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

WELLINGTON NAOKI SONO

ESTUDO COMPARATIVO DE SISTEMAS GERENCIADORES DE BANCO DE DADOS IMPLEMENTADOS COMO APIs JAVA

MARÍLIA
2007

WELLINGTON NAOKI SONO

ESTUDO COMPARATIVO DE SISTEMAS GERENCIADORES DE BANCO DE DADOS IMPLEMENTADOS COMO APIs JAVA

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação do Centro Universitário Eurípides de Marília (UNIVEM), mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Bacharel em Ciência da Computação.

Orientador (a):
Profª. Drª. Fátima L. S. Nunes Marques

MARÍLIA
2007

SONO, Wellington Naoki

Estudo comparativo de Sistemas Gerenciadores de Banco de Dados implementados como APIs Java / Wellington Naoki Sono / Fátima L. S. Nunes Marques. Marília, SP: [s.n], 2007.

Apresentação de Monografia (Graduação em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha.

1. Banco de Dados 2. API Java 3. Software livre

CDD: 005.74

SONO, Wellington Naoki, Estudo comparativo de Sistemas Gerenciadores de Banco de Dados implementados como APIs Java. 2007.
Monografia (Bacharelado em Ciência da Computação) – Curso Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, Marília, 2007.

RESUMO

A distribuição de pequenas aplicações que necessitem de um Banco de Dados pode ser dificultada pela necessidade da instalação de um Sistema Gerenciador de Banco de Dados (SGBD). O objetivo deste trabalho é apresentar um estudo comparativo entre cinco SGBDs gratuitos implementados como APIs (*Application Programming Interface)* Java: Apache Derby, H2 Database, HSQLDB, McKoi, One$DB, que podem ser embarcados dentro de uma aplicação. Foram estudadas as características e tipos de dados suportados de cada SGBD e executada uma análise de desempenho, mostrando quais as vantagens e limitações ao desenvolvimento de *software* utilizando cada um deles.

Palavras-chave: Banco de Dados, API Java, Teste de Desempenho

SONO, Wellington Naoki, Estudo comparativo de Sistemas Gerenciadores de Banco de Dados implementados como APIs Java. 2007.
Monografia (Bacharelado em Ciência da Computação) – Curso Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, Marília, 2007.

ABSTRACT

The distribution of small applications that need a Database can be arduous due to the necessity of the installation of a Database Management System (DBMS). The objective of this work is present a comparative study between five free DBMS implemented as Java APIs: Apache Derby, H2 Database, HSQLDB, McKoi, One$db, that can be embedded in an application. The characteristics and types of supported data of each DBMS were studied and a performance analysis was executed, highlighting the advantages and limitations to the software development by using each one of them.

Keywords: Database, API Java, Performance test

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API: *Application Programming Interface*

BD: Banco de Dados

DDL: *Data Definition Language*

DML: *Data Manipulation Language*

GPL: *General Public License*

JDBC: *Java Database Connectivity*

JVM: *Java Virtual Machine*

MER: Modelo Entidade-Relacionamento

SGBD: Sistema Gerenciador de Banco de Dados

SQL: *Structured Query Language*

TCP/IP: *Transmission Control Protocol / Internet Protocol*

SUMÁRIO

Introdução

Os Sistemas Gerenciadores de Banco de Dados (SGBDs) surgiram para facilitar o armazenamento e o acesso aos dados das aplicações. Desde então, a utilização de SGBDs tem aumentado continuamente, estando presentes nos mais diversos tipos de aplicações, desde as mais robustas de grandes empresas até pequenas aplicações como, por exemplo, em um telefone celular.

Por terem um alto custo, muitos projetos de desenvolvimento de software não dispõem de verba suficiente para obter licenças de utilização de SGBDs comerciais. Uma opção é a utilização de SGBDs gratuitos que têm despontado por disponibilizarem boas ferramentas para uso sem nenhum custo.

Apesar da gratuidade de alguns SGBDs, a distribuição de pequenas aplicações que necessitem de um banco de dados pode ser dificultada pela necessidade da instalação de um SGBD. Desta forma, o estudo de SGBDs que podem ser embarcados em pequenas aplicações torna-se um escopo interessante dentro desta área de pesquisa.

Objetivos

O objetivo geral deste trabalho é apresentar um estudo comparativo entre alguns SGBDs gratuitos, de pequeno porte, que podem ser embarcados em pequenas aplicações. Como objetivos específicos têm-se:

- pesquisar SGBDs gratuitos implementados em Java, que podem ser embarcados dentro de uma aplicação, permitindo que eles sejam distribuídos junto com a aplicação e dispensando sua instalação separadamente;
- instalar e executar os SGBDs encontrados a fim de avaliar possíveis dificuldades e limitações no seu uso;
- executar análise de funcionalidades presentes nos SGBDs pesquisados;
- implementar programa para análise de desempenho dos SGBDs pesquisados, consideração inserção, alteração e exclusão de dados;
- executar análise de desempenho dos SGBDs pesquisados, mostrando quais suas vantagens e limitações em relação ao desenvolvimento de *software* utilizando cada um dos SGBDs pesquisados.

Disposição do trabalho

Além desta introdução, este trabalho apresenta os seguintes capítulos:

- No capítulo 1 abordam-se os conceitos básicos sobre SGBDs e apresentam-se suas vantagens. Também são abordados conceitos sobre Modelo Entidade-Relacionamento, Modelo Entidade-Relacionamento Estendido e Modelo Relacional.
- No capítulo 2 apresentam-se as características e os tipos de dados suportados pelos SGBDs pesquisados.
- No capítulo 3 apresentam-se a Metodologia executada, disponibilizando uma análise comparativa entre os tipos de dados suportados pelos SGBDs, processo de

instalação dos SGBDs, definição de um modelo de dados para teste, definição dos testes, execução dos testes e avaliação dos resultados.

- No capítulo 4 apresentam-se os resultados obtidos a partir dos testes realizados no capítulo 3.
- No capítulo 5 descrevem-se as conclusões que foram obtidas a partir dos testes realizados.

1. BANCOS DE DADOS E SISTEMAS GERENCIADORES DE BANCO DE DADOS

Segundo Silberschatz, Korth e Sudarshan (1999), um Sistema Gerenciador de Banco de Dados (SGBD) é constituído por um conjunto de dados associados a um conjunto de programas para acesso a esses dados. O conjunto de dados, comumente chamado Banco de Dados, contém informações sobre uma empresa em particular. O principal objetivo de um SGBD é proporcionar um ambiente tanto conveniente quanto eficiente para a recuperação e armazenamento das informações do Banco de Dados.

Os Sistemas de Banco de Dados foram criados para gerenciar grandes quantidades de informações. Permitem armazenagem e manipulação de dados, além de garantir a segurança das informações armazenadas contra eventuais problemas com o sistema ou tentativas de acesso não autorizado.

1.1. Vantagens

Antes do aparecimento dos SGBDs, as organizações usavam sistemas de processamento de arquivos para armazenar informações. Nestes sistemas era necessário o desenvolvimento de diversos programas de aplicação que eram usados para extrair e gravar registros nos arquivos apropriados.

Segundo Date (1990), a utilização de Sistemas de Banco de Dados apresenta várias vantagens, entre as quais se podem destacar:

- redução da redundância e inconsistência dos dados: evita que dados sejam duplicados. Em sistemas de processamento de arquivos, os dados são armazenados em vários arquivos, o que pode gerar duplicação dos mesmos dados em locais diferentes. A redundância destes dados pode gerar inconsistência, pois estes arquivos podem ser acessados por diversos programas, sendo possível um programa atualizar os dados apenas em um arquivo, deixando a informação incorreta em outro. O SGBD centraliza todos os dados em um único Sistema de Banco de Dados, evitando esta redundância e, conseqüentemente, a inconsistência;
- eficiência na recuperação da informação: em um sistema de processamento de arquivos é necessário que novas aplicações sejam desenvolvidas para que sejam gerados novos tipos de relatórios; utilizando SGBDs este problema pode ser resolvido facilmente com o uso de linguagens de consulta;
- integridade dos dados: garantia de que os dados estejam corretos de acordo com restrições ou tipos definidos. Em Sistemas de Banco de Dados os tipos de dados e regras que definem certas restrições são definidos no próprio SGBD evitando a necessidade de alterar todas as aplicações que acessam os dados;
- segurança: em Sistemas de Banco de Dados, as informações ficam armazenadas em um único local e são acessadas por vários usuários. O SGBD faz o controle de acesso dando permissão ou não a determinado dado, dependendo do usuário que a esteja requisitando.

1.2. Abstração de dados

Para que se possa usar um sistema, ele precisa ser eficiente na recuperação das informações (Silberschatz, Korth e Sudarshan, 1999).

Esta eficiência está relacionada à forma pela qual foram projetadas as complexas estruturas de representação desses dados no Banco de Dados. Já que muitos dos usuários dos sistemas de bancos de dados não são treinados em computação, os técnicos em desenvolvimento de sistemas omitem essa complexidade da estrutura de representação por meio dos diversos níveis de abstração, de modo a facilitar a interação dos usuários com o sistema:

- nível físico: é o mais baixo nível de abstração que descreve como estes dados estão fisicamente armazenados.
- nível lógico: descreve quais dados estão armazenados no Banco de Dados e quais os inter-relacionamentos entre eles.
- nível de visão: descreve apenas parte do Banco de Dados. Muitos usuários não precisam conhecer todas as suas informações, pelo contrário, os usuários normalmente utilizam apenas parte do Banco de Dados. Assim, para que estas interações sejam simplificadas, um nível de visão é definido. O sistema pode proporcionar diversas visões do mesmo Banco de Dados.

Na Figura 1 ilustra-se o inter-relacionamento entre esses três níveis de abstração.

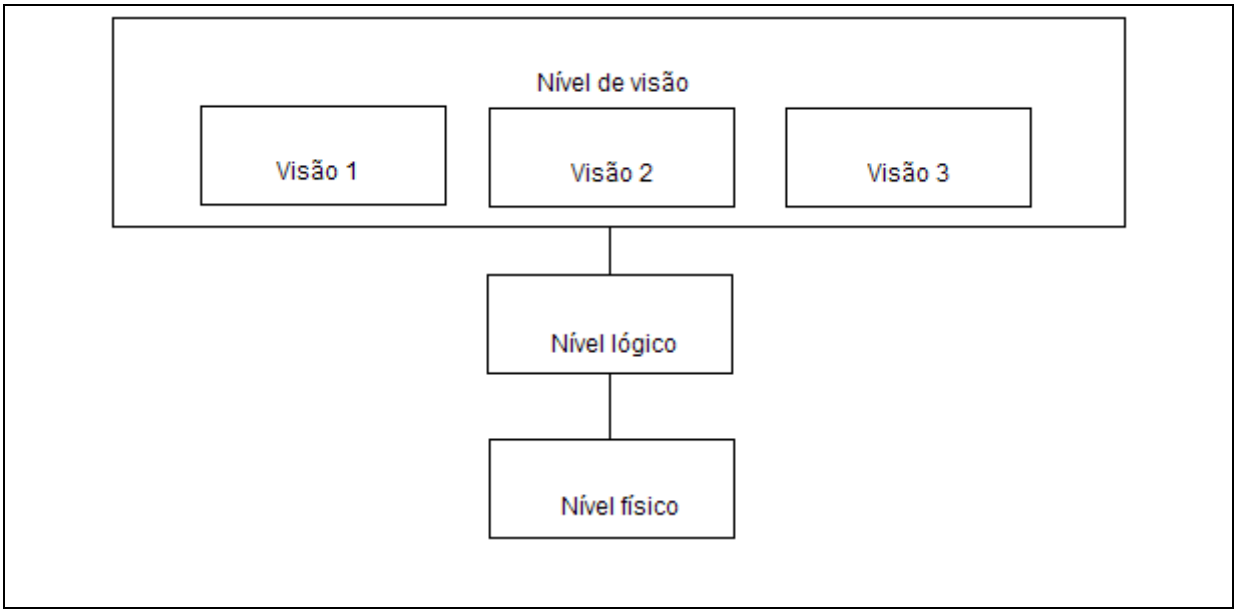


Figura 1 - Níveis de abstração de dados

Fonte: Silberschatz, Korth e Sudarshan, 1999.

1.3. Linguagens de Banco de Dados

Um Sistema de Banco de Dados proporciona dois tipos de linguagens: uma específica para os esquemas do Banco de Dados e outra para expressar consultas e atualizações.

1.3.1. Linguagens de Definição de Dados

Um esquema de dados é especificado por um conjunto de definições expressas por uma linguagem especial chamada Linguagem de Definição de Dados (*Data Definition Language* - *DDL*). O resultado da compilação dos parâmetros DDLs é armazenado em um

conjunto de tabelas que constituem um conjunto especial de arquivos chamado dicionário de dados ou diretório de dados (Silberschatz, Korth e Sudarshan, 1999).

Um dicionário de dados é um arquivo de *metadados*, isto é, dados a respeito de dados. Em um Sistema de Banco de Dados, esse arquivo ou diretório é consultado antes que o dado real seja modificado.

1.3.2. Linguagem de Manipulação de Dados

Os níveis de abstração que foram discutidos anteriormente não se aplicam apenas à definição ou à estrutura dos dados, mas também a sua manipulação. Por manipulação de dados entende-se:

- a recuperação das informações armazenadas no Banco de Dados;
- inserção de novas informações no Banco de Dados;
- a remoção de informações no Banco de Dados;
- a modificação de informações no Banco de Dados.

Silberschatz, Korth e Sudarshan (1999) definem a linguagem de manipulação de dados (*Data Manipulation Language* - DML) como sendo a linguagem que viabiliza o acesso ou a manipulação dos dados de forma compatível ao modelo de dados apropriado. São basicamente dois tipos:

- DMLs procedurais: exigem que o usuário especifique quais dados são necessários, e como obtê-los.

- DMLs não procedurais: exige que o usuário especifique quais os dados são necessários, sem especificar como obtê-los.

1.4. Modelo Entidade-Relacionamento

Segundo Chen (1990), o MER (Modelo Entidade-Relacionamento) é a técnica de modelagem de dados mais utilizada. Ela permite fazer um modelo conceitual dos dados do mundo real que está sendo analisado. Foi criada em 1976 por Peter Chen e baseia-se na idéia de que o mundo real consiste de um conjunto de objetos chamados *entidades* e de um conjunto de relacionamentos entre esses objetos. O Modelo Entidade-Relacionamento baseia-se nos seguintes elementos:

- entidade: é uma representação abstrata de algo do mundo real que pode ser distintamente identificada. São os objetos de significância sobre os quais as informações necessitam ser mantidas. Exemplo: o veículo de placas ABC1234 é uma entidade, o professor José da Silva é uma entidade. Entidades podem ser concretas, conceituais, fatos etc;
- relacionamento: é uma associação entre uma ou mais entidades do mesmo conjunto ou não. Exemplo: José da Silva *possui* um carro de placa ABC1234, neste caso, a entidade José da Silva está relacionada à entidade carro de placa ABC1234;
- atributo: para cada conjunto de entidades, tem-se o interesse em guardar algumas informações relacionadas aos seus elementos, isto é feito através dos atributos.

Exemplo: Carro pode ter seus atributos definidos como placa, chassi, cor, ano, modelo.

1.4.1. Cardinalidade

Os relacionamentos entre entidades possuem importantes restrições chamadas cardinalidades.

Machado e Abreu (1996) definem cardinalidade como sendo o grau do relacionamento entre entidades, sendo expressada pelo número de entidades que pode estar associado via um conjunto de relacionamentos a uma outra entidade. Os tipos de cardinalidade são definidos como:

- um-para-um: neste grau de relacionamento, uma entidade do conjunto A está associada a no máximo uma entidade do conjunto B e uma entidade em B está associada a no máximo uma entidade em A. Por exemplo, em um conjunto de entidades HOMEM, no relacionamento CASAMENTO cada elemento de HOMEM está associado a no máximo um elemento do conjunto de entidades MULHER, assim como cada elemento do conjunto MULHER está associado a no máximo um elemento do conjunto HOMEM;
- um-para-muitos: neste grau de relacionamento, uma entidade do conjunto A pode estar associada a várias entidades do conjunto B, mas cada entidade em B pode estar associada a apenas uma entidade em A. Por exemplo, em um conjunto de entidades PAI, cada entidade pode estar associada a várias entidades no conjunto

de entidades FILHO, porém, cada entidade do conjunto FILHO está associada a apenas uma entidade do conjunto PAI;

- muitos-para-muitos: neste grau de relacionamento, uma entidade do conjunto A pode estar associada a várias entidades do conjunto B, e cada entidade em B pode estar associada a várias entidades em A. Por exemplo, em um conjunto de entidades ALUNO, cada entidade pode estar associada a várias entidades no conjunto de entidades DISCIPLINA, e cada entidade do conjunto DISCIPLINA também pode estar associada a várias entidades do conjunto ALUNO.

1.4.2. Dependência de Existência

A existência de uma entidade pode depender da existência de um outro tipo de entidade. Por exemplo, a existência de entidades no banco de dados depende da existência dos FUNCIONÁRIOS associados. Para que um filho exista deve existir um funcionário que é o seu pai ou mãe, se um funcionário for excluído, as informações sobre seus filhos também devem ser excluídas, pois filho é dependente da existência do funcionário. Por outro lado, um filho pode ser excluído sem a necessidade de se excluir o funcionário, pois funcionário não é dependente da existência do filho. Neste exemplo, a entidade funcionário é chamada *entidade forte* e a entidade filho é chamada *entidade fraca.* Uma entidade fraca depende da existência de outra entidade (CHEN, 1990).

1.4.3. Chaves

É importante especificar como as entidades dentro de um dado conjunto de entidades e os relacionamentos dentro de um conjunto de relacionamentos podem ser identificados.

Conceitualmente, entidades e relacionamentos individuais são distintos, entretanto, na perspectiva do banco de dados, a diferença entre ambos deve ser estabelecida em termos de seus atributos.

Segundo Date (1990), o conceito de *chave* permite-nos fazer tais distinções e define dois tipos de chaves:

- chave primária: a chave primária identifica cada entidade individualmente diferenciado-a de outras entidades dentro de um conjunto. É um atributo único que não se repete entre entidades do mesmo conjunto. Por exemplo, em um conjunto de entidades EMPRESA, pode-se considerar o atributo CNPJ como sendo um candidato a chave primária por ser um identificador único. A chave primária pode também ser composta de dois ou mais atributos, por exemplo, para identificar uma entidade JOGADOR de um determinado time de futebol poderiam ser utilizados os atributos CÓDIGO DO TIME e NÚMERO DA CAMISA para identificar uma entidade JOGADOR específica;
- chave estrangeira: a chave estrangeira identifica outra entidade a qual uma entidade está relacionada. Por exemplo, uma entidade JOGADOR está relacionada a uma entidade TIME, este relacionamento está identificado na entidade JOGADOR pela chave externa CÓDIGO DO TIME que identifica a qual time o jogador pertence.

1.4.4. Modelo Entidade-Relacionamento estendido

Apesar de ser possível modelar a maioria dos Bancos de Dados apenas com os conceitos básicos do MER, alguns aspectos de um Banco de Dados podem ser expressos de modo mais conveniente por meio de algumas extensões do MER básico como: especialização, generalização e agregação (SILBERSCHATZ, KORTH e SUDARSHAN, 1999).

1.4.4.1. Especialização

Segundo Silberschatz, Korth e Sudarshan (1999), um conjunto de entidades pode conter subgrupos de entidades que são, de alguma forma, diferentes de outras entidades do conjunto. Um subconjunto de entidades dentro de um conjunto de entidades pode possuir atributos que não se aplicam às demais entidades do conjunto. Por exemplo, um conjunto de entidades CLIENTE pode conter os atributos de clientes físicos e jurídicos como NOME, ENDEREÇO, TELEFONE, CNPJ, RAZÃO SOCIAL, CPF, etc; apesar de a maioria dos atributos serem comuns a estes dois tipos de clientes, alguns atributos como CNPJ e RAZÃO SOCIAL são aplicados apenas a clientes jurídicos, assim como CPF é aplicado apenas a clientes físicos.

O processo de se projetar os subgrupos dentro de um conjunto de entidades é chamado *especialização*. A especialização de CLIENTE permite-nos distinguir os dois tipos de clientes FÍSICO e JURÍDICO.

1.4.4.2. Generalização

Na prática, a generalização é o processo inverso da especialização. O uso da generalização procede para o reconhecimento de um número de conjuntos de entidades que compartilham características comuns (possuem os mesmo atributos e participam dos mesmo relacionamentos). Com base nestas características comuns, a generalização sintetiza esses conjuntos de entidades em um só conjunto de entidades de nível superior. A generalização é usada para enfatizar as similaridades entre os conjuntos de entidades de nível inferior, omitindo suas diferenças (Silberschatz, Korth e Sudarshan, 1999).

1.4.4.3. Agregação

A agregação é a abstração por meio da qual relacionamentos são tratados como entidades de nível superior. É utilizada quando há a necessidade do estabelecimento de uma associação entre os Conjuntos de Entidades envolvidos em um Conjunto de Relacionamentos.

A situação mais comum ocorre quando há necessidade de associar um relacionamento a um outro relacionamento, o que não é possível no MER. Nesse caso estabelece-se uma agregação (Silberschatz, Korth e Sudarshan, 1999).

Por exemplo, um pedreiro utiliza diversos tipos de materiais de construção em uma obra, para se representar esta associação PEDREIRO, OBRA e MATERIAL é necessário que o relacionamento PEDREIRO-OBRA seja associado à entidade MATERIAL, utiliza-se então a agregação para que o relacionamento PEDREIRO-OBRA seja tratado como uma entidade relacionando-a então a entidade MATERIAL. A Figura 2 ilustra este exemplo:

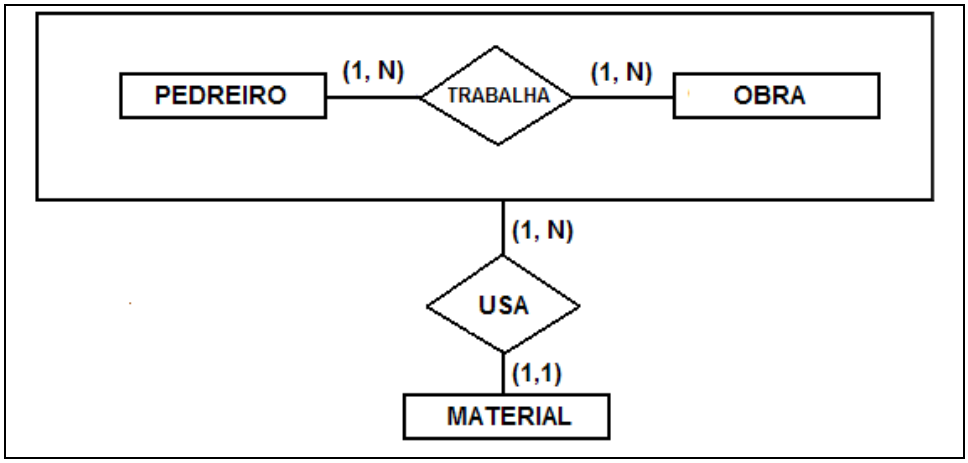


Figura 2 – Exemplo de Agregação

1.4.5. Modelo Relacional

Segundo Guimarães (2003), no modelo relacional a principal construção para representação dos dados são tabelas com linhas não ordenadas e colunas. Uma tabela consiste de um esquema e de uma instância.

O esquema especifica o nome da tabela e o nome do campo e o domínio de cada campo da tabela, também denominado atributo. O domínio do atributo é referenciado no esquema por seu nome e serve para restringir os valores que este atributo pode assumir. Um exemplo de tabela: **Carro** (**chassi**: string, **placa**: string, **modelo**: string, **marca**: string, **ano**: integer).

Neste exemplo está sendo definida a tabela de nome Carro, com os atributos chassi, placa, modelo, marca e ano, cujos domínios são respectivamente string, string, string, string e integer.

A instância de uma relação é o conjunto de linhas, também denominadas tuplas ou registros, que compõem a relação em um dado momento. Ela é variável, já que o número de

tuplas e o conteúdo de seus atributos podem variar ao longo do tempo. A instância de uma tabela deve seguir sempre o seu respectivo esquema, respeitando o número de atributos definidos, bem como os seus domínios.

O modelo relacional somente considera relações que satisfaçam esta restrição. Um exemplo de uma instância para a tabela Carro é ilustrado na Tabela 1.

Tabela 1 - Exemplo de instância para a tabela CARRO no Modelo Relacional

chassi	placa	modelo	marca	Ano
BC9XXZCD90ZGP691410	ABC1234	Gol	Volskwagen	2000
AC3BWZZZ52ZLF582435	XYZ3210	Corsa	GM	2001
BV9WEZKK33ZKP249564	QWE1122	Palio	FIAT	2002

Um banco de dados relacional é um conjunto de uma ou mais tabelas com nomes distintos. O esquema do banco de dados relacional é a coleção dos esquemas de cada tabela que compõe o banco de dados.

1.5. Considerações finais

Este capítulo apresentou uma introdução sobre Banco de Dados e Sistema Gerenciador de Banco de Dados, abordou as vantagens da utilização e as características importantes de um SGBD. Também foram abordados os conceitos sobre Modelo Relacional. Esses conceitos são importantes para a compreensão de SGBDs relacionais que serão apresentados no próximo capítulo.

2. SGBDs GRATUITOS IMPLEMENTADOS EM JAVA

Neste capítulo serão apresentados alguns SGBDs implementados em Java, destacando uma breve história de seu desenvolvimento e suas especificações técnicas.

Os SGBDs escolhidos foram: Derby, H2, HSQLDB, McKoi e One$DB, por serem implementados como APIs Java. Esses SGBDs possuem a característica de poderem ser embarcados em uma aplicação, dispensando assim a necessidade de instalação de um SGBD na máquina em que a aplicação é executada.

2.1. Derby

A primeira versão do Derby foi liberada em 1997 e era chamada de JBMS. Foi desenvolvida pela empresa *Cloudscape*, que foi fundada em 1996 para desenvolver tecnologias de Banco de Dados Java. O produto foi rebatizado para *Cloudscape* e a liberação de novas versões passou a ser feita a cada seis meses (Apache Derby Project Charter, 2007).

Em 1999 a empresa *Cloudscape* foi adquirida pela empresa *Informix Software* que por sua vez teve sua produção de bancos de dados adquirida pela IBM em 2001, incluindo o *Cloudscape*.

Em 2004, a IBM liberou uma cópia do código do *Cloudscape* para a *Apache Software Foundation* que o renomeou para Derby.

O SGBD Derby é um Sistema Gerenciador de Banco de Dados relacional (SGBDR) baseado em SQL.

O Derby pode ser utilizado de três maneiras diferentes:

- incorporado a um aplicativo Java de um único usuário. O Derby pode ficar praticamente invisível ao usuário final porque não requer administração é executado na mesma máquina virtual *Java Virtual Machine* (JVM) que o aplicativo;
- incorporado a um aplicativo multi-usuário como um servidor Web, um servidor de aplicativos;
- incorporado a uma estrutura de servidor. Trabalhando em ambiente cliente/servidor, no qual vários aplicativos se conectam ao Derby através de conexões de rede. Esta estrutura inicia uma instância do Derby e executa em um ambiente incorporado.

O Derby dispõe a utilização de *stored procedures*, *triggers* e integridade referencial, características importantes em um SGBD.

A seguir estão algumas especificações técnicas do Derby segundo o manual de referência (Derby Reference Manual, 2007):

- número máximo de colunas em uma tabela: 1.012;
- número máximo de colunas em uma visão: 5.000;
- número máximo de parâmetros em um procedimento armazenado: 90;
- número máximo de índices em uma tabela: 32.767 ou a capacidade de armazenamento;
- número máximo de tabelas referenciadas em uma instrução SQL ou em uma visão: capacidade de armazenamento;
- número máximo de elementos na lista de seleção: 1.012;
- número máximo de predicados na cláusula WHERE e HAVING: capacidade de armazenamento;

- número máximo de colunas na cláusula GROUP BY: 32.677;
- número máximo de colunas na cláusula ORDER BY: 1.012;
- número máximo de cursores declarados em um programa: capacidade de armazenamento;
- número máximo de cursores abertos ao mesmo tempo: capacidade de armazenamento;
- número máximo de restrições na tabela: capacidade de armazenamento;
- nível máximo de aninhamento da subconsulta: capacidade de armazenamento;
- número máximo de subconsultas em uma única instrução: capacidade de armazenamento;
- número máximo de linhas alteradas em uma unidade de trabalho: capacidade de armazenamento;
- número máximo de constantes em uma instrução: capacidade de armazenamento;
- profundidade máxima de gatilhos em cascata: 16;
- suporte gratuito através de listas de emails (em inglês).

Os tipos de dados suportados pelo Derby estão escritos na Tabela 2:

Tabela 2 - Tipos de dados suportados pelo Apache Derby

Tipo	Faixa / Precisão	Descrição
BIGINT	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Inteiros grandes
BLOB	2.147.483.647 caracteres	Objeto binário de tamanho variável.
CHAR	Até 255 caracteres	Caractere em notação Unicode de 16 bits.
CHAR FOR BIT DATA	1 a 255 bytes	Caracteres binários, tamanho definido.
CLOB	2.147.483.647 caracteres	Dados de tamanho fixo ou variável no formato de caracteres
DATE	0001-01-01 a 9999-12-31	Data. Exibido como YYYY-MM-DD
DECIMAL	Sem limite	Precisão especificada pelo usuário
DOUBLE	-1.7976931348623157E+308 até -2.2250738585072014E-308, 0 e desde 2.2250738585072014E-308 até 1.7976931348623157E+308	Números de ponto flutuante de precisão dupla. O mesmo que FLOAT(8)
FLOAT(precisão)	Depende da precisão	Números de ponto flutuante de precisão simples ou dupla
INTEGER	-2.147.483.648 a 2.147.483.647	Inteiros regulares
LONG VARCHAR	32.700 caracteres	Idêntico a VARCHAR, exceto por não ser necessário especificar o comprimento máximo ao criar colunas deste tipo
Tipo	Faixa / Precisão	Descrição
LONG VARCHAR FOR BIT DATA	32.700 bytes	Cadeias de bits de até 32.700 bytes. É idêntico a VARCHAR FOR BIT DATA, exceto por não ser necessário especificar o comprimento máximo ao criar colunas deste tipo.
REAL	-	O mesmo que DOUBLE
SMALLINT	-32.768 a 32.767	Inteiros pequenos
TIME	00:00:00 a 24:00:00	Hora. Exibido como HH:MM:SS
TIMESTAMP	0001-01-01-00.00.00.000000 a 9999-12-31-24.00.00.000000	yyyy-mm-dd hh:mm:ss[.nnnnnn]
VARCHAR(n)	32.672 caracteres	String de tamanho variável
VARCHAR FOR BIT DATA	32.672 caracteres	Cadeias binárias menores ou iguais ao comprimento especificado

2.2. H2 Database

O início do desenvolvimento do H2 Database deu-se em maio de 2004, mas sua primeira publicação foi em 14 de dezembro de 2005. O autor do H2, Thomas Mueller, foi

também o colaborador original do *SQL Hypersonic* (que originou o HSQLDB) (H2 Database Documentation, 2007).

Em 2001, Thomas Mueller juntou-se à empresa *PointBase*, onde criou a *PointBase micro*, parando então de colaborar com o projeto *SQL Hypersonic*. Para dar continuidade aos trabalhos com o código fonte do *SQL Hypersonic* foi criado o grupo HSQLDB.

O nome H2 veio de "*Hypersonic 2*", entretanto o H2 não compartilha nenhum código com o *SQL Hypersonic* ou o HSQLDB.

O foco do H2 são aplicações com poucas conexões concorrentes e sendo usado embarcado na aplicação, embora também possa ser usado em modo cliente/servidor.

A seguir estão algumas características do H2 segundo a documentação disponibilizada pelo fabricante (H2 Database Documentation, 2007):

- *views;*
- *subqueries;*
- *triggers;*
- *clustering*;
- criptografia;
- *stored functions*;
- armazenamento em disco ou em memória;
- integridade referencial;
- isolamento de transações;
- suporta ORDER BY, GROUP BY, HAVING, UNION, LIMIT, TOP;
- suporte gratuito através de listas email e grupo de discussão no site do desenvolvedor (em inglês).

Os tipos de dados suportados pelo H2 estão descritos na Tabela 3:

Tabela 3 - Tipos de dados suportados pelo H2 Database

Tipo	Faixa / Precisão	Descrição
BIGINT	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Inteiros grandes
BINARY(N)	Até 8.000 bytes	Tamanho fixo do campo, sempre será utilizado o tamanho definido.
BLOB	-	Objeto binário de tamanho variável.
BOOLEAN	-	TRUE ou FALSE
CLOB	-	Igual VARCHAR, mas pretendido para valores muito grandes.
DECIMAL	Sem limite	Precisão especificada pelo usuário
DATE	0001-01-01 a 9999-12-31	Data. Exibido como YYYY-MM-DD
DOUBLE	-1.7976931348623157E+308 até -2.2250738585072014E-308, 0 e desde 2.2250738585072014E-308 até 1.7976931348623157E+308	Números de ponto flutuante de precisão dupla. O mesmo que FLOAT(8)
IDENTITY	-9223372036854775808 a 9223372036854775807	Auto-incremento
INT	-2.147.483.648 a 2.147.483.647	Inteiros regulares
REAL	-	O mesmo que DOUBLE
SMALLINT	-32.768 a 32.767	Inteiros pequenos
TIME	00:00:00 a 24:00:00	Hora. Exibido como HH:MM:SS
TIMESTAMP | DATETIME	0001-01-01-00.00.00.000000 a 9999-12-31-24.00.00.000000	yyyy-mm-dd hh:mm:ss[.nnnnnn]
TINYINT	-128 a 127	O menor tipo de dado inteiro.
VARCHAR	Sem limite	String de tamanho variável
VARCHAR_IGNORECASE	Sem limite	Tipo especial de varchar case-insensitive

2.3. HSQLDB

O desenvolvimento do HSQLDB teve início em 1998, quando foi lançado o projeto *Hypersonic SQL*, de Thomas Mueller, mas em 2000 o projeto foi encerrado. Foi formado então o "*HSQLDB Development Group*" e o projeto foi reiniciado, lançando a primeira versão do HSQLDB (1.6) em Abril de 2001. Desde então vem sendo lançadas novas versões mais aprimoradas (Hsqldb User Guide, 2007).

O HSQLDB é um SGBD relacional, de código aberto. Não é possível compará-lo, em termos de robustez e segurança com outros servidores SGBD, como Oracle ou Microsoft

SQL Server, entretanto o HSQLDB é uma solução simples, que utiliza poucos recursos e que possui bom desempenho.

Caracteriza-se por ser um SGBD pequeno e rápido, que suporta tabelas em disco e em memória. Pode ser usado em *applets*, mídias somente de leitura (CD-ROM) e aplicações embarcadas.

Apesar de algumas limitações, em comparação com outros SGBDs de grande porte, o HSQLDB é *freeware* e apresenta um bom desempenho. O que leva o sistema a ser utilizado por aplicações bastante conhecidas como o OpenOffice.

O HSQLDB é compatível com os comandos SQL e possui driver para acesso através do JDBC. Possui suporte a chaves estrangeiras, *views*, tabelas temporárias, seqüências, transações, sub-consultas, *triggers* e funções para a segurança dos dados. Oferece três modos de funcionamento: Cliente-Servidor, Embarcado no Aplicativo ou apenas na memória.

No modo Servidor, o HSQLDB funciona similarmente aos SGBDs mais populares, como MySql, PostgreSql, Oracle. Ele é executado como um processo à parte e os aplicativos que necessitarem de seus serviços, conectam-se a ele através de uma conexão TCP/IP.

O modo embarcado executa o Banco de Dados como parte do seu aplicativo. Para algumas aplicações este modo poderá ser mais rápido, pois haverá uma economia de recursos, visto que os dados não são convertidos e enviados pela rede. O empecilho deste modo é que não é possível conectar-se ao Banco de Dados de fora da aplicação. Desta forma, não é possível checar o conteúdo dos dados com ferramentas de gerenciamento de Banco de Dados enquanto o aplicativo é executado. Recomenda-se que no desenvolvimento do aplicativo utilize-se o modo Servidor para que se possa empregar ferramentas de gerenciamento, e quando for distribuí-lo, substitua para o modo embarcado.

O modo “Apenas Memória” é muito similar ao embarcado, sendo a diferença que ao invés do Banco de Dados ficar gravado em arquivos, ele estará somente na memória,

permitindo acesso mais rápido aos dados e utilização do Banco de Dados quando não há permissão de acesso em disco, por exemplo, *Applets*.

A quantidade de tabelas, colunas, tamanho de *strings* e dados binários são limitados somente pela quantidade de memória disponível no computador.

A seguir estão algumas características do HSQLDB:

- 100% Java;
- possui as operações básicas de SQL, faz *inner* e *outer joins*;
- suporte a auto-incremento;
- transações COMMIT, ROLLBACK e SAVEPOINT;
- integridade referencial;
- *stored procedures* e *triggers;*
- em memória (como *applets*), embutido (aplicações Java) e modos operacionais com Cliente-servidor;
- pode ser usado dentro de Jars, num CD, em sistema embarcado, etc;
- suporte a JDBC;
- suporte gratuito através de listas de emails e suporte comercial pago (em inglês).

Os tipos de dados suportados pelo HSQLDB estão descritos na Tabela 4:

Tabela 4 - Tipos de dados suportados pelo HSQLDB

Tipo	Faixa / Precisão	Descrição
BIGINT	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Inteiros grandes
BINARY(N)	Até 8.000 bytes	Tamanho fixo do campo, sempre será utilizado o tamanho definido.
BOOLEAN	False, true ou null	False, True ou NULL
CHAR | CHARACTER	Até 255 caracteres	Caractere em notação Unicode de 16 bits.
DATE	0001-01-01 a 9999-12-31	Data. Exibido como YYYY-MM-DD
DECIMAL	Sem limite	Precisão especificada pelo usuário
DOUBLE	-1.7976931348623157E+308 até -2.2250738585072014E-308, 0 e desde 2.2250738585072014E-308 até 1.7976931348623157E+308	Números de ponto flutuante de precisão dupla. O mesmo que FLOAT(8)
INTEGER | INT	-2.147.483.648 a 2.147.483.647	Inteiros regulares
LONGVARCHAR	32.700 caracteres	Idêntico a VARCHAR, exceto por não ser necessário especificar o comprimento máximo ao criar colunas deste tipo
NUMERIC	Sem limite	Precisão especificada pelo usuário
REAL	-1.7976931348623157E+308 até -2.2250738585072014E-308, 0 e desde 2.2250738585072014E-308 até 1.7976931348623157E+308	O mesmo que DOUBLE
SMALLINT	-32.768 a 32.767	Inteiros pequenos
TIME	00:00:00 a 24:00:00	Hora. Exibido como HH:MM:SS
TIMESTAMP | DATETIME	0001-01-01-00.00.00.000000 a 9999-12-31-24.00.00.000000	yyyy-mm-dd hh:mm:ss[.nnnnnn]
TINYINT	-128 a 127	O menor tipo de dado inteiro.
VARBINARY(N)	8.000 bytes de dados binários	O mesmo que BINARY, porem utiliza apenas o espaço necessário.
VARCHAR	32.672 caracteres	String de tamanho variável
VARCHAR_IGNORECASE	32.672 caracteres	Tipo especial de varchar case-insensitive

2.4. McKoi

O início do projeto do Banco de Dados McKoi SQL deu-se em 1998 e desde lá tem evoluído muito. O objetivo principal do projeto era desenvolver um Banco de dados simples de manter, fácil de utilizar e administrar, robusto, que suportasse múltiplos acessos simultâneos e alto desempenho (McKoi SQL Database Manual, 2007).

O McKoi SQL Database é um Banco de Dados relacional de código aberto. É otimizado para executar como um Banco de Dados cliente/servidor com múltiplos clientes, mas também pode ser embarcado em uma aplicação Java.

Pode ser usado em qualquer Sistema Operacional, é pequeno, fornece um driver JDBC e é distribuído sob a licença GNU/GPL.

A seguir estão algumas características do McKoi:

- suporte a *Transactions*;
- suporte a integridade referencial;
- suporte a *Triggers;*
- suporte a índices;
- suporte a multi-tarefa;
- suporte a multi-usuário;
- embarcável;
- suporte gratuito através de listas de email (em inglês).

Os tipos de dados suportados pelo McKoi estão descritos na Tabela 5:

Tabela 5 - Tipos de dados suportados pelo McKoi

Tipo	Faixa / Precisão	Descrição
BOOLEAN	True, false ou null	TRUE, FALSE ou NULL
BIGINT	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Inteiros grandes
BYNARY(N)	2.147.483.647 caracteres	Tamanho fixo do campo, sempre será utilizado o tamanho definido.
BLOB	2.147.483.647 caracteres	Objeto binário de tamanho variável.
CHAR	Até 254 caracteres	Caractere em notação Unicode de 16 bits.
CLOB	2.147.483.647 caracteres	Dados de tamanho fixo ou variável no formato de caracteres
DATE	0001-01-01 a 9999-12-31	Data. Exibido como YYYY-MM-DD
DECIMAL	Sem limite	Precisão especificada pelo usuário
DOUBLE	±1.7976931348623157E+308 a ±2.2250738585072014E-308	Números de ponto flutuante de precisão dupla. O mesmo que FLOAT(8)
INTEGER	-2.147.483.648 a 2.147.483.647	Inteiros regulares
NUMERIC	Sem limite	Precisão especificada pelo usuário
REAL	-3.402E+38 a 3.402E+38	O mesmo que DOUBLE
SMALLINT	-32.768 a 32.767	Inteiros pequenos
TIME	00:00:00 a 24:00:00	Hora. Exibido como HH:MM:SS
TIMESTAMP	0001-01-01-00.00.00.000000 a 9999-12-31-24.00.00.000000	yyyy-mm-dd hh:mm:ss[.nnnnnn]
VARCHAR(n)	32.672 caracteres	String de tamanho variável
JAVA_OBJECT	2.147.483.647 caracteres	Igual ao BLOB

2.5. One$DB

One$db é um Banco de Dados relacional versão de código aberto do Banco de Dados Daffodil (RDBMS comercial da empresa Daffodil Software), e teve seu lançamento em 3 de novembro de 2004. É livre tanto para o uso comercial quanto para uso doméstico. O One$db é um Banco de Dados robusto, e herda todos os princípios do Banco de Dados Daffodil tais como a independência de plataforma, a administração zero e o suporte a JDBC/ODBC. Ele é disponibilizado em duas edições diferentes: embarcada e rede (Daffodil DB SQL Reference Guide, 2007).

A edição embarcada é apropriada para as aplicações multi-tarefas que requerem uma base de dados embarcada. Suporta múltiplas conexões concorrentes e foi projetada

especificamente para ser incluída em aplicações Java. Embarcado em uma aplicação Java mono-usuário, o One$DB pode ser executado praticamente invisível ao usuário, pois neste modo ele não requer nenhuma administração e é executado na mesma máquina virtual Java (JVM) da aplicação.

O One$DB não impõe nenhuma limitação em consideração ao número de conexões, do uso do processador e do armazenamento de dados total. Estes valores ficam limitados apenas pela capacidade do computador onde é executado.

A seguir estão algumas características do One$DB:

- *full-text search;*
- suporte a PSM (*Persistent Stored Module*);
- armazenamento de dados criptografados;
- *triggers;*
- suporte para *Large Object Data types* (BLOB and CLOB);
- *stored procedures;*
- índices;
- *sequences;*
- integridade referencial;
- *open source*, o suporte é feito em grupos de discussões.

Os tipos de dados suportados pelo One$DB estão descritos na Tabela 6:

Tabela 6 - Tipos de dados suportados pelo One$DB

Tipo	Faixa / Precisão	Descrição
BIGINT	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Inteiros grandes
BINARY(N)	Até 8.000 bytes	Tamanho fixo do campo, sempre será utilizado o tamanho definido.
BLOB	2.147.483.647 caracteres	Objeto binário de tamanho variável.
BOOLEAN	True ou false	TRUE ou FALSE
BYTE	-128 a 127	Inteiro de 8 bits em notação de complemento de dois.
CHAR	Até 254 caracteres	Caractere em notação Unicode de 16 bits.
CLOB	-	Igual VARCHAR, mas pretendido para valores muito grandes.
DATE	0001-01-01 a 9999-12-31	Data. Exibido como YYYY-MM-DD
DECIMAL	Sem limite	Precisão especificada pelo usuário
DOUBLE	de -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 e de 2.2250738585072014E-308 a 1.7976931348623157E+308	Números de ponto flutuante de precisão dupla. O mesmo que FLOAT(8)
INTEGER | INT	-2.147.483.648 a 2.147.483.647	Inteiros regulares
LONGVARCHAR	32.700 caracteres	Idêntico a VARCHAR, exceto por não ser necessário especificar o comprimento máximo ao criar colunas deste tipo
LONG VARCHAR FOR BIT DATA	32.700 bytes	Cadeias de bits de até 32.700 bytes. É idêntico a VARCHAR FOR BIT DATA, exceto por não ser necessário especificar o comprimento máximo ao criar colunas deste tipo.
NUMERIC	Sem limite	Precisão especificada pelo usuario
REAL	de -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 e de 2.2250738585072014E-308 a 1.7976931348623157E+308	O mesmo que DOUBLE
SMALLINT	-32.768 a 32.767	Inteiros pequenos
TIME	00:00:00 a 24:00:00	Hora. Exibido como HH:MM:SS
TIMESTAMP	0001-01-01-00.00.00.000000 a 9999-12-31-24.00.00.000000	yyyy-mm-dd hh:mm:ss[.nnnnnn]
TINYINT	-128 a 127	O menor tipo de dado inteiro.
VARBINARY(N)	8.000 bytes de dados binários	O mesmo que BINARY porem utiliza apenas o espaço necessário.
VARCHAR	32.672 caracteres	String de tamanho variável

2.6. Considerações finais

Este capítulo apresentou os SGBDs implementados como APIs Java (Derby, H2 Database, HSQLDB, McKoi e One$DB). Foi abordada uma breve história sobre seus desenvolvimentos, suas principais características e os tipos de dados suportados. A Metodologia para avaliar esses SGBDs é apresentada no próximo capítulo.

3. METODOLOGIA

Neste capítulo, é apresentada a definição do modelo de dados a ser utilizado nos testes utilizando os tipos de dados compatíveis entre os SGBDs, a descrição da instalação de cada um dos SGBDs e definição dos testes de integridade e desempenho a serem realizados.

Como foi citado anteriormente os SGBDs escolhidos foram: Apache Derby, H2 Database, HSQLDB, McKoi e One$DB, por possuírem as características de serem implementados como APIs Java e poderem ser embarcados dentro de uma aplicação. A linguagem utilizada para o desenvolvimento das aplicações dos testes foi a linguagem Java pelo motivo de os SGBDs escolhidos serem APIS Java.

Figura 3 – Fases da Metologia utilizada

3.1. Comparação entre os diversos tipos de dados

Antes da construção do modelo de dados, foi feito um comparativo dos tipos de dados suportados entre os SGBDs para que se pudessem definir os tipos de dados compatíveis entre eles, conforme é mostrado na Tabela 7.

Tabela 7 - Tipos de dados correspondentes entre os SGBDs

Tipo	Apache Derby	H2 Database	HSQLDB	McKoi	One$DB
BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT
BINARY(N)	CHAR FOR BIT DATA	BINARY(N)	BINARY(N)	BYNARY(N)	BINARY(N)
BLOB	BLOB	BLOB	-	BLOB	BLOB
BOOLEAN	-	BOOLEAN	BOOLEAN	BOOLEAN	BOOLEAN
BYTE	-	-	-	-	BYTE
CHAR	CHAR	-	CHAR | CHARACTER	CHAR	CHAR
CHAR FOR BIT DATA	CHAR FOR BIT DATA	BYNARY(N)	BYNARY(N)	BYNARY(N)	BINARY(N)
CLOB	CLOB	CLOB	-	CLOB	CLOB
DATE	DATE	DATE	DATE	DATE	DATE
DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL
DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE
FLOAT(precisão)	FLOAT(precisão)	DOUBLE	DOUBLE	DOUBLE	DOUBLE
IDENTITY	-	IDENTITY	-	-	-
INTEGER	INTEGER	INT	INTEGER | INT	INTEGER	INTEGER | INT
JAVA_OBJECT	BLOB	BLOB	-	JAVA_OBJECT	BLOB
LONG VARCHAR	LONG VARCHAR	-	LONGVARCHAR	-	LONGVARCHAR
LONG VARCHAR FOR BIT DATA	LONG VARCHAR FOR BIT DATA	-	-	-	LONG VARCHAR FOR BIT DATA
NUMERIC	NUMERIC	DECIMAL	NUMERIC	NUMERIC	NUMERIC
REAL	REAL	REAL	REAL	REAL	REAL
SMALLINT	SMALLINT	SMALLINT	SMALLINT	SMALLINT	SMALLINT
TIME	TIME	TIME	TIME	TIME	TIME
TIMESTAMP	TIMESTAMP	TIMESTAMP | DATETIME	TIMESTAMP | DATETIME	TIMESTAMP	TIMESTAMP
TINYINT	-	TINYINT	TINYINT	-	TINYINT
VARCHAR(n)	VARCHAR(n)	VARCHAR	VARCHAR	VARCHAR(n)	VARCHAR
VARCHAR FOR BIT DATA	VARCHAR FOR BIT DATA	-	VARBINARY(N)	-	VARBINARY(N)
VARCHAR_IGNORECASE	-	VARCHAR_IGNORECASE	VARCHAR_IGNORECASE	-	-

3.2. Definição do modelo de dados

O modelo de dados foi baseado em uma empresa fictícia de venda de produtos de diversos fabricantes. Cada fabricante pode ter diversos produtos organizados por categorias. O preço do produto pode variar de acordo com o tipo de comprador, ou seja, o fabricante pode definir tabelas de preços diferentes para pessoa física ou jurídica, por exemplo. A seguir é apresentada uma breve descrição de cada tabela do Banco de Dados definido:

- tabela fabricante: contém o código do fabricante que é o atributo responsável pela identificação do fabricante na base de dados; contém também as informações do fabricante;
- tabela categoria: contém os dados de categoria dos produtos, que são definidas pelo administrador do sistema; os produtos dos fabricantes são classificados de acordo com estas categorias;
- tabela produto: contém os dados dos produtos;
- tabela tabela: contém os dados das tabelas de preços, sendo que cada fabricante pode ter várias tabelas de preços;
- tabela tabela_item: contém a relação da tabela de preços com os produtos que pertencem a ela, cada produto pode estar em mais de uma tabela de preços;
- tabela venda: contém dados da venda;
- tabela venda_item: contém os produtos de cada venda;
- tabela cliente: contém dados do cliente;
- tabela pessoa_fisica: especialização da tabela clientes. Contém o atributo cli_cpf se o cliente for do tipo pessoa física;

- tabela pessoa_juridica: especialização da tabela clientes. Contém o atributo cli_cnpj se o cliente for do tipo pessoa jurídica;
- tabela cor: contém os dados de cores, sendo que cada produto está relacionado a uma cor predominante;
- tabela origem: contém os dados de origem dos produtos, os produtos podem ser nacionais ou importados, esta tabela armazena os dados do país de origem dos produtos.

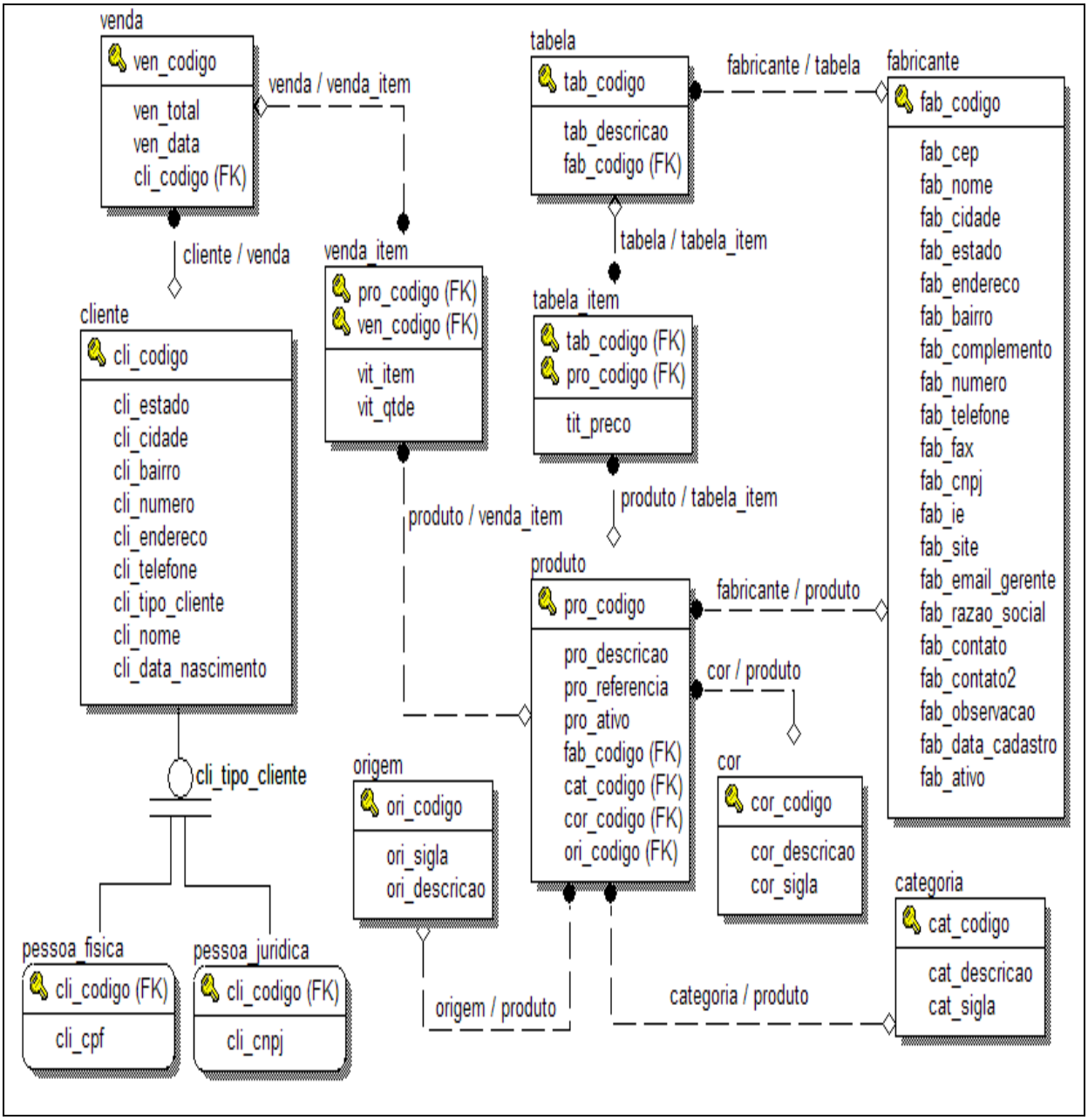


Figura 4 - Modelo de dados utilizado nos testes

3.2.1. Instalação dos SGBDs

Após a definição do modelo de dados a ser implementado para a execução dos testes, foram feitas as instalações dos SGBDs a serem avaliados.

3.2.2. Instalação do Apache Derby

Para instalar o Derby, primeiro é necessário fazer o download do arquivo extensão zip ou tar disponível no site do Derby, em http://db.apache.org/derby/derby_downloads.html. Nesta página estão disponíveis várias versões do Derby em pacotes *bin, lib* e *src*. Deve-se fazer download do pacote *lib*, que contém apenas os arquivos jar necessários para utilização do SGBD. A versão utilizada neste trabalho foi a 10.3.1.4.

Após fazer o download, deve-se extrair os arquivos contidos no pacote. O nome do diretório criado varia de acordo com a versão, assume-se que este diretório se chama *derby* e foi extraído para *C:\DB*. Dentro do diretório *derby* será criado um diretório chamado *lib* que conterá o arquivo *derby.jar*; este deve ser incluído na variável de ambiente *classpath* para que a JVM e os programas Java encontrem os arquivos necessários no momento da execução.

No Windows XP, para configurar a variável de ambiente *classpath*, deve-se ir em Painel de Controle / Sistema / Avançado / Variáveis de Ambiente, em Variáveis do sistema, selecionar a variável *CLASSPATH* e selecionar a opção Editar. Será exibida a janela apresentada na Figura 5, onde se deve acrescentar o caminho do arquivo *derby.jar* no campo Valor da variável:

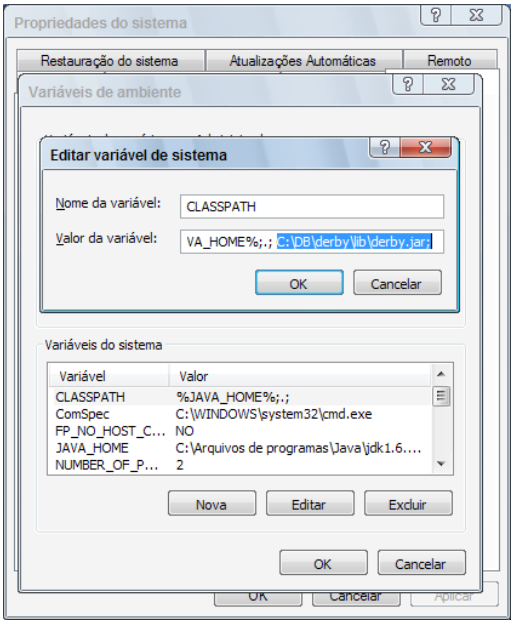


Figura 5 - Configurando a variável de ambiente no Windows XP

A variável *classpath* pode ser definida temporariamente em tempo de execução, para isso um *scritp* contendo o seguinte comando deve ser executado antes da chamada do programa:

set CLASSPATH=C:\DB\derby\lib\derby.jar;%CLASSPATH%

3.2.3. Instalação do H2 Database

Para instalar o H2 Database é necessário fazer o download do arquivo zip disponível em http://www.h2database.com/html/frame.html. Nesta página estão disponíveis duas formas de instalação:

- *Windows Installer*: executável que instala automaticamente os arquivos necessários para executar na plataforma Windows;

- *All plataforms*: arquivo zip contendo os arquivos necessários para executar em qualquer plataforma.

Deve-se fazer download do arquivo *zip*, que contém os arquivos jar necessários para utilização do SGBD. A versão utilizada neste trabalho foi a 1.0.

Após fazer o download, deve-se extrair os arquivos contidos no pacote para um diretório de preferência. Será criado um diretório chamado *h2*, assume-se que este diretório foi extraído para *C:\DB*. Dentro do diretório *h2* será criado um diretório chamado *bin* que conterá o arquivo *h2.jar*; este deve ser incluído na variável de ambiente *classpath* para que a JVM e os programas Java encontrem os arquivos necessários na hora de executar.

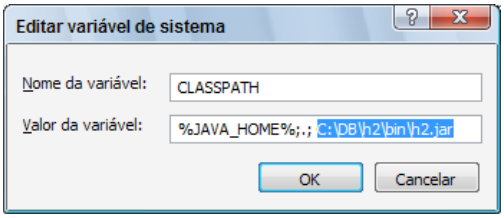


Figura 6 - Configuração da variável de ambiente

A variável *classpath* pode ser definida temporariamente em tempo de execução, para isso um *scritp* contendo o seguinte comando deve ser executado antes da chamada do programa:

set CLASSPATH=C:\DB\h2\bin\h2.jar;%CLASSPATH%

3.2.4. Instalação do HSQLDB

Para instalar o HSQL, é necessário fazer o download do arquivo zip disponível em http://hsqldb.sourceforge.net. Nesta página estão disponíveis várias versões do HSQLDB. Deve-se fazer download do arquivo *zip*, que contém os arquivos jar necessários para utilização do SGBD. A versão utilizada neste trabalho foi a 1.8.0.

Após fazer o download, deve-se extrair os arquivos contidos no pacote para um diretório de preferência. Será criado um diretório chamado *hsqldb*, assume-se que este diretório foi extraído para *C:\DB*. Dentro do diretório *hsqldb* será criado um diretório chamado *lib* que conterá o arquivo *hsqldb.jar*, que deve ser incluído na variável de ambiente *classpath* para que a JVM e os programas Java encontrem os arquivos necessários na hora de executar.

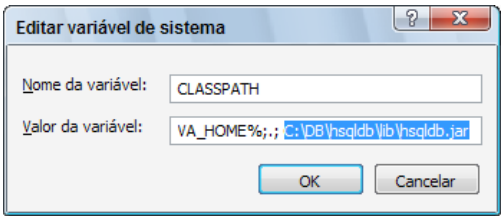


Figura 7 - Configuração da variável de ambiente

A variável *classpath* pode ser definida temporariamente em tempo de execução, para isso um *scritp* contendo o seguinte comando deve ser executado antes da chamada do programa:

set CLASSPATH= C:\DB\hsqldb\lib\hsqldb.jar;%CLASSPATH%

3.2.5. Instalação do McKoi Database

Para instalar o McKoi Database é necessário fazer o download do arquivo zip disponível no site do McKoi, em http://www.mckoi.com/database/index.html#Download. Nesta página, estão disponíveis várias versões do McKoi. Deve-se fazer download do arquivo *zip* que contém os arquivos jar necessários para utilização do SGBD. A versão utilizada neste trabalho foi a 1.0.3.

Após fazer o download, deve-se extrair os arquivos contidos no pacote. O nome do diretório criado varia de acordo com a versão, assume-se que este diretório se chama *mckoi* e foi extraído para *C:\DB*. Dentro do diretório *mckoi* há o arquivo *mckoi.jar*, que deve ser incluído na variável de ambiente *classpath* para que a JVM e os programas Java encontrem os arquivos necessários na hora de executar.

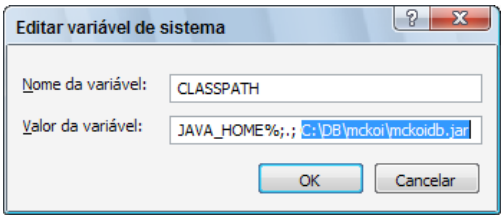


Figura 8 - Configuração da variável de ambiente

A variável *classpath* pode ser definida temporariamente em tempo de execução, para isso um *scritp* contendo o seguinte comando deve ser executado antes da chamada do programa:

set CLASSPATH= C:\DB\mckoi\mckoidb.jar;%CLASSPATH%

3.2.6. Instalação do One$DB

Para instalar o One$DB é necessário fazer o download do arquivo *zip* disponível em http://sourceforge.net/projects/daffodildb/link para download. Nesta página, haverá um link para download do arquivo que ao ser clicado é direcionado para uma página onde estão disponíveis várias versões para download. A versão utilizada neste trabalho foi a 4.1.

Após fazer o download, deve-se extrair o arquivo *jar* para uma pasta de preferência, assume-se que este arquivo chama-se *onedollardb.jar*. Após descompactar o arquivo, no *prompt* de comando dentro do diretório onde está o arquivo *jar* deve ser executado o seguinte comando para que a instalação seja iniciada:

java –jar onedollardb.jar

Na Figura 9 exibe-se a tela de boas vindas, deve ser selecionado o botão Next para dar prosseguimento à instalação. Será exibida a tela com os termos de uso conforme é mostrado na Figura 10. Estando de acordo, deve ser selecionado a opção Yes antes de prosseguir para o próximo passo.

Figura 9 - Tela de boas-vindas da instalação do One$DB

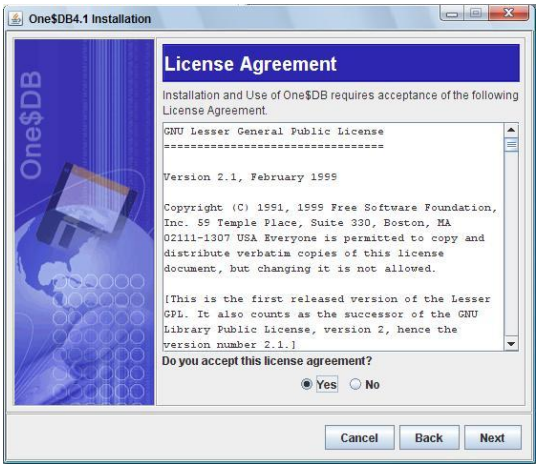


Figura 10 - Termos de uso do One$DB

São então exibidas algumas informações sobre a versão do SGBD, conforme ilustrado na Figura 11.

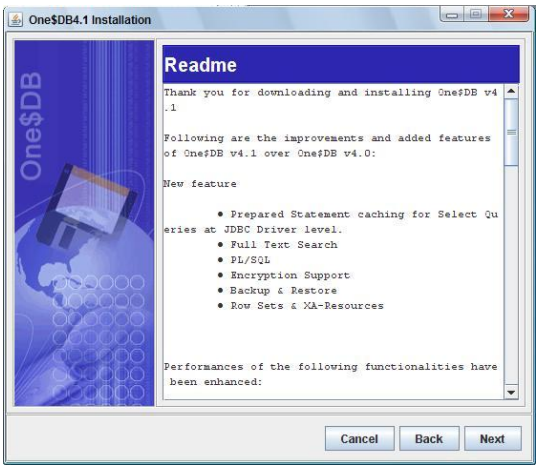


Figura 11 - Notas de versão do One$DB

Antes de prosseguir, o instalador é verificado se outra versão do One$DB está instalada no computador, conforme apresentado na Figura 12.

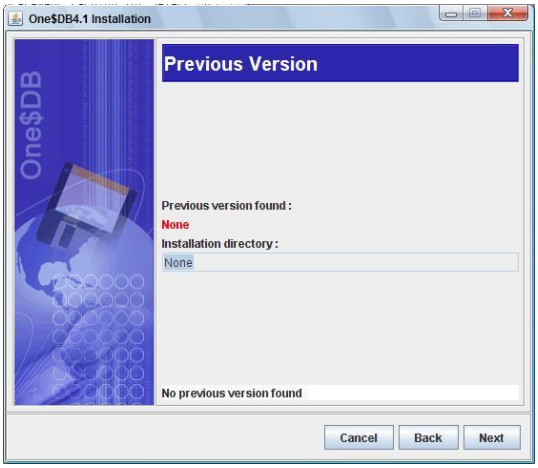


Figura 12 - Verificação de versão anterior do One$DB

Na Figura 13 ilustra-se a próxima tela de instalação onde se pode escolher entre 4 opções:

- *Embedded*: instala os componentes necessários para executar o SGDB em modo embarcado;
- *Server*: instala os componentes necessáios para executar o SGBD em modo servidor;
- *Client*: instala os componentes necessáios para executar o SGBD em modo cliente;
- *Documentation*: documentação completa do one$DB.

Neste trabalho foi utilizada a versão *Embedded*, pois o SGBD foi avaliado embarcado em uma aplicação.

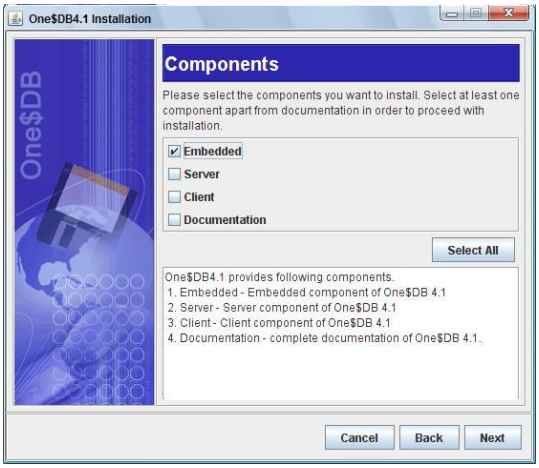


Figura 13 - Escolha do tipo de instalação

Após selecionar os componentes a serem instalados, é necessário informar o caminho onde os componentes serão instalados. Neste caso, foi escolhido o caminho C:\DB\onedollardb, conforme ilustrado na Figura 14.

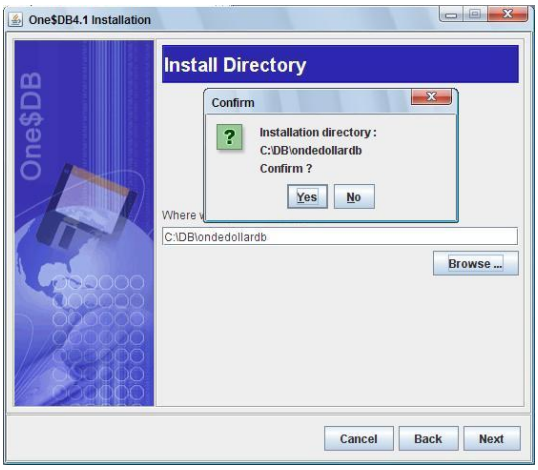


Figura 14 - Escolha do diretório destino da instalação

O programa instalador verifica as máquinas virtuais *Java* instaladas no computador e exibe uma lista para que seja selecionado a JVM a ser utilizada pelo SGBD, conforme ilustrado na Figura 15.

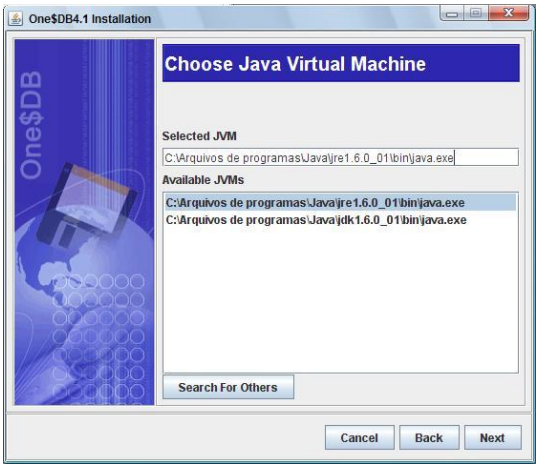


Figura 15 - Escolha da JVM

Após informar a JVM a ser utilizada, é iniciada a cópia dos arquivos, conforme é ilustrado na Figura 16.

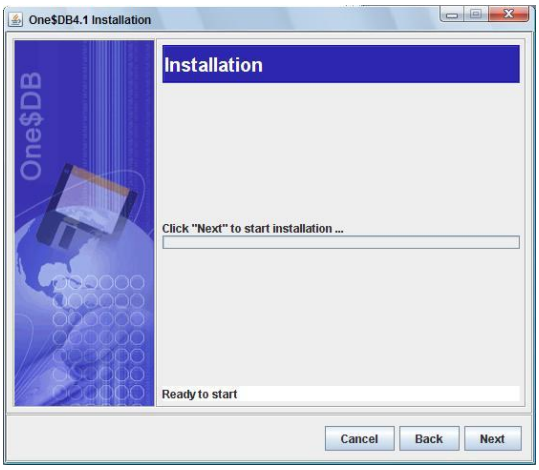


Figura 16 - Tela de início da instalação

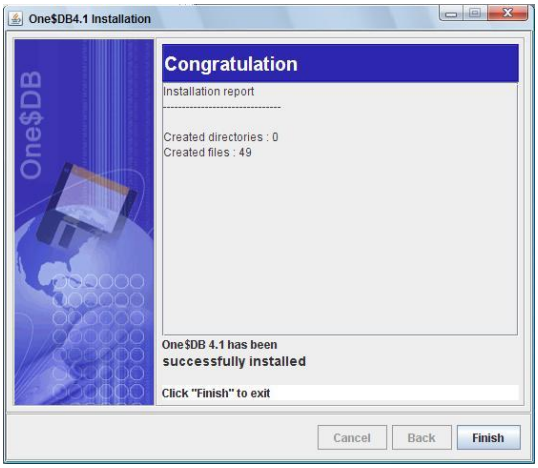


Figura 17 - Tela de término da instalação do One$DB

Ao término da cópia dos arquivos, é necessário configurar a variável *classpath*. Dentro do diretório *onedolladb* há o diretório *lib* que contém os arquivos *DaffodilDB_Common.jar* e *DaffodilDB_Embedded.jar*, que devem ser incluídos na variável de ambiente *classpath* para que a JVM e os programas Java encontrem os arquivos necessários na hora de executar.

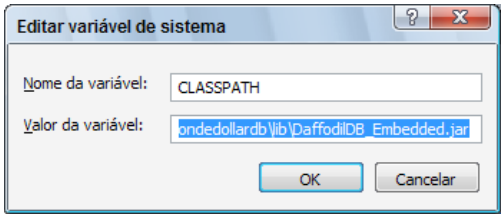


Figura 18 - Configuração da variável de ambiente

A variável *classpath* pode ser definida temporariamente em tempo de execução, para isso um *scritp* contendo o seguinte comando deve ser executado antes da chamada do programa:

set CLASSPATH= C:\DB\ondedollardb\lib\DaffodilDB_Common.jar; C:\DB\ondedollardb\lib\DaffodilDB_Embedded.jar%CLASSPATH%

3.3. Definição dos testes a serem realizados

Após a instalação dos SGBDs, a criação da base de dados proposta foi feita através de scripts executados para cada SGBD separadamente. As bases de dados criadas, foram semelhantes para todos os cinco SGBDs, e foram utilizados os tipos de dados compatíveis entre eles conforme foi definido na Tabela 7.

Para a realização dos testes, foram desenvolvidas duas aplicações em linguagem Java por se tratarem de SGBDs que constituem APIs Java.

Para os testes de integridade, foi desenvolvida uma aplicação que consiste em um programa que permite a escolha de conexão com um entre os cinco SGBDs propostos. Em seguida, permite que um comando seja digitado e executado no SGBD escolhido e um campo onde é exibido o resultado. Os scripts de criação do Banco de Dados também estão presentes neste programa. Na Figura 19 ilustra-se a interface do programa.

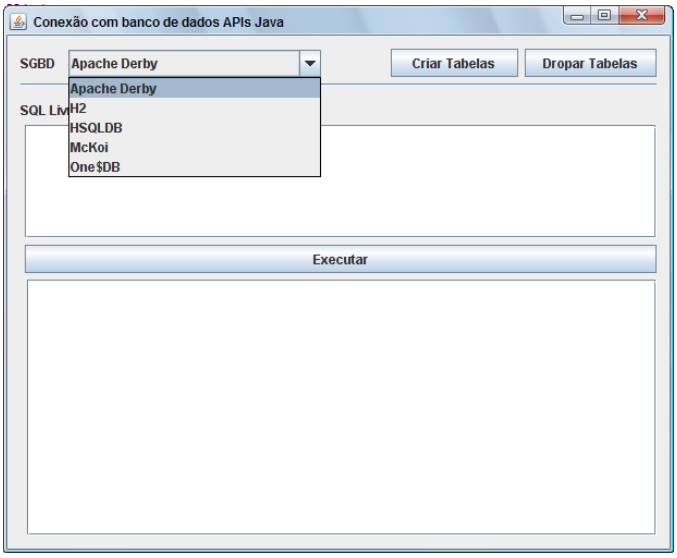


Figura 19 – Interface do programa desenvolvido para os testes de integridade

Para os testes de desempenho de armazenamento e recuperação, foi desenvolvida uma aplicação sem interface gráfica que executa uma seqüência de operações de acordo com os argumentos passados na chamada do programa. Os argumentos são: SGBD, OPERAÇÃO, QUANTIDADE DE REGISTROS e TABELA. A aplicação registra o tempo de cada seqüência de operações em um arquivo texto para posterior análise e comparação.

Deu-se então início à fase de testes para se atingir os objetivos propostos. Foram realizados testes de integridade e desempenho nos cinco SGBDs.

3.3.1. Teste de integridade dos tipos de dados

Este teste constituiu na verificação quanto ao controle automático do SGBD em relação à integridade dos tipos de dados oferecidos.

Para este teste, foram feitas tentativas de inserção de dados fora da especificação dos tipos definidos nas tabelas e analisado como cada SGBD se comporta com relação à validação dos dados durante a inserção. Por exemplo: na tabela FABRICANTE, há um atributo chamado fab_data_cadastro cujo propósito é armazenar a data em que o fabricante foi cadastrado, esse atributo foi definido como tipo *DATE,* um dos testes consistiu na tentativa de inserir valores que não correspondiam ao formato de data, que em todos os SGBDs estudados é YYY-MM-DD, e verificar como cada SGBD trata este tipo de erro.

3.3.2. Teste de integridade de chave primária

Este teste constituiu na verificação do controle de unicidade de chaves primárias nas tabelas e o controle de tentativa de inserção de chaves nulas.

Neste teste foram inseridos registros em tabelas que possuíam chave primária e feitas tentativas de inserções com chaves primárias duplicadas e chaves primárias nulas. Foi verificado como cada SGBD se comporta no controle destas chaves. Por exemplo: na tabela CLIENTE, há um atributo chamado cli_codigo definido como chave primária do tipo inteiro. Um dos testes consistiu na tentativa de inserir duas vezes um cliente com o mesmo valor para o atributo cli_codigo e verificar como cada SGBD se comporta com relação a esta

integridade. O outro teste consistiu na tentativa de inserir um cliente com o campo cli_codigo nulo e verificar como cada SGBD trata este tipo de erro.

3.3.3. Teste de integridade referencial

A finalidade deste teste é verificar o controle de integridade automático do SGBD na inclusão, remoção ou alteração de dados em tabelas relacionadas com outras tabelas.

Este teste consistiu em verificar a garantia de integridade referencial em cada um dos SGBDs na inserção, remoção e alteração dos registros.

Com relação à inclusão, foram feitas tentativas de inserção em tabelas filhas, sem que o registro correspondente estivesse presente na tabela pai.

Para o teste de exclusão, foram feitas tentativas de exclusão de registros na tabela pai, que estivessem relacionados a algum registro na tabela filha.

No teste de alteração, foram feitas tentativas de alteração do campo de chave estrangeira nas tabelas filhas, tentando alterar para um registro pai não existente e, tentativas de alteração da chave primária na tabela pai que possuía um correspondente na tabela filha.

3.3.4. Desempenho com a utilização de tabelas solitárias

Nesta fase foi verificado o tempo de armazenamento e recuperação em tabelas sem relacionamento.

Para esses testes foram utilizadas as tabelas do modelo de dados proposto na Figura 4 que não possuíam relacionamentos com tabelas filhas.

Os testes de armazenamento e recuperação, consistiram em realizar várias seqüências de inserções e seleções com quantidades de registros variadas. Por exemplo, na tabela fabricante, foram executadas seqüências de inserção de 50, 500, 5000 e 50000 registros.

Cada seqüência de inserções e de seleções foi executada 10 vezes através de um script para assegurar que o tempo obtido não fosse influenciado pelo estado da máquina no momento da execução, conforme é ilustrado na Figura 20. Em seguida, foi calculada a média entre os tempos obtidos.

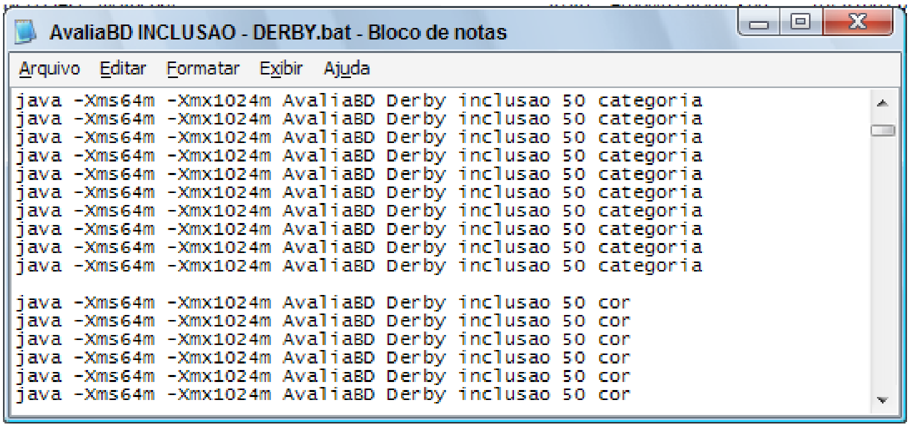


Figura 20 – Chamada do programa através de um script com passagem de argumentos para teste de desempenho de inclusão

3.3.5. Desempenho com a utilização de tabelas relacionadas

Neste teste o objetivo foi verificar o tempo de armazenamento e recuperação em tabelas que têm um ou mais relacionamentos.

Este teste consiste basicamente no que foi executado para os testes de desempenho em tabelas comuns, com a diferença de que neste caso, as tabelas utilizadas foram as tabelas que possuem um ou mais relacionamentos.

3.3.6. Considerações Finais

Neste capítulo foi definido o modelo de dados e os testes de integridade e desempenho realizados para se fazer o comparativo entre os SGBDs. Também foi apresentada a instalação de cada SGBD.

Os resultados obtidos com a execução dos testes serão discutidos no próximo capítulo.

4. RESULTADOS E DISCUSSÕES

Após a execução dos testes descritos no capítulo anterior e da coleta dos resultados obtidos, algumas considerações podem ser levantadas a respeito dos SGBDs avaliados.

Os testes foram realizados em uma máquina Pentium IV 2.8 GHz HT com 1 Gigabyte de memória RAM e 200 Gigabytes de Disco Rígido.

4.1.1. Teste de integridade dos tipos de dados

Foram realizados testes com os tipos DATE, DOUBLE, CHAR, INTEGER E VARCHAR conforme ilustrado na Tabela 8.

Tabela 8 - Teste de tipos de dados

Tipo do Campo	**Valor inserido**	***Resultado Derby***	***Resultado H2***	***Resulta HSQLDB***	***Resultado McKoi***	***Resultado One$DB***
DATE	1	ERRO: *formato de data inválida*	ERRO: *formato de data inválida*	ERRO: *formato de data inválida*	**SUCESSO: *data inserida (1969-12-31)***	ERRO: *formato de data inválida*
DATE	20	ERRO: *formato de data inválida*	ERRO: *formato de data inválida*	ERRO: *formato de data inválida*	**SUCESSO: *data inserida (1969-12-31)***	ERRO: *formato de data inválida*
DATE	"A"	ERRO: *formato de data inválida*	ERRO: *formato de data inválida*	ERRO: *formato de data inválida*	ERRO: *formato de data inválida*	ERRO: *formato de data inválida*
DOUBLE	1	**SUCESSO: *valor inserido***	**SUCESSO: *valor inserido***	**SUCESSO: *valor inserido***	**SUCESSO: *valor inserido***	**SUCESSO: *valor inserido***
DOUBLE	"A"	ERRO: *tipo incompatível*	ERRO: *tipo incompatível*	ERRO: *tipo incompatível*	**SUCESSO: *inserido o valor 0.0***	ERRO: *tipo incompatível*
CHAR(2)	"ABC"	ERRO: *tamanho incompatível*	ERRO: *tamanho incompatível*	**SUCESSO: *palavra inserida***	**SUCESSO: *truncamento para 'AB'***	ERRO: *tamanho incompatível*
INTEGER	"A"	ERRO: *tipo incompatível*	ERRO: *tipo incompatível*	ERRO: *tipo incompatível*	**SUCESSO: *inserido o valor 0***	ERRO: *tipo incompatível*
INTEGER	1.99	**SUCESSO: *truncamento para 1***	**SUCESSO: *truncamento para 1***	**SUCESSO: *truncamento para 1***	**SUCESSO: *truncamento para 1***	**SUCESSO: *truncamento para 1***
VARCHAR (255)	Palavra de tamanho maior que 255	ERRO: *tamanho incompatível*	ERRO: *tamanho incompatível*	**SUCESSO: *palavra inserida***	**SUCESSO: *palavra inserida***	ERRO: *tamanho incompatível*

Os três primeiros testes foram realizados com o tipo DATE, sendo que os dois primeiros consistiram em inserir os valores 1 e 20. O SGBD Mckoi inseriu a data 1969-12-31 para os dois valores, os outros 4 SGBDs exibiram erro de formato de data inválido. O terceiro teste consistiu em inserir a letra A, neste caso, todos os SGBDs exibiram erro de formato de data inválido.

O quarto e o quinto testes foram realizados com o tipo DOUBLE, sendo que no quarto teste, foi inserido o valor 1 e em todos os SGBDs, tendo sido incluído com sucesso. No quinto teste foi inserida a letra A e, com exceção do SGBD Mckoi que, permitiu a inclusão convertendo o valor para 0.0, todos os outros exibiram erro de tipo incompatível.

No sexto teste foi testado o tipo CHAR, definido com o tamanho 2. Foi inserida a palavra ABC. Neste caso, o SGBD HSQLDB permitiu a inclusão da palavra completa, ignorando o limite definido de 2 caracteres, já o Mckoi realizou o truncamento da palavra para AB e permitiu a inclusão. Os outros 3 SGBDs apresentaram erro de tamanho inválido.

O sétimo e o oitavo testes foram realizados com o tipo INTEGER. No sétimo teste foi inserida a letra A e, assim como para o tipo DOUBLE, o SGBD Mckoi realizou a conversão da letra A para o valor 0 e permitiu a inclusão. Todos os outros SGBDs apresentaram erro de tipo incompatível. No oitavo teste foi inserido o valo 1.99, e todos os SGBDs realizaram o truncamento para o valor 1 e permitiram a inclusão.

No nono teste foi realizado o teste com o tipo VARCHAR definido com o tamanho 255. Neste teste foi inserida uma palavra com mais de 255 caracteres; os SGBDs HSQLDB e Mckoi permitiram a inclusão da palavra completa, fora do limite definido.

Com relação à garantia da integridade dos tipos de dados, os SGBDs Apache Derby, H2 Database e One$DB mostraram-se eficazes com relação a este quesito. Estes três SGBDs apresentaram controle automático eficaz de tipos de dados em todos os testes realizados.

4.1.2. Teste de integridade de chave primária

Com relação à garantia de integridade de chave primária, os cinco SGBDs mostraram-se eficazes para este tipo de controle. Foram feitas tentativas de inclusão de registros com a chave primária duplicada e registros com a chave primária nula. Nenhum dos SGBDs permitiu a inclusão dos registros, conforme é mostrado na Tabela 9.

Tabela 9 - Teste de chave primária

Descrição	***Resultado Derby***	***Resultado H2***	***Resulta HSQLDB***	***Resultado McKoi***	***Resultado One$DB***
Inserção de 2 registros com a mesma chave primária na tabela CATEGORIA	ERRO: *duplicação de chave primária*	ERRO: *duplicação de chave primária*	ERRO: *duplicação de chave primária*	ERRO: *duplicação de chave primária*	ERRO: *duplicação de chave primária*
Inserção de 2 registros com a mesma chave primária na tabela PRODUTO	ERRO: *duplicação de chave primária*	ERRO: *duplicação de chave primária*	ERRO: *duplicação de chave primária*	ERRO: *duplicação de chave primária*	ERRO: *duplicação de chave primária*
Inserção de 1 registro com chave primária nula na tabela FABRICANTE	ERRO: *inserção de chave nula*	ERRO: *inserção de chave nula*	ERRO: *inserção de chave nula*	ERRO: *inserção de chave nula*	ERRO: *inserção de chave nula*
Inserção de 1 registro com chave primária nula na tabela VENDA	ERRO: *inserção de chave nula*	ERRO: *inserção de chave nula*	ERRO: *inserção de chave nula*	ERRO: *inserção de chave nula*	ERRO: *inserção de chave nula*

4.1.3. Teste de integridade referencial

Foram feitas tentativas de inclusão de registros em tabelas filhas que não possuíam correspondente na tabela Pai conforme é mostrado na Tabela 10.

Tabela 10 - Teste de garantia integridade referencial na inclusão

Descrição	***Resultado Derby***	***Resultado H2***	***Resulta HSQLDB***	***Resultado McKoi***	***Resultado One$DB***
Inserção na tabela FABRICANTE	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*
Inserção na tabela TABELA que possui um correspondente na tabela FABRICANTE	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*
Inserção na tabela TABELA que não possui um correspondente na tabela FABRICANTE	ERRO: *inserção não realizada*	ERRO: *inserção não realizada*	ERRO: *inserção não realizada*	ERRO: *inserção não realizada*	ERRO: *inserção não realizada*
Inserção na tabela PRODUTO que possui correspondentes nas tabelas FABRICANTE, CATEGORIA, COR E ORIGEM	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*	SUCESSO: *inserção realizada*
Inserção na tabela PRODUTO que possui correspondentes nas tabelas FABRICANTE, CATEGORIA, COR, mas não possui correspondente na tabela ORIGEM	ERRO: *inserção não realizada*	ERRO: *inserção não realizada*	ERRO: *inserção não realizada*	ERRO: *inserção não realizada*	ERRO: *inserção não realizada*

Um dos testes realizados foi a tentativa de inclusão na tabela TABELA, que contém os dados das tabelas de preço de cada fabricante. Foi inserido um registro na tabela TABELA com um fabricante não incluído na tabela FABRICANTE, todos os SGBDs retornaram erro de chave estrangeira não encontrada.

Com relação à garantia de integridade referencial na exclusão de registros, todos os SGBDs também se mostraram eficazes neste tipo de controle. Foram feitas tentativas de exclusão em tabelas Pai que possuíam correspondentes nas tabelas Filhas e nenhum dos SGBDs permitiu a exclusão, conforme é mostrado na Tabela 11.

.

Tabela 11 - Teste de garantia integridade referencial na exclusão

Descrição	***Resultado Derby***	***Resultado H2***	***Resulta HSQLDB***	***Resultado McKoi***	***Resultado One$DB***
Exclusão na tabela VENDA_ITEM	SUCESSO: *exclusão realizada*	SUCESSO: *exclusão realizada*	SUCESSO: *exclusão realizada*	SUCESSO: *exclusão realizada*	SUCESSO: *exclusão realizada*
Exclusão na tabela FABRICANTE que possui um correspondente na tabela PRODUTO	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*
Exclusão na tabela TABELA que possui um correspondente na tabela TABELA_ITEM	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*
Exclusão na tabela CLIENTE que possui correspondentes nas tabelas PESSOA_FISICA e PESSOA_JURIDICA	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*	ERRO: *exclusão não realizada*

Como se pode observar, todos os SGBDs garantiram a integridade referencial na exclusão, em todos os testes realizados.

No teste de alteração, todos os SGBDs também se mostraram eficazes no controle de integridade referencial. Foram feitas tentativas de alterar a chave estrangeira em uma tabela filha, alterando para um correspondente não existente na tabela Pai e tentativas de alteração da chave primária na tabela pai que possuía um correspondente na tabela filha. Nenhum SGBD permitiu a alteração, conforme é mostrado na Tabela 12.

Tabela 12 - Teste de garantia integridade referencial na alteração

Descrição	*Resultado Derby*	*Resultado H2*	*Resulta HSQLDB*	*Resultado McKoi*	*Resultado One$DB*
Alteração na tabela FABRICANTE	SUCESSO: *alteração realizada*	SUCESSO: *alteração realizada*	SUCESSO: *alteração realizada*	SUCESSO: *alteração realizada*	SUCESSO: *alteração realizada*
Alteração na tabela PRODUTO para um fabricante que não existe na tabela FABRICANTE	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*
Alteração na tabela VENDA para um cliente que não exista na tabela CLIENTE	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*
Alteração na tabela TABELA_ITEM para um produto que não exista na tabela PRODUTO	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*
Alteração do código tabela CATEGORIA que possui um correspondente na tabela PRODUTO	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*	ERRO: *alteração não realizada*

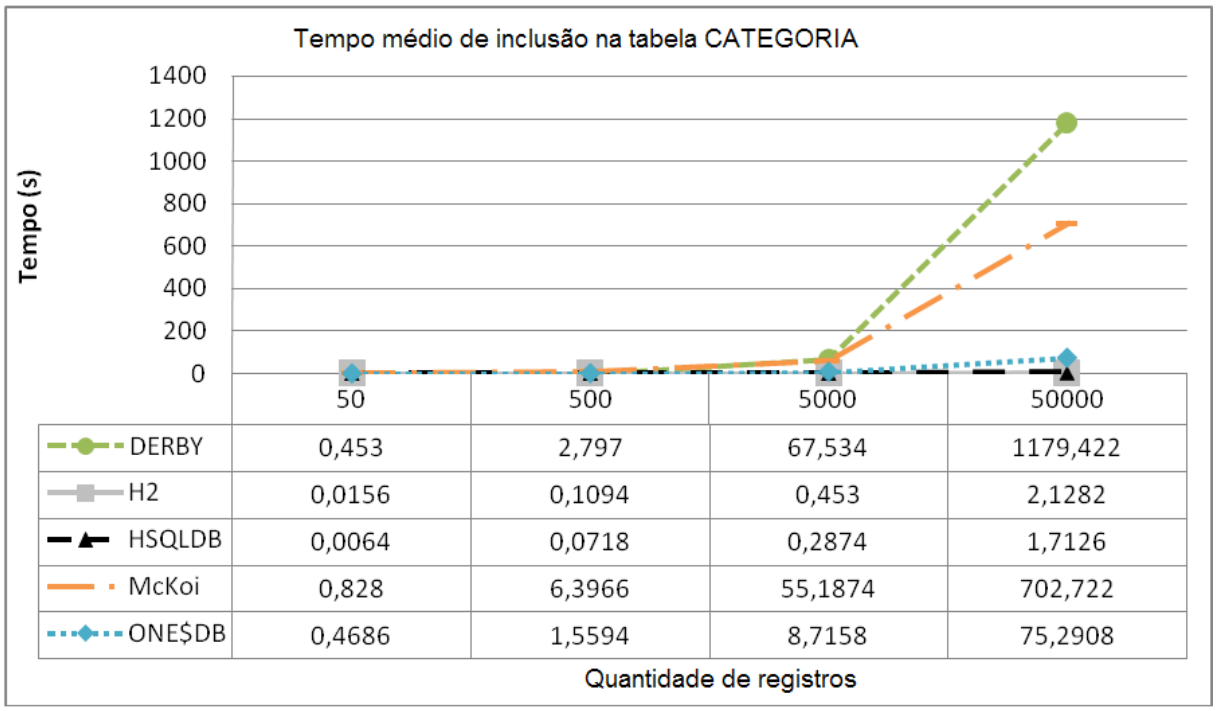
Com relação ao controle de integridade referencial, os cinco SGBDs propostos apresentaram mecanismos eficazes de controle, não permitindo que operações indesejadas fossem realizadas e mantendo a integridade do Banco de Dados.

4.1.4. Teste de desempenho de armazenamento em tabelas solitárias

Foram realizadas seqüências de inclusões de volumes de registros variados em tabelas solitárias, e o desempenho de cada SGBD foi avaliado de acordo com o tempo dispendido para que a inclusão fosse concluída.

Os testes consistiram em inserir seqüências de registros nas tabelas CATEGORIA, CLIENTE, COR, FABRICANTE e ORIGEM. As seqüências de registros utilizadas foram de 50, 500, 5000 e 50000 registros.

Como a estrutura das tabelas CATEGORIA, COR e ORIGEM é a mesma, os resultados obtidos com a inclusão de registros nestas tabelas é semelhante conforme são ilustrados na Figura 21, Figura 22 e Figura 23.



	50	500	5000	50000
DERBY	0,453	2,797	67,534	1179,422
H2	0,0156	0,1094	0,453	2,1282
HSQLDB	0,0064	0,0718	0,2874	1,7126
McKoi	0,828	6,3966	55,1874	702,722
ONE$DB	0,4686	1,5594	8,7158	75,2908

Figura 21 - Desempenho de inclusão na tabela CATEGORIA

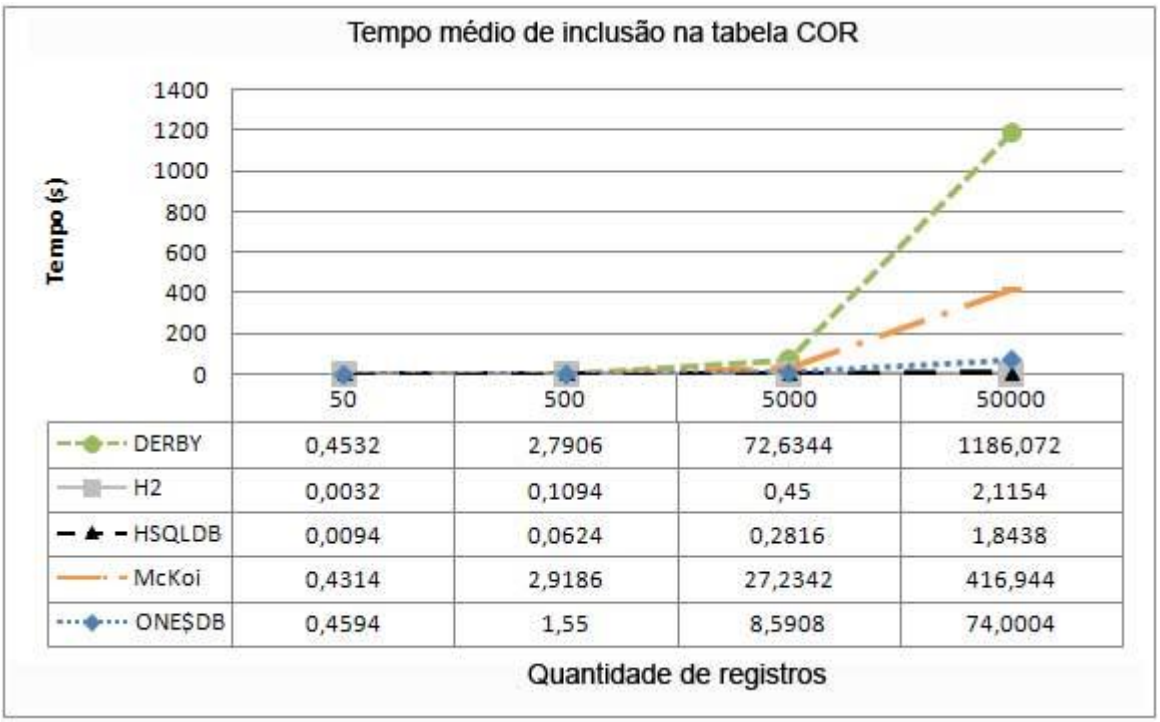


Figura 22 - Desempenho de inclusão na tabela COR

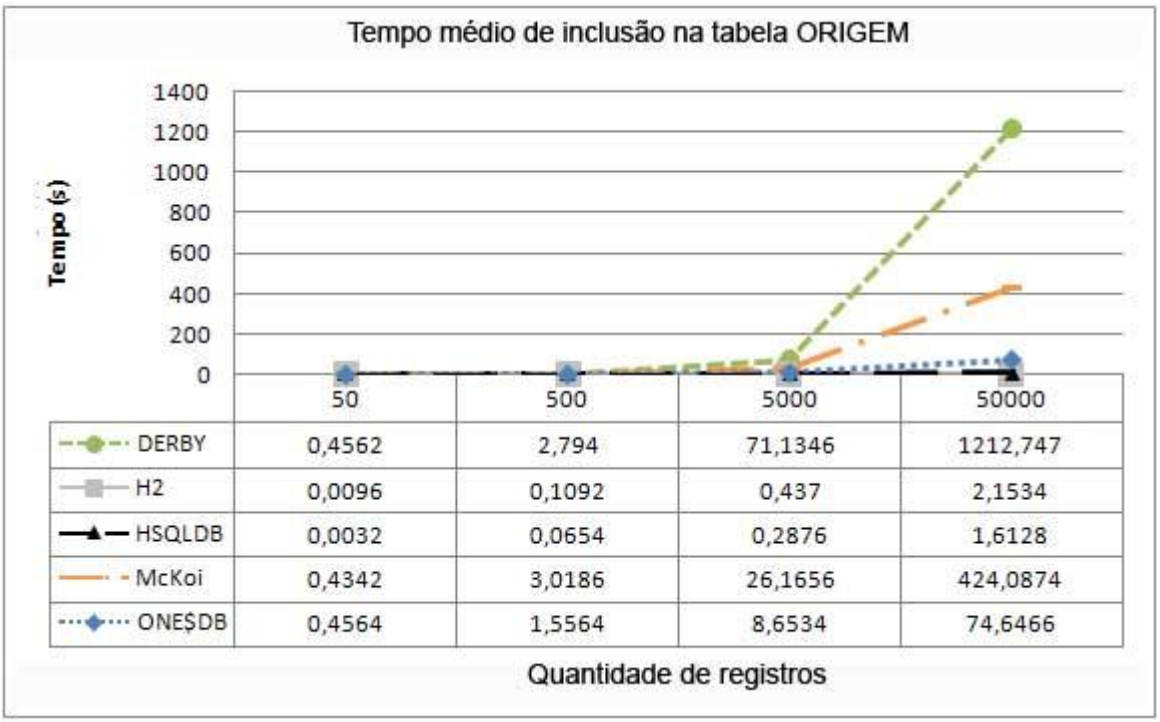


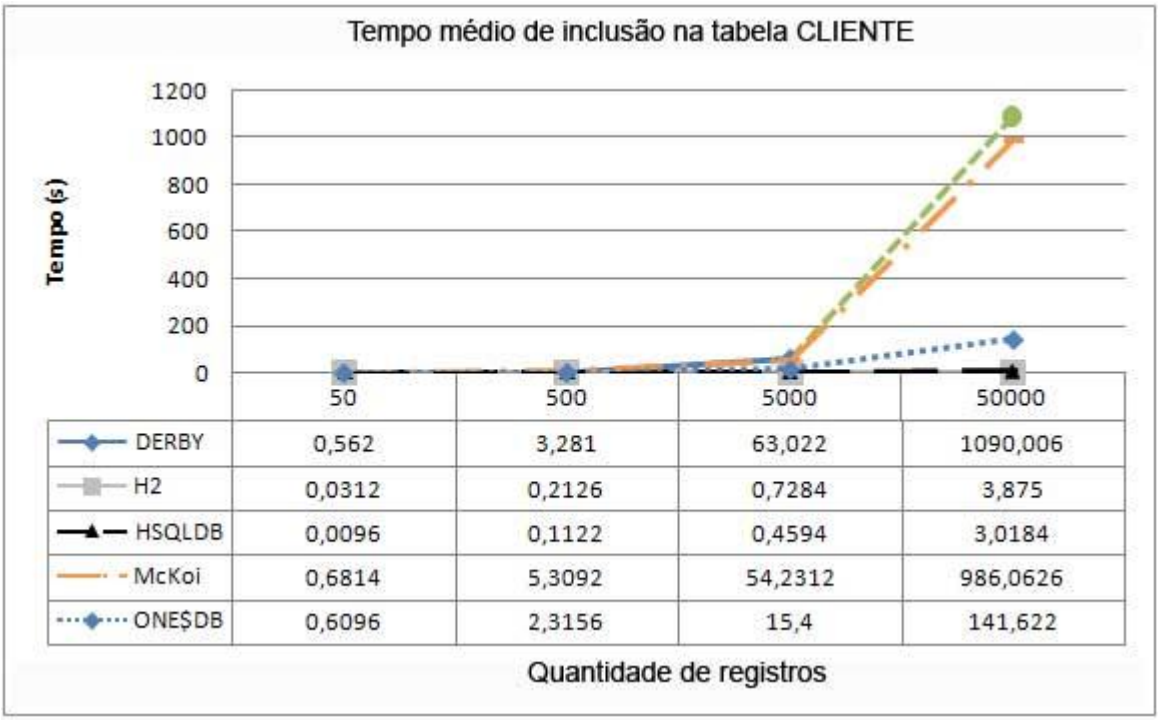
Figura 23 - Desempenho de inclusão na tabela ORIGEM

Pode-se observar que em tabelas com poucas colunas, neste caso as tabelas possuíam 3 atributos, a velocidade dos SGBDs H2 e HSQLDB é muito superior aos demais quando é inserido um grande volume de dados. Também se pode observar que incluindo um volume de

dados menor, na casa de algumas centenas de registros, o SGBD Derby é mais veloz que o Mckoi, mas à medida em que a quantidade de registros aumenta, o desempenho do Derby diminui em uma proporção maior que o do Mckoi.

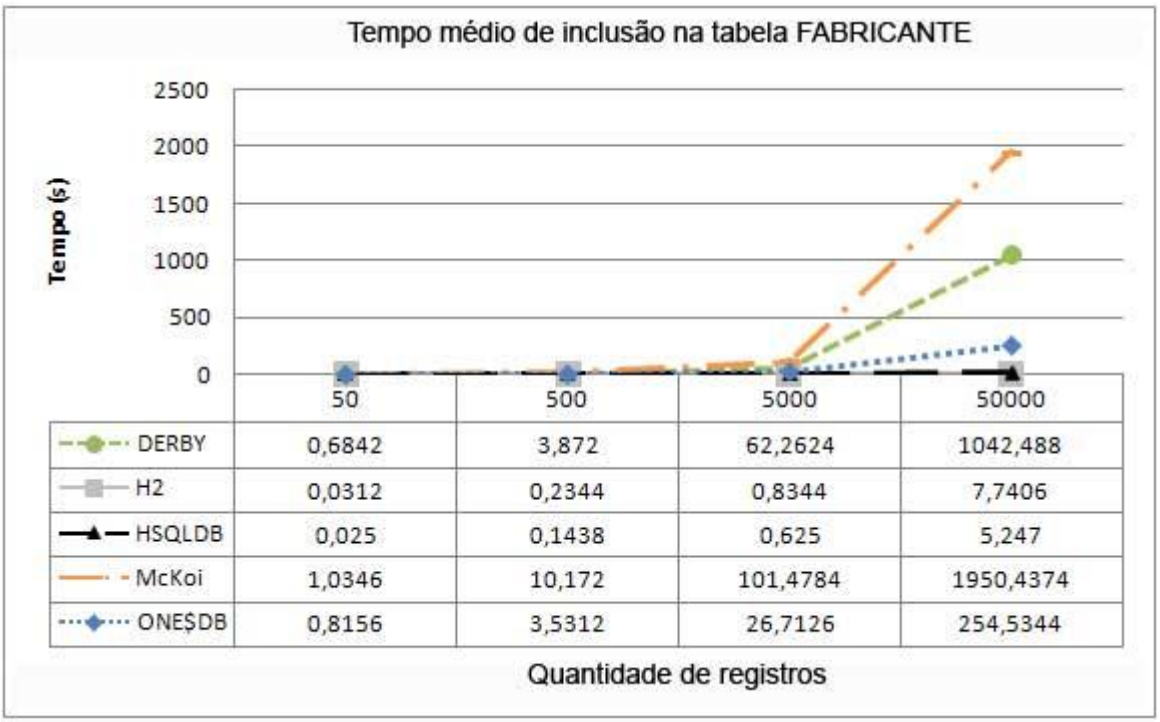
O SGBD One$B apesar de apresentar uma velocidade muito superior ao Derby e Mckoi, se comparado ao H2 e ao HSQLDB se mostra muito inferior a estes dois SGBDs.

A Figura 24 e a Figura 25 mostram os resultados utilizando tabelas sem relacionamento com uma quantidade maior de colunas.



	50	500	5000	50000
DERBY	0,562	3,281	63,022	1090,006
H2	0,0312	0,2126	0,7284	3,875
HSQLDB	0,0096	0,1122	0,4594	3,0184
McKoi	0,6814	5,3092	54,2312	986,0626
ONE$DB	0,6096	2,3156	15,4	141,622

Figura 24 - Desempenho de inclusão na tabela CLIENTE



	50	500	5000	50000
DERBY	0,6842	3,872	62,2624	1042,488
H2	0,0312	0,2344	0,8344	7,7406
HSQLDB	0,025	0,1438	0,625	5,247
McKoi	1,0346	10,172	101,4784	1950,4374
ONE$DB	0,8156	3,5312	26,7126	254,5344

Figura 25 - Desempenho de inclusão na tabela FABRICANTE

Pode-se observar que em tabelas com uma quantidade maior de colunas, o tempo de inclusão dos SGBDs aumentou, com exceção do SGBD Derby que manteve um desempenho semelhante aos testes anteriores em tabelas com poucas colunas. Isto fica claro ao realizar o teste na tabela FABRICANTE, que possui 20 colunas: o tempo de inclusão dos SGBDs aumenta de forma proporcional, sendo o Derby o que mantém o tempo estável se comparado aos testes anteriores, ficando abaixo do tempo do Mckoi que para inserir 50000 registros levou 1950,4374 segundos, e na tabela ORIGEM que possui apenas 3 atributos levou 424,087 segundos para incluir este mesmo montate.

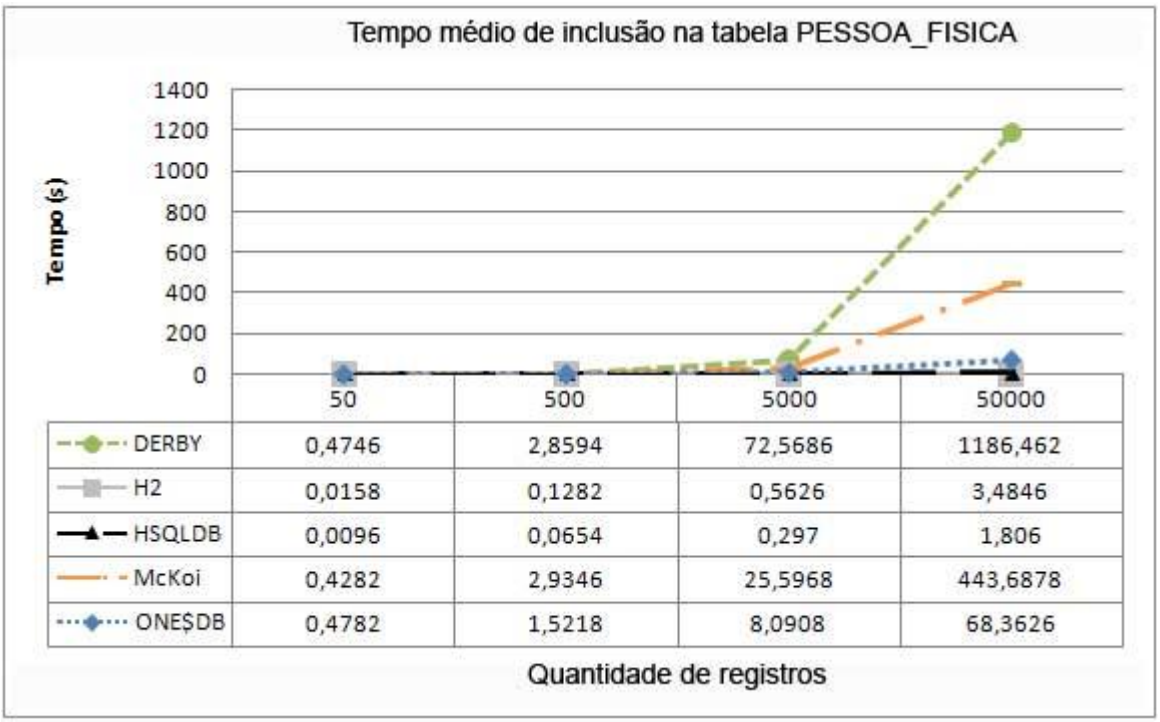
Com relação ao desempenho na inserção em tabelas sem relacionamento, os SGBDs H2 Database e HSQLDB mostraram-se mais velozes que os demais.

4.1.5. Teste de desempenho de armazenamento em tabelas com um relacionamento

Este teste verificou o desempenho com a utilização de tabelas que possuem somente um relacionamento, ou seja, são filhas de uma única tabela pai. Da mesma forma que no teste anterior, foram inseridas seqüências de 50, 500, 5000 e 50000 registros e o desempenho dos SGBDs foi avaliado de acordo com os tempos que cada SGBD levou para concluir a inclusão.

Os testes foram realizados com as tabelas PESSOA_FISICA, PESSOA_JURIDICA, TABELA e VENDA.

Pode-se observar que se tratando de tabelas com apenas um relacionamento, a garantia de integridade referencial afeta o desempenho dos SGBDs de forma pouco significante, principalmente com relação ao Derby, que manteve praticamente o mesmo desempenho dos testes anteriores com tabelas sem relacionamento, conforme ilustram a Figura 26, Figura 27, Figura 28 e Figura 29.



	50	500	5000	50000
DERBY	0,4746	2,8594	72,5686	1186,462
H2	0,0158	0,1282	0,5626	3,4846
HSQLDB	0,0096	0,0654	0,297	1,806
McKoi	0,4282	2,9346	25,5968	443,6878
ONE$DB	0,4782	1,5218	8,0908	68,3626

Figura 26 - Desempenho de inclusão na tabela PESSOA_FISICA

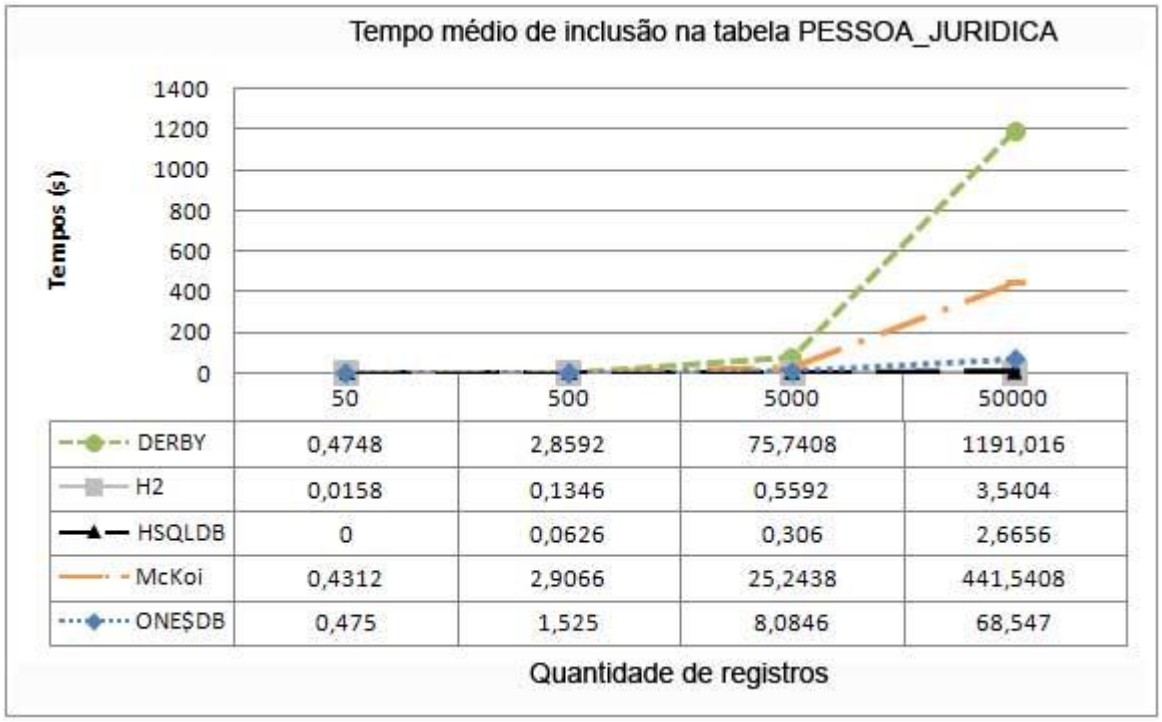


Figura 27 - Desempenho de inclusão na tabela PESSOA_JURIDICA

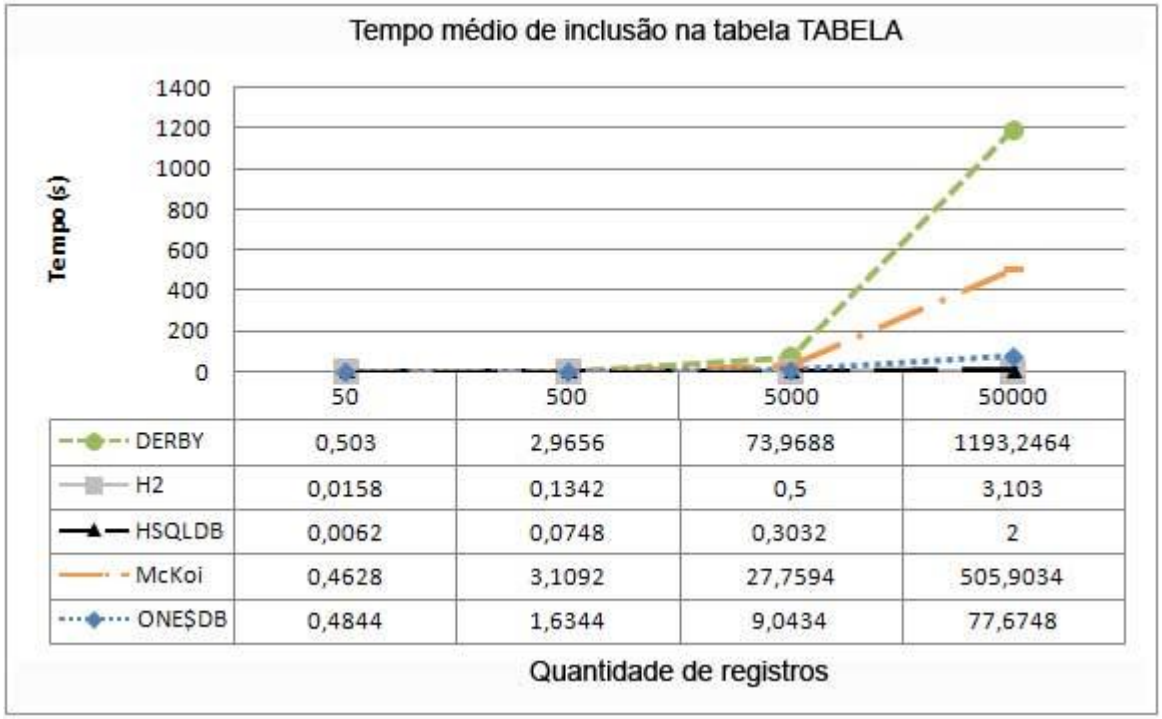


Figura 28 - Desempenho de inclusão na tabela TABELA

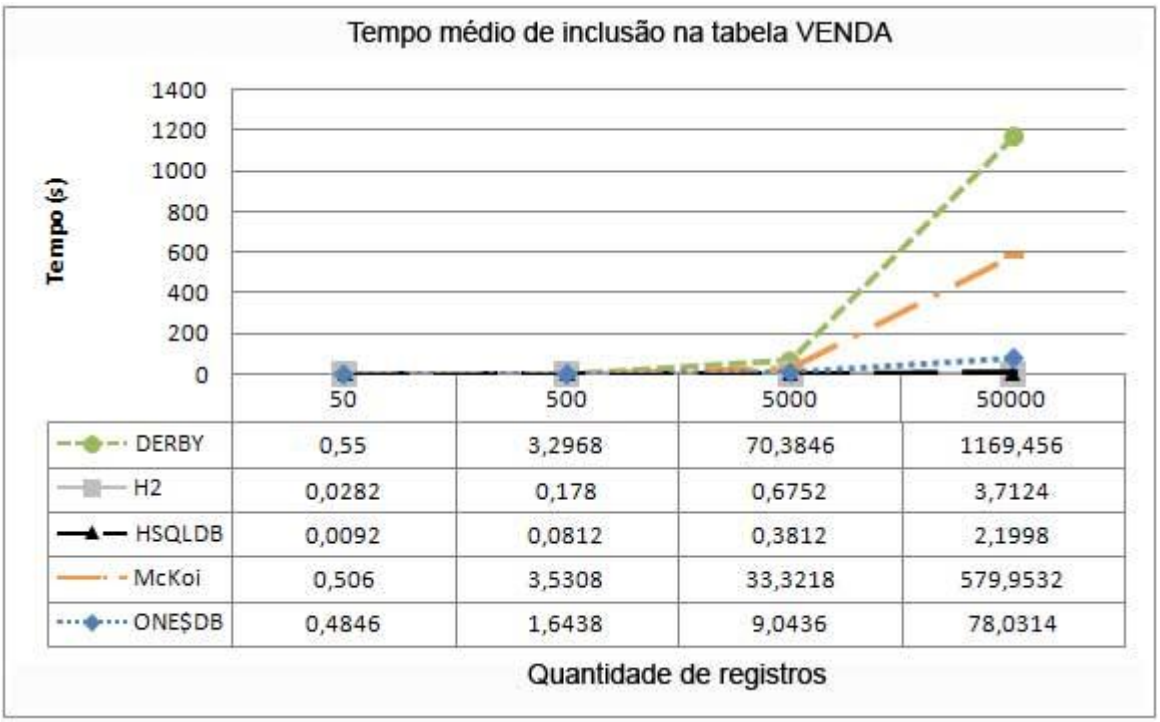


Figura 29 - Desempenho de inclusão na tabela VENDA

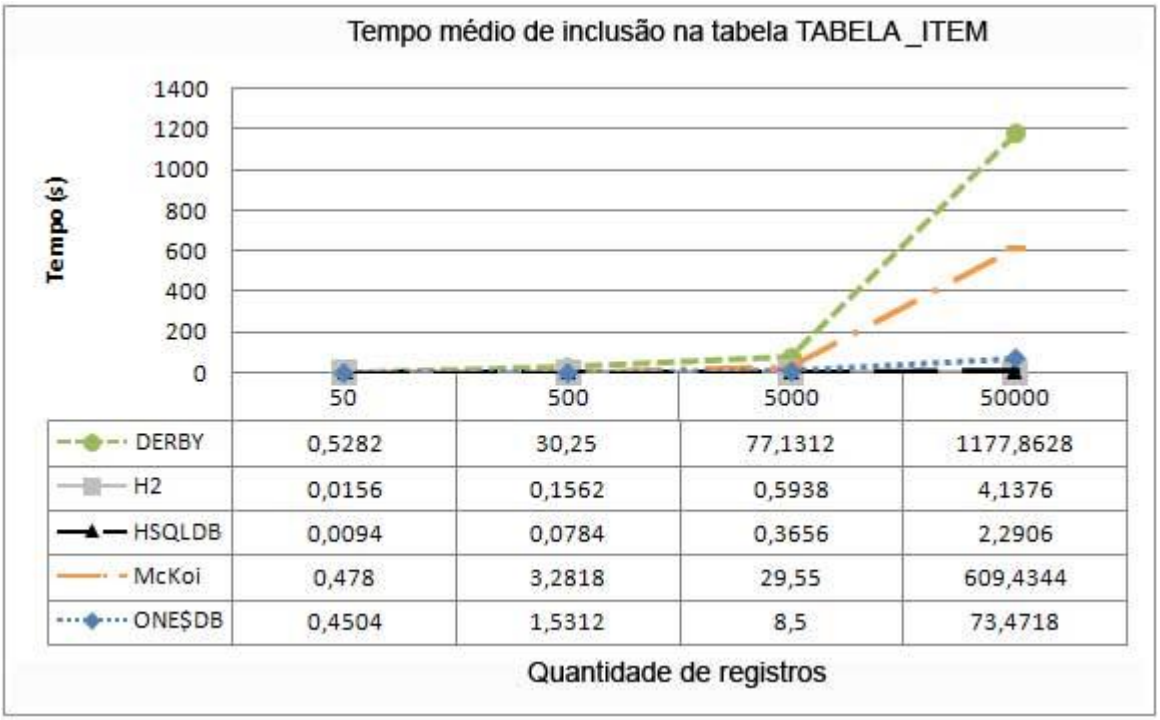
Pode-se observar que em tabelas com um relacionamento, os resultados se assemelham aos resultados obtidos no teste de armazenamento em tabelas sem relacionamento, mantendo-se H2 e HSQLDB entre os mais rápidos.

4.1.6. Teste de desempenho de armazenamento em tabelas com dois relacionamentos

Neste teste foram utilizadas as tabelas que possuíam dois relacionamentos. As tabelas utilizadas foram TABELA_ITEM e VENDA_ITEM.

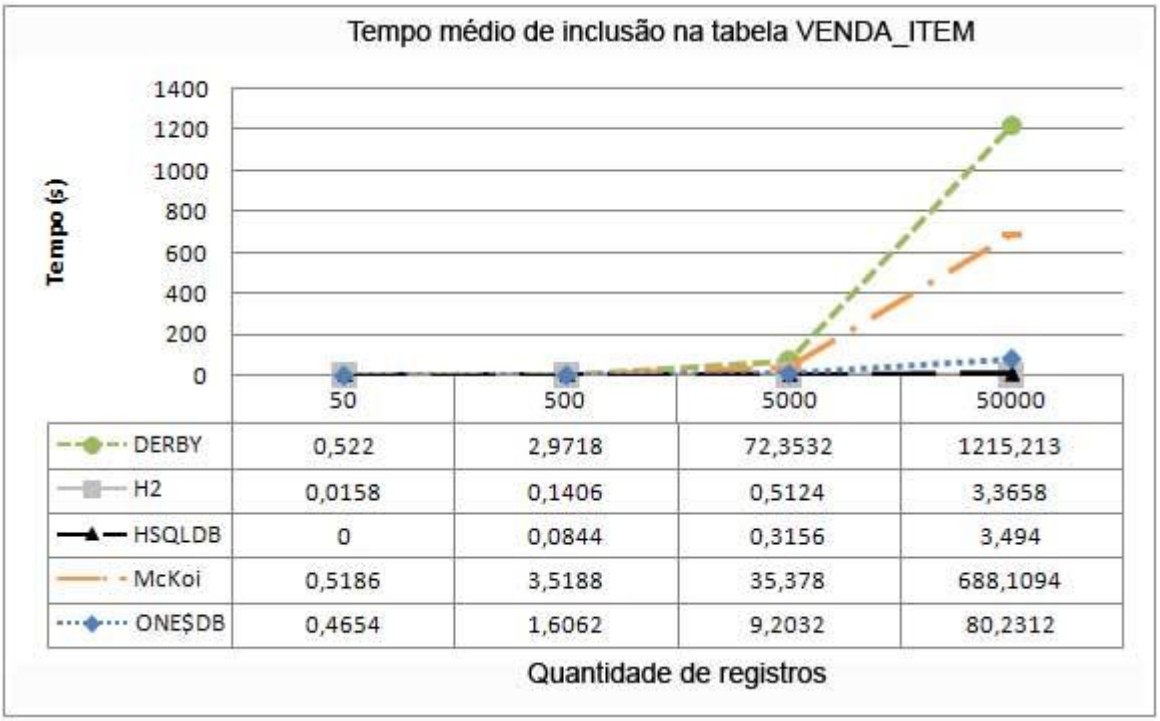
Observou-se que Derby, H2 Database, HSQLDB e One$DB mativeram suas médias de desempenho dos testes anteriores, mostrando que com dois relacionamentos, o desempenho também não é afetado de forma significante para estes SGBDs. Já com o SGBD

Mckoi, verificou-se que houve uma queda de desempenho se comparado aos testes com tabelas solitárias e com apenas um relacionamento. Os resultados são ilustrados na Figura 30 e Figura 31.



	50	500	5000	50000
DERBY	0,5282	30,25	77,1312	1177,8628
H2	0,0156	0,1562	0,5938	4,1376
HSQLDB	0,0094	0,0784	0,3656	2,2906
McKoi	0,478	3,2818	29,55	609,4344
ONE$DB	0,4504	1,5312	8,5	73,4718

Figura 30 - Desempenho de inclusão na tabela TABELA_ITEM



	50	500	5000	50000
DERBY	0,522	2,9718	72,3532	1215,213
H2	0,0158	0,1406	0,5124	3,3658
HSQLDB	0	0,0844	0,3156	3,494
McKoi	0,5186	3,5188	35,378	688,1094
ONE$DB	0,4654	1,6062	9,2032	80,2312

Figura 31 - Desempenho de inclusão na tabela VENDA_ITEM

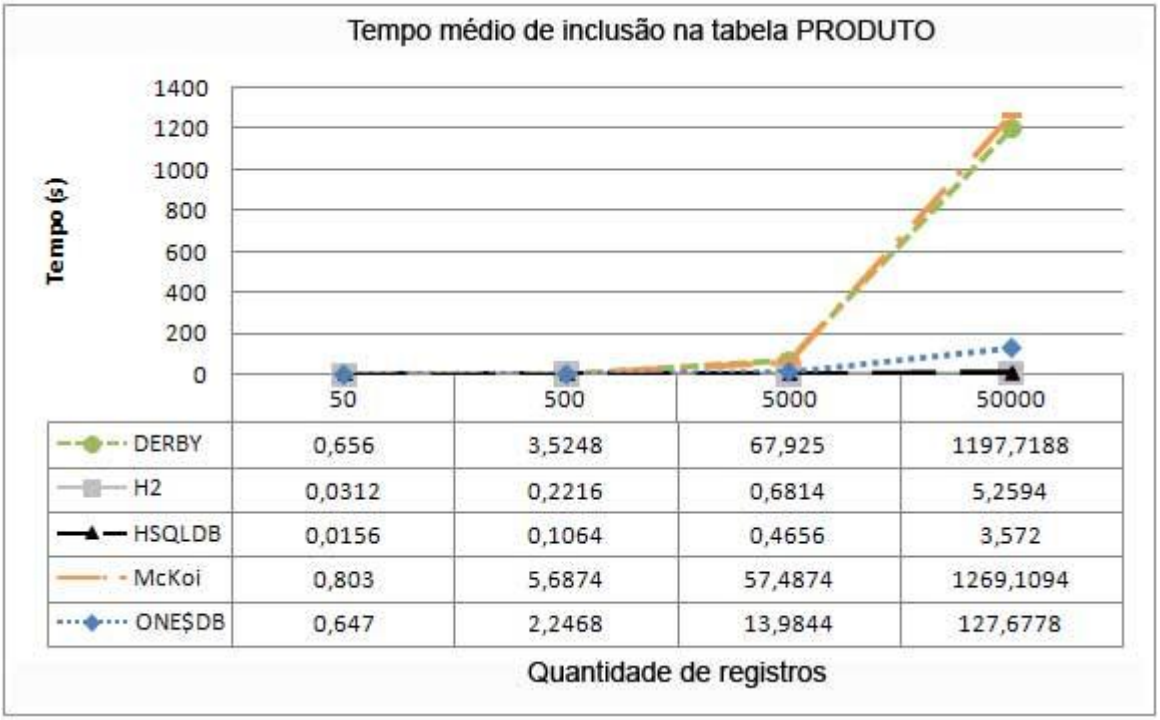
4.1.7. Teste de desempenho de armazenamento em tabela com quatro relacionamentos

Da mesma forma que nos testes anteriores, foram inseridas seqüências com volumes variados de registros. A tabela utilizada foi a tabela PRODUTO, por possuir 4 relacionamentos.

Desta vez, se pode observar que, com exceção do Derby, houve um aumento significante nos tempos de inclusão dos SGBDs, mostrando o quanto a garantia de integridade referencial pode degradar o desempenho do Banco de Dados.

Mais uma vez, o SGBD Derby manteve o desempenho anterior, ficando com o tempo de inclusão abaixo do tempo do Mckoi.

Os outros SGBDs apresentaram uma queda de desempenho proporcional, e os SGBDs H2 e HSQLDB se mantiveram entre os mais velozes, conforme pode ser observado na Figura 32.



	50	500	5000	50000
DERBY	0,656	3,5248	67,925	1197,7188
H2	0,0312	0,2216	0,6814	5,2594
HSQLDB	0,0156	0,1064	0,4656	3,572
McKoi	0,803	5,6874	57,4874	1269,1094
ONE$DB	0,647	2,2468	13,9844	127,6778

Figura 32 - Desempenho de inclusão na tabela PRODUTO

4.1.8. Teste de desempenho de recuperação em tabelas solitárias

Neste teste foram recuperados volumes variados de registros em tabelas solitárias, sendo utilizadas as tabelas CATEGORIA, COR, ORIGEM e FABRICANTE.

Os volumes de registros utilizados nos testes de recuperação, foram os mesmos utilizados nos testes de inclusão: 50, 500, 5000 e 50000.

A análise foi separada em duas partes: tabelas com poucas colunas e tabela com maior quantidade de colunas. Os resultados foram obtidos com tabelas com poucas colunas como são mostrados na Figura 33, Figura 34 e Figura 35.

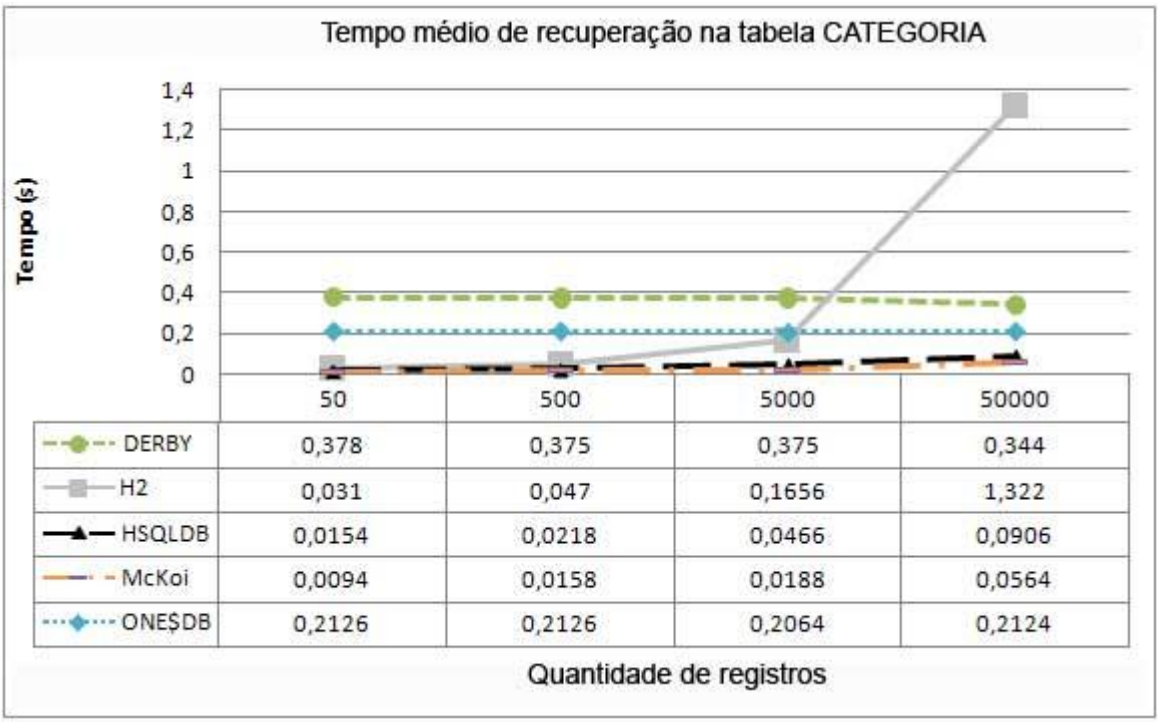


Figura 33 - Desempenho de recuperação na tabela CATEGORIA

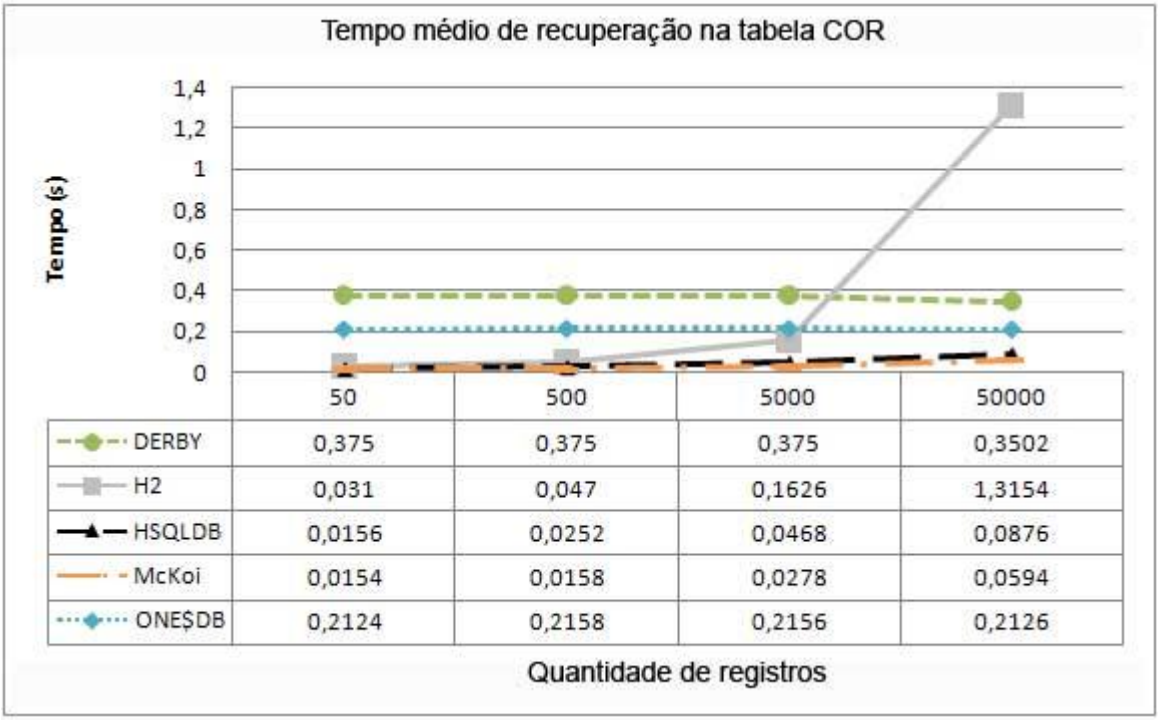
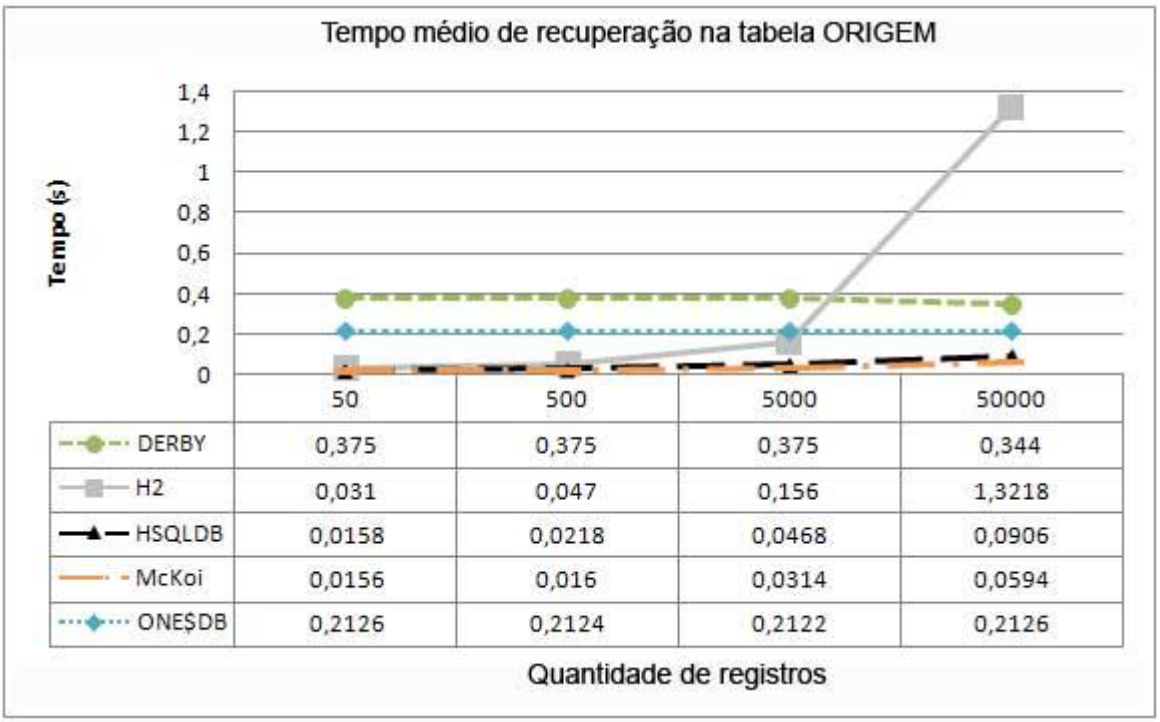


Figura 34 – Desempenho de recuperação na tabela COR



	50	500	5000	50000
DERBY	0,375	0,375	0,375	0,344
H2	0,031	0,047	0,156	1,3218
HSQLDB	0,0158	0,0218	0,0468	0,0906
McKoi	0,0156	0,016	0,0314	0,0594
ONE$DB	0,2126	0,2124	0,2122	0,2126

Figura 35 - Desempenho de recuperação na tabela ORIGEM

Devido à semelhança na estrutura entre as tabelas CATEGORIA, COR e ORIGEM, os resultados observados são praticamente os mesmos para recuperação de dados nestas três tabelas.

Comparando-se o desempenho apresentado pelos cinco SGBDs, pode-se fazer as seguintes observações:

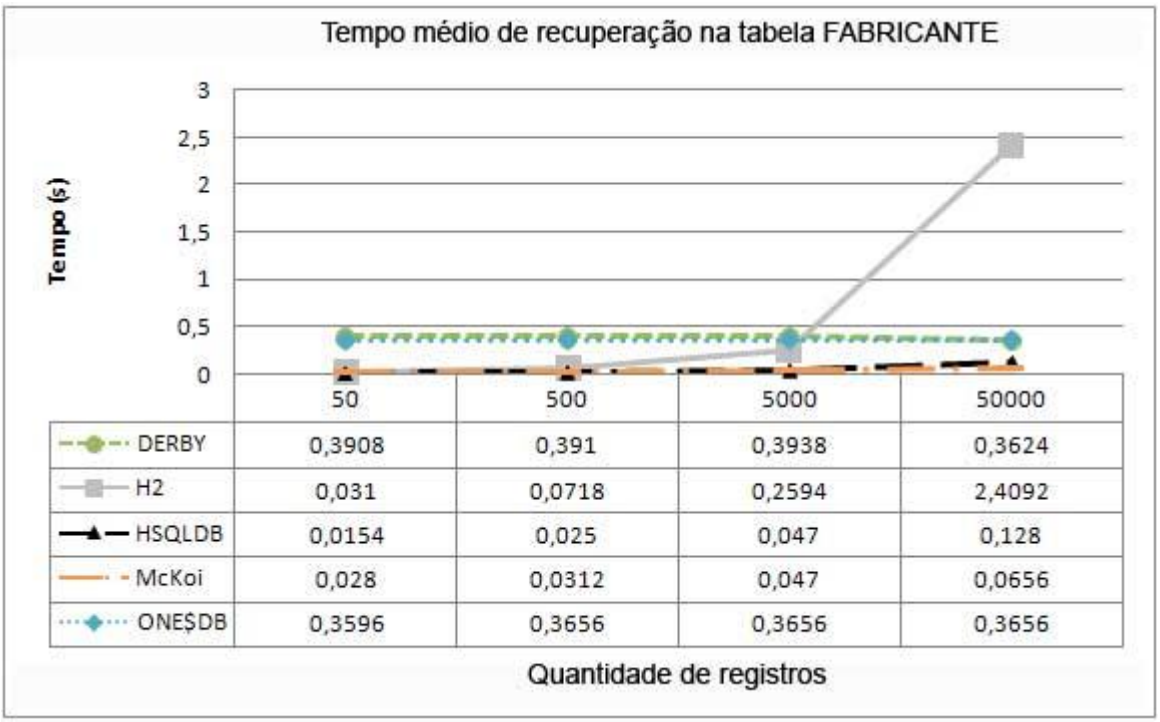
- O Mckoi que se apresentou entre os mais lentos nos testes de inclusão, foi o mais veloz na recuperação em tabelas solitárias levando 0,05 segundos para recuperar 50000 registros;
- O HSQLDB também apresentou um bom desempenho, mostrando-se veloz também na recuperação de dados, sendo o segundo mais rápido com o tempo de 0,09 segundos para recuperar 50000 registros;
- O H2 que se apresentou entre os mais rápidos nos testes de inserção, apresentou a maior queda de desempenho na medida em que a quantidade de registros recuperados foi aumentada, mostrando-se o mais lento na recuperação de 50000

registros, apresentando um tempo de 1,3218 segundos contra 0,0594 segundos apresentados pelo Mckoi que foi o mais rápido;

- O One$DB apresentou um tempo constante de 0,21 segundos, mostrando que seu desempenho não é afetado com relação à quantidade de registros;
- O Derby mostrou-se o mais lento recuperando um volume aproximado de até 5000 registros, apresentando uma pequena melhora de desempenho na medida em que a quantidade de registros foi aumentada.

Com relação aos testes com tabelas que possuem uma quantidade grande de colunas, foi utilizada a tabela FABRICANTE, e pode-se fazer as seguintes observações:

- O H2 mostrou ser o que tem o desempenho mais afetado na medida em que a quantidade de registros aumenta, sendo o mais lento, levando 2,4 segundos para recuperar 50000 registros enquanto para recuperar este volume na tabela CATEGORIA levou 1,3 segundos;
- Derby, HSQLDB e Mckoi não apresentaram queda de desempenho muito significativa se comparados aos testes em tabelas com poucas colunas;
- One$DB apresentou um pequeno aumento no tempo de recuperação em relação ao teste em tabelas com poucos registros, passando de 0,21 segundos para 0,36 segundos, mas manteve um desempenho constante, não sendo afetada pela quantidade de registros;
- Mckoi permaneceu sendo o mais rápido mesmo em tabelas com uma quantidade maior de colunas, levando 0,06 segundos para recuperar um volume de 50000 registros.



	50	500	5000	50000
DERBY	0,3908	0,391	0,3938	0,3624
H2	0,031	0,0718	0,2594	2,4092
HSQLDB	0,0154	0,025	0,047	0,128
McKoi	0,028	0,0312	0,047	0,0656
ONE$DB	0,3596	0,3656	0,3656	0,3656

Figura 36 - Desempenho de recuperação na tabela FABRICANTE

4.1.9. Teste de desempenho de recuperação em tabelas com um relacionamento

Este teste verificou o desempenho na recuperação em tabelas relacionadas, consistiu em recuperar os dados de um registro na tabela filho e de seu correspondente na tabela pai através do comando *JOIN*, por exemplo, para se recuperar os dados de 50 registros da tabela PESSOA_FISICA e de seu correspondente da tabela CLIENTE utilizou-se o seguinte comando:

```
SELECT * FROM PESSOA_FISICA
JOIN CLIENTE ON PESSOA_FISICA.cli_codigo = CLIENTE.cli_codigo
WHERE PESSOA_FISICA.cli_codigo < 50;
```

Figura 37 – Exemplo de comando utilizado na recuperação em tabelas relacionadas

Assim como nos testes anteriores, foram recuperados volumes variados de registros nas tabelas e os tempos foram registrados. Para este teste foram utilizadas as tabelas PESSOA_FISICA, TABELA e VENDA.

Neste teste pode-se observar uma queda de desempenho muito grande do SGBD Mckoi com relação aos testes em tabelas solitárias, onde ele se apresentou o mais rápido. Nesta situação ele se apresentou entre os mais lentos, mostrando o quanto a integridade referencial afeta seu desempenho.

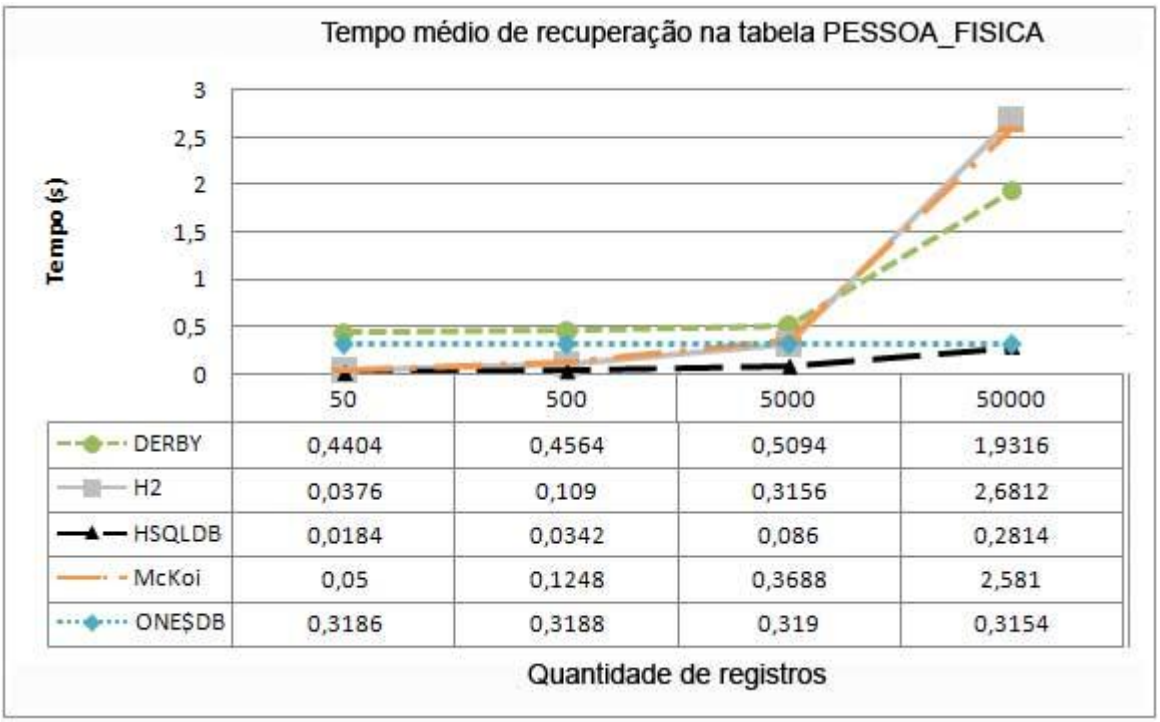
O H2 apresentou um bom desempenho sendo o segundo mais rápido até um volume aproximado de 5000 registros, mas ao se recuperar 50000 registros passa a ser o mais lento entre os cinco SGBDs.

O HSQLDB também apresentou um bom desempenho neste teste, sendo o mais rápido na recuperação de todos os volumes de registros testados.

O One$DB, assim como no teste anterior, manteve um tempo constante de recuperação, e levou praticamente o mesmo tempo para recuperar 50 e 50000 registros, sendo o segundo mais rápido na recuperação de 50000 com o tempo de 0,31 segundos.

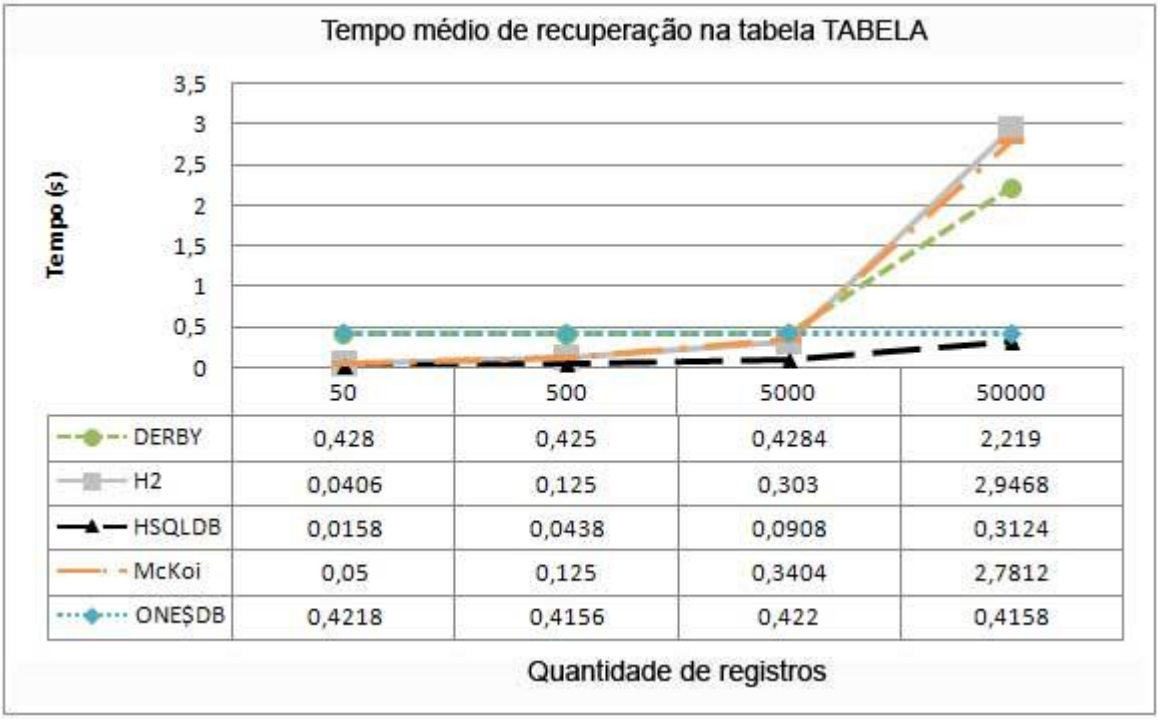
Já o Derby, que assim como o One$DB havia apresentado um tempo constante de recuperação em todos os volumes de registros em tabelas solitárias, neste teste, apresentou queda de desempenho conforme a quantidade de registros aumentou, sendo o mais lento na recuperação de 50, 500 e 5000 registros e o terceiro mais lento na recuperação de 50000 registros.

Os resultados podem ser observados na Figura 38, Figura 39 e Figura 40.



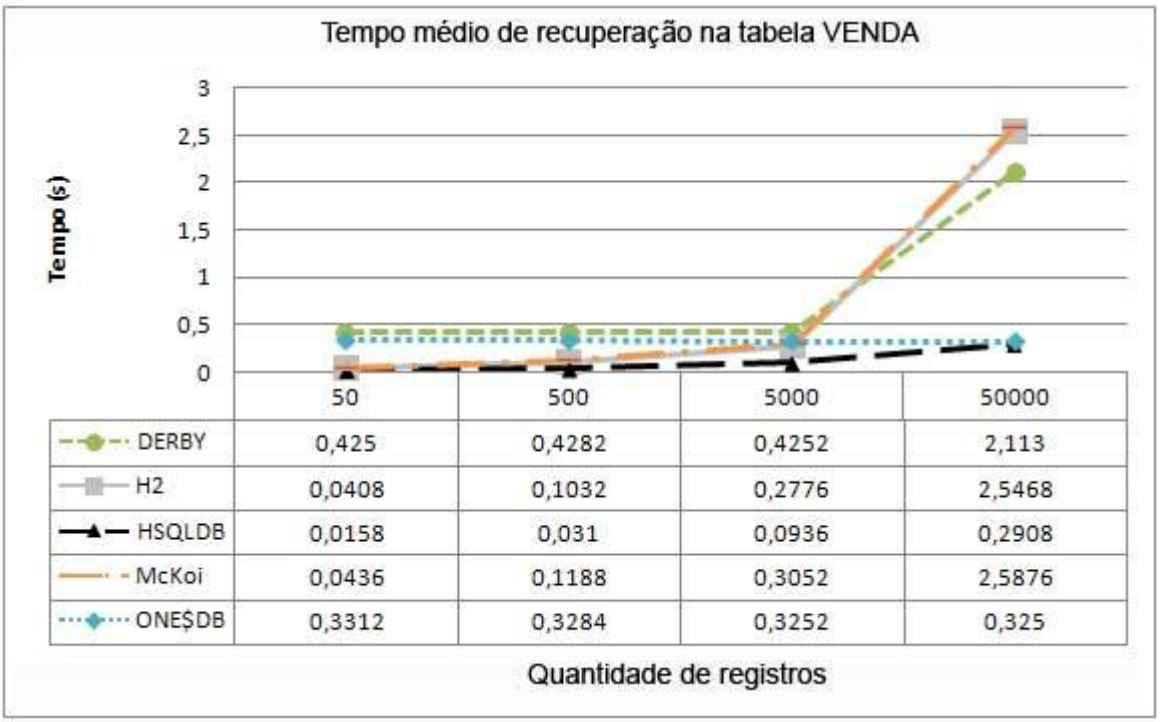
	50	500	5000	50000
DERBY	0,4404	0,4564	0,5094	1,9316
H2	0,0376	0,109	0,3156	2,6812
HSQLDB	0,0184	0,0342	0,086	0,2814
McKoi	0,05	0,1248	0,3688	2,581
ONE$DB	0,3186	0,3188	0,319	0,3154

Figura 38 - Desempenho de recuperação na tabela PESSOA_FISICA



	50	500	5000	50000
DERBY	0,428	0,425	0,4284	2,219
H2	0,0406	0,125	0,303	2,9468
HSQLDB	0,0158	0,0438	0,0908	0,3124
McKoi	0,05	0,125	0,3404	2,7812
ONE$DB	0,4218	0,4156	0,422	0,4158

Figura 39 - Desempenho de recuperação na tabela TABELA

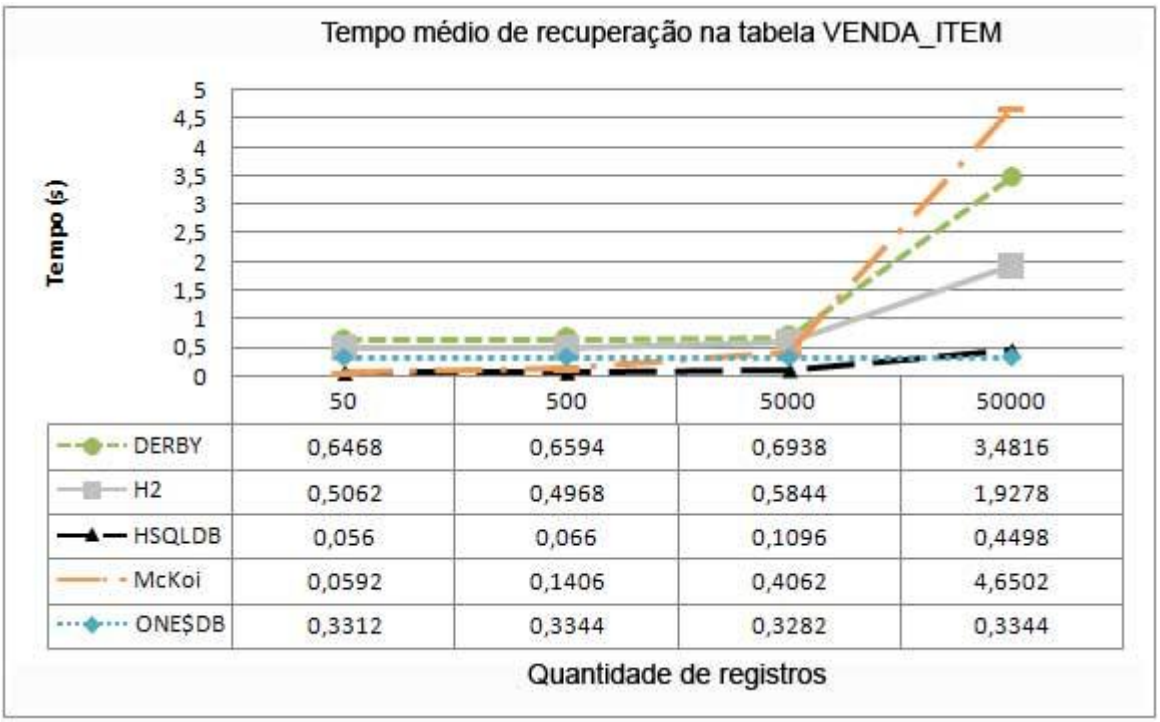


	50	500	5000	50000
DERBY	0,425	0,4282	0,4252	2,113
H2	0,0408	0,1032	0,2776	2,5468
HSQLDB	0,0158	0,031	0,0936	0,2908
McKoi	0,0436	0,1188	0,3052	2,5876
ONE$DB	0,3312	0,3284	0,3252	0,325

Figura 40 - Desempenho de recuperação na tabela VENDA

4.1.10. Teste de desempenho de recuperação em tabelas com dois relacionamentos

Este teste verificou o desempenho de recuperação em tabelas com dois relacionamentos. A tabela utilizada neste teste foi a VENDA_ITEM, o resultado pode ser observado na Figura 41.



	50	500	5000	50000
DERBY	0,6468	0,6594	0,6938	3,4816
H2	0,5062	0,4968	0,5844	1,9278
HSQLDB	0,056	0,066	0,1096	0,4498
McKoi	0,0592	0,1406	0,4062	4,6502
ONE$DB	0,3312	0,3344	0,3282	0,3344

Figura 41 - Desempenho de recuperação na tabela VENDA_ITEM

Pode-se observar que o HSQLDB mostrou-se o mais rápido até o teste com 5000 registros, e ao recuperar 50000 foi mais lento que o One$DB que apresentou uma velocidade constante de 0,3 segundos em todos os volumes de registros recuperados.

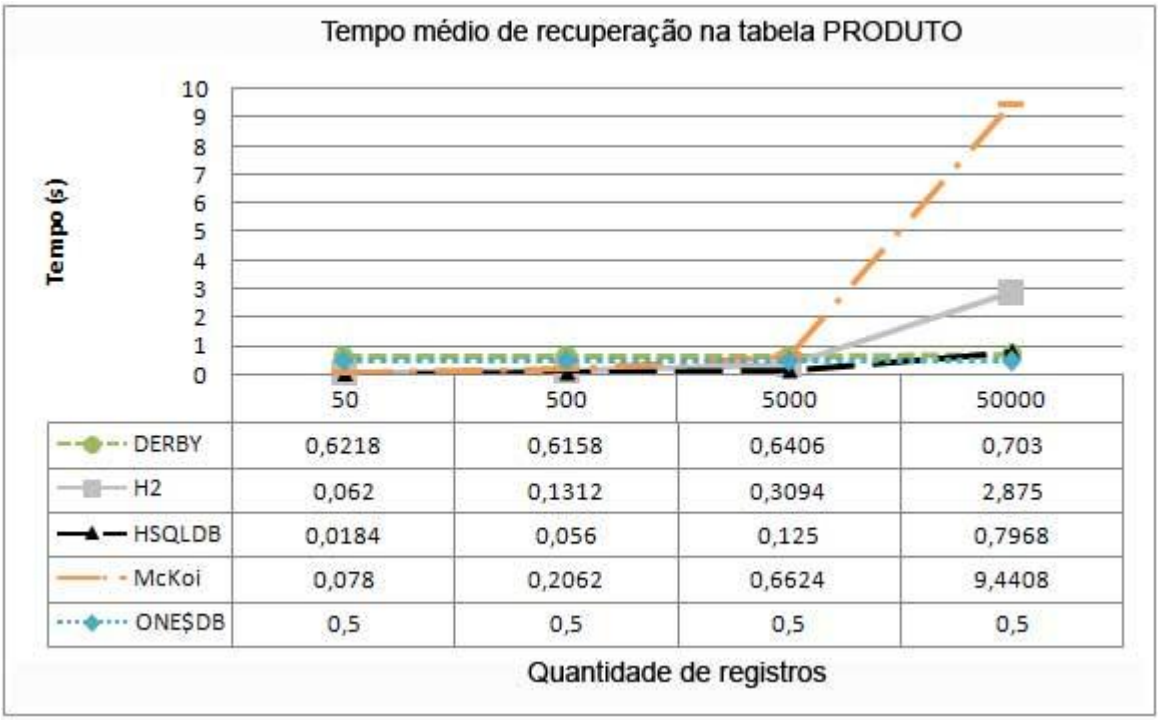
O Mckoi apresentou grande queda de desempenho ainda maior que nos testes com tabelas com um relacionamento, levando 4,6 segundos para recuperar 50000 registros, mostrando novamente como a integridade referencial afeta seu desempenho.

Assim como no teste em tabelas com um relacionamento, o Derby foi o mais lento até uma quantidade de 5000 registros, apresentando uma pequena queda de desempenho na medida em que a quantidade de registros aumentou, sendo o segundo mais lento na recuperação de 50000 registros com o tempo de 3,4816.

O H2 mostrou-se o segundo mais lento na recuperação de até 5000 registros e apresentou uma queda de desempenho constante na medida em que a quantidade de registros aumentou, mas em uma proporção menor que o Derby e o Mckoi, sendo o terceiro mais lento na recuperação de 50000 registros.

4.1.11. Teste de desempenho de recuperação em tabelas com quatro relacionamentos

Este teste verificou o desempenho na recuperação em tabelas com quatro relacionamentos. Foram recuperados os dados da tabela PRODUTO e das quatro tabelas relacionadas a ela: CATEGORIA, COR, ORIGEM e FABRICANTE. Os resultados podem ser observados na Figura 42.



	50	500	5000	50000
DERBY	0,6218	0,6158	0,6406	0,703
H2	0,062	0,1312	0,3094	2,875
HSQLDB	0,0184	0,056	0,125	0,7968
McKoi	0,078	0,2062	0,6624	9,4408
ONE$DB	0,5	0,5	0,5	0,5

Figura 42 - Desempenho de recuperação na tabela PRODUTO

Assim como já foi observado nos testes anteriores de recuperação em tabelas relacionadas, o Mckoi apresentou uma queda de desempenho muito grande neste teste conforme a quantidade de registrou aumentou, mostrando claramente como a integridade referencial afeta o desempenho deste SGBD quando se tratam de grandes quantidades de dados. Foi o mais lento dos SGBDs, com o tempo de 9,4 segundos na recuperação de 50000 registros.

Pode-se observar que a integridade referencial não afeta o desempenho dos SGBDs Derby e One$DB com relação à quantidade de registros. Eles apresentaram-se como sendo os mais lentos com pequena quantidade de registros, com o tempo de 0,6 e 0,5, respectivamente, mas mantiveram um desempenho constante na medida em que a quantidade de registros foi aumentando e acabaram sendo os dois SGBDs mais rápidos na inserção de 50000 registros.

O H2 apresentou-se como o segundo mais rápido na recuperação de 50 registros, com o tempo de 0,06 segundos, mas teve uma queda de desempenho proporcional na medida em que a quantidade de registrou aumentou, sendo o segundo mais lento na recuperação de 50000 registros com o tempo de 2,8 segundos.

O HSQLDB foi o mais rápido até um volume de 5000 registros, mas como apresentou uma queda de desempenho proporcional na medida em que a quantidade de registros aumentou, foi mais lento que o One$DB na recuperação de 50000 registros, pois este SGBD apresentou um tempo constante para todos os volumes de registros recuperados, sendo o mais rápido na recuperação deste montante com o tempo de 0,5 segundos.

4.2. Análise Geral

A Tabela 13 foi construída de acordo com os resultados apresentados neste capítulo. A primeira coluna indica o requisito da aplicação e as demais colunas indicam os SGBDs com os resultados obtidos. Os resultados que apresentam números se referem a um comparativo entre os resultados obtidos nos testes, por ordem crescente de desempenho, já os resultados exibidos com a letra X, identificam que o SGBD satisfez o requisito apontado.

Tabela 13 – Resumo comparativo dos SGBDs

Descrição	*Derby*	*H2*	*HSQLDB*	*McKoi*	*One$DB*
Integridade dos tipos de dados	**X**	**X**			**X**
Integridade de chave primária	**X**	**X**	**X**	**X**	**X**
Integridade referencial (inclusão)	**X**	**X**	**X**	**X**	**X**
Integridade referencial (exclusão)	**X**	**X**	**X**	**X**	**X**
Integridade referencial (alteração)	**X**	**X**	**X**	**X**	**X**
Desempenho de armazenamento em tabelas solitárias (50 registros)	**3**	**2**	**1**	**5**	**4**
Desempenho de armazenamento em tabelas solitárias (50000 registros)	**5**	**2**	**1**	**4**	**3**
Desempenho de armazenamento em tabelas com Um relacionamento (50 registros)	**4**	**2**	**1**	**3**	**5**
Desempenho de armazenamento em tabelas com Um relacionamento (50000 registros)	**5**	**2**	**1**	**4**	**3**
Desempenho de armazenamento em tabelas com Dois relacionamentos (50 registros)	**5**	**2**	**1**	**4**	**3**
Desempenho de armazenamento em tabelas com Dois relacionamentos (50000 registros)	**5**	**2**	**1**	**4**	**3**
Desempenho de armazenamento em tabelas com Quatro relacionamentos (50 registros)	**4**	**2**	**1**	**5**	**3**
Desempenho de armazenamento em tabelas com Quatro relacionamentos (50000 registros)	**4**	**2**	**1**	**5**	**3**
Desempenho de recuperação em tabelas solitárias (50 registros)	**5**	**3**	**2**	**1**	**4**
Desempenho de recuperação em tabelas solitárias (50000 registros)	**4**	**5**	**2**	**1**	**3**
Desempenho de recuperação em tabelas com Um relacionamento (50 registros)	**5**	**2**	**1**	**3**	**4**
Desempenho de recuperação em tabelas com Um relacionamento (50000 registros)	**3**	**5**	**1**	**4**	**2**
Desempenho de recuperação em tabelas com Dois relacionamentos (50 registros)	**5**	**4**	**1**	**2**	**3**

Tabela 14 – Resumo comparativo dos testes de integridade dos SGBDs (continuação)

Descrição	*Derby*	*H2*	*HSQLDB*	*McKoi*	*One$DB*
Desempenho de recuperação em tabelas com Dois relacionamentos (50000 registros)	**4**	**3**	**2**	**5**	**1**
Desempenho de recuperação em tabelas com Quatro relacionamentos (50 registros)	**5**	**2**	**1**	**3**	**4**
Desempenho de recuperação em tabelas com Quatro relacionamentos (50000 registros)	**2**	**4**	**3**	**5**	**1**

Conforme é mostrado na tabela Tabela 13, pode-se concluir que a escolha de um SGBD depende muito da aplicação, uma vez que se pode observar que alguns SGBDs tem desempenhos diferentes dependendo da quantidade de dados e da quantidade de relacionamentos que são processados.

CONCLUSÕES

O objetivo deste trabalho foi apresentar um estudo comparativo entre cinco SGBDs gratuitos implementados como APIs Java (Apache Derby, H2 Database, HSQLDB, McKoi, One$DB) e que podem ser embarcados dentro de uma aplicação, para isto foram realizados testes de integridade e desempenho. A partir dos resultados apresentados no capítulo anterior, são esboçadas as conclusões descritas a seguir.

O SGBD Derby, apesar de ser eficaz no controle dos tipos de dados e integridade de chave primária e referencial, esteve sempre entre os mais lentos nos teste de desempenho de armazenamento e recuperação.

O H2, assim como o HSQLDB, apresentou desempenho muito superior aos outros SGBDs em relação ao armazenamento. Também apresentou controle eficaz de tipos de dados, integridade de chave primária e integridade referencial, mostrou-se também veloz na recuperação de poucos registros, porém obteve um baixo desempenho nos testes de recuperação de grandes volumes de dados, sendo um dos mais lentos neste quesito. Pode-se concluir que é uma boa opção para aplicações que não manipulem um grande volume de dados.

Dentre os SGBDs estudados, pode-se concluir que o HSQLDB foi o que se manteve entre os mais rápidos em todos os testes de desempenho realizados, tanto no armazenamento quanto na recuperação de dados. Também apresentou controle eficaz de chave primária e integridade referencial, mas no teste de tipos de dados, falhou no controle de tamanhos de campos char e varchar, permitindo que fossem inseridos valores maiores do que os definidos. Uma observação importante a ser feita, é que o HSQLDB apresentou o erro

“java.lang.OutOfMemoryError: Java heap space” e foi necessário a liberação de 1Gb de memória RAM para a JVM para que se pudessem concluir os testes com 50000 registros.

O SGBD Mckoi, mostrou não controlar os tipos de dados de forma eficaz, falhando em quase todos os testes deste tipo. Com relação à integridade de chave primária e referencial, mostrou controle eficaz, mas em relação ao desempenho, se manteve entre os mais lentos em todos os testes de armazenamento. Nos testes de recuperação, foi o mais rápido em tabelas solitárias, mas mostrou grande queda de desempenho em tabelas relacionadas.

O One$DB apresentou controle eficaz de tipos de dados, integridade de chave primária e referencial. Com relação ao desempenho, mostrou-se muito superior no armazenamento de grandes quantidades se comparado a Derby e Mckoi, mas teve um desempenho muito menor se comparado a HSQLDB e H2. Já nos testes de recuperação, mostrou ser uma boa opção para recuperação de grandes volumes de registros, apresentando pouca variação de tempo em relação à quantidade de registros, sendo o mais rápido na recuperação de 50000 registros em alguns testes.

Como foi dito anteriormente, a escolha de um SGBD depende muito dos requisitos da aplicação, quantidade de dados, velocidade, segurança, e também requisitos de hardware. Como se pode observar, o SGBD HSQLDB obteve melhor desempenho, mas exigiu que se fossem liberados 1Gb de memória RAM para a JVM e também não foi eficaz no controle de tipos de dados char e varchar. Apresentou-se como outra boa opção o SGBD H2 para aplicações que não requerem recuperação de grandes volumes de dados, pois mostrou controle eficaz de integridade e bom desempenho de armazenamento e recuperação de até 5000 registros.

Referências bibliográficas

Apache Derby Project Charter. Disponível em http://db.apache.org/derby/derby_charter.html. Acesso em: 29 março 2007.

CHEN, Peter. **Modelagem de dados:** A abordagem entidade-relacionamento para projeto lógico. São Paulo: McGraw-Hill: Makron, 1990.

Daffodil DB SQL Reference Guide. Disponível em http://www.daffodildb.com/. Acesso em 10 abril 2007.

DEITEL, Harvey M.; DEITEL, Paul J. **Java Como Programar**. 4. ed. Porto Alegre: Bookman, 2003.

Derby Reference Manual. Disponível em http://db.apache.org/derby/docs/10.2/ref/. Acesso em: 29 março 2007.

GUIMARÃES, C. C. **Fundamentos de banco de dados:** modelagem, projeto e linguagem SQL. Campinas: Editora da Unicamp, 2003.

H2 Database Documentation. Disponível em http://www.h2database.com/h2.pdf . Acesso em: 01 abril 2007.

HSQLDB. Disponível em http://pt.wikipedia.org/wiki/HSQLDB. Acesso em 03 abril 2007**.**

MACHADO, Felipe; ABREU, Mauricio. **Projeto de Banco de Dados:** Uma visão Prática. 6. ed. São Paulo: Érica, 1996.

McKoi SQL Database Manual. Disponível em http://mckoi.com/database/docindex.html. Acesso em 05 abril 2007.

SILBERSCHATZ, A., KORTH, H.F, SUDARSHAN, S., **Sistemas de Banco de Dados**. 3. ed. São Paulo: Makron Books, 1999.

SIMPSON, Blaine; TOUSSI Fred. **Hsqldb User Guide**. Disponível em http://www.hsqldb.org/web/hsqlDocsFrame.html. Acesso em 03 abril 2007.

APÊNDICE A – Código fonte da aplicação implementada em *Java* para realização dos testes de desempenho

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.io.*;

import com.sun.org.apache.xpath.internal.operations.Mod;

public class AvaliaBD {
   public static void main(String args[]) {
      Connection conn = null;
      Statement  stm;
      ResultSet  result = null;
      long       t, hora_inicio, hora_fim;
      int        QTD, i, inicio, fim;
      String     BANCO, OPERACAO, TABELA;

      if (args.length != 4) {
         System.out.println("Banco, Operação, Quantidade ou Tabela não informado na chamada
do programa.");
         System.exit(0);
      } else {
         BANCO    = args[0];
         OPERACAO = args[1];
         QTD      = Integer.parseInt(args[2]);
         TABELA   = args[3];

         try {
            if (BANCO.equals("Derby")) {
               Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
               conn =
DriverManager.getConnection("jdbc:derby:.\\DATABASE\\DERBY\\bdteste;create=true");
            } else if (BANCO.equals("H2")) {
               Class.forName("org.h2.Driver");
               conn =
DriverManager.getConnection("jdbc:h2:.\\DATABASE\\H2\\bdteste;create=true");
            } else if (BANCO.equals("HSQLDB")) {
               Class.forName("org.hsqldb.jdbcDriver");
               conn =
DriverManager.getConnection("jdbc:hsqldb:.\\DATABASE\\HSQLDB\\bdteste;create=true");
            } else if (BANCO.equals("McKoi")) {
               Class.forName("com.mckoi.JDBCDriver");
               conn =
DriverManager.getConnection("jdbc:mckoi:local://./DATABASE/MCKOI/bdteste.conf?create_or
_boot=true","mckoi","mckoi");
            } else if (BANCO.equals("One$DB")) {
               Class.forName("in.co.daffodil.db.jdbc.DaffodilDBDriver");
               conn =
```

```
DriverManager.getConnection("jdbc:daffodilDB_embedded:bdteste;create=true;path=.//DATAB
ASE//ONEDB");
          } else {
             System.out.println("\r\nERRO: banco "+ BANCO +" NAO ENCONTRADO!!!!");
                System.exit(0);
          }

          //inicializa o statement com a conexao e  seta autocommit
          stm = conn.createStatement();
          conn.setAutoCommit(true);

          //EXECUTA INCLUSÃO NA TABELA INDICADA
          if (OPERACAO.equals("inclusao")) {
             if (TABELA.equals("categoria")) {
                try {
                   result = stm.executeQuery("SELECT max(cat_codigo) FROM categoria");
                   result.next();

                   i     = result.getInt(1) + 1;
                   inicio = i;
                   fim    = inicio + (QTD - 1);

                   hora_inicio = System.currentTimeMillis();
                   for (i=inicio;i<=fim;i++){
                      stm.executeUpdate(
                         "INSERT INTO categoria("+
                            "cat_codigo, "+
                            "cat_descricao, "+
                            "cat_sigla"+
                         ")VALUES("+
                            i+","+
                            "'Categoria',"+
                            "'CT'"+
                         ")"
                      );
                   }
                   hora_fim = System.currentTimeMillis();
                   t = hora_fim - hora_inicio;

                   try{
                      PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                      out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                      out1.close();
                   } catch (Exception e) {
                      System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                   }
                   System.out.println("concluido");
                } catch (Exception e){
                    System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
                }
             }

             else if (TABELA.equals("cor")) {
```

```
                try {
                   result = stm.executeQuery("SELECT max(cor_codigo) FROM cor");
                   result.next();

                   i     = result.getInt(1) + 1;
                   inicio = i;
                   fim    = inicio + (QTD - 1);

                   hora_inicio = System.currentTimeMillis();
                   for (i=inicio;i<=fim;i++){
                      stm.executeUpdate(
                         "INSERT INTO cor("+
                            "cor_codigo, "+
                            "cor_descricao, "+
                            "cor_sigla"+
                         ")VALUES("+
                            i+","+
                            "'Cor',"+
                            "'CR'"+
                         ")"
                      );
                   }
                   hora_fim = System.currentTimeMillis();
                   t = hora_fim - hora_inicio;

                   try{
                      PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                      out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                      out1.close();
                   } catch (Exception e) {
                      System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                   }
                   System.out.println("concluido");
                } catch (Exception e){
                   System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
                }
             }

             else if (TABELA.equals("origem")) {
                try {
                   result = stm.executeQuery("SELECT max(ori_codigo) FROM origem");
                   result.next();

                   i     = result.getInt(1) + 1;
                   inicio = i;
                   fim    = inicio + (QTD - 1);

                   hora_inicio = System.currentTimeMillis();
                   for (i=inicio;i<=fim;i++){
                      stm.executeUpdate(
                         "INSERT INTO origem("+
                            "ori_codigo, "+
                            "ori_descricao, "+
```

```
                        "ori_sigla"+
                      ")VALUES("+
                        i+","+
                        "'Origem',"+
                        "'OR'"+
                      ")"
                    );
                  }
                  hora_fim = System.currentTimeMillis();
                  t = hora_fim - hora_inicio;

                  try{
                    PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                    out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                    out1.close();
                  } catch (Exception e) {
                    System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                   }
                   System.out.println("concluido");
                } catch (Exception e){
                   System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
                }
              }

              else if (TABELA.equals("fabricante")) {
                try {
                  result = stm.executeQuery("SELECT max(fab_codigo) FROM fabricante");
                  result.next();

                  i     = result.getInt(1) + 1;
                  inicio = i;
                  fim    = inicio + (QTD - 1);

                  hora_inicio = System.currentTimeMillis();
                  for (i=inicio;i<=fim;i++){
                    stm.executeUpdate(
                      "INSERT INTO fabricante("+
                        "fab_codigo, "+
                        "fab_cep, "+
                        "fab_nome, "+
                        "fab_cidade, "+
                        "fab_estado, "+
                        "fab_endereco, "+
                        "fab_bairro, "+
                        "fab_complemento, "+
                        "fab_numero, "+
                        "fab_telefone, "+
                        "fab_fax, "+
                        "fab_cnpj, "+
                      "fab_ie, "+
                        "fab_site, "+
                        "fab_email_gerente, "+
                        "fab_razao_social, "+
```

```
                        "fab_contato, "+
                        "fab_contato2, "+
                        "fab_observacao, "+
                        "fab_ativo"+
                      ")VALUES("+
                        i+", "+
                        "'CEP', "+
                        "'Fabrica', "+
                        "'Cidade', "+
                        "'SP', "+
                        "'Endereco', "+
                        "'Bairro', "+
                        "'Complemento', "+
                        "'nº', "+
                        "'Tel', "+
                        "'Fax', "+
                        "'CNPJ', "+
                        "'IE', "+
                        "'www.fabrica.com.br', "+
                        "'gerente@fabrica.com.br', "+
                        "'Fabrica LTDA', "+
                        "'Contato1 Fabrica', "+
                        "'Contato2 Fabrica', "+
                        "'Esta é a fábrica', "+
                        "'t'"+
                      ")"
                    );
                }
                hora_fim = System.currentTimeMillis();
                t = hora_fim - hora_inicio;

                try{
                  PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                  out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                  out1.close();
                } catch (Exception e) {
                  System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                }
                System.out.println("concluido");
              } catch (Exception e){
                System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
              }
            }

            else if (TABELA.equals("cliente")) {
              try {
                result = stm.executeQuery("SELECT max(cli_codigo) FROM cliente");
                result.next();

                i     = result.getInt(1) + 1;
                inicio = i;
                fim    = inicio + (QTD - 1);
```

```
                    hora_inicio = System.currentTimeMillis();
                    for (i=inicio;i<=fim;i++){
                       stm.executeUpdate(
                          "INSERT INTO cliente ("+
                             "cli_codigo, "+
                             "cli_estado, "+
                             "cli_cidade, "+
                             "cli_bairro, "+
                             "cli_numero, "+
                             "cli_endereco, "+
                             "cli_telefone, "+
                             "cli_tipo_cliente, "+
                             "cli_nome, "+
                             "cli_data_nascimento"+
                          ")VALUES("+
                             i+","+
                             "'SP', "+
                             "'Cidade', "+
                             "'Bairro', "+
                             "'nº', "+
                             "'endereco', "+
                             "'Tel', "+
                             "'F', "+
                             "'Cliente', "+
                             "current_date" +
                          ")"
                       );
                    }
                    hora_fim = System.currentTimeMillis();
                    t = hora_fim - hora_inicio;

                    try{
                       PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                       out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                       out1.close();
                    } catch (Exception e) {
                       System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                    }
                    System.out.println("concluido");
                 } catch (Exception e){
                    System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
                 }
              }

              else if (TABELA.equals("pessoa_fisica")) {
                 try {
                    result = stm.executeQuery("SELECT max(cli_codigo) FROM pessoa_fisica");
                    result.next();

                    i     = result.getInt(1) + 1;
                    inicio = i;
                    fim    = inicio + (QTD - 1);
```

```
                    hora_inicio = System.currentTimeMillis();
                    for (i=inicio;i<=fim;i++){
                       stm.executeUpdate(
                          "INSERT INTO pessoa_fisica("+
                             "cli_codigo,"+
                             "cli_cpf"+
                          ")VALUES("+
                             i+","+
                             "'CPF'"+
                          ")"
                       );
                    }
                    hora_fim = System.currentTimeMillis();
                    t = hora_fim - hora_inicio;

                    try{
                       PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                       out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                       out1.close();
                    } catch (Exception e) {
                       System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                    }
                    System.out.println("concluido");
                 } catch (Exception e){
                    System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
                 }
              }

              else if (TABELA.equals("pessoa_juridica")) {
                 try {
                    result = stm.executeQuery("SELECT max(cli_codigo) FROM pessoa_juridica");
                    result.next();

                    i     = result.getInt(1) + 1;
                    inicio = i;
                    fim    = inicio + (QTD - 1);

                    hora_inicio = System.currentTimeMillis();
                    for (i=inicio;i<=fim;i++){
                       stm.executeUpdate(
                          "INSERT INTO pessoa_juridica("+
                             "cli_codigo,"+
                             "cli_cnpj"+
                          ")VALUES("+
                             i+","+
                             "'CNPJ'"+
                          ")"
                       );
                    }
                    hora_fim = System.currentTimeMillis();
                    t = hora_fim - hora_inicio;

                    try{
```

```
                    PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                    out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                    out1.close();
                  } catch (Exception e) {
                    System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                  }
                  System.out.println("concluido");
                } catch (Exception e){
                  System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
                }
              }

              else if (TABELA.equals("produto")) {
                try {
                  result = stm.executeQuery("SELECT max(pro_codigo) FROM produto");
                  result.next();

                  i     = result.getInt(1) + 1;
                  inicio = i;
                  fim    = inicio + (QTD - 1);

                  hora_inicio = System.currentTimeMillis();
                  for (i=inicio;i<=fim;i++){
                    stm.executeUpdate(
                      "INSERT INTO produto("+
                        "pro_codigo,"+
                        "pro_descricao,"+
                        "pro_referencia,"+
                        "pro_ativo,"+
                        "fab_codigo,"+
                        "cat_codigo,"+
                        "cor_codigo,"+
                        "ori_codigo"+
                      ")VALUES("+
                        i+","+
                        "'Produto',"+
                        "'REF',"+
                        "'t',"+
                        "1,"+
                        "1,"+
                        "1,"+
                        "1"+
                      ")"
                    );
                  }
                  hora_fim = System.currentTimeMillis();
                  t = hora_fim - hora_inicio;

                  try{
                    PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                    out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                    out1.close();
```

```
                } catch (Exception e) {
                   System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                }
                System.out.println("concluido");
             } catch (Exception e){
                System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
             }
          }

          else if (TABELA.equals("tabela")) {
             try {
                result = stm.executeQuery("SELECT max(tab_codigo) FROM tabela");
                result.next();

                i     = result.getInt(1) + 1;
                inicio = i;
                fim    = inicio + (QTD - 1);

                hora_inicio = System.currentTimeMillis();
                int inc = 1;
                for (i=inicio;i<=fim;i++){
                   stm.executeUpdate(
                      "INSERT INTO tabela("+
                         "tab_codigo, "+
                         "tab_descricao, "+
                         "fab_codigo"+
                      ")VALUES("+
                         i+","+
                         "'Tabela',"+
                            inc+
                      ")"
                   );
                   if (i % 2 != 0) inc++;
                }
                hora_fim = System.currentTimeMillis();
                t = hora_fim - hora_inicio;

                try{
                   PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                   out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                   out1.close();
                } catch (Exception e) {
                   System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                }
                System.out.println("concluido");
             } catch (Exception e){
                System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
             }
          }

          else if (TABELA.equals("tabela_item")) {
             try {
                result = stm.executeQuery("SELECT max(tab_codigo) FROM tabela_item");
```

```
                    result.next();

                    i     = result.getInt(1) + 1;
                    inicio = i;
                    fim    = inicio + (QTD - 1);

                    hora_inicio = System.currentTimeMillis();
                    for (i=inicio;i<=fim;i++){
                      stm.executeUpdate(
                        "INSERT INTO tabela_item("+
                          "tab_codigo, "+
                          "pro_codigo, "+
                          "tit_preco"+
                        ")VALUES("+
                          i+","+
                          "1,"+
                          "10"+
                        ")"
                      );
                    }
                    hora_fim = System.currentTimeMillis();
                    t = hora_fim - hora_inicio;

                    try{
                      PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                      out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                      out1.close();
                    } catch (Exception e) {
                      System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                    }
                    System.out.println("concluido");
                  } catch (Exception e){
                    System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
                  }
                }

                else if (TABELA.equals("venda")) {
                  try {
                    result = stm.executeQuery("SELECT max(ven_codigo) FROM venda");
                    result.next();

                    i     = result.getInt(1) + 1;
                    inicio = i;
                    fim    = inicio + (QTD - 1);

                    hora_inicio = System.currentTimeMillis();
                    for (i=inicio;i<=fim;i++){
                      stm.executeUpdate(
                        "INSERT INTO venda("+
                          "ven_codigo, "+
                          "ven_total, "+
                          "cli_codigo"+
                        ")VALUES("+
```

```
                        i+","+
                        "100,"+
                        "1"+
                      ")"
                    );
                  }
                  hora_fim = System.currentTimeMillis();
                  t = hora_fim - hora_inicio;

                  try{
                    PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                    out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                    out1.close();
                  } catch (Exception e) {
                    System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                  }
                  System.out.println("concluido");
                } catch (Exception e){
                  System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
                }
              }

              else if (TABELA.equals("venda_item")) {
                try {
                  result = stm.executeQuery("SELECT max(ven_codigo) FROM venda_item");
                  result.next();

                  i     = result.getInt(1) + 1;
                  inicio = i;
                  fim    = inicio + (QTD - 1);

                  hora_inicio = System.currentTimeMillis();
                  for (i=inicio;i<=fim;i++){
                    stm.executeUpdate(
                      "INSERT INTO venda_item("+
                        "pro_codigo,"+
                        "ven_codigo,"+
                        "vit_item,"+
                        "vit_qtde"+
                      ")VALUES("+
                        "1,"+
                        "1,"+
                        i+","+
                        "1"+
                      ")"
                    );
                  }
                  hora_fim = System.currentTimeMillis();
                  t = hora_fim - hora_inicio;

                  try{
                    PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
```

```
                    out1.write(QTD +" Registros INSERIDOS em: "+t+"ms\r\n\r\n");
                    out1.close();
                  } catch (Exception e) {
                    System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                  }
                  System.out.println("concluido");
                } catch (Exception e){
                  System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
                }
              }

              else {
                System.out.println("\r\nERRO: Tabela "+ TABELA +" NAO ENCONTRADA!");
                System.exit(0);
              }
            }

            //EXECUTA TESTE DE RECUPERAÇÃO EM TABELAS SOLITÁRIAS
            if (OPERACAO.equals("recuperacao_solitarias")) {
              if (TABELA.equals("categoria") ||
                  TABELA.equals("cor") ||
                  TABELA.equals("origem") ||
                  TABELA.equals("fabricante") ||
                  TABELA.equals("cliente"))
              {
                String sigla = TABELA.substring(0, 3);

                try {
                  hora_inicio = System.currentTimeMillis();
                  result = stm.executeQuery("SELECT * FROM "+TABELA+" WHERE
"+sigla+"_codigo < "+(QTD+1));
                  hora_fim = System.currentTimeMillis();
                  t = hora_fim - hora_inicio;

                  i=0;
                  while (result.next()){
                    i++;
                  }

                  try{
                    PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                    out1.write(i +" Registros RECUPERADOS em: "+t+"ms\r\n\r\n");
                    out1.close();
                  } catch (Exception e) {
                    System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                  }
                  System.out.println("concluido");
                } catch (Exception e){
                  System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
                }
              }

              else {
```

```
            System.out.println("\r\nERRO: Tabela "+ TABELA +" NAO ENCONTRADA!");
            System.exit(0);
        }
    }

    //EXECUTA TESTE DE RECUPERAÇÃO EM TABELAS RELACIONADAS
    if (OPERACAO.equals("recuperacao_relacionadas")) {
        if (TABELA.equals("pessoa_fisica") ||
            TABELA.equals("pessoa_juridica") ||
            TABELA.equals("produto") ||
            TABELA.equals("tabela") ||
            TABELA.equals("tabela_item") ||
            TABELA.equals("venda") ||
            TABELA.equals("venda_item"))
        {
            try {
                t = 0;

                //1 RELACIONAMENTO
                if (TABELA.equals("pessoa_fisica")) {
                    hora_inicio = System.currentTimeMillis();
                    result = stm.executeQuery(
                        "SELECT * FROM pessoa_fisica " +
                        "JOIN cliente ON pessoa_fisica.cli_codigo = cliente.cli_codigo " +
                        "WHERE pessoa_fisica.cli_codigo < "+(QTD+1)
                    );
                    hora_fim = System.currentTimeMillis();
                    t = hora_fim - hora_inicio;
                }
                if (TABELA.equals("pessoa_juridica")) {
                    hora_inicio = System.currentTimeMillis();
                    result = stm.executeQuery(
                        "SELECT * FROM pessoa_juridica " +
                        "JOIN cliente ON pessoa_juridica.cli_codigo = cliente.cli_codigo " +
                        "WHERE pessoa_juridica.cli_codigo < "+(QTD+1)
                    );
                    hora_fim = System.currentTimeMillis();
                    t = hora_fim - hora_inicio;
                }
                if (TABELA.equals("tabela")) {
                    hora_inicio = System.currentTimeMillis();
                    result = stm.executeQuery(
                        "SELECT * FROM tabela " +
                        "JOIN fabricante ON tabela.fab_codigo = fabricante.fab_codigo " +
                        "WHERE tabela.tab_codigo < "+(QTD+1)
                    );
                    hora_fim = System.currentTimeMillis();
                    t = hora_fim - hora_inicio;
                }
                if (TABELA.equals("venda")) {
                    hora_inicio = System.currentTimeMillis();
                    result = stm.executeQuery(
                        "SELECT * FROM venda " +
                        "JOIN cliente ON venda.cli_codigo = cliente.cli_codigo " +
```

```
                    "WHERE venda.ven_codigo < "+(QTD+1)
                  );
                  hora_fim = System.currentTimeMillis();
                  t = hora_fim - hora_inicio;
                }

                //2 RELACIONAMENTOS
                if (TABELA.equals("venda_item")) {
                  hora_inicio = System.currentTimeMillis();
                  result = stm.executeQuery(
                    "SELECT * FROM venda_item " +
                    "JOIN venda ON venda_item.ven_codigo = venda.ven_codigo " +
                    "JOIN produto ON venda_item.pro_codigo = produto.pro_codigo " +
                    "WHERE venda_item.vit_item < "+(QTD+1)
                  );
                  hora_fim = System.currentTimeMillis();
                  t = hora_fim - hora_inicio;
                }

                //4 RELACIONAMENTOS
                if (TABELA.equals("produto")) {
                  hora_inicio = System.currentTimeMillis();
                  result = stm.executeQuery(
                    "SELECT * FROM produto " +
                    "JOIN categoria ON produto.cat_codigo = categoria.cat_codigo " +
                    "JOIN cor ON produto.cor_codigo = cor.cor_codigo " +
                    "JOIN fabricante ON produto.fab_codigo = fabricante.fab_codigo " +
                    "JOIN origem ON produto.ori_codigo = origem.ori_codigo " +
                    "WHERE produto.pro_codigo < "+(QTD+1)
                  );
                  hora_fim = System.currentTimeMillis();
                  t = hora_fim - hora_inicio;
                }

                i=0;
                while (result.next()){
                  i++;
                }

                try{
                  PrintWriter out1 = new PrintWriter(new BufferedWriter(new FileWriter("log
"+BANCO+"-"+TABELA+".txt",true)));
                  out1.write(i +" Registros RECUPERADOS em: "+t+"ms\r\n\r\n");
                  out1.close();
                } catch (Exception e) {
                  System.out.println("Ocorreu o seguinte erro1: " + e.getMessage());
                }
                System.out.println("concluido");
              } catch (Exception e){
                System.out.println("Ocorreu o seguinte erro2: " + e.getMessage());
              }
            }

            else {
```

```
                    System.out.println("\r\nERRO: Tabela "+ TABELA +" NAO ENCONTRADA!");
                    System.exit(0);
                }
            }
        } catch (Exception e) {
            System.out.println("Ocorreu o seguinte erro3: " + e.getMessage());
        } finally {
            try {
                if (BANCO.equals("H2") || BANCO.equals("HSQLDB"))
conn.createStatement().execute("shutdown");
                conn.close();
            } catch (Exception onConClose) {
                onConClose.printStackTrace();
                System.out.println("ERRO AO ENCERRAR A CONEXAO");
            }
        }
    }
  }
}
```

APÊNDICE B – Código fonte da aplicação implementada em *Java* para realização dos testes de integridade

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import javax.swing.JOptionPane;


public class JavaDB extends javax.swing.JFrame {
  //Declaração de variáveis
  private javax.swing.JSeparator  jSeparator1;
  private javax.swing.JButton     jButtonExecutar, jButtonCriarTabelas,
jButtonDroparTabelas;
  private javax.swing.JLabel      jLabelSGBD,jLabelSQL;
  private javax.swing.JComboBox   jComboBoxSGBD;
  private javax.swing.JPanel      jPanel1;
  private javax.swing.JScrollPane jScrollPaneSQL, jScrollPaneResultado;
  private javax.swing.JTextArea   jTextAreaSQL,jTextAreaResultado;

  Connection conn = null;
  Statement  stm;
  ResultSet  result = null;
  long       hora_inicio;
  long       hora_fim;
  String     MSG_ERRO = "";
  String     SQL;
  int        QTD, i, inicio;

  public JavaDB() {
    super("Conexão com banco de dados APIs Java");
    criaInterface();
  }

  private void criaInterface() {
    jComboBoxSGBD = new javax.swing.JComboBox();
    jLabelSGBD = new javax.swing.JLabel();
    jButtonExecutar = new javax.swing.JButton();
    jPanel1 = new javax.swing.JPanel();
    jScrollPaneResultado = new javax.swing.JScrollPane();
    jTextAreaResultado = new javax.swing.JTextArea();
    jLabelSQL = new javax.swing.JLabel();
    jScrollPaneSQL = new javax.swing.JScrollPane();
    jTextAreaSQL = new javax.swing.JTextArea();
    jSeparator1 = new javax.swing.JSeparator();
```

```
    jButtonCriarTabelas = new javax.swing.JButton();
    jButtonDroparTabelas = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    jComboBoxSGBD.setModel(new javax.swing.DefaultComboBoxModel(new String[] {
"Apache Derby", "H2", "HSQLDB", "McKoi", "One$DB" }));

    jLabelSGBD.setText("SGBD");

    jButtonExecutar.setText("Executar");
    jButtonExecutar.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonExecutarActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(
      jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGap(0, 0, Short.MAX_VALUE)
    );
    jPanel1Layout.setVerticalGroup(
      jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGap(0, 0, Short.MAX_VALUE)
    );

    jTextAreaResultado.setColumns(20);
    jTextAreaResultado.setEditable(false);
    jTextAreaResultado.setRows(5);
    jScrollPaneResultado.setViewportView(jTextAreaResultado);

    jLabelSQL.setText("SQL Livre:");

    jTextAreaSQL.setColumns(20);
    jTextAreaSQL.setRows(4);
    jScrollPaneSQL.setViewportView(jTextAreaSQL);

    jButtonCriarTabelas.setText("Criar Tabelas");
    jButtonCriarTabelas.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonCriarTabelasActionPerformed(evt);
      }
    });

    jButtonDroparTabelas.setText("Dropar Tabelas");
    jButtonDroparTabelas.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonDroparTabelasActionPerformed(evt);
      }
```

```
    });

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
              .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jSeparator1, javax.swing.GroupLayout.DEFAULT_SIZE, 554,
Short.MAX_VALUE)
                .addGroup(layout.createSequentialGroup()
                  .addComponent(jLabelSGBD)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                  .addComponent(jComboBoxSGBD,
javax.swing.GroupLayout.PREFERRED_SIZE, 232,
javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
64, Short.MAX_VALUE)
                  .addComponent(jButtonCriarTabelas,
javax.swing.GroupLayout.PREFERRED_SIZE, 117,
javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                  .addComponent(jButtonDroparTabelas))
                .addGroup(layout.createSequentialGroup()
                  .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                  .addComponent(jScrollPaneResultado,
javax.swing.GroupLayout.DEFAULT_SIZE, 548, Short.MAX_VALUE))))
            .addGroup(layout.createSequentialGroup()
              .addContainerGap()
              .addComponent(jLabelSQL))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
              .addGap(16, 16, 16)
              .addComponent(jScrollPaneSQL, javax.swing.GroupLayout.DEFAULT_SIZE,
548, Short.MAX_VALUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
              .addGap(16, 16, 16)
              .addComponent(jButtonExecutar, javax.swing.GroupLayout.DEFAULT_SIZE,
548, Short.MAX_VALUE)))
          .addContainerGap())
    );
    layout.setVerticalGroup(
```

```
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
          .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(jLabelSGBD)
            .addComponent(jComboBoxSGBD, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(jButtonDroparTabelas)
            .addComponent(jButtonCriarTabelas))
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 10,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(jLabelSQL)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(jScrollPaneSQL, javax.swing.GroupLayout.PREFERRED_SIZE,
104, javax.swing.GroupLayout.PREFERRED_SIZE)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(jButtonExecutar)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
              .addGap(90, 90, 90)
              .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGroup(layout.createSequentialGroup()
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
              .addComponent(jScrollPaneResultado,
javax.swing.GroupLayout.DEFAULT_SIZE, 235, Short.MAX_VALUE)
              .addContainerGap())))
    );
    pack();
    setLocationRelativeTo( null ); //cloca frame no meio da tela
  }

  private void jButtonExecutarActionPerformed(java.awt.event.ActionEvent evt) {
    //SE TIVER ALGUM SQL LIVRE DIGITADO EM jComboBoxSQL EXECUTA
    try {
      jTextAreaResultado.setText("Conectando ao banco de dados
"+jComboBoxSGBD.getSelectedItem().toString()+"...\n");

      //conecta ao banco de dados selecionado
      switch (jComboBoxSGBD.getSelectedIndex()) {
        //derby
        case 0:   Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
            conn =
DriverManager.getConnection("jdbc:derby:.\\DATABASE\\DERBY\\bdteste;create=true");
            break;
```

```
            //h2
            case 1: Class.forName("org.h2.Driver");
                conn =
DriverManager.getConnection("jdbc:h2:.\\DATABASE\\H2\\bdteste;create=true");
                break;
            //hsqldb
            case 2: Class.forName("org.hsqldb.jdbcDriver");
                conn =
DriverManager.getConnection("jdbc:hsqldb:.\\DATABASE\\HSQLDB\\bdteste;create=true")
;
                break;
            //mckoi
            case 3: Class.forName("com.mckoi.JDBCDriver");
                conn =
DriverManager.getConnection("jdbc:mckoi:local://./DATABASE/MCKOI/bdteste.conf?creat
e_or_boot=true","mckoi","mckoi");
                break;
            //one$db
            case 4: Class.forName("in.co.daffodil.db.jdbc.DaffodilDBDriver");
                conn =
DriverManager.getConnection("jdbc:daffodilDB_embedded:bdteste;create=true;path=.//DAT
ABASE//ONEDB");
                break;
        }

        jTextAreaResultado.setText(jTextAreaResultado.getText() + "Conectado com
Sucesso!");

        //inicializa o statement com a conexao e  seta autocommit
        stm = conn.createStatement();
        conn.setAutoCommit(true);

        //SQL LIVRE
        if (!jTextAreaSQL.getText().equals("")) {
            jTextAreaResultado.setText(jTextAreaResultado.getText() +"\n\n+Executando em
modo SQL Livre");
            jTextAreaResultado.setText(jTextAreaResultado.getText() +"\n\n"+
jTextAreaSQL.getText()+"\n");

            //pega hora antes de executar o SQL
            hora_inicio = System.currentTimeMillis();

            //se não for SELECT, executa "executeUpdate" senão "executeQuery"
            if (jTextAreaResultado.getText().toUpperCase().indexOf("SELECT") == -1) {
              stm.executeUpdate(jTextAreaSQL.getText());
              hora_fim = System.currentTimeMillis();
            } else {
              result = stm.executeQuery(jTextAreaSQL.getText());
              hora_fim = System.currentTimeMillis();
```

```
            String resultado = new String();
            ResultSetMetaData rsmd = result.getMetaData();
            int colunas = rsmd.getColumnCount();
            jTextAreaResultado.setText(jTextAreaResultado.getText() +
Integer.toString(colunas)+" Colunas\n");
            while (result.next()) {
              resultado = resultado +"\n"+ result.getString(1);
              for (int i=2;i<=colunas;i++){
                resultado = resultado+" | "+result.getString(i);
              }
            }
            jTextAreaResultado.setText(jTextAreaResultado.getText()+resultado);
          }

          long t = hora_fim - hora_inicio;
          jTextAreaResultado.setText(jTextAreaResultado.getText() + "\n\nOperação
concluída!" + "\nTempo = " + t + " milisegundos");
        }
      } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Ocorreu o seguinte erro: " + e.getMessage());
      } finally {
        try {
          conn.close();
          jTextAreaResultado.setText(jTextAreaResultado.getText() + "\n\nDesconectado!");
        } catch (Exception onConClose) {
          onConClose.printStackTrace();
        }
      }
    }

    private void jButtonCriarTabelasActionPerformed(java.awt.event.ActionEvent evt) {
      jTextAreaResultado.setText("Criando base de dados do
"+jComboBoxSGBD.getSelectedItem().toString()+"...\n");

      //conecta ao banco de dados selecionado
      switch (jComboBoxSGBD.getSelectedIndex()) {
        //derby
        case 0: try {
              Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
              conn =
DriverManager.getConnection("jdbc:derby:.\\DATABASE\\DERBY\\bdteste;create=true");

              //inicializa o statement com a conexao e  seta autocommit
              stm = conn.createStatement();
              conn.setAutoCommit(true);
              SQL = "CREATE TABLE fabricante ("+
                  "fab_codigo       INTEGER    NOT NULL PRIMARY KEY   ,"+
                  "fab_cep          VARCHAR(8)                        ,"+
                  "fab_nome         VARCHAR(30) NOT NULL              ,"+
                  "fab_cidade       VARCHAR(30) NOT NULL              ,"+
```

```
                "fab_estado        CHAR(2)     NOT NULL             ,"+
                "fab_endereco      VARCHAR(30)                      ,"+
                "fab_bairro        VARCHAR(30)                      ,"+
                "fab_complemento   VARCHAR(20)                      ,"+
                "fab_numero        VARCHAR(10)                      ,"+
                "fab_telefone      VARCHAR(20)                      ,"+
                "fab_fax           VARCHAR(20)                      ,"+
                "fab_cnpj          VARCHAR(14)                      ,"+
                "fab_ie            VARCHAR(12)                      ,"+
                "fab_site          VARCHAR(50)                      ,"+
                "fab_email_gerente VARCHAR(30)                      ,"+
                "fab_razao_social  VARCHAR(30)                      ,"+
                "fab_contato       VARCHAR(30)                      ,"+
                "fab_contato2      VARCHAR(30)                      ,"+
                "fab_observacao    VARCHAR(255)                     ,"+
                "fab_data_cadastro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,"+
                "fab_ativo         CHAR(1)                          "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE categoria ("+
                "cat_codigo    INTEGER    NOT NULL PRIMARY KEY,"+
                "cat_descricao VARCHAR(20) NOT NULL          ,"+
                "cat_sigla     CHAR(10)   NOT NULL           "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE cor ("+
                "cor_codigo    INTEGER    NOT NULL PRIMARY KEY,"+
                "cor_descricao VARCHAR(20) NOT NULL          ,"+
                "cor_sigla     CHAR(10)   NOT NULL           "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE origem ("+
                "ori_codigo    INTEGER    NOT NULL PRIMARY KEY,"+
                "ori_descricao VARCHAR(20) NOT NULL          ,"+
                "ori_sigla     CHAR(10)   NOT NULL           "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE produto ("+
                "pro_codigo    INTEGER    NOT NULL PRIMARY KEY
,"+
                "pro_descricao VARCHAR(50) NOT NULL
,"+
                "pro_referencia VARCHAR(20) NOT NULL
,"+
                "pro_ativo     CHAR(1)                                  ,"+
                "fab_codigo    INTEGER                                    ,"+
```

```
                "cat_codigo     INTEGER                                         ,"+
                "cor_codigo     INTEGER                                         ,"+
                "ori_codigo     INTEGER                                         ,"+
                "CONSTRAINT produto_fab_codigo_FK FOREIGN KEY (fab_codigo)
references fabricante(fab_codigo),"+
                "CONSTRAINT produto_cat_codigo_FK FOREIGN KEY (cat_codigo)
references categoria(cat_codigo) ,"+
                "CONSTRAINT produto_cor_codigo_FK FOREIGN KEY (cor_codigo)
references cor(cor_codigo)      ,"+
                "CONSTRAINT produto_ori_codigo_FK FOREIGN KEY (ori_codigo)
references origem(ori_codigo)     "+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE tabela ("+
                "tab_codigo     INTEGER     NOT NULL PRIMARY KEY
,"+
                "tab_descricao  VARCHAR(30) NOT NULL
,"+
                "fab_codigo     INTEGER                                         ,"+
                "CONSTRAINT tabela_fab_codigo_FK FOREIGN KEY (fab_codigo)
references fabricante(fab_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE tabela_item ("+
                "tab_codigo INTEGER                                             ,"+
                "pro_codigo INTEGER                                             ,"+
                "tit_preco  DOUBLE                                             ,"+
                "CONSTRAINT tabela_item_tab_codigo_FK FOREIGN KEY (tab_codigo)
references tabela(tab_codigo),"+
                "CONSTRAINT tabela_item_pro_codigo_FK FOREIGN KEY (pro_codigo)
references produto(pro_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE cliente ("+
                "cli_codigo         INTEGER       NOT NULL PRIMARY KEY,"+
                "cli_estado         CHAR(2)       NOT NULL          ,"+
                "cli_cidade         VARCHAR(30)   NOT NULL          ,"+
                "cli_bairro         VARCHAR(30)                     ,"+
                "cli_numero         VARCHAR(10)                     ,"+
                "cli_endereco       VARCHAR(30)                     ,"+
                "cli_telefone       VARCHAR(20)                     ,"+
                "cli_tipo_cliente   CHAR(1)       NOT NULL          ,"+
                "cli_nome           VARCHAR(50)   NOT NULL          ,"+
                "cli_data_nascimento DATE         NOT NULL          "+
                ")";
            stm.executeUpdate(SQL);
```

```
            SQL = "CREATE TABLE pessoa_fisica ("+
                "cli_codigo INTEGER      NOT NULL PRIMARY KEY,"+
                "cli_cpf    CHAR(14)     NOT NULL          ,"+
                "CONSTRAINT pessoa_fisica_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE pessoa_juridica ("+
                "cli_codigo INTEGER  NOT NULL PRIMARY KEY,"+
                "cli_cnpj   CHAR(14) NOT NULL          ,"+
                "CONSTRAINT pessoa_juridica_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE venda ("+
                "ven_codigo INTEGER NOT NULL PRIMARY KEY
,"+
                "ven_total  DOUBLE                                                ,"+
                "cli_codigo INTEGER                                               ,"+
                "CONSTRAINT venda_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE venda_item ("+
                "pro_codigo INTEGER                                                ,"+
                "ven_codigo INTEGER                                               ,"+
                "vit_item   INTEGER                                              ,"+
                "vit_qtde   INTEGER                                              ,"+
                "CONSTRAINT venda_item_pro_codigo_FK FOREIGN KEY (pro_codigo)
references produto(pro_codigo),"+
                "CONSTRAINT venda_item_ven_codigo_FK FOREIGN KEY (ven_codigo)
references venda(ven_codigo)   "+
                ")";
            stm.executeUpdate(SQL);


            jTextAreaResultado.setText(jTextAreaResultado.getText() + "Banco de Dados
criado com sucesso!\n");
            break;
          } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Ocorreu o seguinte erro: " +
e.getMessage());
          } finally {
            try {
              conn.close();
              jTextAreaResultado.setText(jTextAreaResultado.getText() +
"\n\nDesconectado!");
```

```
            } catch (Exception onConClose) {
              onConClose.printStackTrace();
            }
          }

      //h2
      case 1: try {
            Class.forName("org.h2.Driver");
            conn =
DriverManager.getConnection("jdbc:h2:.\\DATABASE\\H2\\bdteste;create=true");

            //inicializa o statement com a conexao e  seta autocommit
            stm = conn.createStatement();
            conn.setAutoCommit(true);
            SQL = "CREATE TABLE fabricante ("+
                "fab_codigo        INTEGER     NOT NULL PRIMARY KEY   ,"+
                "fab_cep           VARCHAR(8)                         ,"+
                "fab_nome          VARCHAR(30) NOT NULL               ,"+
                "fab_cidade        VARCHAR(30) NOT NULL               ,"+
                "fab_estado        CHAR(2)     NOT NULL               ,"+
                "fab_endereco      VARCHAR(30)                        ,"+
                "fab_bairro        VARCHAR(30)                        ,"+
                "fab_complemento   VARCHAR(20)                        ,"+
                "fab_numero        VARCHAR(10)                        ,"+
                "fab_telefone      VARCHAR(20)                        ,"+
                "fab_fax           VARCHAR(20)                        ,"+
                "fab_cnpj          VARCHAR(14)                        ,"+
                "fab_ie            VARCHAR(12)                        ,"+
                "fab_site          VARCHAR(50)                        ,"+
                "fab_email_gerente VARCHAR(30)                        ,"+
                "fab_razao_social  VARCHAR(30)                        ,"+
                "fab_contato       VARCHAR(30)                        ,"+
                "fab_contato2      VARCHAR(30)                        ,"+
                "fab_observacao    VARCHAR(255)                       ,"+
                "fab_data_cadastro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,"+
                "fab_ativo         CHAR(1)                            "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE categoria ("+
                "cat_codigo    INTEGER    NOT NULL PRIMARY KEY,"+
                "cat_descricao VARCHAR(20) NOT NULL           ,"+
                "cat_sigla     CHAR(10)   NOT NULL            "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE cor ("+
                "cor_codigo    INTEGER    NOT NULL PRIMARY KEY,"+
                "cor_descricao VARCHAR(20) NOT NULL           ,"+
                "cor_sigla     CHAR(10)   NOT NULL            "+
```

```
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE origem ("+
                "ori_codigo     INTEGER     NOT NULL PRIMARY KEY,"+
                "ori_descricao  VARCHAR(20) NOT NULL          ,"+
                "ori_sigla      CHAR(10)    NOT NULL          "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE produto ("+
                "pro_codigo     INTEGER     NOT NULL PRIMARY KEY
,"+
                "pro_descricao  VARCHAR(50) NOT NULL
,"+
                "pro_referencia VARCHAR(20) NOT NULL
,"+
                "pro_ativo      CHAR(1)                                    ,"+
                "fab_codigo     INTEGER                                     ,"+
                "cat_codigo     INTEGER                                     ,"+
                "cor_codigo     INTEGER                                     ,"+
                "ori_codigo     INTEGER                                     ,"+
                "CONSTRAINT produto_fab_codigo_FK FOREIGN KEY (fab_codigo)
references fabricante(fab_codigo),"+
                "CONSTRAINT produto_cat_codigo_FK FOREIGN KEY (cat_codigo)
references categoria(cat_codigo) ,"+
                "CONSTRAINT produto_cor_codigo_FK FOREIGN KEY (cor_codigo)
references cor(cor_codigo)       ,"+
                "CONSTRAINT produto_ori_codigo_FK FOREIGN KEY (ori_codigo)
references origem(ori_codigo)     "+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE tabela ("+
                "tab_codigo     INTEGER     NOT NULL PRIMARY KEY
,"+
                "tab_descricao  VARCHAR(30) NOT NULL
,"+
                "fab_codigo     INTEGER                                     ,"+
                "CONSTRAINT tabela_fab_codigo_FK FOREIGN KEY (fab_codigo)
references fabricante(fab_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE tabela_item ("+
                "tab_codigo INTEGER                                      ,"+
                "pro_codigo INTEGER                                      ,"+
                "tit_preco  DOUBLE                                      ,"+
                "CONSTRAINT tabela_item_tab_codigo_FK FOREIGN KEY (tab_codigo)
references tabela(tab_codigo),"+
```

```
                "CONSTRAINT tabela_item_pro_codigo_FK FOREIGN KEY (pro_codigo)
references produto(pro_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE cliente ("+
                "cli_codigo         INTEGER     NOT NULL PRIMARY KEY,"+
                "cli_estado         CHAR(2)     NOT NULL          ,"+
                "cli_cidade         VARCHAR(30) NOT NULL          ,"+
                "cli_bairro         VARCHAR(30)                   ,"+
                "cli_numero          VARCHAR(10)                  ,"+
                "cli_endereco        VARCHAR(30)                  ,"+
                "cli_telefone        VARCHAR(20)                  ,"+
                "cli_tipo_cliente    CHAR(1)     NOT NULL         ,"+
                "cli_nome           VARCHAR(50) NOT NULL          ,"+
                "cli_data_nascimento DATE        NOT NULL          "+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE pessoa_fisica ("+
                "cli_codigo INTEGER       NOT NULL PRIMARY KEY,"+
                "cli_cpf    CHAR(14)      NOT NULL          ,"+
                "CONSTRAINT pessoa_fisica_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE pessoa_juridica ("+
                "cli_codigo INTEGER  NOT NULL PRIMARY KEY,"+
                "cli_cnpj   CHAR(14) NOT NULL          ,"+
                "CONSTRAINT pessoa_juridica_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE venda ("+
                "ven_codigo INTEGER NOT NULL PRIMARY KEY
,"+
                "ven_total  DOUBLE                                          ,"+
                "ven_data   DATE DEFAULT CURRENT_DATE
,"+
                "cli_codigo INTEGER                                         ,"+
                "CONSTRAINT venda_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE venda_item ("+
                "pro_codigo INTEGER                                       ,"+
                "ven_codigo INTEGER                                       ,"+
```

```
                "vit_item   INTEGER                                                 ,"+
                "vit_qtde   INTEGER                                                 ,"+
                "CONSTRAINT venda_item_pro_codigo_FK FOREIGN KEY (pro_codigo)
references produto(pro_codigo),"+
                "CONSTRAINT venda_item_ven_codigo_FK FOREIGN KEY (ven_codigo)
references venda(ven_codigo)   "+
                ")";
            stm.executeUpdate(SQL);

            jTextAreaResultado.setText(jTextAreaResultado.getText() + "Banco de Dados
criado com sucesso!\n");
            break;
          } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Ocorreu o seguinte erro: " +
e.getMessage());
          } finally {
            try {
              conn.close();
              jTextAreaResultado.setText(jTextAreaResultado.getText() +
"\n\nDesconectado!");
            } catch (Exception onConClose) {
              onConClose.printStackTrace();
            }
          }
      //hsqldb
      case 2: try {
            Class.forName("org.hsqldb.jdbcDriver");
            conn =
DriverManager.getConnection("jdbc:hsqldb:.\\DATABASE\\HSQLDB\\bdteste;create=true")
;

            //inicializa o statement com a conexao e  seta autocommit
            stm = conn.createStatement();
            conn.setAutoCommit(true);
            SQL = "CREATE TABLE fabricante ("+
                "fab_codigo       INTEGER    NOT NULL PRIMARY KEY   ,"+
                "fab_cep          VARCHAR(8)                        ,"+
                "fab_nome         VARCHAR(30) NOT NULL              ,"+
                "fab_cidade       VARCHAR(30) NOT NULL              ,"+
                "fab_estado       CHAR(2)    NOT NULL               ,"+
                "fab_endereco     VARCHAR(30)                       ,"+
                "fab_bairro       VARCHAR(30)                       ,"+
                "fab_complemento  VARCHAR(20)                       ,"+
                "fab_numero       VARCHAR(10)                       ,"+
                "fab_telefone     VARCHAR(20)                       ,"+
                "fab_fax          VARCHAR(20)                       ,"+
                "fab_cnpj         VARCHAR(14)                       ,"+
                "fab_ie           VARCHAR(12)                       ,"+
                "fab_site         VARCHAR(50)                       ,"+
                "fab_email_gerente  VARCHAR(30)                     ,"+
```

```
                "fab_razao_social   VARCHAR(30)                    ,"+
                "fab_contato        VARCHAR(30)                   ,"+
                "fab_contato2       VARCHAR(30)                    ,"+
                "fab_observacao     VARCHAR(255)                    ,"+
                "fab_data_cadastro  TIMESTAMP DEFAULT CURRENT_TIMESTAMP,"+
                "fab_ativo          CHAR(1)                      "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE categoria ("+
                "cat_codigo     INTEGER     NOT NULL PRIMARY KEY,"+
                "cat_descricao  VARCHAR(20) NOT NULL           ,"+
                "cat_sigla      CHAR(10)    NOT NULL           "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE cor ("+
                "cor_codigo     INTEGER     NOT NULL PRIMARY KEY,"+
                "cor_descricao  VARCHAR(20) NOT NULL           ,"+
                "cor_sigla      CHAR(10)    NOT NULL           "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE origem ("+
                "ori_codigo     INTEGER     NOT NULL PRIMARY KEY,"+
                "ori_descricao  VARCHAR(20) NOT NULL           ,"+
                "ori_sigla      CHAR(10)    NOT NULL           "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE produto ("+
                "pro_codigo     INTEGER     NOT NULL PRIMARY KEY
,"+
                "pro_descricao  VARCHAR(50) NOT NULL
,"+
                "pro_referencia VARCHAR(20) NOT NULL
,"+
                "pro_ativo      CHAR(1)                                    ,"+
                "fab_codigo     INTEGER                                     ,"+
                "cat_codigo     INTEGER                                     ,"+
                "cor_codigo     INTEGER                                     ,"+
                "ori_codigo     INTEGER                                     ,"+
                "CONSTRAINT produto_fab_codigo_FK FOREIGN KEY (fab_codigo)
references fabricante(fab_codigo),"+
                "CONSTRAINT produto_cat_codigo_FK FOREIGN KEY (cat_codigo)
references categoria(cat_codigo) ,"+
                "CONSTRAINT produto_cor_codigo_FK FOREIGN KEY (cor_codigo)
references cor(cor_codigo)       ,"+
                "CONSTRAINT produto_ori_codigo_FK FOREIGN KEY (ori_codigo)
references origem(ori_codigo)     "+
```

```
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE tabela ("+
                "tab_codigo     INTEGER     NOT NULL PRIMARY KEY
,"+
                "tab_descricao  VARCHAR(30) NOT NULL
,"+
                "fab_codigo     INTEGER                                                 ,"+
                "CONSTRAINT tabela_fab_codigo_FK FOREIGN KEY (fab_codigo)
references fabricante(fab_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE tabela_item ("+
                "tab_codigo INTEGER                                                 ,"+
                "pro_codigo INTEGER                                                 ,"+
                "tit_preco  DOUBLE                                                 ,"+
                "CONSTRAINT tabela_item_tab_codigo_FK FOREIGN KEY (tab_codigo)
references tabela(tab_codigo),"+
                "CONSTRAINT tabela_item_pro_codigo_FK FOREIGN KEY (pro_codigo)
references produto(pro_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE cliente ("+
                "cli_codigo         INTEGER     NOT NULL PRIMARY KEY,"+
                "cli_estado         CHAR(2)     NOT NULL          ,"+
                "cli_cidade         VARCHAR(30) NOT NULL          ,"+
                "cli_bairro         VARCHAR(30)                   ,"+
                "cli_numero         VARCHAR(10)                   ,"+
                "cli_endereco       VARCHAR(30)                   ,"+
                "cli_telefone       VARCHAR(20)                   ,"+
                "cli_tipo_cliente   CHAR(1)     NOT NULL          ,"+
                "cli_nome           VARCHAR(50) NOT NULL          ,"+
                "cli_data_nascimento DATE       NOT NULL          "+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE pessoa_fisica ("+
                "cli_codigo INTEGER      NOT NULL PRIMARY KEY,"+
                "cli_cpf    CHAR(14)     NOT NULL          ,"+
                "CONSTRAINT pessoa_fisica_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE pessoa_juridica ("+
                "cli_codigo INTEGER  NOT NULL PRIMARY KEY,"+
                "cli_cnpj   CHAR(14) NOT NULL          ,"+
```

```
                "CONSTRAINT pessoa_juridica_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE venda ("+
                "ven_codigo INTEGER NOT NULL PRIMARY KEY
,"+
                "ven_total  DOUBLE                                                  ,"+
                "ven_data   DATE DEFAULT CURRENT_DATE
,"+
                "cli_codigo INTEGER                                                 ,"+
                "CONSTRAINT venda_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE venda_item ("+
                "pro_codigo INTEGER                                                 ,"+
                "ven_codigo INTEGER                                                 ,"+
                "vit_item   INTEGER                                                ,"+
                "vit_qtde   INTEGER                                                ,"+
                "CONSTRAINT venda_item_pro_codigo_FK FOREIGN KEY (pro_codigo)
references produto(pro_codigo),"+
                "CONSTRAINT venda_item_ven_codigo_FK FOREIGN KEY (ven_codigo)
references venda(ven_codigo)   "+
                ")";
            stm.executeUpdate(SQL);


            jTextAreaResultado.setText(jTextAreaResultado.getText() + "Banco de Dados
criado com sucesso!\n");
            break;
          } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Ocorreu o seguinte erro: " +
e.getMessage());
          } finally {
            try {
              conn.close();
              jTextAreaResultado.setText(jTextAreaResultado.getText() +
"\n\nDesconectado!");
            } catch (Exception onConClose) {
              onConClose.printStackTrace();
            }
          }
      //mckoi
      case 3: try {
            Class.forName("com.mckoi.JDBCDriver");
            conn =
DriverManager.getConnection("jdbc:mckoi:local://./DATABASE/MCKOI/bdteste.conf?creat
```

```
e_or_boot=true","mckoi","mckoi");

            //inicializa o statement com a conexao e  seta autocommit
            stm = conn.createStatement();
            conn.setAutoCommit(true);
            SQL = "CREATE TABLE fabricante ("+
                "fab_codigo        INTEGER     NOT NULL PRIMARY KEY   ,"+
                "fab_cep           VARCHAR(8)                         ,"+
                "fab_nome          VARCHAR(30) NOT NULL               ,"+
                "fab_cidade        VARCHAR(30) NOT NULL               ,"+
                "fab_estado        CHAR(2)     NOT NULL               ,"+
                "fab_endereco      VARCHAR(30)                        ,"+
                "fab_bairro        VARCHAR(30)                        ,"+
                "fab_complemento   VARCHAR(20)                        ,"+
                "fab_numero        VARCHAR(10)                        ,"+
                "fab_telefone      VARCHAR(20)                        ,"+
                "fab_fax           VARCHAR(20)                        ,"+
                "fab_cnpj          VARCHAR(14)                        ,"+
                "fab_ie            VARCHAR(12)                        ,"+
                "fab_site          VARCHAR(50)                        ,"+
                "fab_email_gerente VARCHAR(30)                        ,"+
                "fab_razao_social  VARCHAR(30)                        ,"+
                "fab_contato       VARCHAR(30)                        ,"+
                "fab_contato2      VARCHAR(30)                        ,"+
                "fab_observacao    VARCHAR(255)                       ,"+
                "fab_data_cadastro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,"+
                "fab_ativo         CHAR(1)                            "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE categoria ("+
                "cat_codigo    INTEGER     NOT NULL PRIMARY KEY,"+
                "cat_descricao VARCHAR(20) NOT NULL             ,"+
                "cat_sigla     CHAR(10)    NOT NULL             "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE cor ("+
                "cor_codigo    INTEGER     NOT NULL PRIMARY KEY,"+
                "cor_descricao VARCHAR(20) NOT NULL             ,"+
                "cor_sigla     CHAR(10)    NOT NULL             "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE origem ("+
                "ori_codigo    INTEGER     NOT NULL PRIMARY KEY,"+
                "ori_descricao VARCHAR(20) NOT NULL             ,"+
                "ori_sigla     CHAR(10)    NOT NULL             "+
               ")";
            stm.executeUpdate(SQL);
```

```
            SQL = "CREATE TABLE produto ("+
                "pro_codigo     INTEGER     NOT NULL PRIMARY KEY
,"+
                "pro_descricao  VARCHAR(50) NOT NULL
,"+
                "pro_referencia VARCHAR(20) NOT NULL
,"+
                "pro_ativo      CHAR(1)                                          ,"+
                "fab_codigo     INTEGER                                           ,"+
                "cat_codigo     INTEGER                                           ,"+
                "cor_codigo     INTEGER                                           ,"+
                "ori_codigo     INTEGER                                           ,"+
                "CONSTRAINT produto_fab_codigo_FK FOREIGN KEY (fab_codigo)
references fabricante(fab_codigo),"+
                "CONSTRAINT produto_cat_codigo_FK FOREIGN KEY (cat_codigo)
references categoria(cat_codigo) ,"+
                "CONSTRAINT produto_cor_codigo_FK FOREIGN KEY (cor_codigo)
references cor(cor_codigo)      ,"+
                "CONSTRAINT produto_ori_codigo_FK FOREIGN KEY (ori_codigo)
references origem(ori_codigo)     "+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE tabela ("+
                "tab_codigo     INTEGER     NOT NULL PRIMARY KEY
,"+
                "tab_descricao  VARCHAR(30) NOT NULL
,"+
                "fab_codigo     INTEGER                                           ,"+
                "CONSTRAINT tabela_fab_codigo_FK FOREIGN KEY (fab_codigo)
references fabricante(fab_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE tabela_item ("+
                "tab_codigo INTEGER                                           ,"+
                "pro_codigo INTEGER                                           ,"+
                "tit_preco  DOUBLE                                           ,"+
                "CONSTRAINT tabela_item_tab_codigo_FK FOREIGN KEY (tab_codigo)
references tabela(tab_codigo),"+
                "CONSTRAINT tabela_item_pro_codigo_FK FOREIGN KEY (pro_codigo)
references produto(pro_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE cliente ("+
                "cli_codigo         INTEGER     NOT NULL PRIMARY KEY,"+
                "cli_estado         CHAR(2)     NOT NULL          ,"+
                "cli_cidade         VARCHAR(30) NOT NULL          ,"+
```

```
                "cli_bairro         VARCHAR(30)                ,"+
                "cli_numero         VARCHAR(10)                ,"+
                "cli_endereco       VARCHAR(30)                ,"+
                "cli_telefone       VARCHAR(20)                ,"+
                "cli_tipo_cliente   CHAR(1)    NOT NULL        ,"+
                "cli_nome           VARCHAR(50) NOT NULL       ,"+
                "cli_data_nascimento DATE      NOT NULL        "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE pessoa_fisica ("+
                "cli_codigo INTEGER      NOT NULL PRIMARY KEY,"+
                "cli_cpf    CHAR(14)     NOT NULL          ,"+
                "CONSTRAINT pessoa_fisica_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE pessoa_juridica ("+
                "cli_codigo INTEGER  NOT NULL PRIMARY KEY,"+
                "cli_cnpj   CHAR(14) NOT NULL          ,"+
                "CONSTRAINT pessoa_juridica_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE venda ("+
                "ven_codigo INTEGER NOT NULL PRIMARY KEY
,"+
                "ven_total  DOUBLE                                          ,"+
                "ven_data   DATE DEFAULT CURRENT_DATE
,"+
                "cli_codigo INTEGER                                         ,"+
                "CONSTRAINT venda_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE venda_item ("+
                "pro_codigo INTEGER                                          ,"+
                "ven_codigo INTEGER                                          ,"+
                "vit_item   INTEGER                                         ,"+
                "vit_qtde   INTEGER                                         ,"+
                "CONSTRAINT venda_item_pro_codigo_FK FOREIGN KEY (pro_codigo)
references produto(pro_codigo),"+
                "CONSTRAINT venda_item_ven_codigo_FK FOREIGN KEY (ven_codigo)
references venda(ven_codigo)   "+
               ")";
            stm.executeUpdate(SQL);
```

```
            jTextAreaResultado.setText(jTextAreaResultado.getText() + "Banco de Dados
criado com sucesso!\n");
            break;
          } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Ocorreu o seguinte erro: " +
e.getMessage());
          } finally {
            try {
              conn.close();
              jTextAreaResultado.setText(jTextAreaResultado.getText() +
"\n\nDesconectado!");
            } catch (Exception onConClose) {
              onConClose.printStackTrace();
            }
          }
      //one$db
      case 4: try {
            Class.forName("in.co.daffodil.db.jdbc.DaffodilDBDriver");
            conn =
DriverManager.getConnection("jdbc:daffodilDB_embedded:bdteste;create=true;path=.//DAT
ABASE//ONEDB");

            //inicializa o statement com a conexao e  seta autocommit
            stm = conn.createStatement();
            conn.setAutoCommit(true);
            SQL = "CREATE TABLE fabricante ("+
                "fab_codigo       INTEGER    NOT NULL PRIMARY KEY   ,"+
                "fab_cep          VARCHAR(8)                        ,"+
                "fab_nome         VARCHAR(30) NOT NULL              ,"+
                "fab_cidade       VARCHAR(30) NOT NULL              ,"+
                "fab_estado       CHAR(2)    NOT NULL               ,"+
                "fab_endereco     VARCHAR(30)                       ,"+
                "fab_bairro       VARCHAR(30)                       ,"+
                "fab_complemento  VARCHAR(20)                       ,"+
                "fab_numero       VARCHAR(10)                       ,"+
                "fab_telefone     VARCHAR(20)                       ,"+
                "fab_fax          VARCHAR(20)                       ,"+
                "fab_cnpj         VARCHAR(14)                       ,"+
                "fab_ie           VARCHAR(12)                       ,"+
                "fab_site         VARCHAR(50)                       ,"+
                "fab_email_gerente  VARCHAR(30)                     ,"+
                "fab_razao_social   VARCHAR(30)                     ,"+
                "fab_contato      VARCHAR(30)                       ,"+
                "fab_contato2     VARCHAR(30)                       ,"+
                "fab_observacao   VARCHAR(255)                      ,"+
                "fab_data_cadastro  TIMESTAMP DEFAULT CURRENT_TIMESTAMP,"+
                "fab_ativo        CHAR(1)                         "+
               ")";
            stm.executeUpdate(SQL);
```

```
            SQL = "CREATE TABLE categoria ("+
                "cat_codigo     INTEGER     NOT NULL PRIMARY KEY,"+
                "cat_descricao  VARCHAR(20) NOT NULL          ,"+
                "cat_sigla      CHAR(10)    NOT NULL          "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE cor ("+
                "cor_codigo     INTEGER     NOT NULL PRIMARY KEY,"+
                "cor_descricao  VARCHAR(20) NOT NULL          ,"+
                "cor_sigla      CHAR(10)    NOT NULL          "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE origem ("+
                "ori_codigo     INTEGER     NOT NULL PRIMARY KEY,"+
                "ori_descricao  VARCHAR(20) NOT NULL          ,"+
                "ori_sigla      CHAR(10)    NOT NULL          "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE produto ("+
                "pro_codigo     INTEGER     NOT NULL PRIMARY KEY
,"+
                "pro_descricao  VARCHAR(50) NOT NULL
,"+
                "pro_referencia VARCHAR(20) NOT NULL
,"+
                "pro_ativo      CHAR(1)                                    ,"+
                "fab_codigo     INTEGER                                     ,"+
                "cat_codigo     INTEGER                                     ,"+
                "cor_codigo     INTEGER                                     ,"+
                "ori_codigo     INTEGER                                     ,"+
                "CONSTRAINT produto_fab_codigo_FK FOREIGN KEY (fab_codigo)
references fabricante(fab_codigo),"+
                "CONSTRAINT produto_cat_codigo_FK FOREIGN KEY (cat_codigo)
references categoria(cat_codigo) ,"+
                "CONSTRAINT produto_cor_codigo_FK FOREIGN KEY (cor_codigo)
references cor(cor_codigo)       ,"+
                "CONSTRAINT produto_ori_codigo_FK FOREIGN KEY (ori_codigo)
references origem(ori_codigo)     "+
               ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE tabela ("+
                "tab_codigo     INTEGER     NOT NULL PRIMARY KEY
,"+
                "tab_descricao  VARCHAR(30) NOT NULL
,"+
                "fab_codigo     INTEGER                                     ,"+
```

```
                "CONSTRAINT tabela_fab_codigo_FK FOREIGN KEY (fab_codigo)
references fabricante(fab_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE tabela_item ("+
                "tab_codigo INTEGER                                          ,"+
                "pro_codigo INTEGER                                          ,"+
                "tit_preco  DOUBLE PRECISION                                      ,"+
                "CONSTRAINT tabela_item_tab_codigo_FK FOREIGN KEY (tab_codigo)
references tabela(tab_codigo),"+
                "CONSTRAINT tabela_item_pro_codigo_FK FOREIGN KEY (pro_codigo)
references produto(pro_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE cliente ("+
                "cli_codigo         INTEGER     NOT NULL PRIMARY KEY,"+
                "cli_estado         CHAR(2)     NOT NULL          ,"+
                "cli_cidade         VARCHAR(30) NOT NULL          ,"+
                "cli_bairro         VARCHAR(30)                   ,"+
                "cli_numero          VARCHAR(10)                  ,"+
                "cli_endereco        VARCHAR(30)                  ,"+
                "cli_telefone       VARCHAR(20)                   ,"+
                "cli_tipo_cliente    CHAR(1)     NOT NULL          ,"+
                "cli_nome           VARCHAR(50) NOT NULL          ,"+
                "cli_data_nascimento DATE        NOT NULL          "+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE pessoa_fisica ("+
                "cli_codigo INTEGER       NOT NULL PRIMARY KEY,"+
                "cli_cpf    CHAR(14)      NOT NULL          ,"+
                "CONSTRAINT pessoa_fisica_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE pessoa_juridica ("+
                "cli_codigo INTEGER  NOT NULL PRIMARY KEY,"+
                "cli_cnpj   CHAR(14) NOT NULL          ,"+
                "CONSTRAINT pessoa_juridica_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                ")";
            stm.executeUpdate(SQL);

            SQL = "CREATE TABLE venda ("+
                "ven_codigo INTEGER NOT NULL PRIMARY KEY
,"+
                "ven_total  DOUBLE PRECISION                                      ,"+
```

```
                    "ven_data   DATE DEFAULT CURRENT_DATE
,"+
                    "cli_codigo INTEGER                                          ,"+
                    "CONSTRAINT venda_cli_codigo_FK FOREIGN KEY (cli_codigo)
references cliente(cli_codigo)"+
                   ")";
                stm.executeUpdate(SQL);

                SQL = "CREATE TABLE venda_item ("+
                    "pro_codigo INTEGER                                          ,"+
                    "ven_codigo INTEGER                                          ,"+
                    "vit_item   INTEGER                                         ,"+
                    "vit_qtde   INTEGER                                         ,"+
                    "CONSTRAINT venda_item_pro_codigo_FK FOREIGN KEY (pro_codigo)
references produto(pro_codigo),"+
                    "CONSTRAINT venda_item_ven_codigo_FK FOREIGN KEY (ven_codigo)
references venda(ven_codigo)   "+
                   ")";
                stm.executeUpdate(SQL);

                jTextAreaResultado.setText(jTextAreaResultado.getText() + "Banco de Dados
criado com sucesso!\n");
                break;
              } catch (Exception e) {
                JOptionPane.showMessageDialog(null, "Ocorreu o seguinte erro: " +
e.getMessage());
              } finally {
                try {
                  conn.close();
                  jTextAreaResultado.setText(jTextAreaResultado.getText() +
"\n\nDesconectado!");
                } catch (Exception onConClose) {
                  onConClose.printStackTrace();
                }
              }
      }
    }

    private void jButtonDroparTabelasActionPerformed(java.awt.event.ActionEvent evt) {
      try {
        jTextAreaResultado.setText("Dropando a base de dados do
"+jComboBoxSGBD.getSelectedItem().toString()+"...\n");

        //conecta ao banco de dados selecionado
        switch (jComboBoxSGBD.getSelectedIndex()) {
          //derby
          case 0:   Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
                conn =
DriverManager.getConnection("jdbc:derby:.\\DATABASE\\DERBY\\bdteste;create=true");
                break;
```

```
            //h2
            case 1: Class.forName("org.h2.Driver");
                conn =
DriverManager.getConnection("jdbc:h2:.\\DATABASE\\H2\\bdteste;create=true");
                break;
            //hsqldb
            case 2: Class.forName("org.hsqldb.jdbcDriver");
                conn =
DriverManager.getConnection("jdbc:hsqldb:.\\DATABASE\\HSQLDB\\bdteste;create=true")
;
                break;
            //mckoi
            case 3: Class.forName("com.mckoi.JDBCDriver");
                conn =
DriverManager.getConnection("jdbc:mckoi:local://./DATABASE/MCKOI/bdteste.conf?creat
e_or_boot=true","mckoi","mckoi");
                break;
            //one$db
            case 4: Class.forName("in.co.daffodil.db.jdbc.DaffodilDBDriver");
                conn =
DriverManager.getConnection("jdbc:daffodilDB_embedded:bdteste;create=true;path=.//DAT
ABASE//ONEDB");
                break;
        }
        stm = conn.createStatement();
        conn.setAutoCommit(true);

         try {
           SQL = "DROP TABLE venda_item";
           stm.executeUpdate(SQL);
        } catch (Exception e) {
           jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
         }
        try {
           SQL = "DROP TABLE venda";
           stm.executeUpdate(SQL);
        } catch (Exception e) {
           jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
         }
        try {
           SQL = "DROP TABLE pessoa_fisica";
           stm.executeUpdate(SQL);
        } catch (Exception e) {
           jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
         }
        try {
           SQL = "DROP TABLE pessoa_juridica";
           stm.executeUpdate(SQL);
        } catch (Exception e) {
           jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
```

```
 }
try {
   SQL = "DROP TABLE cliente";
   stm.executeUpdate(SQL);
} catch (Exception e) {
   jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
 }
try {
   SQL = "DROP TABLE tabela_item";
   stm.executeUpdate(SQL);
} catch (Exception e) {
   jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
 }
try {
   SQL = "DROP TABLE tabela";
   stm.executeUpdate(SQL);
} catch (Exception e) {
   jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
 }
try {
   SQL = "DROP TABLE produto";
   stm.executeUpdate(SQL);
} catch (Exception e) {
   jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
 }
try {
   SQL = "DROP TABLE origem";
   stm.executeUpdate(SQL);
} catch (Exception e) {
   jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
 }
try {
   SQL = "DROP TABLE cor";
   stm.executeUpdate(SQL);
} catch (Exception e) {
   jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
 }
try {
   SQL = "DROP TABLE categoria";
   stm.executeUpdate(SQL);
} catch (Exception e) {
   jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
 }
try {
   SQL = "DROP TABLE fabricante";
   stm.executeUpdate(SQL);
} catch (Exception e) {
   jTextAreaResultado.setText(jTextAreaResultado.getText() + e.getMessage() + "\n");
 }
```

```
        jTextAreaResultado.setText(jTextAreaResultado.getText() + "Banco de Dados
dropado!\n");
      } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Ocorreu o seguinte erro: " + e.getMessage());
      }  finally {
        try {
          conn.close();
          jTextAreaResultado.setText(jTextAreaResultado.getText() + "\n\nDesconectado!");
        } catch (Exception onConClose) {
          onConClose.printStackTrace();
        }
      }
    }

    public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(
        new Runnable() {
          public void run() {
            new JavaDB().setVisible(true);
          }
        }
      );
    }
}
```