgreSQL	*Insert*	0 minutos, 1 segundos e 453 milésimos de segundo.
		Select	0 minutos, 0 segundos e 226 milésimos de segundo.
		Delete	0 minutos, 0 segundos e 107 milésimos de segundo.
		Drop Tabela	0 minutos, 0 segundos e 100 milésimos de segundo.
		Drop Banco	0 minutos, 0 segundos e 359 milésimos de segundo.

Em algumas situações nos testes aplicados, vê-se que rotinas com quantidades de registros distintas são executadas em tempos quase que equivalentes, inclusive quando a quantidade de registros foi maior o tempo diminuiu (*Firebird*: *Select* de 1000 e 5000 registros). Isso se dá pelo fato de os valores dos tempos estarem numa margem de ocorrência, pois existem fatores (hardware e processamento) que interferem no transcorrer das transações, porém não se nota divergências acentuadas.

Baseado nas tabelas 5,6 e 7, nota-se que o *MySQL* destacou-se em relação aos demais, sendo superior na maioria dos testes no que diz respeito ao tempo de processamento das

transações. O *Firebird* por sua vez teve o seu desempenho bem razoável, diminuindo em alguns pontos à medida que a quantidade de dados aumenta. Já o *PostgreSQL* teve seu rendimento um pouco menor quando usam-se poucos registros, estabilizando-se, em média, quando a quantidade de registros inseridos aumentam.

6 CONCLUSÕES

Foi notado, a partir deste trabalho, o quão é importante o uso do computador, através de *Software*, para se automatizar processos. E, para cada rotina específica, existe um programa para realizá-la, onde, quase sempre, tem-se alternativas livres. No tocante do tema abordado, para preencher esta lacuna comentada, existem aplicativos de criação e gerência de dados livres, em que foram escolhidos três dos mais usados para fins de pesquisa: *Firebird*, *MySQL* e *PostgreSQL*.

Buscou-se, ao longo desta pesquisa, estudar conceitos e paradigmas relacionados com esta temática, e a partir daí levantaram-se argumentos para formular a opinião de profissionais e estudantes da computação a respeito de *Software* livre, banco de dados, os SGDBs escolhidos e o desempenho destes em testes específicos realizados. Porém, a escolha em si de um sistema de banco de dados é apenas um ponto no que diz respeito a construção ideal de um projeto, havendo outros tópicos e pré-requisitos pertinentes para tal processo.

Como etapas que necessitam da atenção do desenvolvedor, já até abordados anteriormente, precisam-se levantar requisitos. Estes servirão para se planejar as ações e culminar com o desenvolvimento sadio de um sistema. Em específico, na adoção de um SGDB e modelagem de uma base de dados, tem-se que levar em conta a problemática; se a mesma precisa de fato de um sistema gerenciador de banco de dados, e se precisar, qual das metodologias de modelagem de dados; e seguir as etapas de um projeto de banco de dados.

Explicitadas as definições e etapas necessárias para a modelagem e construção de uma base de dados em um sistema, são buscadas funcionalidades, inerentes ao projeto, em SGDBs, que aperfeiçoem o construir e o funcionar deste. Em particular, nos gerenciadores de bancos de dados escolhidos, foram levantadas e cruzadas informações a respeito destes, ressaltando alguns dos pontos positivos e negativos, traçando o paralelo de ambos. Comprovou-se na prática, por meio do programa FMPSys, a carga e a integridade dos dados, e o tempo gasto em cada transação suportada pelo programa.

Desta maneira, coletadas as informações de cada etapa referida, conclui-se que ambos gerenciadores de banco de dados gozam de características comuns que são importantes. Entretanto, constatou-se que em ordem de grandeza, no que diz respeito às funcionalidades, tem-se, os SGDBs estudados, distribuídos da seguinte forma: *Firebird*, *MySQL* e *PostgreSQL*.

Em situações predeterminadas, abaixo encontram-se, respectivamente, a cada um dos gerenciadores de banco de dados vistos, ocasiões em que o uso destes se adequam melhor,

baseados em nosso estudo e testes:

- ***Firebird***: projetos de pequena à média escala; aplicações desktop; preferencialmente em conjunto com linguagens de programação da empresa de desenvolvimento Borland, é o que diz Freitas et. al. (2007). O mesmo, nestas situações, tem seu potencial melhor usufruído. Como resultado nos testes, mediante o sistema FMPSys e os parâmetros adotados, o mesmo tem sua performance razoável quando tem-se poucos registros, e diminuída, em média, proporcionalmente à quantidade de registros inseridos; e os dados são mantidos .
- ***MySQL***: projetos de pequena e média escala; de acordo com Manzieiro e Sanches (2009) para aplicações *web* e preferencialmente em conjunto com linguagens que tem suporte nativo na plataforma Linux. Como resultado nos testes, mediante o sistema FMPSys e os parâmetros adotados, o mesmo tem sua performance relativamente elevada e estável, em média, diminuindo proporcionalmente à quantidade de registros inseridos.
- ***PostgreSQL***: segundo Scherer et. al. (2007), este é aplicável em projetos de pequena à larga escala; que necessitam de segurança reforçada; preferencialmente em conjunto com plataformas Linux. Como resultado nos testes, mediante o sistema FMPSys e os parâmetros adotados, o mesmo tem sua performance ligeiramente menor quando usam-se poucos registros, tornado-se estável, em média, à medida que a quantidade de registros inseridos aumentam.

Neste sentido percebeu-se, baseado nos testes, que os dados e transações ocorreram sem haver preocupação do sistema FMPSys, relativo à integridade dos mesmos, ou seja, os SGDBs comportaram-se bem quanto às transações e a segurança e disponibilidade dos dados.

Assim, o trabalho foi essencial para mostrar alguns dos mais importantes pontos de vista para a escolha do gerenciador de banco de dados ideal para cada problema em específico, dentre eles: *Firebird, MySQL* e *PostgreSQL*. Este estudo nos remete a refletir no uso racional destes SGDBs, tendo em vista à performance, para que se possa aprimorar a construção e o funcionamento de um *Software*, e que venha a se utilizar de alguns deles.

Neste cenário, acredita-se que a abordagem discutida na realização deste trabalho possa ser significativa, para a pesquisa de banco de dados, no tocante do uso das ferramentas livres *Firebird, MySQL e PostreSQL*. Portanto, este trabalho não se conclui totalmente, pois no meio computacional os conceitos e ferramentas são muito dinâmicos, necessitando assim,

de futuras pesquisas e aprimoramento, com outras funcionalidades e testes, com o aplicativo FMPSys, transportando-o futuramente para a *web*, obtendo dessa maneira mais resultados relevantes com os referidos SGDBs utilizados neste estudo.

REFERÊNCIAS

ABREU, Bruno Rodrigo Cunha. **Unindo Sistemas e Bancos de Dados Distribuídos.** Recife: UFPE, 2006. 42 p. Trabalho de Conclusão de Curso (TCC), curso de Ciência da Computação. Universidade Federal de Pernambuco, Recife, 2006.

BACIC, Nicolas Michel. **O software livre como alternativa ao aprisionamento tecnológico imposto pelo software proprietário**. Trabalho de Conclusão do Curso (TCC), curso de Ciência da Computação. Campinas: UNICAMP, 2003. 135 p. Universidade Estadual de Campinas, Campinas, 2003.

BELINE, William; MENTA, Eziquiel; SALVI, Rosana Figueiredo. **EaD no Mundo Open Source**: Construindo Conhecimento com Liberdade. Paraná, 2006.(dizer o que é)

BUBLITZ, Jorge Luis. **Novidades do Firebird 2.0.** ClubeDelphi, Rio de Janeiro, ano 6, n. 81, 2006.

CAELUM. **Java e Orientação a Objetos -FJ11**. Disponível em: <http;//www.caelum.com.br>. Acesso em: dez. 2010.

CANTU, Carlos Henrique. **Firebird Essencial**. Rio de Janeiro: Editora Ciência Moderna Ltda, 2005.

CASSINO, João. Cidadania Digital: Os Telecentros do Município de São Paulo. In: SILVEIRA, Sérgio Amadeu da & CASSINO, João. (Orgs) **Software livre e Inclusão Digital.** São Paulo: Cortez, 2003.

DATE, C. J.. **Introdução a Sistemas de Banco de Dados**. 8.ed. Rio de Janeiro: Elsevier, 2005.

Elmasri, Ramez; NAVATHE, Shamrant B . **Sistemas de Banco de Dados**. 4. ed. São Paulo: Addison Wesley, 2005.

FREITAS, Gustavo André de; PARIS, Riliane Alpoim. **Firebird**. Trabalho acadêmico apresentado como requisito para avaliação em seminário interdisciplinar, no Curso de Bacharelado em Sistemas de Informação. Linhares: UNILINHARES, 2007.

FERRAZ, Nelson Corrêa de Toledo. **Vantagens do Software Livre para o Ambiente**

Corporativo. São Paulo: PUC, 2002. 114 p. Monografia (Máster Business Information Systems) – Pontifica Universidade Católica, São Paulo, 2002.

GALANTE, Alan Carvalho; MOREIRA, Elvis Leonardo Rangel; BRANDÃO, Flávio Camilo. **Banco de Dados Orientado a Objetos: Uma Realidade**. Macaé, 2007.

HEUSER, Carlos Alberto. **Projeto de Banco de Dados**. 4.ed. Instituto de Informática da URFGS: Sagra Luzzatto, 1998. (Série Livros Didáticos).

HEXSEL, Roberto A. **Software livre: Propostas de Ações de Governo para Incentivar o Uso de software livre**. Curitiba, 2002. 53 f. Relatório Técnico – Departamento de Informática, Universidade Federal do Paraná, 2002. Licença Pública GNU. GNU GENERAL PUBLIC LICENCE Version 2, June 1991. Disponível em: <http://liebr.conectiva.com.br/licenca_gnu.html>. Acesso em: 10 setembro. 2010.

KOFLER, Michael. **The Definitive Guide to MySQL**. 5. 3.ed. Berkeley: Apress, 2007.

MACAFERI, Leniel Braz de Oliveira. **Abordagem TopDown em Banco de Dados Distribuídos**. Trabalho acadêmico realizado na disciplina de Banco de Dados Distribuídos, no Curso de Engenharia de Computação. Recife: UFPE, 2007.

MECENAS, Ivan. **Firebird. SQL magazine**, Rio de Janeiro, ano 3, n. 37, 2006.

MACHADO, Felipe; ABREU, Maurício. **Projeto de Banco de Dados**. 11.ed. São Paulo: Editora Érica Ltda, 1996.

MANZIEIRO, Lucas Bonzanini; SANCHES, Renata Barbosa. **MySQL**. Trabalho acadêmico realizado na disciplina de Sistema de Banco de Dados II, no Curso de Administração com Habilitação em Análise de Sistemas. Tupã: Faculdades Faccat, 2009.

MARQUES, Nuno Miguel Cavalheiro. **Linguagens de Programação**. Universidade Aberta, 2000.

MATTOSO, Marta. **Bancos de Dados Distribuídos**. Universidade Federal do Rio de Janeiro, 1998.

NETO, Álvaro Pereira. **PostgresSQL - Técnicas Avançadas: Versões Open Source 7.x e 8.x**. 4.ed. São Paulo: Érica, 2007.

OZSU, M. Tamer; VALDURIEZ, Patric. **Notes for "Principles of Distributed Database Systems"**. 1999. Disponível em: <http://www.softbase.uwaterloo.ca/ozsu/ddbook/notes/design/design.pdf>. Acesso em: 8 nov. 2010.

REBECCA, M. Riordan. **Microsoft SQL Server 2000 Passo**. São Paulo: Pearson Education, 2002.

RIBEIRO, Daniel Darlen Correa. **Software livre na Administração Pública: estudo de caso sobre a adoção do SAMBA na Auditoria Geral do Estado de Minas Gerais**. Lavras, 2004. 104 f. Monografia (Especialista em Administração em Redes Linux) – Centro de Ciências Exatas, Universidade Federal do Lavras, 2004.

RICARTE, Ivan Luiz Marques. **Sistemas de Bancos de Dados Orientados a Objetos.** Campinas: UEC, 1998.

ROCHA, A. R. R. et al. **Unified Modeling Language**. 2001. Disponível em: <http://www.dei.unicap.br/~almir/seminarios/2001.1/5mno/uml/>. Acesso em 12/11/2010.

SANTOS, Paulo Sérgio M. dos. **Comparando Firebird, MySQL, PostegreSQL e SQLite. SQL magazine.** Rio de Janeiro, ano 4, n. 41, 2007.

SCHERER, Adriana Paula Zamin; JACOBSON, Daniel Gonçaves; SANTOS, Marcelo Luis dos. **PostgreSQL: instalando e conhecendo seus recursos**. Porto Alegre, RS: Faculdade Dom Bosco de Porto Alegre, 2007.

SHIBAYAMA, Eric Teruo. **Estudo Comparativo entre Banco de Dados Distribuídos.** Londrina: UEL, 2004. 65 p. Trabalho de Conclusão de Curso (TCC), curso de Ciência da Computação. Universidade Estadual de Londrina, Londrina, 2004.

SILBERSCHARTZ, Abraham; SUNDARSHA, S; KORTH, Henry F. (trad.) Daniel Vieira. **Sistema de Banco de Dados**. 3ª imp. 5.ed. Rio de Janeiro: Elsevier, 2006.

SILVA, Felipe Augusto; SILVA, Fernanda Brandão. **Máquinas Virtuais Java e. NET.** Campinas, SP: UEC, 2008.

STALLMAN, Richard. (trad.) Fernando Lozano. **O que é Software Livre?** Boston, 2010. Disponível em: <http:// www.gnu.org./philosophy/freesw.pt-br.html>. Acesso em: 13 set. 2010.

TAKAI, Osvaldo Kotaro; ITALIANO, Isabel Cristina; FERREIRA, João Eduardo. **Introdução a Banco de Dados.** São Paulo: DCC-IME-USP, 2005.