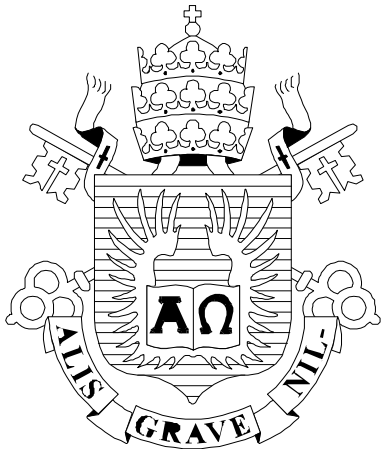

PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 15/11

GearDB: Uma Nova Ferramenta para Geração de Dados

Janaina de Souza Horácio
Andrew Diniz da Costa

Carlos José Pereira de Lucena

Soeli Terezinha Fiorini

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 15/11 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Outubro, 2011

GearDB: Uma Nova Ferramenta para Geração de Dados

Janaina de Souza Horácio, Andrew Diniz da Costa,
Carlos José Pereira de Lucena, Soeli Terezinha Fiorini

{janaina,soeli}@les.inf.puc-rio.br, acosta@inf.puc-rio.br, lucena@inf.puc-rio.br

Abstract. This paper presents GearDB, a new tool for automatically generating data for databases created using different Database Management Systems (DBMS). Aiming at a powerful result, we decided to integrate GearDB with JMeter, since JMeter is a tool used for loading and performance testing, which usually require large volumes of data to carry on their tests. Besides, an industrial large system responsible for controlling the inventory and supply of petroleum and derived products (e.g. gasoline, kerosene, etc) has been used to illustrate a real situation in which GearDB has proven to be useful.

Keywords: Data Generation, Database, Tests, JMeter.

Resumo. Este artigo apresenta o GearDB, uma nova ferramenta responsável por gerar dados de forma automática e inseri-los em bancos de dados criados a partir de diferentes Sistemas de Gerenciamento de Dados (SGBDs). Visando uma maior contribuição do trabalho, decidimos integrar o GearDB com o JMeter, já que o JMeter é uma ferramenta voltada para testes de carga e performance e que geralmente precisam de grandes volumes de dados para realizar seus testes. Além disso, um sistema industrial grande e complexo, responsável por controlar o estoque e suprimento de petróleo e produtos derivados (ex: gasolina, querosene etc), é usado para exemplificar situações reais em que o GearDB foi utilizado.

Palavras-chave: Geração de Dados, Banco de Dados, Testes, JMeter.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introdução

Torna-se cada vez mais comum, empresas adotarem o conceito segurança da informação [Fontes 2005] devido à necessidade de proteger determinados dados. A principal motivação de tal necessidade é a relação do conceito segurança com três princípios básicos: confidencialidade, disponibilidade e integridade. Confidencialidade diz respeito à garantia de que a informação será acessível apenas àqueles que possuem autorização de acesso. Disponibilidade garante que a informação acessada estará disponível o máximo de tempo possível. Já integridade garante que a informação manipulada deve manter as características originais estabelecidas pelo proprietário da informação, incluindo controle de mudanças e garantia do seu ciclo de vida, isto é, informações sobre sua criação, manutenção e destruição.

Assim, empresas têm adotado a política de proteger qualquer tipo de informação que revele o motivo do seu crescimento e prosperidade. Diversos trabalhos na literatura tratam esse tema, como, por exemplo, [Resende 2000]. Tal trabalho considera que toda informação desempenha um papel na empresa e elas devem ser protegidas de acordo com sua importância.

Devido à crescente importância em proteger tais informações, tornou-se comum equipes de desenvolvimento, principalmente equipes terceirizadas, não terem acesso a dados reais usados em ambientes de produção. Conseqüentemente, realizar testes [Myers, 2004] que validem o sistema desenvolvido sem que dados reais sejam utilizados, torna-se muitas das vezes um desafio. Torna-se evidente quando certos requisitos precisam ser validados a partir de testes que usam um grande volume de dados. Testes de desempenho [Denaro et al 2004] [Molyneaux 2009] e de carga [Garousi et al 2006] são alguns tipos de teste [Black et al. 2007] que geralmente precisam dessa grande quantidade de dados. No entanto, a geração deles pode ser custosa, principalmente se for feita de forma manual, já que em diversas situações tais dados devem obedecer a regras pré-definidas pelo sistema.

Alguns exemplos de critérios que definem quais dados devem ser gerados são: tamanho da informação, formato dos dados e regras de negócio. Assim, para realizar a geração adequada, tais informações precisam obedecer a critérios que podem conduzir a um grande volume de dados diferentes.

Diversas ferramentas propostas na literatura visam ajudar nessa geração de dados, como, por exemplo, [Fresh Trash Generator 2011] e [Test Dictionary 2011]. No entanto, tais ferramentas apresentam um conjunto de desvantagens, como: (i) dificuldade de uso por não proverem uma boa documentação, (ii) não é possível realizar extensões por não serem *open source*, e (iii) não permitem a geração de dados em bases já existentes e que tenham algum tipo de restrição, como, por exemplo, colunas que aceitem somente números inteiros, colunas que definem uma quantidade máxima de caracteres, assim como colunas que não aceitam valores nulos ou que sejam chaves primárias ou chaves estrangeiras.

Visando sanar as desvantagens presentes nas ferramentas propostas na literatura, esse artigo apresenta o GearDB, uma nova ferramenta de geração de dados criada a partir da experiência em testar sistemas grandes e complexos (mais de 150 mil linhas de codigo com 300 a 400 regras de negócio) no Laboratório de Engenharia de Software da Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) [LES 2011]. Uma das principais características dessa ferramenta é a integração com o [JMeter 2011], ferra-

menta voltada para testes de carga e performance. Tal integração visa permitir que a geração de dados seja feita de forma automática a partir do JMeter sem que seja necessário sair dela, possibilitando maior agilidade na execução dos testes criados.

Esse artigo está organizado da seguinte maneira. Na Seção 2 são apresentados os procedimentos empíricos adotados para que a nova ferramenta de geração de dados fosse proposta. Na Seção 3 são apresentados alguns trabalhos relacionados. Na Seção 4 a ferramenta GeardDB é apresentada, enquanto que na Seção 5 são descritas algumas situações em que a ferramenta foi utilizada em um sistema industrial. Tal sistema controla o estoque e suprimento de petróleo e seus derivados (ex: gasolina, querosene etc). Por fim, na Seção 6 são apresentadas as conclusões e os trabalhos futuros.

2 Procedimentos Empíricos

Na Figura 1 são ilustrados os procedimentos seguidos para que uma nova ferramenta de geração de dados pudesse ser proposta. Esse trabalho teve a duração de quatro anos e foi composto por cinco etapas. A primeira etapa foi responsável pela criação e execução de diferentes tipos de teste em aplicações grandes e complexas. As ferramentas usadas foram [Rational Manual Test 2011], [Rational Functional Test 2011], [Rational Performance Tester, 2011] e [DBUnit 2011], responsáveis, respectivamente, por criar testes manuais, testes funcionais automatizados, testes de performance automatizado e testes unitários de banco. Essa etapa durou três anos, enquanto que o restante foi realizado no quarto ano.

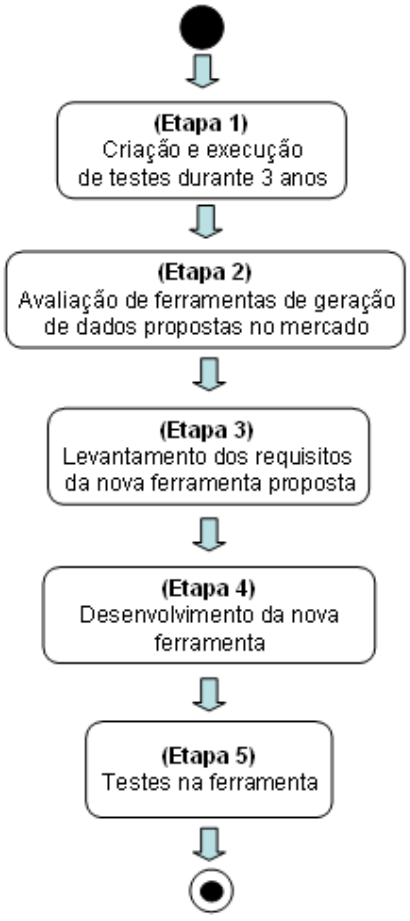

Figura 1. Procedimentos empíricos seguidos no trabalho.

Enquanto testes eram criados e executados, percebeu-se a necessidade de utilizar ferramentas que realizassem a geração automática de dados para bancos de dados existentes. A partir dessa necessidade, decidimos avaliar quais ferramentas oferecidas no mercado seriam interessantes de serem usadas (segunda etapa). A partir desse estudo, percebemos que todas as abordagens analisadas apresentavam desvantagens o que nos impedia de utilizá-las, como, por exemplo: (i) o alto valor para utilizá-las, (ii) não possuíam uma boa documentação, (iii) não realizavam a geração automática de dados a partir de uma base já criada e que apresentasse regras pré-definidas, (iv) não inseriam dados gerados em tabelas, e (v) a instalação das ferramentas era trabalhosa e pouco intuitiva.

Na terceira etapa decidimos realizar o levantamento de quais requisitos deveriam ser implementados na nova ferramenta a fim de atender as necessidades dos sistemas testados. Assim, diversas reuniões foram feitas envolvendo os líderes dos projetos.

Após esse levantamento, o desenvolvimento da ferramenta foi realizado na quarta etapa, enquanto que na quinta etapa a ferramenta foi testada e utilizada em diferentes sistemas. A partir desse uso, novos requisitos foram incluídos na ferramenta tornando-a estável.

3 Trabalhos Relacionados

Muitas ferramentas de geração de dados foram propostas na literatura. [Spawner Data Generator, 2011] e [DGMaster, 2011], são exemplos de ferramentas que oferecem uma interface simples e intuitiva. Apesar disso, não permitem a inserção de dados gerados em tabelas de algum banco de dados já existente.

[DataGenerator, 2011] apresenta uma ferramenta de geração de dados baseada em regras de testes (ex: gerar dados considerando um intervalo de números, quantidade de caracteres etc) que são fornecidas pelo usuário. Essa ferramenta gera dados para tabelas de bancos criadas a partir de diferentes SGBDs, como, por exemplo, Oracle e Firebird. Uma das principais características da ferramenta é que ela exibe todas as tabelas do esquema e muitas delas não são referentes ao negócio do sistema, isto é, são tabelas internas do SGBD. Além disso, ela não permite flexibilidade para que dados possam ser gerados para tabelas de bancos criados a partir de outros SGBDs, como, por exemplo, SQL Server.

[DBMonster, 2011] propõe uma ferramenta desenvolvida em Java que gera dados de forma randômica e os armazena em um banco de dados a partir de scripts SQL gerados. Essa ferramenta é voltada para desenvolvedores de aplicações com banco de dados que possuem a necessidade de ajustar índices ou testar o desempenho da aplicação sob um grande volume de carga. Na versão analisada (v1.0.1), ela não possui uma interface gráfica, dificultando seu uso, pois obriga os usuários entenderem sua lógica de desenvolvimento e configuração. Visando evitar tais problemas, a nova ferramenta proposta no artigo pretende prover uma interface intuitiva para que usuários consigam utilizá-la sem a necessidade de um treinamento longo ou de uma documentação extensa e complexa.

Tabela 1. Pontos fortes e fracos das ferramentas analisadas.

Nome da Ferramenta	Pontos Fortes	Pontos Fracos
Fresh Trash Generator	Oferece suporte a diversos SGBDs. Além disso, as regras de geração de dados podem ser especificadas pelo usuário.	Todas as tabelas, inclusive tabelas internas de SGBDs são apresentadas. Assim, a identificação das tabelas do negócio pode se tornar confusa.
Data Generator	Permite o uso de diversos SGBDs. Possui alta performance, resolve relacionamentos de tabelas de forma automática (chave estrangeira), cria tabelas de teste e exibe procedures desenvolvidas, assim como dados armazenados em tabelas.	Tela poluída com diversas opções, impedindo boa usabilidade da ferramenta. Além disso, não é uma ferramenta gratuita.
DGMaster	Permite o tratamento de muitos tipos de dados, além de gerar dados em diversos formatos: texto, BD, xml etc.	Apresenta instabilidade operacional.
Spawner Data Generator	Insere dados no MYSQL 5.x, e possui interface simples e intuitiva.	As colunas das tabelas têm que ser configuradas, ou seja, para cada tabela o usuário tem que informar qual o tipo e tamanho de informação de cada uma das colunas.
CSV Data Generator	Gera um arquivo CSV e os atributos das colunas que vão ser geradas e que podem ser personalizadas pelo usuário.	Gera dados somente no formato CSV.
Test Dictionary	Gera nomes próprios, termos e trata relacionamentos de tabelas. Bastante útil para preencher tabelas com dados de teste mais realistas.	Instalação e uso da ferramenta pouco intuitivo.
GenerateData	Tem uma versão online gratuita que gera dados em diversos formatos, assim como grandes volumes de dados.	Usuário tem que especificar as regras de geração de dados para todas as colunas das tabelas, além de não permitir a inserção dos dados diretamente em tabelas de diferentes bancos.
DBMonster	Gera dados de forma randômica armazenando-os em um banco de dados.	Não possui uma interface gráfica para gerar dados.

Outras ferramentas analisadas, como [CSV Data Generator, 2011], [GenerateData, 2011] e [Fresh Trash Generator, 2011] apresentam alguns problemas adicionais e si-

milares aos mencionados acima, como: interfaces complexas para uso, (ii) não oferecem flexibilidade para que dados sejam inseridos em bancos de dados criados a partir de diferentes SGBDs, (iii) geram dados sem considerar restrições pré-estabelecidas por alguma tabela, (iv) apresentam tabelas que não fazem parte do negócio, e (iv) são muito caras para uso. Para sanar tais limitações, decidimos propor a nova ferramenta de geração de dados chamada GearDB. Na Tabela 1 são apresentados os principais pontos fortes e fracos de cada ferramenta analisada.

4 GearDB

GearDB é uma ferramenta de geração de dados que ao ser acionada pelo usuário gera informações aleatórias que são inseridas diretamente em uma ou mais tabelas de um banco de dados. A ferramenta foi projetada para dar suporte a bancos de dados criados a partir de diversos SGBDs, pois cada SGBD possui alguma peculiaridade quando armazenam dados referentes a tabelas, como, por exemplo, restrições de tabelas (ex: chaves primárias, chaves estrangeiras, limite de caracteres etc). Assim, a ferramenta foi projetada com pontos flexíveis (*hot spots*) [Fayad e Johnson 1999] para que fosse possível trabalhar com bancos de dados criados a partir de diferentes SGBDs (ex: SQL Server, Oracle, DB2 etc).

Nesta Seção 4.1 é apresentada a idéia geral do GeardDB, seguida por uma explicação mais detalhada dos tratamentos realizados pela ferramenta para realizar a geração de dados automática na Seção 4.2. Na Seção 4.3 é apresentado como o GearDB pode ser usado a partir do JMeter, e por último, na seção 4.4, são listados os pontos fixos e flexíveis da ferramenta.

4.1 Idéia Geral da Ferramenta

A Figura 2 ilustra a idéia geral da ferramenta GearDB, que é gerar dados para bancos de dados criados a partir de diferentes SGBDs. Ela oferece uma arquitetura facilmente estendida a fim de tratar peculiaridades de diferentes SGBDs. Atualmente, a ferramenta provê implementações que reconhecem bancos criados pelo SGBD Oracle [Oracle, 2010].

Na Figura 3 são apresentadas as principais classes da ferramenta. A classe *LoginApp* é responsável por recuperar informações de login, senha e a url do esquema a ser acessado pela ferramenta. A partir disso, uma instância da classe *Login* é criada para armazenar as informações recuperadas. Já a classe *ConexaoStrategy* define qual estratégia será adotada para acessar o banco de dados desejado. Quando o banco é acessado pela ferramenta, suas tabelas são recuperadas pela classe com prefixo *Concrete*, que é responsável por seguir as regras do banco utilizado. As restrições de cada tabela são representadas pela classe *TabelasDao*, que é lida pela *TabelaApp*. Já a classe *InserirValores* é usada para realizar inserções de dados na tabela correspondente. Tal classe usa a classe *GeraValoresContinuos* para gerar dados de forma automática. Essa classe gera dados totalmente aleatórios considerando a quantidade de tuplas e a quantia máxima de caracteres em cada coluna onde as informações deverão ser inseridas, o tipo de dado e as restrições de integridade definidas por coluna (ex: chave primária e chave estrangeira). Caso a tabela, em que terá dados gerados, tenha uma chave estrangeira para outra tabela, a classe *Coluna* é instanciada. Tal instância irá guardar as informações de cada uma das colunas da tabela referenciada.

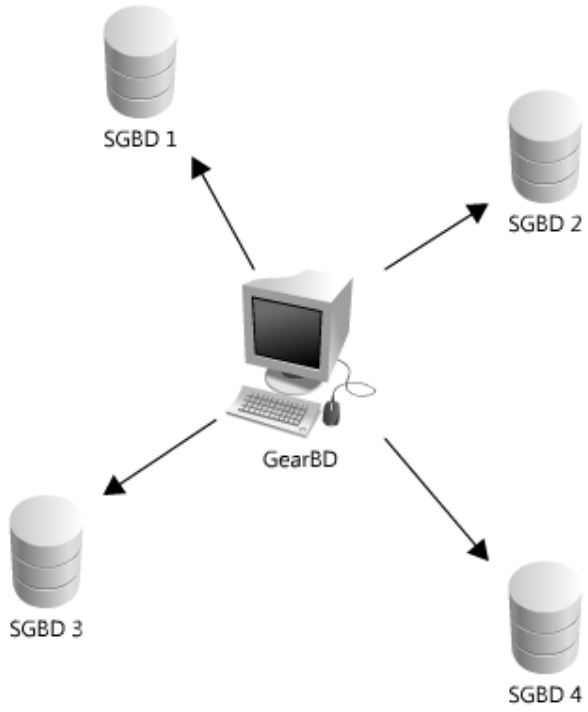

Figura 2. Modelo Conceitual da Ferramenta.

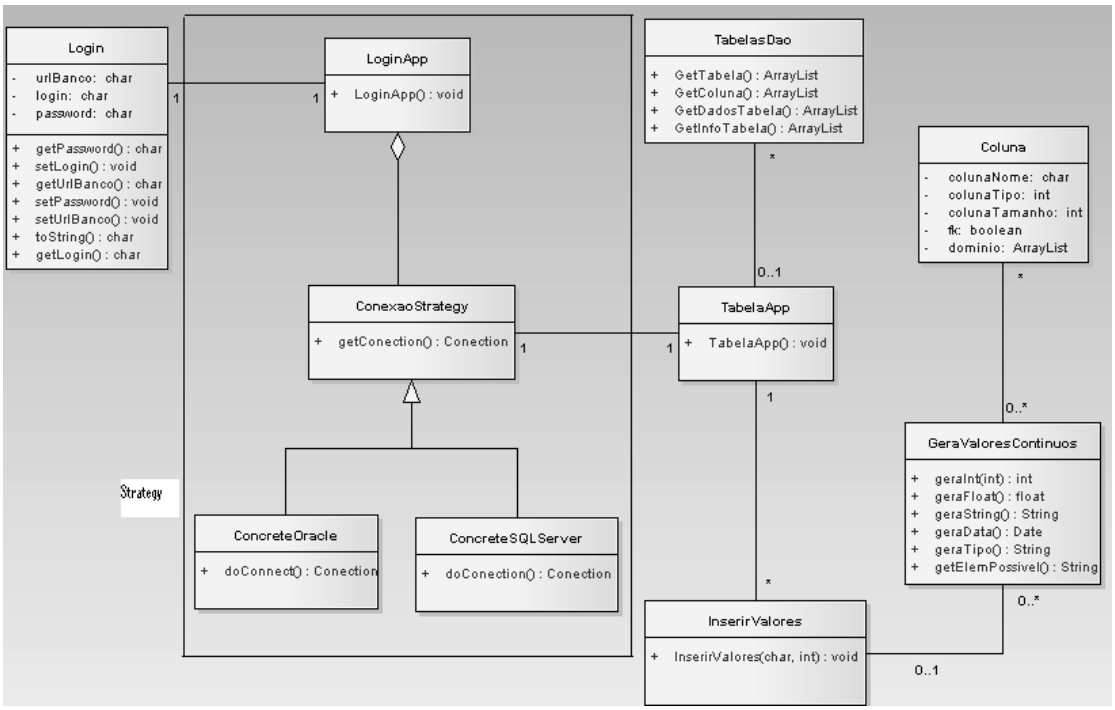

Figura 3. Diagrama com as principais classes.

No momento em que o usuário solicita uma conexão com um banco de dados e esta conexão é realizada com sucesso, a ferramenta recupera apenas as tabelas do modelo de dados em que o usuário tem permissão de acesso para escrita. Assim, não haverá o

problema de um usuário solicitar a geração de dados para alguma tabela que não possua tal permissão.

Como diferentes SGBDs podem ser usados, diferentes estratégias de conexão podem ser adotadas. Assim, para representar essas possíveis conexões, o padrão Strategy [Gamma et al. 2000] foi utilizado.

4.2 Tratamento para Geração de Dados

Para que a ferramenta proposta conseguisse realizar a geração de dados para diferentes esquemas de banco de dados, diversos tratamentos foram realizados. Inicialmente a ferramenta recupera o tipo de dado armazenado por uma coluna de uma tabela, e em seguida verifica se existe uma correspondência deste tipo de dado do SQL com um tipo primitivo do Java (ex: boolean, short, int, etc). Caso exista tal correspondência, a quantidade de dados solicitada pelo usuário será gerada pela ferramenta e inserida na tabela correspondente. Caso contrário, a ferramenta verifica se existe alguma classe Java que corresponda ao dado solicitado. Se houver, a ferramenta instancia esta classe para que possa tratar o tipo de dado a ser gerado. Um exemplo seria uma tabela escolhida pelo usuário com uma coluna do tipo Date no SQL. Nesse caso, o GearDB instancia a classe Java.sql.Date no Java para gerar os dados solicitados. Na Tabela 2 são apresentadas as correspondências entre os tipos de dados em Java e SQL tratadas pela ferramenta.

Outro tratamento realizado pelo GearDB é referente aos relacionamentos que podem existir entre tabelas de um banco de dados, isto é, chaves estrangeiras. Quando é solicitada a geração de dados para uma tabela que possui relacionamentos com outras tabelas, a ferramenta gera dados para as tabelas envolvidas a fim de atender a solicitação realizada. No entanto, GearDB gera a quantidade mínima de tuplas possíveis para as tabelas relacionadas a partir de alguma chave estrangeira válida. Por outro lado, se as tabelas relacionadas já possuírem dados, a ferramenta os utiliza.

Outro ponto importante tratado pela ferramenta é a geração de dados considerando o conceito de chave primária simples e composta. Chave primária simples é identificada a partir de uma única coluna, enquanto que chave composta usa duas ou mais colunas de uma tabela. Tais tratamentos são feitos pela classe *InserirValores*, ilustrada na Figura 3, pois nela deve estar presente a estratégia responsável por gerar os dados aguardados. Essa classe é um ponto flexível do framework, já que diferentes estratégias podem ser representadas.

Além disso, quando uma conexão é realizada com um banco de dados, a ferramenta recupera apenas as tabelas do modelo de dados que o usuário tem permissão de acesso para escrita. Assim, não haverá o problema de um usuário solicitar a geração de dados para alguma tabela que não possua essa permissão de escrita. A classe responsável por isso é a TableApp (ver Figura 3).

Tabela 2. Relação dos tipos de dados do Java e SQL.

Tipo de Dados em Java	Tipos de Dados em SQL
String	VARCHAR, LONGVARCHAR
Java.math.BigDecimal	NUMERIC
Boolean	BIT
Byte	TINYINT
Short	SMALLINT
Int	INTEGER
Long	BIGINT
Float	REAL
Double	DOUBLE
Byte[]	VARBINARY, LONGVARBINARY
Java.sql.Date	DATE
Java.sql.Time	TIME
Java.sql.Timestamp	TIMESTAMP

4.3 Acessando GearDB a partir do JMeter

Como os diversos projetos testados pelo laboratório de engenharia de software precisam de grandes volumes de dados para executar testes de performance e carga, decidimos integrar o GearDB com o JMeter, ferramenta voltada para a criação de tais testes e usada em alguns desses projetos. Para que isso fosse possível, criamos uma opção chamada "Gerar dados" no menu da tela principal do JMeter (ver Figura 4). A partir dessa opção a GeardDB pode ser usado para gerar de forma automática um conjunto de dados que deverão ser incluídos em uma ou mais tabelas de algum banco de dados.

Na Figura 5 é ilustrado um processo composto por quatro passos exemplificando como o GearDB pode ser utilizado. No primeiro passo, o usuário deve fornecer três informações: (i) url, (ii) usuário e (iii) senha de acesso a algum banco de dados. No segundo passo, após realizar a conexão com o banco, o usuário deve escolher qual tabela terá dados incluídos, gerados pelo GearDB. As tabelas disponíveis são recuperadas pelo GearDB a partir das informações fornecidas pelo usuário no passo 1. Ao selecionar a tabela desejada, uma tela é apresentada ao usuário (terceiro passo) solicitando a quantidade de tuplas que devem ser inseridas. Após fornecer tal quantidade, os dados são gerados de forma automática e inseridos na tabela. Caso haja chaves estrangeiras, o sistema insere dados nas tabelas relacionadas. Essas inserções somente são realizadas se tais tabelas não possuírem dados que possam ser usados como chave estrangeira. Já no quarto passo, o usuário poderá escolher qual tabela deseja visualizar os dados gerados, pois por cada geração n tabelas podem estar envolvidas.

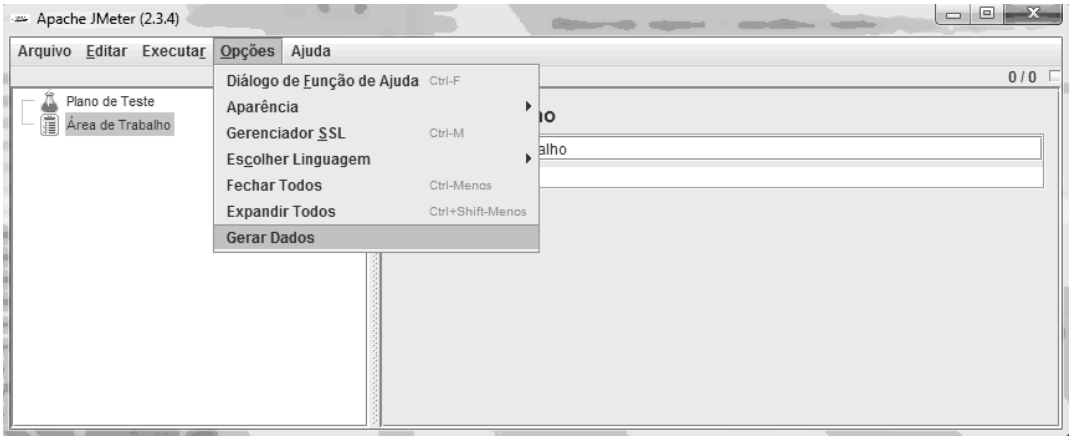

Figura 4. Integração com JMeter.

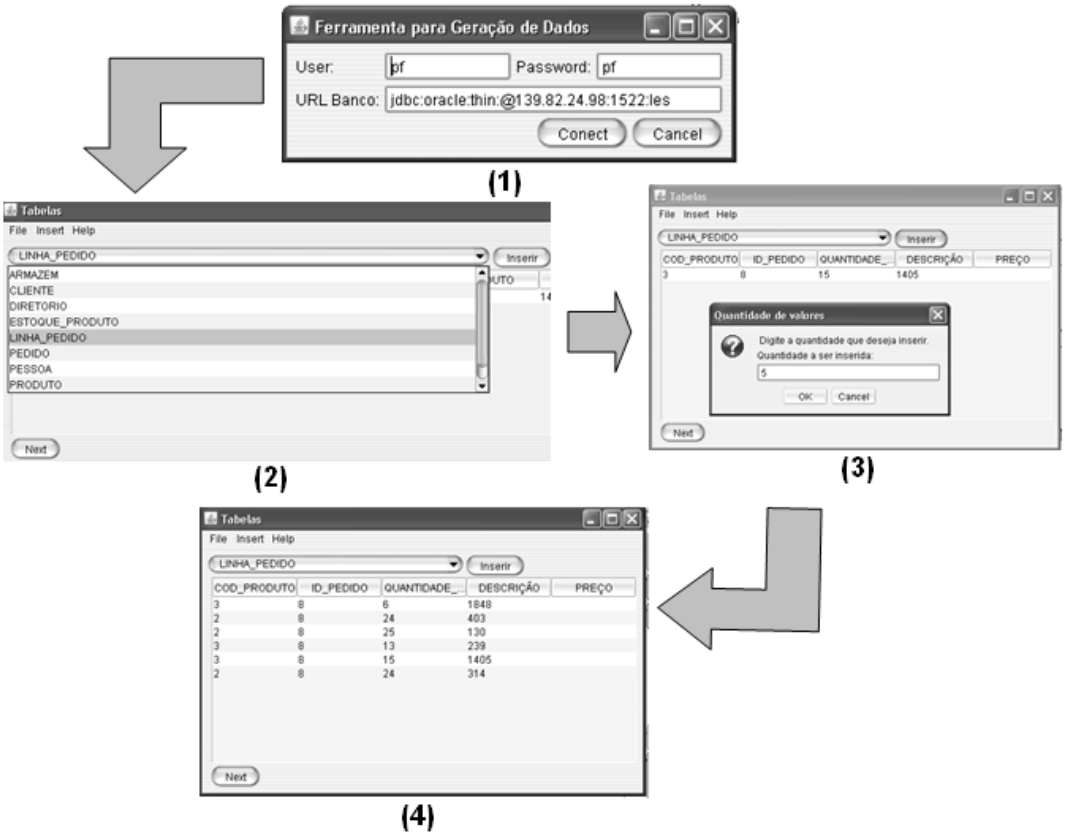

Figura 5. Utilizando a ferramenta GearDB.

4.4 Pontos Fixos e Flexíveis

Como o GearDB foi criado para permitir o uso de bases criadas a partir de diferentes SGBDs, as classes ilustradas na Figura 3 são consideradas pontos fixos (frozen-spots) ou pontos flexíveis (hot-spots) [Fayad e Johnson 1999] da ferramenta. Os pontos fixos são os seguintes:

- **LoginApp:** Classe responsável por apresentar a tela ao usuário com informações de login e que devem ser providas pelo usuário.

- **Login:** Classe responsável por guardar as informações de login fornecidas pelo usuário.
- **GeraValoresContinuos:** Classe responsável por tratar a relação entre os dados SQL e Java. A partir disso é realizada a geração dos dados em Java. Para maiores detalhes, ver seção 4.2.
- **TableApp:** Classe responsável por recuperar e montar com a ajuda da TableDao uma tabela fictícia em Java com informações referentes a cada coluna de uma determinada tabela.
- **Coluna:** Classe que guarda informações referentes a cada coluna de uma tabela fictícia.
- **TabelaApp:** Classe responsável por guardar uma tabela fictícia em Java (uma instância criada na classe TableApp) até que a geração e inserção de dados seja finalizada.
- **ConcreteOracle:** Classe responsável por receber informações referentes ao banco de dados Oracle e realizar a conexão desejada.

Já os pontos flexíveis (hot-spots) da ferramenta são os seguintes:

- **ConexaoStrategy:** Responsável por representar estratégias de conexão com bancos de dados criados a partir de diferentes SGBDs. Para criar uma nova conexão, basta estendê-la e implementar o método *doConnect()*.
- **InserirValores:** Classe responsável por coletar os dados gerados e inseri-los em uma tabela. Como algumas bases podem definir diferentes tipos de dados, formas distintas de inserção podem ser definidas, como, por exemplo, inserir números inteiros a partir de seqüência definida.

5 Estudo de Caso

Para exemplificar situações reais em que o GearDB foi utilizado, escolhemos um dos sistemas mais complexos trabalhado no Laboratório de Engenharia de Software. Assim, inicialmente apresentamos em detalhe a idéia geral do sistema, seguida pela descrição de situações em que o GearDB foi utilizado.

5.1 Idéia Geral do Sistema

Este estudo de caso foi realizado no Laboratório de Engenharia de Software com uma equipe especializada na área de testes. Essa equipe tem trabalhado intensivamente na criação e execução de testes em diversos sistemas desenvolvidos para uma grande empresa de petróleo e gás no Brasil.

Para demonstrar a utilidade da ferramenta decidimos usar como estudo de caso um dos maiores sistemas testado no laboratório. Esse sistema tem como objetivo central realizar o controle do estoque e suprimento de petróleo e seus produtos derivados (ex: gasolina, querosene etc.)

Alguns dos objetivos do sistema são: (i) registrar rotas (caminhos que ligam terminais e refinarias localizadas no Brasil) com base em dutos e navios que poderiam ser usados para o transporte de produtos (ex: gasolina, óleos lubrificantes, querosene etc); (ii) cadastrar planejamentos de quando esses produtos chegarão a terminais e refinarias; (iii) definir quais são as melhores rotas para o transporte de um determinado produto; (iv) registrar os dados reais de quando e quais produtos chegaram nos terminais e refinarias; (v) comparar dados reais com dados planejados; (vi) gerar diferentes tipos de relatórios e gráficos para ajudar na análise das diferentes atividades realizadas a partir do sistema; e (vii) controlar importações e exportações de produtos para outros países.

O sistema foi desenvolvido por quatro equipes: interface, banco de dados, requisitos e teste. A equipe de teste é composta por sete pessoas responsáveis por criar e executar testes unitário de banco de dados criados a partir do DBUnit, testes funcionais criados pela Rational Functional Tester e Rational ManualTest, além de testes de desempenho criados pelo Rational Performance Tester. A Tabela 3 apresenta algumas informações adicionais do sistema testado enquanto que a Tabela 4 apresenta a relação da quantidade de casos de uso testados com os tipos de teste criados.

Tabela 3. Informações sobre o sistema usado no estudo de caso.

Tempo do Projeto	Pessoas Envolvidas	Time de Teste	Tamanho do Modelo	Tamanho do BD/Código	Casos de Teste
7 anos	25 pessoas	7 pessoas	49 casos de uso 400 regras de negócio 712 classes	213 Tabelas 156.602 linhas de código	46 testes de banco 17 testes de performance 120 testes funcionais automatizados 600 testes funcionais manuais

5.2 Cenários de Uso

Alguns casos de teste criados para o sistema descrito utilizaram o GearDB, que ficou responsável por gerar dados que seriam usados na execução dos testes. Para exemplificar algumas situações práticas do uso da ferramenta, quatro testes de diferentes funcionalidades e que a utilizaram são apresentados nessa seção.

O primeiro exemplo é referente a um teste funcional automatizado responsável por testar um relatório cujo objetivo era apresentar o quanto existia de estoque em refinarias e terminais em um dia específico fornecido pelo usuário. Esse relatório recupera tais informações a partir de diversas tabelas. Assim, para que o teste pudesse validar a consistência dos dados apresentados na interface com os dados armazenados em banco o GearDB foi utilizado para popular as tabelas acessadas pelo teste.

Tabela 4. Relação dos casos de teste com os casos de uso testados.

Tipos de teste	Quantidade de testes	Casos de Uso Testados	Média de Testes por Caso de Uso
Testes de banco	46	4	11,5
Teste de Performance	17	6	2,8
Testes Funcionais Automatizados	120	11	10,9
Testes Funcionais Manuais	600	45	13,3

Outro exemplo é um teste unitário de banco responsável por verificar se um cenário criado a partir de outro cenário apresenta as informações esperadas. Segundo o sistema testado, cenário é uma entidade que apresenta um conjunto de informações mensais e planejadas referente a diversos produtos, como, por exemplo, produtos importados ou exportados, assim como a quantia de cada produto em terminais e refinarias. Assim, quando um cenário é criado a partir de outro, os dados do cenário base são copiados para o novo cenário. A partir disso, o objetivo do teste é verificar se os dados do cenário base foram copiados corretamente para o cenário derivado. No entanto, para executar tal análise há a necessidade de criar um cenário base que recupera dados de diversas tabelas. Visando prover tais dados, o GeardDB foi utilizado.

Um teste de performance realizado pela equipe de teste foi referente à medição do tempo para abrir um cenário base, já que informações são recuperadas de diversas tabelas. Após tais recuperações, esses dados são estruturados para que fiquem legíveis aos usuários. Assim, dependendo da quantidade de dados recuperados, o sistema pode ter um comportamento mais lento para abri-lo. Para testar diferentes volumes de dados, o GeardDB foi utilizado.

Por fim, há um teste funcional para a tela de cadastro de rotas. Nessa tela são apresentadas diversas informações, como, por exemplo, quais terminais ou refinarias serão escolhidas como origem e destino da rota, assim como quais produtos poderão ser transportados entre esses lugares. A partir disso, um dos testes criados foi responsável por verificar se as informações apresentadas na tela de cadastro estão corretas. Para testar tais valores, um conjunto de dados foi gerado pelo GearDB.

6 Conclusões e Trabalhos Futuros

Nesse artigo foi apresentada a ferramenta de geração de dados GearDB, que permite a inserção de informações em uma base de dados já criada e informada pelo usuário. Além disso, as regras definidas na base também são respeitadas no momento da geração dos dados.

Visando uma maior contribuição, realizamos a integração do GearDB com o JMeter. Tal integração tornou mais fácil a inserção de dados quando testes de carga e performance forem criados a partir do JMeter. Apesar do GearDB permitir essa fácil inserção de dados em tabelas existentes, as seguintes melhorias serão realizadas: (i) tratar bases

de dados que possuam tabelas com referencia circular, (ii) prover implementações que tratem a comunicação e as restrições definidas em outros SGBDs, como, por exemplo, SQL Server, MySql e DB2, (iii) melhorar o tratamento para a inserção em tabelas que possuam referência para outras tabelas (chave estrangeira) que o usuário não possua permissão de escrita e leitura, e (iv) realizar um tratamento de concorrência na inserção de dados nas tabelas.

Referências

Black, R., Eldh, S., Evan, I., et al. "Glossário Padrão de Termos Utilizados em Teste de Software", http://www.bstqb.org.br/uploads/docs/glossario_pb_1.4.pdf, Versão 1.3br, 2007.

CSV Data Generator, http://rubyforge.org/projects/datagen/. Último acesso em Março de 2011.

Databene, http://databene.org/. Último acesso em Fevereiro de 2011.

DataGenerator, http://datagenerator.sourceforge.net/?q=node. Último acesso em Março de 2011.

DBMonster, http://sourceforge.net/projects/dbmonster/. Último acesso em Março de 2011.

DBUnit web site, http://www.dbunit.org/. Último acesso em Fevereiro de 2011.

Denaro, G., Polini, A., and Emmerich,W. "Early performance testing of distributed software applications", SIGSOFT Software Engineering Notes, 2004, 29(1):94–103.

DGMaster, http://dgmaster.sourceforge.net/. Último em Março de 2010.

Fayad, M., Johnson, R. "Building Application Frameworks: Object-Oriented Foundations of Framework Design (Hardcover)", Wiley publisher, first edition, ISBN-10: 0471248754, 1999.

Fontes, E. "Segurança da Informação", editora: Saraiva, 1a edição, ISBN 8502054422., 2005

Fresh Trash Generator, http://sourceforge.net/projects/freshtrash/. Último acesso em Março de 2011.

Gamma, E.; Helm, R., Johnson, R., Vlissides, J. "Padrões de Projeto: Soluções reutilizáveis de software orientado a objeto", 1a edição, ISBN 8573076100, 2000.

Garousi, V., Briand, L., and Labiche, Y. "Traffic-aware stress testing of distributed systems based on UML models", in Proceedings of the 28th International Conference on Software Engineering (ICSE), Shanghai, China, 2006, pages 391–400.

GenerateData, http://www.generatedata.com/#about. Último acesso em Fevereiro de 2011.

Interbase, http://www.embarcadero.com/products/interbase-smp. Último acesso em Fevereiro de 2011.

JMeter, http://jakarta.apache.org/jmeter/. Último acesso em Fevereiro de 2011.

LES – Laboratório de Engenharia de Software, http://www.les.inf.puc-rio.br/. Último acesso em Fevereiro de 2011.

Molyneaux, I. The Art of Application Performance Testing: Help for Programmers and Quality Assurance. O'Reilly Media. 1a edição, ISBN 0596520662, 2009.

Myers, G. "The Art of Software Testing", John Wiley & Sons., 2a edição, 2004.

Pressman, R. "Engenharia de Software", McGraw-Hill, 6a edição, ISBN 0072853182, 2005.

Resende, A. D. Pedro: http://www.cic.unb.br/~pedro/segdados.htm. Último acesso em Fevereiro de 2011.

Rational Functional Tester, http://www-01.ibm.com/software/awdtools/tester/functional/. Último acesso em Fevereiro de 2011.

Rational ManualTest, http://www-01.ibm.com/software/awdtools/test/manager/. Último acesso em Fevereiro de 2011.

Rational Performance Tester, http://www.acutest.co.uk/acutest/testing-rational-ibm. Último acesso em Fevereiro de 2011.

Spawner Data Generator, http://sourceforge.net/projects/spawner/. Último acesso em Março de 2011.

Test Dictionary, http://sourceforge.net/projects/test-dictionary/. Último acesso em Março de 2011.