

# **XXI SIMPÓSIO BRASILEIRO DE BANCO DE DADOS**

## **ANAIS DA III SESSÃO DE DEMOS**

**16 a 20 de outubro de 2006  
Florianópolis, Santa Catarina, Brasil**

### **Promoção**

SBC – Sociedade Brasileira de Computação  
Comissão Especial de Banco de Dados

ACM – Association for Computing Machinery  
SIGMOD – Special Interest Group on Management of Data

VLDB – Very Large Data Base Endowment Inc.

### **Edição**

Ângelo Brayner – UNIFOR

Cristina Dutra de Aguiar Ciferri – ICMC/USP São Carlos

### **Organização**

UFSC – Universidade Federal de Santa Catarina

IDESTI – Instituto de Desenvolvimento, Pesquisa e Capacitação  
em Gestão Social de Tecnologia de Informação

### **Realização**

Departamento de Informática e Estatística – UFSC

## FICHA CATALOGRÁFICA

Catálogo na fonte pela Biblioteca Universitária da UFSC

S612a Simpósio Brasileiro de Banco de Dados (21. : 2006 : Florianópolis, SC).  
Anais da III Sessão de Demos / XXI Simpósio Brasileiro de Banco de Dados ; edição Ângelo Brayner, Cristina Dutra de Aguiar Ciferri. – Florianópolis : Sociedade Brasileira de Computação, 2006.  
48 p. : il.

ISBN: 85-7669-084-5

1. Banco de Dados – Congressos. I. Brayner, Ângelo. II. Ciferri, Cristina Dutra de Aguiar. III. Sociedade Brasileira de Computação. IV. Sessão de Demos (3. : 2006 : Florianópolis, SC). V. SBBD (21. : 2006 : Florianópolis, SC). VI. Título.

CDU: 681.31:061.68

“Esta obra foi impressa a partir de originais entregues, já compostos pelos autores”.

Editoração: Frank Siqueira  
Arte Gráfica do Evento/Capa: Sandro Coutinho de Azevedo  
Impressão: Mattes Agência Gráfica

## Prefácio

A Sessão de Demos, em sua terceira edição, demonstra sua consolidação como um evento satélite ao Simpósio Brasileiro de Banco de Dados (SBDD). O objetivo básico da Sessão de Demos é proporcionar um fórum para intercâmbio de experiências e de soluções automatizadas entre pesquisadores e desenvolvedores da área de Banco de Dados no Brasil. Portanto, a Sessão de Demos apresenta um foco prioritário em produtos e protótipos na área de Banco de Dados.

A Sessão de Demos 2006 recebeu um total de 17 submissões de três países diferentes (Brasil, Portugal e Suíça). Este fato demonstra que o evento já tem visibilidade até fora do Brasil. Do total de submissões foram selecionadas 8 ferramentas para participar do processo de escolha das três melhores ferramentas. Mantendo as estatísticas passadas, mais de 70% dos trabalhos enviados estão diretamente relacionados a trabalhos de mestrado e doutorado, bem como a projetos de pesquisa. Vale ressaltar que os trabalhos submetidos encontram-se, em sua grande maioria, dentro das grandes áreas de Banco de Dados.

O processo de avaliação das ferramentas é realizado em duas etapas. Na primeira etapa, os autores enviam artigos descrevendo as principais características de suas ferramentas, para avaliação por parte dos membros do Comitê de Avaliação. Na segunda etapa, os trabalhos selecionados são apresentados, sob a forma de demonstrações *in loco*, durante o evento, quando então são novamente avaliados por membros do Comitê. Esta segunda avaliação visa garantir que as ferramentas selecionadas sejam operacionais e funcionalmente compatíveis com a descrição contida nos documentos avaliados na primeira etapa. Além disso, os resultados das duas avaliações fornecem subsídios para que o Comitê de Avaliação possa escolher as três melhores ferramentas. Em sua terceira edição, a apresentação de ferramentas durante o evento consiste apenas das demonstrações práticas dos trabalhos selecionadas pelo Comitê de Avaliação. As demonstrações estão distribuídas nos turnos da manhã e da tarde, nos horários das 9h às 12h30, e das 14h às 16h30 do dia 17 de outubro de 2006.

Gostaríamos de agradecer ao Comitê de Avaliação pelo excelente trabalho realizado e pelo atendimento aos prazos. Agradecemos ainda ao Comitê Diretor do SBDD que entendeu a importância dos anais impressos para a Sessão de Demos.

Finalmente, desejamos a todos os participantes um excelente e motivante evento.

Ângelo Brayner – UNIFOR

Cristina Dutra de Aguiar Ciferri – ICMC/USP – São Carlos

Coordenadores da Sessão de Demos 2006 do SBDD



**Angelo Brayner** concluiu o doutorado em Ciência da Computação pela Universität Kaiserslautern, Alemanha, em 1999. Atualmente é professor titular da Universidade de Fortaleza, onde é responsável pela cadeira de Banco de Dados dos cursos de graduação e Mestrado em Ciência da Computação da UNIFOR. É autor do livro *Transaction Management in Multidatabases Systems*, publicado pela Shaker-Verlag, Alemanha, em 1999. Prof. Angelo Brayner é autor de vários artigos publicados em periódicos e conferências internacionais e nacionais.



**Cristina Dutra de Aguiar Ciferri** é docente do Departamento de Ciências de Computação da Universidade de São Paulo/Campus de São Carlos desde abril de 2005. Obteve seu título de doutor em Ciência da Computação pela Universidade Federal de Pernambuco em 2002, na área de concentração de banco de dados. Entre 1996 e 2005 foi professora do Departamento de Informática da Universidade Estadual de Maringá, onde participou de diversos projetos de pesquisa financiados pelo CNPq, MCT e Fundação Araucária. Atualmente, profa. Cristina é responsável pela disciplina de Banco de Dados, tem vários artigos publicados e participa de diversos projetos de pesquisa dentre os quais destacam-se o projeto FragDW (Fragmentação dos Dados em Ambientes de Data Warehousing) e o projeto MIRVisIM (Mineração, Indexação, Recuperação e Visualização de Dados em Sistemas de Arquivamento de Imagens Médicas) financiado pela FAPESP. Suas principais áreas de interesse são data warehousing, bancos de dados heterogêneos, bancos de dados distribuídos e sistemas de informação geográfica.

## Sumário

PgSimilar: uma ferramenta open source para suporte a consultas por similaridade no PostgreSQL .....	01
Eduardo Borges (UFRGS), Carina Dorneles (UPF)	
XSDelta - uma ferramenta visual para comparação de esquemas XML .....	07
Augusto Belotto Perini (UFRGS), Vincent Nelson Kellers da Silveira (UFRGS), Renata de Matos Galante (UFRGS)	
GML Publisher: um framework para publicação de feições geográficas armazenadas em banco de dados relacional como GML .....	13
Fábio Feitosa (UFC), Fernando Lemos (UFC), Vânia Vidal (UFC)	
Postgres-REP: uma ferramenta para configuração de esquemas de replicação em bancos de dados .....	19
Daniel Vanin Macedo (UCS), José Maurício Carré Maciel (UCS)	
DESANA - uma ferramenta para extração de dados da Web considerando contextos fracos .....	25
Sérgio Afonso L. F. de Sá Júnior (UFAM), Daniel Oliveira (UFAM), Altigran Soares da Silva (UFAM)	
FastMapDB: uma ferramenta para visualização em SGBDRs com uma implementação interativa do algoritmo para detecção de agrupamentos k-medoid .....	31
Gabriel Fedel (USP/São Carlos), Humberto Razente (USP/ São Carlos), Agma Traina (USP/São Carlos), Caetano Traina Júnior (USP/São Carlos)	
LABRADOR: um sistema para consultas baseadas em palavras-chave sobre bancos de dados relacionais .....	37
Filipe de Sá Mesquita (UFAM), Dinorah Monteiro (UFAM), Altigran Soares da Silva (UFAM)	
BioProvider: uma ferramenta para o provimento de bases de biosseqüências .....	43
Maíra Ferreira de Noronha (PUC-Rio), Sérgio Lifschitz (PUC-Rio)	

# **XXI Simpósio Brasileiro de Banco de Dados**

## **III Sessão de Demos**

### **Comitê Avaliador**

Angelo Brayner (UNIFOR)  
Cristina Dutra de Aguiar Ciferri (USP/São Carlos)  
*Presidentes*

Angelo Perkusich (UFCG)  
Bernadete Faria Lóscio (UECE)  
Caetano Traina Júnior (USP/São Carlos)  
Carmem Satie Hara (UFPR)  
João Eduardo Ferreira (USP/IME)  
Marcelo Finger (USP/IME)  
Marco Antônio Casanova (PUC/Rio)  
Nabor Mendonça (UNIFOR)  
Nina Edelweiss (UFRGS)  
Renato Fileto (UFSC)  
Valéria Cesário Times (UFPE)  
Vânia Vidal (UFC)

### **Avaliadores Externos**

Angelo Augusto Frozza (UFSC)  
Robson do Nascimento Fidalgo (UFPE)

### **Comitê Organizador**

Raul Sidnei Wazlawick, UFSC <i>Coordenador Geral</i>	Ronaldo dos Santos Mello, UFSC <i>Coordenador SBB</i>
Daniele Wazlawick, UFSC <i>Coordenador Administrativo-Financeiro</i>	Eugênio C. E. Vieira, PGE-SC <i>Coordenador de Relações Institucionais</i>
Olinto José Varela Furtado, UFSC <i>Coordenador de Inscrições</i>	Frank Siqueira, UFSC <i>Coordenador de Publicações</i>
Ricardo Pereira e Silva, UFSC <i>Coordenador de Publicidade</i>	Maria Marta Leite, UFSC <i>Coordenador de Eventos</i>
Masanao Ohira, UFSC <i>Coordenador de Equipamentos</i>	Everton Luiz Vieira, UFSC <i>Webmaster</i>
Juliana Sellmer, UFSC	Rogério Cid Bastos, UFSC
Álvaro G. Rojas Lezana, UFSC	Rosvelter Coelho da Costa, UFSC
José Eduardo de Lucca, UFSC	Eratóstenes Araújo, SOFTEX
Heitor Blum S. Thiago, SUCE-SU-SC	Maria Angela Alves, CenPRA
Adoniran B. Cantelli, PowerSolutions	Gislaine Parra Freund, CTAI-SENAI-SC
Everaldo Artur Grahl, FURB	Mirela S.M.A. Notare, Faculdades Barddal
Fernanda dos Santos Cunha, UNIVALI	Mauro N. Madeira, UNISUL
Jolmar L. Hawerth, Univ. Estácio de Sá	

## **PgSimilar: Uma ferramenta *open source* para suporte a consultas por similaridade no PostgreSQL**

**Eduardo N. Borges<sup>\*1</sup>, Carina F. Dorneles<sup>2</sup>**

<sup>1</sup>Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)  
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

<sup>2</sup>Ciência da Computação - Instituto de Ciências Exatas e Geociências – ICEG  
Universidade de Passo Fundo – UPF

enborges@inf.ufrgs.br, dorneles@upf.br

**Abstract.** *Currently, the commercial database management systems (DBMS) do not support approximate queries directly. There is a need for technologies on DBMS to support this type of queries. This paper presents a tool which was developed to attach some functionality to the PostgreSQL DBMS. A similarity open source module named PgSimilar was developed to implement some similarity functions that handle similarity between strings. This module was attached to the DBMS to provide support for approximate queries. Furthermore, operators for similarity comparison for metric domains have also been developed.*

**Resumo.** *Atualmente, os sistemas gerenciadores de bancos de dados (SGBD) comerciais não suportam diretamente consultas aproximadas. Faz-se necessário um SGBD que implemente este tipo de pesquisa. O presente trabalho apresenta uma ferramenta desenvolvida no intuito de anexar funcionalidades ao SGBD PostgreSQL. Foi desenvolvido um módulo de similaridade open source denominado PgSimilar que implementa funções e procedimentos que tratam vários tipos de similaridade entre strings. Este módulo foi anexado ao SGBD para dar suporte às consultas aproximadas. Além disso, operadores de comparação por similaridade para domínios métricos foram desenvolvidos.*

### **1. Introdução**

Variações na representação das informações permitem diversas instâncias de um mesmo objeto do mundo real. Esta variação deve-se ao fato de que os dados armazenados são digitados por diferentes usuários ou gerados por diferentes aplicações. Os mecanismos tradicionais de pesquisa em banco de dados utilizam uma busca por comparação exata.

---

<sup>\*</sup> Este trabalho foi realizado na Fundação Universidade Federal do Rio Grande (FURG) durante o desenvolvimento do Trabalho de Conclusão de Curso do autor.

Em uma base de dados de cidades do Rio Grande do Sul, por exemplo, o nome de uma cidade pode estar armazenado de diversas formas como na Figura 1: Rio Grande, R. Grande, ou ainda Rio Graande. Todas as representações são consideradas objetos diferentes, mas fazem referência a uma mesma cidade. A recuperação da informação por meio de consultas exatas pode ser ineficiente neste caso. Portanto, torna-se necessária a utilização de um mecanismo de pesquisa mais elaborado com suporte a consultas imprecisas ou aproximadas, também conhecidas como consultas por similaridade [Jagadish 1995].

“Selecione as cidades que sejam, no mínimo, 70% similares a Rio Grande”

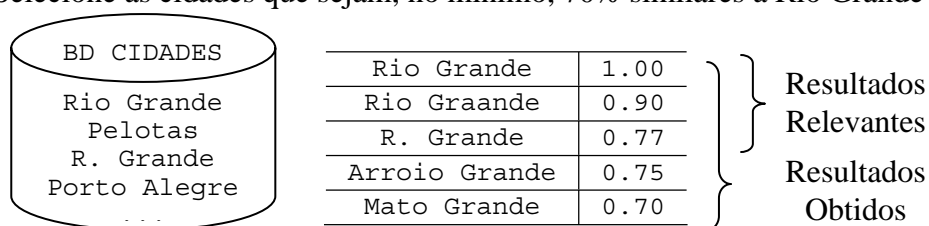


Figura 1. Consulta por similaridade aplicada a uma base de cidades.

Algoritmos de pesquisa baseados em funções de similaridade são aplicados nas bases de dados onde a consulta deve ser precisa o suficiente para que possa retornar resultados satisfatórios (os objetos que representam a cidade Rio Grande). Ao mesmo tempo a consulta deve ser flexível o bastante a fim de retornar todos, ou a maior parte, dos resultados relevantes (R. Grande neste exemplo).

Os resultados obtidos por meio da consulta representada na Figura 1 são formados por uma lista de cinco cidades, mas apenas três delas são resultados relevantes, os quais representam a cidade de Rio Grande. Nota-se que a consulta abrange mais cidades do que deveria. O limiar de 70% de similaridade adotado neste caso faz com que mais objetos façam parte do raio de busca da consulta. Definições adequadas de valores de limiar são discutidas na literatura [Stasiu et al. 2005] e não fazem parte do escopo do trabalho apresentado neste artigo.

O sistema de indexação de citações CiteSeer [Guiles et al. 1998] é um exemplo de base de dados cujas informações são originadas por meio da integração de várias outras bases. O sistema indexa literatura acadêmica no formato eletrônico (arquivos *Postscript* na *Web*) e identifica citações do mesmo artigo em diferentes formatos. Desta forma, a base de dados contém informações repetidas, mas que são diferentes na maneira como são representadas.

Atualmente, os sistemas gerenciadores de bancos de dados (SGBD) comerciais não suportam diretamente consultas aproximadas [Gravano et al. 2001]. Para que informações similares (dados representados de maneira diferente, mas com o mesmo significado) possam ser recuperadas através de uma consulta simples é necessário que o SGBD possua esta funcionalidade implementada em seu *engine* de consulta.

Visando preencher uma lacuna não abordada pelos SGBD no que tange ao tratamento de consultas aproximadas, o presente trabalho apresenta uma contribuição para um SGBD que permite solucionar problemáticas referentes à ineficiência das consultas por comparação exata. Esta contribuição é uma extensão para o SGBD comercial PostgreSQL [2005] denominada **PgSimilar**. Esta ferramenta é constituída de



um módulo de pesquisa por similaridade composto de um conjunto de funções de similaridade entre *strings*, operadores SQL e procedimentos para o suporte a consultas aproximadas.

Este artigo está dividido da seguinte forma: a seção 1 descreve a motivação e os objetivos da ferramenta. O funcionamento, o pacote de extensão do PostgreSQL e as funções de similaridade implementadas são apresentados na seção 2. E por fim, na seção 3, são descritas as considerações finais, conclusões e trabalhos futuros.

## 2. PgSimilar

O módulo **PgSimilar** foi projetado com o objetivo de possibilitar ao usuário a realização de consultas utilizando as funções de forma transparente, pois estas são executadas internamente no SGBD. Após a análise realizada pelo Analisador Léxico e Sintático do PostgreSQL, o Sistema de Reescrita analisa semanticamente a consulta. Nesta etapa, o SGBD traduz as chamadas das funções do módulo **PgSimilar** e reescreve a árvore de análise. Após, são verificadas as regras no catálogo do sistema e é elaborado um plano de execução otimizado. Os dados são buscados no banco através do Processador de Consultas e exibidos ao usuário através da interface de conexão.

Na primeira vez que uma função definida no **PgSimilar** é chamada em uma sessão, o carregador dinâmico carrega o arquivo objeto na memória para que a função possa ser chamada. O arquivo objeto carregado dinamicamente é mantido em memória após a primeira carga, ou seja, a primeira utilização de uma das funções contidas no código objeto. Chamadas posteriores das funções somente envolvem acessos à tabela de símbolos. Isto ocasiona um ganho no desempenho pelo fato das funções já estarem contidas na memória. A Figura 2 exhibe os passos da execução de uma consulta aproximada utilizando funções do **PgSimilar**.

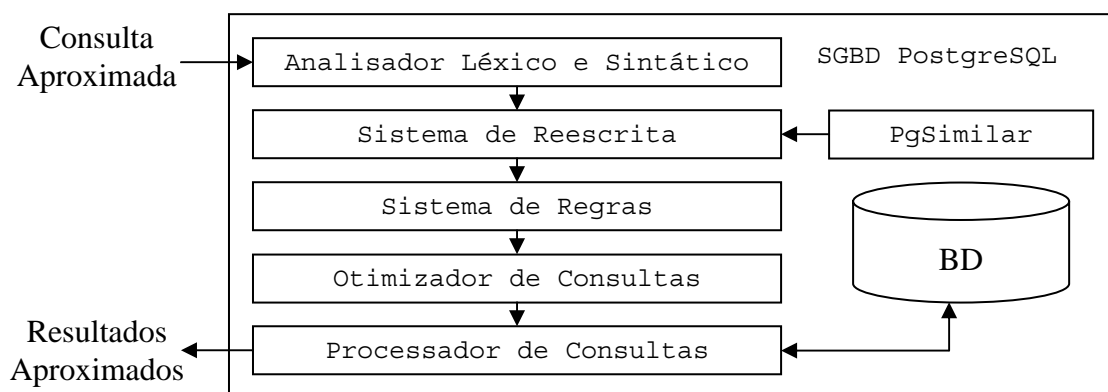


Figura 2. Execução de uma consulta utilizando as funções do PgSimilar.

### 2.1. Pacote

A ferramenta desenvolvida consiste em um módulo de extensão composto por vários arquivos organizados através da estrutura representada na Figura 3. A seguir serão descritas as funções de cada arquivo do pacote.

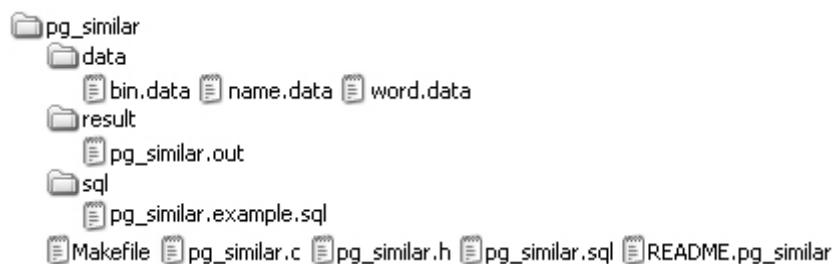


Figura 3. Estrutura de arquivos do pacote PgSimilar.

- `pg_similar.example.sql`: conjunto de testes que abrange todas as funções de similaridade do pacote. Neste *script*, as consultas fazem referência aos dados de teste contidos nos arquivos `bin.data`, `name.data` e `word.data`.
- `pg_similar.out`: contém a saída esperada para a execução dos testes.
- `pg_similar.h`: arquivo de biblioteca que contém os cabeçalhos das funções de similaridade e funções auxiliares.
- `pg_similar.c`: código-fonte do software que contém os procedimentos e funções de similaridade que tratam de vários tipos de similaridade entre *strings*.
- `pg_similar.sql`: *script* para instalar o pacote.
- `Makefile`: *script* para compilar o pacote. Faz uso do `Makefile.global` do código-fonte do SGBD PostgreSQL.
- `README.pg_similar`: arquivo de ajuda do software.

A instalação é simples. O pacote pode ser encontrado no endereço eletrônico do Instituto de Informática da UFRGS [http://www.inf.ufrgs.br/~enborges/pg\\_similar.zip](http://www.inf.ufrgs.br/~enborges/pg_similar.zip). Basta o usuário descompactar o pacote na pasta de contribuições do PostgreSQL, compilar o código-fonte e executar o *script* `pg_similar.sql` sobre uma base de dados `dbname`.

```
psql -d dbname -f /usr/local/pgsql/share/contrib/pg_similar.sql
```

No desenvolvimento das funções de similaridade anexadas ao PostgreSQL foi utilizada uma convenção de chamada para as funções em C denominada versão 1. Esta convenção faz uso de macros definidas nas bibliotecas do SGBD. As macros são utilizadas no intuito de otimizar a legibilidade, manutenibilidade e inteligibilidade do código fonte. Além disso, suprimem a maior parte da complexidade da passagem de argumentos e resultados.

```
PG_FUNCTION_INFO_V1(setlimit);
Datum setlimit(PG_FUNCTION_ARGS)
{
    float4 limit = PG_GETARG_FLOAT4(0);
    if (limit < 0 || limit > 1.0)
        elog(ERROR, "Should be between 0 and 1");
    pgsimlimit = limit;
    PG_RETURN_FLOAT4(pgsimlimit);
}
```

Figura 4. Uso de macros no desenvolvimento das funções.

Na Figura 4 é definida a função que altera o limiar de similaridade adotado nas consultas. O acesso aos argumentos é realizado utilizando a macro `PG_GETARG_X()` correspondente ao tipo de dado do argumento. A macro recebe como parâmetro o número do argumento da função, contado a partir de 0. O resultado é retornado utilizando a macro `PG_RETURN_X()` para o tipo de dado do parâmetro.

**Tabela 1. Funções de gerenciamento de limiar.**

Função	Descrição
<code>pg_similar_limit()</code>	Retorna o limiar de similaridade atual
<code>pg_similar_setlimit (real)</code>	Atribui um valor ao limiar de similaridade adotado nas consultas subseqüentes

## 2.2. Funções de Similaridade

A avaliação da similaridade entre objetos de um mesmo domínio (*strings*) é realizada por meio de funções baseadas na diferença entre palavras. Alguns exemplos destas funções são Hamming, Levenshtein [Levenshtein 1966], Jaro [Jaro 1989], Jaro-Winkler [Winkler 1990] e Carla [Mergen & Heuser 2005]. Conforme Cohen [et al. 2003], tais funções são utilizadas nas mais diversas áreas, como: checagem de significado, reconhecimento de pronúncia, análise de DNA, detecção de plágios, entre outras.

**Tabela 2. Operadores de similaridade propostos.**

Função de Similaridade	Operador Binário
<code>pg_similar_hamming (text, text)</code>	Não Disponível
<code>pg_similar_levenshtein (text, text)</code>	<code>~=</code>
<code>pg_similar_jaro (text, text)</code>	Não Disponível
<code>pg_similar_jarowinkler (text, text)</code>	<code>~~=</code>
<code>pg_similar_carla (text, text)</code>	<code>==</code>

Foram implementadas as funções de similaridade entre *strings* citadas, funções de gerenciamento de limiar (*threshold* ou *limit*) e operadores de similaridade visando o suporte a consultas por abrangência (*Range Query*). A Tabela 1 descreve as funções de limiar. Já a Tabela 2 apresenta os operadores e as respectivas funções.

```
SELECT pg_similar_setlimit(0.7);
```

Consulta 1:

```
SELECT nome, pg_similar_levenshtein(nome, 'rio grande') as similaridade
FROM cidades
WHERE nome ~= 'rio grande'
ORDER BY similaridade DESC, nome;
```

Consulta 2:

```
SELECT nome, pg_similar_hamming(nome, 'rio grande') as similaridade
FROM cidades
WHERE pg_similar_hamming(nome, 'rio grande') > pg_similar_limit()
ORDER BY similaridade DESC, nome;
```

**Figura 5. Exemplos de consultas.**

Na Figura 5 é exemplificado o uso de algumas das funções contidas no módulo **PgSimilar**. A primeira consulta realiza a busca apresentada na Figura 1. Já a segunda consulta mostra como é possível realizar a mesma busca utilizando a função Hamming, a qual opera somente palavras de tamanho igual. O operador correspondente não foi

implementado, pois dificilmente as palavras de uma coluna possuem a mesma largura. Já o operador correspondente à função Jaro não foi implementado porque a função Jaro-Winkler retorna resultados mais significativos por tratar-se de uma otimização deste algoritmo.

### **3. Conclusões e trabalhos futuros**

Este trabalho visa preencher uma lacuna não abordada pelos SGBDs no que tange ao tratamento de consultas aproximadas. O desenvolvimento de uma contribuição para um SGBD permite demonstrar as facilidades e vantagens do uso da similaridade nestes sistemas e, além disso, solucionar problemáticas referentes à ineficiência das consultas por coincidência exata em certas bases de dados.

Novos tipos de dados e funções podem ser adicionados ao módulo de similaridade, aumentando a capacidade do sistema sem comprometer sua robustez e velocidade ou funcionalidade das consultas.

Métodos e índices de acessos adequados a cada uma das funções de similaridade devem futuramente ser acrescentados a esta proposta, pois o tempo de comparação entre dois objetos pode ser muito alto. Também deve ser anexado à ferramenta o suporte a consultas por similaridade aos k vizinhos mais próximos (kNNQ), tornando o **PgSimilar** uma ferramenta completa para a similaridade de texto em SGBD.

### **Referências**

- Cohen, W., Ravikumar, P., and Fienberg S. (2003): “A Comparison of String Distance Metrics for Name-Matching Tasks” in IIWeb 2003: 73-78.
- Giles, C.L., Bollacker, K., and Lawrence, S. “CiteSeer: An Automatic Citation Indexing System”, Digital Libraries 98: Third ACM Conf. Digital Libraries, ACM Press, New York, 1998, pp. 89-98.
- Gravano, L., Ipeirotis, P. G., Jagadish, H. V., Koudas, N., Muthukrishnan, S., Pietarinen, L., and Srivastava, D. (2001). “Using q-grams in a DBMS for Approximate String Processing”. IEEE Data Eng. Bull. 24(4): 28-34.
- Jagadish, H. V., Mendelzon A. O., and Milo T. “Similarity-Based Queries”, Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California. ACM Press. pages: 36-45.
- Jaro, M. A. (1989). Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. Journal of the American Statistical Association 84:414–420.
- Levenshtein, I. V. (1966). Binary codes capable of correcting deletions, insertions and reversals. Cybernetics and Control Theory, 10(8):707–710.
- Mergen, S. L. S., Heuser, C. A. “Carla: Uma técnica para comparação de cadeias de caracteres”. Escola Regional de Banco de Dados (ERBD), 2005, Porto Alegre. p. 55-60.
- PostgreSQL (2005), “Sistema Gerenciador de Bancos de Dados PostgreSQL 8.0.3”, url: <http://www.postgresql.org/download>, acesso em outubro de 2005.
- Stasiu, R.K., Heuser, C.A., and Silva, R.: Estimating Recall and Precision for Vague Queries in Databases. CAiSE 2005: 187-200.
- Winkler, W. E. (1990). String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. Proceedings of the Section on Survey Research Methods, American Statistical Association. 354-359.

## XSDelta – Uma Ferramenta Visual para Comparação de Esquemas XML

Augusto Belotto Perini, Vincent Nelson Kellers da Silveira e  
Renata de Matos Galante

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)  
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{aperini, vincent, galante}@inf.ufrgs.br

**Abstract.** *This paper describes XSDelta, a graphical tool designed for comparison of XML schemas expressed through both DTD and XML Schema. By using a diff algorithm for detecting changes in XML documents, XSDelta exposes to the user all operations required to transform an original XML schema into its new version. XSDelta presents schemas textually, making modified parts stand out through distinct colors and identifying the schemas changes performed. As output, XSDelta generates XML files containing identified differences, assisting semi-structured databases administrators in the XML schemas evolution process management.*

**Resumo.** *Este trabalho apresenta o XSDelta, uma ferramenta gráfica capaz de comparar versões de esquemas XML definidos tanto em DTD quanto em XML Schema. Utilizando um algoritmo de detecção de diferenças em documentos XML, o XSDelta expõe ao usuário todas as operações envolvidas na transformação de um esquema original em sua nova versão. O XSDelta é capaz de exibir textualmente os esquemas, realçando através de cores distintas as partes modificadas e identificando as mudanças de esquema realizadas. Como saída, o XSDelta gera arquivos XML contendo as diferenças identificadas, auxiliando administradores de bancos de dados semi-estruturados na gerência do processo de evolução de esquemas XML.*

### 1. Introdução

A utilização de esquemas XML como um meio de definir e especificar regras de validação sobre as estruturas e conteúdo de documentos XML tornou a gerência da evolução de esquemas XML um foco de atenção por parte de administradores de bancos de dados semi-estruturados.

Manutenções em bases de dados semi-estruturadas que incluam a modificação de esquemas podem comprometer consultas e a validade de documentos XML [Guerrini 2005]. Essa evolução pode ser entendida como um processo natural de amadurecimento das estruturas de dados que descrevem as informações disponíveis aos sistemas computacionais, podendo ocorrer sob diversas circunstâncias, como refinamento de requisitos de aplicações XML, alteração na modelagem lógica de um banco de dados, eliminação de *bugs*, etc. A detecção da evolução no contexto de esquemas XML possibilita que os administradores de bases de dados semi-estruturadas compreendam as

modificações ocorridas, de modo que elas sejam propagadas aos documentos que referenciam tais esquemas, impedindo que se tornem inválidos [Guerrini 2005].

A Figura 1 é um exemplo de evolução de esquema (representado por uma árvore XML) onde o atributo “id” e o elemento “ano” são adicionados ao elemento “artigo”.

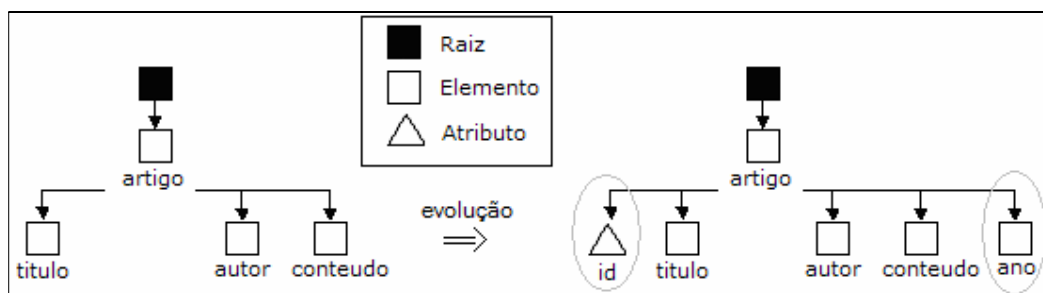


Figura 1. Exemplo de evolução de esquema

Na literatura há diversos algoritmos capazes de identificar diferenças em árvores XML. O XyDiff foi o algoritmo escolhido para implementar no XSDelta o processo de detecção de diferenças em função de sua ordem de complexidade, do conjunto de operações de *diff* suportadas e da implementação *open source* disponível. Cabe ressaltar que o XSDelta é projetado de maneira que o algoritmo escolhido possa ser facilmente substituído por outro de mesmo propósito que se mostre mais adequado segundo os critérios abordados na Seção 2.

O objetivo do XSDelta é detectar as diferenças entre duas versões de um esquema por meio de um algoritmo de detecção de diferenças em documentos XML. Esquemas definidos em XML *Schema* possuem sintaxe XML [W3C 2004A], sendo documentos bem formados e aptos ao processo de detecção de diferenças proposto. Esquemas DTD possuem sintaxe própria. Devido à simplicidade e limitações da DTD, o XSDelta possui a funcionalidade de conversão do formato DTD para o formato XML *Schema*, viabilizando a submissão de esquemas definidos em ambas linguagens ao mesmo processo de detecção de diferenças.

O XSDelta trata a transformação de um esquema em outro como a aplicação de sucessivas operações de modificação nas estruturas das suas árvores XML. As operações em questão incluem a inserção e remoção de nodos, atualização de valores dos nodos e movimentação de sub-árvores. O exemplo na Figura 1 indica a inserção de dois nodos na árvore XML. Além disso, o XSDelta exibe textualmente os esquemas, realçando através de cores distintas as partes modificadas, juntamente com a lista de operações de edição realizadas. Como saída, o XSDelta gera arquivos XML contendo as diferenças identificadas.

Este trabalho está dividido da seguinte maneira: A Seção 2 apresenta as principais características dos algoritmos de detecção de diferenças em documentos XML e justifica a escolha do XyDiff. A Seção 3 apresenta a arquitetura e as funcionalidades do XSDelta. Seção 4 encerra o trabalho com considerações finais, relacionando trabalhos futuros.

## 2. Algoritmos de detecção de diferenças em documentos XML

Diversos algoritmos surgiram para o cômputo de diferenças entre dados hierarquicamente estruturados, representados sob o formato de árvores. Como saída, eles retornam um *script*, também conhecido como *script delta*, que contém todas as operações para transformação de um documento em outro. Após comparar os esquemas, o XSDelta processa este *script delta* para apresentar as operações ao usuário.

No estudo de [Peters 2005] são comparados dezessete algoritmos que detectam diferenças em árvores XML, que variam quanto ao foco da detecção, vantagens e desvantagens sob diferentes cenários, otimizações e requisitos peculiares. Os itens comparados entre os algoritmos incluem o tratamento de árvores ordenadas ou não-ordenadas (se a ordem entre nodos irmãos é relevante ou não); a corretude (qualidade da saída gerada, o *script delta*), estando relacionada com a busca pelo *matching* ótimo entre nodos; o uso de memória e tamanho das entradas (que pode ser um limitante para certos algoritmos); o uso de semântica para a comparação de nodos, como utilização de chaves ou *id's*, proporcionando uma condição única para o casamento de elementos; tamanho da saída (tamanho dos *scripts delta* gerados, influenciado pelo suporte a determinadas operações de *diff*); suporte a operações especializadas como *move* e *copy*, garantindo *scripts* mais compactos e complexidade de tempo e de recursos.

### 2.1 XyDiff - Detecting Changes in XML Documents

O XyDiff [Cobéna 2002] mostrou-se adequado ao XSDelta, na medida em que suporta as operações de *diff* do tipo *insert*, *delete*, *update* e *move*, o que contribui para diminuir o tamanho do *script delta*, e que sua complexidade de execução é da ordem de  $O(n \log n)$ , adotando práticas que focam na otimização de uso de memória e tempo de processamento, e lida com árvores ordenadas (considera que cada nodo da árvore contém uma lista ordenada de nodos filhos). Em [ZHANG 1992] demonstra-se que a detecção de mudanças em árvores não-ordenadas é substancialmente mais difícil e custosa. Para o cômputo das diferenças e geração do *delta*, o XyDiff é dividido em cinco fases:

- Fase 1 - busca pelo casamento único entre nodos, por meio de atributos *ID*;
- Fase 2 - atribuição de assinaturas e pesos aos nodos e ordenamento de sub-árvores através de uma fila de prioridade;
- Fase 3 - procura por casamentos priorizando nodos e sub-árvores de maior peso;
- Fase 4 - otimização dos casamentos pela procura por nodos casados cujos pais possuam mesma assinatura, com varredura *bottom-up* e posterior varredura *top-down* das árvores;
- Fase 5 – reconhecimento das operações *insert*, *delete*, *update* e *move* e geração do *script delta*.

## 3. XSDelta - Uma Ferramenta Visual para Comparação de Esquemas XML

Esta seção descreve o XSDelta, uma ferramenta visual para comparação de esquemas XML desenvolvida para auxiliar administradores de bancos de dados semi-estruturados na gerência do processo de evolução de esquemas XML.

### 3.1. Arquitetura da Ferramenta

Os quatro casos de uso (Editar esquema, Converter esquema, Detectar diferenças e Gerar delta) ilustrados na Figura 2, identificados no momento da modelagem do XSDelta, encapsulam todas as funcionalidades da ferramenta (vide Seção 3.2).

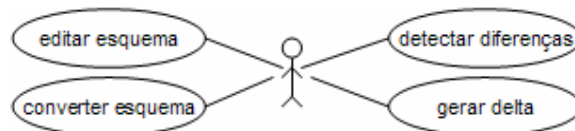


Figura 2. Diagrama de casos de uso do XSDelta

A linguagem de programação Java foi utilizada na implementação do XSDelta. Para a diagramação relativa a parte de análise e projeto foi utilizado o ArgoUML<sup>1</sup>, que é uma ferramenta *open source* para modelagem UML que roda sobre a plataforma Java. O ambiente de desenvolvimento utilizado foi o Eclipse 3.1<sup>2</sup>.

Para a fase de experimentação da ferramenta XSDelta estão sendo usadas bases de dados oriundas da DBLP<sup>3</sup> (*Digital Bibliography & Library Project*), entre outras, que modelam o domínio de artigos científicos em Ciência da Computação. Estas bases de dados<sup>4</sup> utilizadas na etapa de testes são importantes para o refinamento do XSDelta.

### 3.2. Funcionalidades do XSDelta

O XSDelta possui três funcionalidades principais: (i) conversão de esquemas DTD para esquemas XML *Schema*; (ii) processamento dos *scripts delta* para exibição visual das operações de evolução; e (iii) geração do arquivo *delta*. Duas perspectivas agrupam essas funcionalidades: esquema para o item (i) e evolução para os itens (ii) e (iii).

O fluxo básico de utilização do XSDelta está detalhado na Figura 3. Os esquemas XML são submetidos ao processo de detecção de diferenças. Esquemas que não são documentos XML (esquemas DTD) seguem o fluxo alternativo, descrito por setas tracejadas, para que sejam convertidos para o formato XML *Schema*.

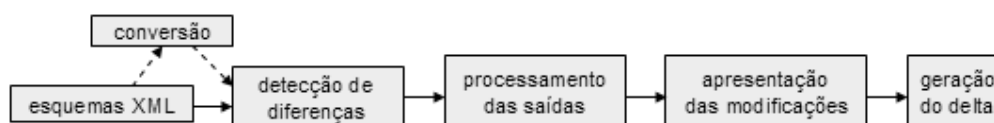


Figura 3. Fluxo básico do XSDelta

Após detectar as diferenças, o XSDelta processa as saídas do algoritmo escolhido para apresentar as modificações entre os esquemas, juntamente com a opção de geração do arquivo XML (geração do delta). O delta gerado pelo XSDelta é um documento XML. Esse documento contém os conteúdos dos esquemas comparados e as operações de evolução detectadas pelo algoritmo de *diff* escolhido.

<sup>1</sup> <http://argouml.tigris.org/>

<sup>2</sup> <http://www.eclipse.org/>

<sup>3</sup> <http://dblp.uni-trier.de/>

<sup>4</sup> Bases disponíveis em <<http://dblp.uni-trier.de/xml>>. Acesso em 03/05/2006.



### 3.2.1. Conversão de esquemas

A Figura 3 ilustra um esquema em formato DTD (lado esquerdo) sendo convertido para o formato XML Schema (lado direito). É através da perspectiva ESQUEMA que o XSDelta viabiliza a análise da evolução de esquemas DTD, pois, após serem convertidos para XML Schema, esquemas DTD tornam-se entradas válidas ao algoritmo de detecção de diferenças utilizado na perspectiva EVOLUÇÃO.

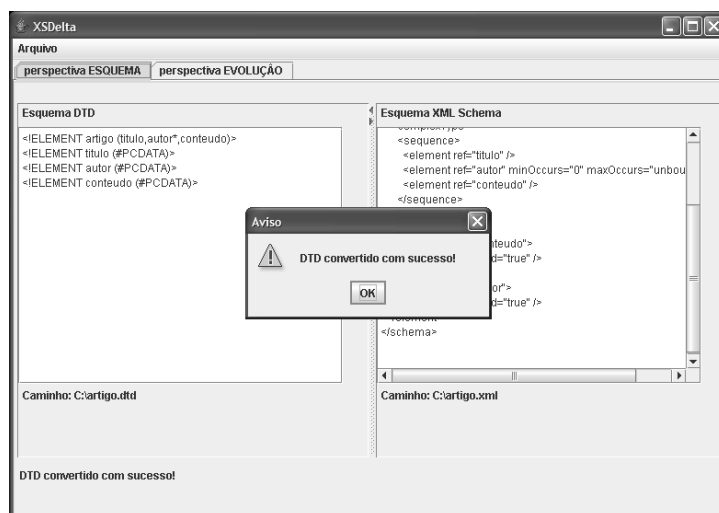


Figura 3. Perspectiva ESQUEMA

### 3.2.2. Visualização da evolução do esquema

A Figura 4 ilustra o resultado da comparação de um esquema com sua nova versão.

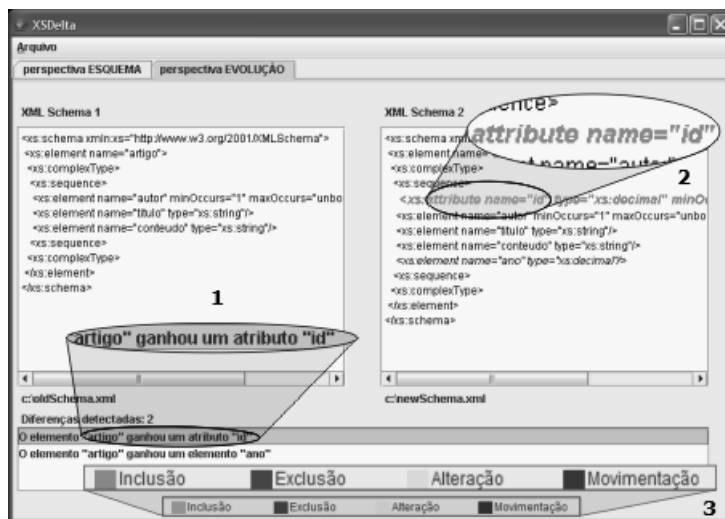


Figura 4. Perspectiva EVOLUÇÃO

Neste exemplo o XSDelta identificou as operações de inserção do novo atributo **id** e do novo elemento **ano**, ambas ao elemento já existente **artigo**. Todas as operações identificadas aparecem na lista de diferenças detectadas. O usuário do XSDelta pode seleccionar livremente as operações desta lista (ampliação 1) para que porções dos

conteúdos dos esquemas XML recebam destaques coloridos (ampliação 2). Na parte inferior da tela encontra-se a legenda com o significado das cores utilizadas no destaque das porções de interesse dos esquemas (ampliação 3).

As ampliações numeradas são recursos ilustrativos, que visam facilitar a identificação dos elementos na interface e não integram a implementação do XSDelta.

#### **4. Conclusões e trabalhos futuros**

Este trabalho apresentou o XSDelta, uma ferramenta visual capaz de descrever textual e graficamente modificações realizadas em esquemas definidos em DTD e *XML Schema*, no intuito de possibilitar aos administradores de bancos de dados semi-estruturados uma melhor compreensão a respeito do processo de evolução de esquemas XML.

Atualmente os módulos do XSDelta responsáveis por processar a saída do algoritmo de detecção de diferenças em documentos XML escolhido e por gerar o arquivo XML *delta* estão sendo testados e necessitam de refinamentos. O módulo processamento das saídas vem sendo projetado para que a integração de outros algoritmos de mesmo propósito seja facilitada. Esse módulo da ferramenta deve ser estendido para cada algoritmo que se deseje incluir suporte, pois eles não retornam o *script* com as operações em formato padrão. A ferramenta não se limita a comparar esquemas XML, podendo inclusive detectar diferenças em documentos XML de conteúdo, dada a natureza dos algoritmos estudados.

Há um trabalho de mestrado em Ciência da Computação no Instituto de Informática da UFRGS em andamento que aborda a propagação de modificações realizadas em esquemas XML para documentos XML, particularmente interessado nos resultados alcançados pelo XSDelta, pois seu primeiro passo consiste na identificação das modificações realizadas nos esquema XML. Este, e outros trabalhos relacionados, representam estímulos à melhoria dos resultados atualmente obtidos pelo XSDelta.

#### **5. Referências**

- Cobéna, G., Abiteboul, S. and Marian, A. (2002) "Detecting Changes in XML Documents", In: 18th IEEE International Conference on Data Engineering (ICDE), Washington, DC, USA. Proceedings... IEEE Computer Society, p. 41-52.
- Guerrini G., Mesiti M., Rossi D. (2005) "Impact of XML Schema Evolution on Valid Documents", In: 7th annual ACM international Workshop on Web Information and data Management (WIDM), Bremen, Germany. Proceedings... ACM Press p. 39-44.
- Peters, L. J. (2005) "Change Detection in XML Tress: a Survey", In: 3rd Twente Student Conference on IT, University of Twente, Enschede, Netherlands. June.
- W3C. (2004). "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C Recommendation, <http://www.w3.org/TR/2004/REC-xml-20040204/>, February.
- W3C. (2004). "XML Schema Part 0: Primer Second Edition", W3C Recommendation, <http://www.w3.org/TR/xmlschema-0/>, October.
- Zhang, K., Statman, R., Shasha, D. (1992) "On the Editing Distance between Unordered Labeled Trees", In: Information Processing Letters, 42(3), Amsterdam, The Netherlands. Proceedings... Elsevier North-Holland, Inc. p. 133-139.

## **GML Publisher: Um Framework para Publicação de Feições Geográficas armazenadas em Banco de Dados Relacional como GML**

**Fábio Bezerra Feitosa, Fernando Cordeiro Lemos, Vânia Maria Ponte Vidal**

Departamento de Computação, Universidade Federal do Ceará

email: {fabiofbf, fernandocl, vvidal}@lia.ufc.br

**Resumo.** *O propósito da especificação WFS, proposta pelo consórcio OpenGis (OGC), é descrever operações de manipulação de dados geoespaciais codificados em GML. Servidores WFS publicam visões GML de feições geográficas armazenadas em fontes de dados de forma que o usuário pode consultar e atualizar tais fontes através do esquema da visão. Neste trabalho, propomos GML Publisher, um framework para publicação de dados geográficos armazenados em banco de dados relacional como GML. O framework proposto deverá atender à especificação WFS e solucionar limitações apresentadas por outras implementações abertas de servidores WFS. Desenvolvemos também um ambiente para facilitar publicação e manutenção de visões GML no GML Publisher.*

### **1. Introdução**

A missão do consórcio OpenGIS (OGC) [7] é definir soluções padrões e recomendações que garantam a interoperabilidade entre Sistemas de Informações Geoespaciais (SIGs) [1]. Entre as iniciativas do OpenGis estão a Geography Markup Language (GML) [8] e a especificação Web Feature Service (WFS) [9]. O propósito da especificação WFS é descrever as operações de manipulação de instâncias de feições geográficas codificadas em GML. GML é uma extensão de XML, que foi proposta pelo OGC como formato padrão para representação de dados geográficos na *Web*. Servidores que implementam a especificação WFS são chamados de Servidores WFS e têm como objetivo proporcionar a consulta, atualização, troca e transporte de dados geoespaciais. Dessa forma, o usuário pode consultar e atualizar as fontes de dados através de uma visão GML.

Nesse trabalho, apresentamos um *framework* para publicação de dados geográficos armazenados em banco de dados relacional como GML. O *framework* deverá atender às especificações WFS e deverá solucionar limitações apresentadas por outras implementações abertas de servidores WFS. Uma requisição WFS consiste de uma descrição de operação de consulta ou atualização de dados e é aplicada a uma ou mais visões GML. Dentre os cinco tipos de operações WFS, destacamos as operações *GetFeature* e *Transaction*, as quais permitem a consulta e atualização de visões GML.

O artigo está organizado como se segue. Na seção 2, apresentamos o framework proposto. Na seção 3, mostramos, de forma sucinta, o ambiente para publicação de Visões GML no GML Publisher. Na seção 4 e 5, mostramos um estudo de caso e os trabalhos relacionados, respectivamente. Na seção 6, apresentamos a conclusão.

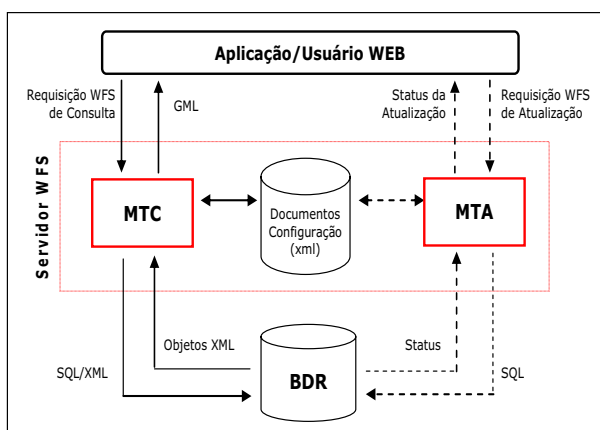


Figura 1 – Arquitetura do GML Publisher

```

<wfs:GetFeature outputFormat="GML2" ...>
<wfs:Query typeName="F_Posto">
  <wfs:PropertyName>nome</wfs:PropertyName>
  <wfs:PropertyName>endereco/cidade</wfs:PropertyName>
  <wfs:PropertyName>pluviometria</wfs:PropertyName>
  <wfs:PropertyName>geometria</wfs:PropertyName>
  <ogc:Filter>
    <ogc:PropertyIsLessThan>
      <ogc:PropertyName>endereco/cidade/area</ogc:PropertyName>
      <ogc:Literal>7000</ogc:Literal>
    </ogc:PropertyIsLessThan>
  </ogc:Filter>
</wfs:Query></wfs:GetFeature>

```

Figura 2 - Requisição WFS de Consulta W

## 2. Framework Proposto

### 2.1 Arquitetura

No *framework* proposto, os dados geográficos armazenados em banco de dados relacionais são publicados através de uma visão GML. Consultas e atualizações definidas sobre o esquema da visão GML são traduzidas em consultas e atualizações sobre o esquema do banco. Figura 1 mostra os principais componentes do GML Publisher. O Módulo Tradutor de Consultas (MTC) é responsável pela tradução de requisições WFS de consulta (operação *GetFeature*) definida sobre visões GML e o Módulo Tradutor de Atualização (MTA) é responsável pela tradução das requisições WFS de atualizações (operação *Transaction*) definida sobre as visões GML. As visões GML publicadas podem ser acessadas por aplicações *Web* através de requisições WFS utilizando o protocolo HTTP.

### 2.2 Tradução de Requisições WFS de Consulta no GML Publisher

Uma requisição WFS de consulta contém uma ou mais consultas, cada uma encapsulada em um elemento `<wfs:Query>`. No GML Publisher, como mostrado na Figura 1, cada elemento `<wfs:Query>` de uma requisição WFS é traduzido em uma consulta SQL/XML [11] definida sobre o esquema do banco de dados. Os resultados das consultas são em seguida compilados em um único documento GML o qual é enviado ao usuário. Com a introdução do tipo de dados XML e do padrão SQL/XML como parte do SQL:2003 [3], usuários podem utilizar as funções de publicação do SQL/XML para definir consultas que exportam dados relacionais no formato XML. A vantagem dessa abordagem está na capacidade do SGBD de processar as consultas SQL/XML juntamente com as cláusulas SQL. A Figura 2 mostra um exemplo de requisição WFS de consulta sobre a visão GML *F\_Posto* (Figura 4).

No enfoque proposto, uma visão GML é especificada através do esquema da visão e de um conjunto de assertivas de correspondência [13], as quais especificam o mapeamento entre o esquema da visão e o esquema do banco de dados relacional. Baseado nas assertivas de correspondência da visão, o GML Publisher traduz consultas WFS em consultas SQL/XML de forma eficiente e correta. Em [13] apresentamos o algoritmo que realiza a tradução de requisições WFS em consultas SQL/XML

### 2.3 Tradução de Requisições WFS de Atualização no GML Publisher

Atualizações sobre visões GML são definidas através da operação *Transaction*. Uma requisição WFS de atualização contém uma ou mais atualizações encapsuladas nos elementos `<wfs:Insert>`, `<wfs:Update>` ou `<wfs>Delete>`. No GML Publisher, como mostrado na Figura 1, uma requisição WFS de atualização é traduzida em atualizações definidas sobre o esquema do banco de dados. Após processar as atualizações no banco, o resultado sobre o sucesso ou não de cada atualização é retornado ao usuário. No enfoque proposto, a tradução das requisições WFS de atualização é realizada baseada nas assertivas de correspondência da visão GML. Os algoritmos usados na tradução de requisições WFS de atualização são similares aos apresentados em [6].

### 2.4 Acessando o GML Publisher através da Web

O GML Publisher é disponibilizado como uma aplicação Web, de forma que seus serviços podem ser acessados por um cliente através de requisições HTTP. Essas requisições têm o seguinte formato: `http://host/path?REQUEST=operation&param=value`, onde `host` denota o endereço do servidor da aplicação Web que hospeda o GML Publisher, `path` é o caminho da aplicação GML Publisher na aplicação Web e `REQUEST` é o parâmetro da requisição que indica a operação (`operation`) submetida ao GML Publisher. O nome (`param`) e o valor (`value`) para o segundo parâmetro da requisição variam de acordo com a operação. Dentre as cinco operações descritas na especificação WFS, as implementadas pelo GML Publisher são: (i) *DescribeFeatureType*: requisita a descrição do esquema da visão GML publicada; (ii) *GetCapabilities*: requisita os nomes das visões GML publicadas, e ações (consultas e/ou atualizações) suportadas por estas; (iii) *GetFeature*: descreve operações de consulta sobre visões GML; e (iv) *Transaction*: descreve operações de atualização sobre visões GML publicadas.

## 3. Ambiente para Publicação de Visões GML no GML Publisher

A publicação de uma visão GML no GML Publisher não é uma tarefa simples, pois envolve a manipulação de vários arquivos XML de configuração, exigindo do administrador conhecimentos avançados de XML e XML Schema. Assim, desenvolvemos **FBA** (Feature-By-Assertion), um ambiente visual para facilitar a publicação e a administração das visões GML publicadas no GML Publisher. O processo de publicação compreende os seguintes passos: (1) Primeiro, o administrador registra o SGBD que irá prover os dados; (2) Em seguida, o administrador seleciona uma tabela de feição ou tabela pivô e o **FBA** publica automaticamente a visão GML baseado no esquema da tabela pivô e possíveis tabelas relacionadas; (3) O administrador então poderá incluir ou alterar novos mapeamentos entre o esquema da visão GML e o esquema do banco de dados; (4) Por fim, são gerados os arquivos de configuração que publicam a visão GML no GML Publisher. **FBA** é uma adaptação da ferramenta DFP [14] que foi desenvolvida para o Degree WFS [2].

## 4. Estudo de Caso

Considere o esquema relacional POSTOS mostrado na Figura 3. *Posto\_rel* é uma tabela de feição, cujo dado geográfico está representado pelo atributo geométrico `geom_point`. Suponha que o projetista publique a visão GML *F\_Posto* que exporta instâncias do tipo *TPosto* cujo esquema é mostrado na Figura 4. Abaixo mostramos os passos de uma tradução de consulta e de atualização no GML Publisher.

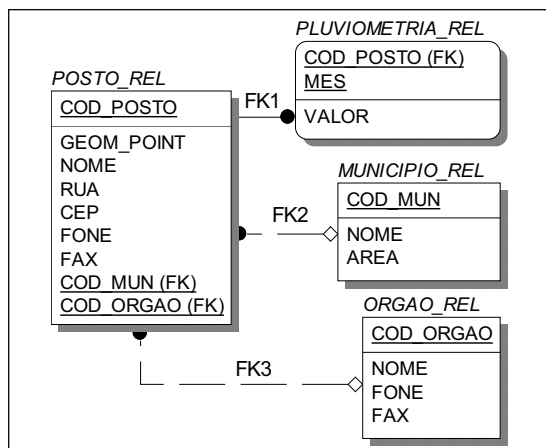


Figura 3 – Esquema do Banco

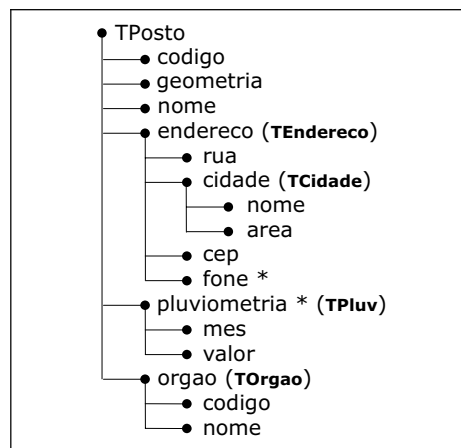


Figura 4 – Visão GML F\_Posto

```
SELECT XMLELEMENT("gml:featureMember",
XMLELEMENT("F_Posto",
XMLFOREST(P.NOME AS "nome"),
(SELECT XMLELEMENT("cidade",
XMLFOREST(M.NOME AS "nome"),
XMLFOREST(M.AREA AS "area") )
FROM MUNICIPIO_REL M,
WHERE P.COD_MUN = M.COD_MUN),
(SELECT XMLAGG(
XMLFOREST(
XMLFOREST(PL.MES AS "mes"),
XMLFOREST(PL.VALOR AS "valor")
AS "pluviometria") )
FROM PLUVIOMETRIA_REL PL
WHERE P.CODPOSTO = PL.CODPOSTO),
XMLFOREST(
SDO_UTIL.TO_GMLGEOMETRY(P.GEOM_POINT)
AS "geometria")
) )
FROM POSTO_REL P, MUNICIPIO_REL M
WHERE P.CODMUN = M.CODMUN
AND M.AREA < 7000
```

Figura 5 - Consulta SQL/XML Q

```
<wfs:FeatureCollection
xmlns=http://www.server.com/myns...>
<gml:boundedBy>
<gml:Box srsName="http://www.opengis.net/...">
<gml:coordinates>-4.1,-40.666,-3.85,-39.23
</gml:coordinates></gml:Box></gml:boundedBy>
<gml:featureMember>
<F_Posto>
<nome>Paramoti</nome>
<cidade><nome>Paramoti</nome><area>6910</area>
</cidade>
<pluviometria><mes>02</mes><valor>95.9</valor>
</pluviometria>
<pluviometria><mes>03</mes><valor>150</valor>
</pluviometria>
<geometria><gml:Point srsName="EPSG:4326">
<gml:coordinates cs="," decimal="." ts=" ">
-4.1,-39.233</gml:coordinates>
</gml:Point></geometria>
</F_Posto></gml:featureMember>
<gml:featureMember><F_Posto>
<nome>Serragem</nome>
...
</F_Posto></gml:featureMember>
</wfs:FeatureCollection>
```

Figura 6 – Documento GML Resultante

```
<wfs:Transaction>
<wfs:Insert>
<F_Posto><codigo>295</codigo>
<geometria>
<gml:Point srsName = '4326'>
<gml:coordinates>-7.183,-40.01</gml:coordinates>
</gml:Point></geometria>
<nome>Brejinho</nome>
<endereco>
<rua>Rua São Jose, 356</rua>
<cidade>
<codigo>2301307</codigo>
<nome>ARARIPE</nome></cidade>
<cep>60000000</cep>
<fone>885315284</fone> <fone>885315385</fone>
</endereco>
<pluviometria><mes>1</mes><valor>84.5</valor></pluviometria>
<pluviometria><mes>2</mes><valor>110</valor></pluviometria>
<orgao><codigo>4</codigo><nome>DNOCs</nome></orgao>
</F_Posto></wfs:Insert></wfs:Transaction>
```

Figura 7 - Requisição WFS de Atualização A

```
INSERT INTO POSTO_REL
(CODPOSTO, GEOM_POINT, NOME ,RUA,
CEP, FONE, FAX, CODMUN, CODORG)
VALUES
(295,
MDSYS.SDO_GEOMETRY( 2001, NULL,
MDSYS.SDO_POINT_TYPE(
-7.183, -40.01, NULL),
NULL, NULL)),
"Brejinho", "R. São Jose, 356",
"60000000", "885315284",
"885315385", 2301307, 4);

INSERT INTO PLUVIOMETRIA_REL
(CODPOSTO,MES,VALOR)
VALUES (295, 1, 84.5);

INSERT INTO PLUVIOMETRIA_REL
(CODPOSTO,MES,VALOR)
VALUES (295, 2, 110);
```

Figura 8 – Atualizações sobre o banco

- **Exemplo de Tradução de Consulta no GML Publisher**

Considere a requisição WFS de consulta **W** mostrada na Figura 2, a qual obtém o nome, dados da cidade, pluviometrias e a geometria dos postos das cidades com área menor que 7000 m<sup>2</sup>. O processamento dessa consulta no GML Publisher é realizado nos seguintes passos: (i) O elemento `<wfs:Query>` da consulta **W** é traduzido pelo **MTC** na consulta SQL/XML **Q** mostrada na Figura 5; (ii) A consulta **Q** é executada pelo SGBD; (iii) Com o resultado da consulta **Q**, o **MTC** monta o documento GML mostrado na Figura 6 e envia ao usuário.

- **Exemplo de Tradução de Atualização no GML Publisher**

Considere a requisição WFS de atualização **A** (Figura 7), a qual solicita a inserção de uma nova feição na visão GML F\_Posto. O processamento dessa requisição no GML Publisher é realizado nos seguintes passos: (i) A requisição **A** é traduzida pelo **MTA** nas atualizações sobre o esquema do banco de dados mostradas na Figura 8; (ii) As atualizações são executadas pelo SGBD e os estados de sucesso ou não são enviados ao GML Publisher; (iii) Com base nos estados das atualizações no banco, o **MTA** monta o documento GML de resposta com o estado da atualização da requisição **A** e envia ao usuário.

## 5. Trabalhos Relacionados

Já existem disponíveis no mercado algumas implementações da especificação WFS [2][10][5][4]. Dentre os trabalhos citados, destacamos o Degree [2] por ser o projeto de software livre que melhor implementa a especificação WFS. No Degree WFS, uma visão GML é publicada através de um arquivo de mapeamentos próprio, o qual especifica as correspondências entre os elementos da visão e os respectivos atributos na fonte de dados. No nosso uso do Degree WFS identificamos as seguintes limitações: (i) Propriedades de tipo complexo devem ter correspondência direta com uma tabela. Por exemplo, não seria possível definir o mapeamento para a propriedade endereço da visão F\_Posto, apesar de rua, cidade e cep possuírem correspondências com *POSTO\_REL*; (ii) Propriedades de tipo complexo distintas que são mapeadas em uma mesma tabela devem ter tipos iguais; (iii) Não permite especificar mapeamentos envolvendo chaves estrangeiras compostas. (iv) Não permite definir expressões de caminho em uma requisição WFS de consulta. Por exemplo, a expressão endereço/cidade na consulta da Figura 6 não é válida no Degree.

O GML Publisher, além de resolver as limitações acima citadas, apresenta um processamento de consultas mais eficiente uma vez que uma requisição WFS é traduzida em uma única consulta SQL/XML, enquanto que no Degree uma requisição WFS de consulta pode ser reformulada em várias consultas SQL. Não temos conhecimento de nenhuma outra implementação WFS que use o padrão SQL/XML. A vantagem dessa abordagem está na capacidade do SGBD de processar as consultas SQL/XML juntamente com as cláusulas SQL e retornar o resultado já no formato XML.

## 6. Conclusão

Neste trabalho apresentamos um *framework* para Publicação de visões GML de feições armazenadas em Banco de Dados Relacional. No enfoque proposto uma visão GML é definida pelo esquema da visão e um conjunto de Assertivas de Correspondência [12]

que especificam como as instâncias do tipo da visão são sintetizadas a partir das tuplas da tabela pivô. As assertivas são usadas para gerar arquivos de configuração da visão, os quais permitem que a tradução de consultas e atualizações WFS possam ser realizadas de forma eficiente.

Foi desenvolvido também um ambiente para facilitar publicação e manutenção de visões GML no GML Publisher. A ferramenta de publicação permite que o usuário defina, através de uma GUI, as propriedades da visão e as suas assertivas, e então, são gerados automaticamente os arquivos de configuração da visão. No caso de modificações no esquema do banco de dados, a ferramenta é capaz de identificar as visões GML afetadas e solicitar ao usuário que redefina as assertivas da visão.

Como trabalho futuro, pretendemos estender o GML Publisher para bancos de dados que não suportem SQL/XML. Neste caso, as requisições de consulta serão traduzidas em procedimentos/funções proprietárias do SGBD.

## **Referências**

- [1] Bishr, Y.: *Overcoming the Semantic and Other Barriers to GIS Interoperability*. International Journal of Geographical Information Science, v. 12, n. 4, p. 299-314, 1998
- [2] Deegree. <http://deegree.sourceforge.net/>
- [3] Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.E. and Zemke, F., *SQL:2003 has been published*. In: ACM SIGMOD Record, v. 33, n.1, p. 119–126, 2004
- [4] GeoServer. <http://geoserver.sourceforge.net/html/index.php>
- [5] GO Loader. <http://www.snowflakesoft.co.uk/products/goloader/>
- [6] Oliveira, W. G. C.: *Atualização de Bancos de Dados Objeto Relacionais Através de Visões XML*. Dissertação de Mestrado em Ciência da Computação, Universidade Federal do Ceará, Setembro/2004.
- [7] OpenGIS Consortium. <http://www.opengis.org>
- [8] OpenGIS Consortium: *Schema for Geography Markup Language (GML)*. <http://www.opengis.org>
- [9] OpenGIS Consortium: *Web Feature Service Implementation Specification*. <http://www.opengis.org>
- [10] OS Master Map. <http://www.ordnancesurvey.co.uk/oswebsite/>
- [11] SQL/XML. <http://www.sqlx.org/> (visited on September 26th, 2005).
- [12] Vidal, V. M. P., Araujo, V. S., Casanova, M. A., *Towards Automatic Generation of Rules for the Incremental Maintenance of XML Views over Relational Data*. In: WISE 2005, Pages 189-202, November, 2005.
- [13] Vidal, V. M. P., Lemos, F. C., Feitosa, F. B.: *Translating WFS Query to SQL/XML Query*. In VII Brazilian Symposium on GeoInformatics, 2005
- [14] Vidal, V. M. P., Teixeira, G. M. L., Feitosa, F. B.: *Towards Automatic Feature Type Publication*. In VI Brazilian Symposium on GeoInformatics, 2004



## Postgres-REP: Uma Ferramenta para Configuração de Esquemas de Replicação em Bancos de Dados

Daniel Vanin Macedo<sup>1</sup>, José Maurício Carré Maciel<sup>1</sup>

<sup>1</sup>Universidade de Caxias do Sul (UCS) - Campus de Vacaria

Av. Dom Frei Cândido Maria Bampi, 2800 - CEP 95200-000 - Vacaria – RS – Brasil

dvmacedo@terra.com.br, jmmaciel@ucs.br

**Resumo.** O trabalho apresenta o Postgres-REP, uma ferramenta para configuração de esquemas de replicação assíncrona em bancos de dados homogêneos. A ferramenta oferece uma interface gráfica intuitiva e suporte a definição de critérios para fragmentação horizontal, vertical e mista dos dados a serem replicados aos sites participantes.

**Abstract.** The work presents the Postgres-REP, a tool for configuration of asynchronous replication schemas in homogeneous databases. The tool offers an intuitive graphic interface and supports the definition of criteria for mixed, vertical, and horizontal fragmentation of the data that will be replicated to the participant sites.

### 1. Introdução

A melhoria das tecnologias para transmissão de dados, assim como seu custo decrescente está incentivando cada vez mais organizações que possuem divisões dispersas geograficamente a desenvolverem aplicações com utilização de banco de dados distribuídos. Torna-se conveniente, nessas circunstâncias, o emprego da replicação para melhoria de desempenho e disponibilidade.

A grande maioria dos sistemas gerenciadores de bancos de dados comerciais ou de código fonte aberto oferecem recursos para replicação de dados. Para a realização deste trabalho foi escolhido o sistema gerenciador PostgreSQL [Postgres 2006] devido à sua disponibilidade em várias plataformas e conformidade com os padrões da indústria como ACID e SQL. Resultados semelhantes poderão ser obtidos com a utilização de outros sistemas em conformidade com os mesmos padrões.

O trabalho apresenta a ferramenta Postgres-REP que contribui na implementação de distribuições de dados através da provisão de um ambiente gráfico intuitivo, para configuração e gerenciamento de esquemas de replicação. A ferramenta oferece recursos para sincronização inicial dos esquemas de bancos de dados, replicação completa ou parcial, com atualizações assíncronas em ambientes homogêneos.

O trabalho está organizado como segue: A seção 2 discute as técnicas de fragmentação e replicação de dados. A seção 3 discute trabalhos relacionados ao tema. A seção 4 apresenta a ferramenta mostrando sua interface. Na seção 5 são apresentadas as considerações sobre os resultados obtidos e a indicação de trabalhos futuros.

## 2. Conceitos de Fragmentação e Replicação

Elmasri (2005) descreve bancos de dados distribuídos como sendo constituídos de um ou mais bancos de dados, logicamente inter-relacionados, distribuídos ao longo de uma rede de computadores. Cada banco de dados participante da distribuição é chamado de *site* e deverá dispor de um sistema gerenciador de banco de dados. Caso os *sites* possuam sistemas gerenciadores idênticos, a distribuição é homogênea, caso contrário, é heterogênea.

Segundo Silberschatz *et al* (1999), a fragmentação consiste em dividir um banco de dados em unidades lógicas chamadas fragmentos, que poderão ser armazenados em um ou mais *sites*. De acordo com o critério utilizado no particionamento do banco de dados a fragmentação pode ser classificada em horizontal ou vertical. Um fragmento horizontal de uma relação é formado por um subconjunto de determinadas tuplas desta relação, especificadas através de uma condição em um ou mais atributos da mesma. Um fragmento vertical é constituído por um subconjunto de atributos desta relação. O resultado da combinação destas duas técnicas é a fragmentação híbrida ou mista.

De acordo com Özsu e Valduriez (2001), a replicação consiste em alocar cópias de dados em um ou mais *sites*. Pode haver a replicação de todo o banco de dados, de determinados fragmentos, ou mesmo pode não haver replicação. Em bancos totalmente replicados, todos os *sites* contêm todos os dados. Em bancos parcialmente replicados, cada fragmento poderá estar presente em um ou mais sites. Por último, em bases não replicadas, cada fragmento estará presente em somente um *site*.

Buretta (1997) considera o tempo entre a atualização dos dados replicados para classificar o tipo de replicação como assíncrona ou síncrona. Gray *et al* (1996) e Kemme *et al* (2000) comparam em seus trabalhos as abordagens síncrona e assíncrona para replicação, destacando os principais problemas e propondo algumas soluções.

Na replicação assíncrona todas as modificações são propagadas para outros sites em um segundo momento, em uma transação separada, sendo que esta poderá ocorrer em segundos, minutos, horas ou até mesmo dias depois. Sua principal desvantagem é que este tipo de replicação tende a atrasar a detecção de conflitos entre as operações. Na replicação síncrona, se algum item de dado for modificado, essa modificação será imediatamente propagada a todos os sites que contêm uma cópia deste item.

A replicação com atualizações centralizadas ou *master-slave* é aquela que permite as atualizações em somente uma das réplicas chamada primária. Todas as operações de modificação são realizadas primeiramente nesta réplica para depois serem distribuídas para as demais réplicas. Quando a execução das atualizações é permitida em todas as réplicas, caracteriza-se a replicação com atualizações distribuídas ou *multimaster*. Esta última, por sua vez, requer atenção especial de forma a prevenir que atualizações replicadas de A para B sejam replicadas de volta para A e assim por diante.

## 3. Trabalhos relacionados

Devido a natureza de código fonte aberto do PostgreSQL, há várias ferramentas de replicação disponíveis, como as descritas por Gborg (2006), Command Prompt (2005), Pgfoundry (2006) e Wieck (2005), algumas ainda em estágio inicial não possuindo maturidade para utilização em ambientes de produção.

Pedroni (2006) e Johann (2005) comparam em seus trabalhos as principais ferramentas disponíveis para replicação de bancos de dados PostgreSQL. Uma síntese desta comparação pode ser observada no Quadro 1. A totalidade das ferramentas comparadas disponibiliza apenas uma interface em linha de comando, para configuração dos esquemas de replicação. Outra característica das mesmas é que não contemplam a propagação de modificações no esquema do banco de dados. Pedroni (2006) contribui estendendo o sistema gerenciador PostgreSQL de forma a oferecer este suporte.

**Quadro 1- Comparativo entre ferramentas**

	<b>Slony-I</b>	<b>Mammoth</b>	<b>Postgres-R</b>	<b>PGCluster</b>
<b>Técnica</b>	Master-Slave	Master-slave	Multi-master	Multi-master
<b>Tipo</b>	Assíncrona	Síncrona e Assíncrona	Síncrona	Síncrona
<b>Suporte a fragmentação</b>	Sim	Sim	Não	Não
<b>Mecanismo de Replicação</b>	Triggers	Log de Transação, envia as alterações através de mensagens	Replicador Manager, envia as alterações às réplicas	Replicação baseada em Log de Transações
<b>Propaga Modificações no Esquema</b>	Não	Não	Não	Não
<b>Interface</b>	Linha de Comando	Linha de Comando	Linha de Comando	Linha de Comando

Fonte: Adaptado de Pedroni (2006, pg.03).

#### **4. A ferramenta Postgres-REP**

Ao ser executada, a ferramenta exibe uma interface inicial, ilustrada na Figura 1, listando os esquemas de replicação já configurados, juntamente com opções para incluir, editar, excluir, ativar ou desativar, configurar o particionamento dos esquemas de replicação e determinar o intervalo de sincronização para as atualizações das réplicas.

A Figura 2 mostra a interface exibida ao selecionar-se as opções de inclusão e edição de *sites*, permitindo determinar o nome, a localização, o usuário e a senha dos sites participantes.

Na Figura 3, é apresentada a interface para determinação do intervalo de atualização das réplicas, na qual o usuário deverá selecionar um dos grupos de opções apresentados.

Por fim, a Figura 4 exibe a interface de configuração dos critérios de fragmentação a serem empregados na replicação na qual o usuário poderá selecionar quais tabelas e os seus respectivos atributos cujos valores deverão ser replicados.

Para o gerenciamento do processo de replicação, são empregados: mecanismos de bancos de dados ativos na forma de gatilhos, tabelas que armazenam os dados que serão replicados no momento de sincronização entre os servidores participantes da distribuição; e arquivos contendo dados sobre sites, sincronização, ativação e intervalos de atualização. Tais estruturas são geradas a partir das informações fornecidas pelo usuário na interface da ferramenta. A seguir, as mesmas são descritas de forma simplificada:

*gsq\_l\_nomeesquema* é a tabela acrescida ao banco de dados origem, utilizada para armazenar as operações de manipulação executados e que suprem as condições configuradas no esquema de replicação, sendo composta pelos campos: *data*, *sql* e *user\_name*.

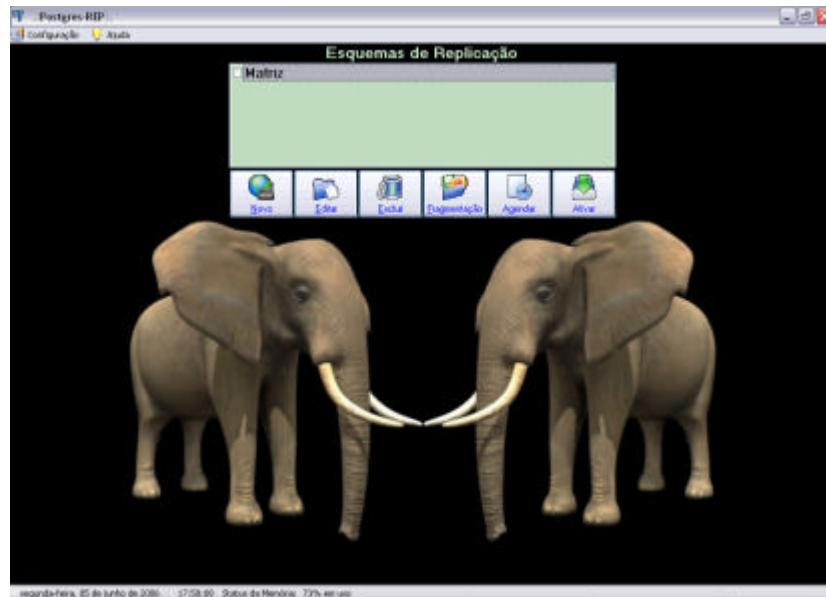


Figura 1. Interface inicial do Postgres-REP

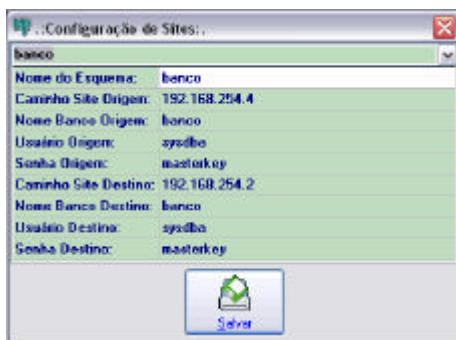


Figura 2. Diálogo de configuração de sites

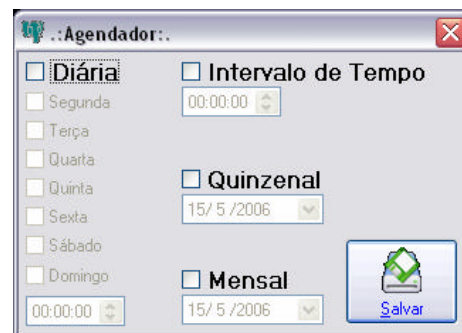


Figura 3. Diálogo para agendamento de replicação



Figura 4. Diálogo Configuração de fragmentação

*fv\_nomeesquema* é a tabela utilizada para guardar no banco de dados origem a configuração da fragmentação vertical, na qual é guardado o nome da tabela e quais os campos devem ser replicados. É composta pelos campos: *código*, *tabela* e *campo*.

*fh\_nomeesquema* é a tabela utilizada para guardar no banco de dados origem a configuração da fragmentação horizontal, na qual é guardado o nome da tabela e qual filtro deve ser respeitado para que a replicação seja executada. É composta pelos campos: *código*, *tabela* e *filtro*. Para cada tabela configurada no esquema de replicação, a ferramenta cria um gatilho *t\_sql\_nomeesquema\_nometabela* que será executado quando ocorrer uma inserção, alteração ou exclusão nesta tabela.

O gatilho *t\_sql\_nomeesquema\_nometabela* aponta para execução de uma função *f\_sql\_nomeesquema\_nometabela* que também é criada pela ferramenta, na qual são contidas todas as condições de fragmentação vertical e horizontal, que devem ser respeitadas para que a replicação siga corretamente as configurações feitas no esquema de replicação. Caso essas condições não sejam respeitadas não haverá gravação na tabela *gsql\_nomeesquema*, e caso contrário, essa função retornará o comando SQL montado pela função.

Quando um esquema de replicação é criado, e o banco de dados Destino não possui a estrutura de tabelas existente no banco de dados Origem, a ferramenta permite ao usuário recriá-la de maneira automática. Na escolha de quais tabelas devem ser replicadas, se a tabela selecionada não existir no banco de dados Destino, a ferramenta também permite a criação individual de cada tabela no banco de dados Destino.

Após a estrutura criada, se algum esquema estiver ativo para execução, a ferramenta fica monitorando os esquemas ativos em busca da condição de agendamento, que corresponda ao momento. Se algum esquema apresentar correspondência, a ferramenta localiza-o, conecta-se com o banco de dados que estão os dados a serem replicados (Origem) e conecta-se com o banco de dados que esses serão replicados (Destino). Após conectado, ela busca todos os registros existentes em *gsql\_nomeesquema* no *Origem* e executa um-a-um em *Destino*. Se for executado com sucesso um registro de *Origem* em *Destino*, esse registro é imediatamente apagado de *gsql\_nomeesquema* em *Origem*.

## **5. Considerações Finais e Trabalhos Futuros**

Date (2000) considera que organizações normalmente já são distribuídas logicamente e provavelmente também fisicamente. Conseqüentemente, os dados também estão normalmente distribuídos, porque cada unidade organizacional dentro da empresa manterá dados que são relevantes para sua própria operação. Para suprir essas necessidades, a tecnologia de sistemas de bancos de dados distribuídos vem ganhando mercado crescentemente.

Este trabalho teve como principal contribuição a criação de um ambiente que auxilie os usuários na configuração e implementação de esquemas de replicação de bancos de dados PostgreSQL. A ferramenta disponibiliza suporte para replicação *master-slave* assíncrona, fragmentação e propagação inicial de esquemas e instâncias.

Alguns aspectos, entretanto, poderão ser considerados para continuidade deste trabalho, entre eles o tratamento para replicação síncrona, suporte a esquemas de replicação *multi-master* e suporte assistido para resolução de conflitos de atualização.

## **Referências**

- BURETTA, M. Data Replication: tools and techniques for managing distributed information. New York: John Wiley e Sons, Inc, 1997. 360p.
- COMMAND PROMPT, Inc. Mammoth Postgresql + Replicator. Disponível em: <http://www.commandprompt.com/products/mammothreplicator/>. Acesso em: 30/09/2005.
- DATE, C. J. An introduction to database systems. 7.ed. Massachusetts: Addison-Wesley, 2000. 938 p.
- ELMASRI, Ramez; NAVATHE, Shamkant B. Sistemas de banco de dados. 4.ed. São Paulo: Addison-Wesley, 2005. 724 p.
- GBORG. The pgreplication Project -- PostgreSQL Replication. Disponível em: <http://gborg.postgresql.org/project/pgreplication/projdisplay.php>. Acesso em: 10/03/2006.
- GRAY, J., HELLAND, P., O'NEIL, P., SHASHA, D. The Dangers of Replication and a Solution, ACM SIGMOD, Montreal, Quebec, Canadá, 1996.
- JOHANN, E, KROTH E. Um Middleware para Replicação entre Banco de Dados Usando Sistemas de Comunicação em Grupo. Anais da I Escola Regional de Banco de Dados. Porto Alegre, RS, 2005.
- KEMME, B., ALONSO, G. Don't Be Lazy, Be Consistent: Postgres-R, A New Way to Implement Database Replication. Proceedings of the 26th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc, 2000 p. 134-143.
- ÖZSU, M. Tamer; VALDURIEZ, Patrick. Princípios de sistemas de bancos de dados distribuídos. Rio de Janeiro: Campus, 2001. 711 p.
- PEDRONI, M. Lisiane; Ribeiro, G. Helena. Replicação em Banco de Dados PostgreSQL. Anais da II Escola Regional de Banco de Dados. Passo Fundo, RS, 2006.
- PGFOUNDRY. PGCluster synchronous replication system multi-master for PostgreSQL. Disponível em: <http://pgfoundry.org/projects/pgcluster/>. Acesso em: 05/04/2006.
- POSTGRES. PostgreSQL Global Development Group. Disponível em: <http://www.postgresql.org>. Acesso em: 05/05/2006.
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. Sistema de banco de dados. 3.ed. São Paulo: Makron Books, 1999. 778 p.
- WIECK, J. Slony-I - A replication system for postgresql. Disponível em: <http://developer.postgresql.org/wieck/slony1/Slony-I-concept.pdf>. Acesso em: 30/09/2005.

## DESANA – Uma Ferramenta para Extração de Dados da Web Considerando Contextos Fracos

Sérgio A. L. F. de Sá Júnior<sup>1</sup>, Daniel Oliveira<sup>1</sup>, Altigran Soares da Silva<sup>1</sup>,

<sup>1</sup>Departamento de Ciência da computação  
Universidade Federal do Amazonas (UFAM)  
Manaus – AM – Brasil

sergioafonso@ufam.edu.br, alti@dcc.ufam.edu.br, dpo@dcc.ufam.edu.br

**Resumo.** Neste artigo apresentamos a ferramenta DESANA (Data Extraction and Structural ANALysis) que implementa uma nova abordagem para geração semi-automática de extratores de dados de páginas da Web. Com base em exemplos, padrões de extração são automaticamente inferidos para extrair valores de atributos mesmo quando a informação contextual nas páginas é pobre. Uma vez extraídos, os valores são agrupados para construir objetos complexos, de acordo como uma estrutura que é automaticamente inferida pela análise de suas posições nas páginas de origem. A ferramenta dispõe de diversos recursos que facilitam a especificação de exemplos através de refinamentos sucessivos e testes interativos dos extratores sendo gerados.

**Abstract.** In this paper we present DESANA (Data Extraction and Structural ANALysis), a tool that implements a new approach for semi-automatic wrapper generation. Based on examples, extraction patterns are automatically inferred for identifying attribute values, even when only poor contextual information is available. Once values are extracted, they are grouped to form complex objects according to a structure that is automatically inferred by analyzing their positioning on the page they occur. A number of features are available in the tool for assisting in example specification through successive refining and interactive testing of the wrappers being generated.

### 1. Introdução

Um número cada vez maior de páginas da Web apresenta dados *semi-estruturados*, os quais possuem uma estrutura que não é explicitamente declarada e que é irregular. Tipicamente, essas páginas são geradas dinamicamente a partir de consultas a um banco de dados, seguida pela aplicação de um *template* HTML que formata estes dados para apresentá-los de forma conveniente e intuitiva aos usuários. Exemplos destas páginas podem ser encontrados em sites de comércio eletrônico, anúncios, catálogos on-line, notícias, bibliotecas digitais etc.

A extração de dados de páginas HTML é normalmente realizada usando programas chamados *extratores* ou *wrappers*, que localizam e extraem dados das páginas e os convertem em um formato adequado tal como tabelas relacionais ou arquivos XML [Arasu and Garcia-Molina 2003, Baumgartner et al. 2001, Crescenzi et al. 2001, Liu et al. 2000, Laender et al. 2002, Sahuguet and Azavant 2001]. A partir daí esses dados podem ser processados de forma mais simples. Os dados de interesse são localizados através da identificação automática ou semi-automática de características do contexto

sintático em que eles se encontram, tais como sua formatação, seu posicionamento, seu tipo etc.

O desenvolvimento de extratores para páginas HTML apresenta grandes desafios. A maior parte oriunda do fato que o formato HTML é destinado à visualização pelo usuário, onde normalmente os itens de dados aparecem misturados aos elementos de marcação e hiperlinks, além de texto irrelevante. Com isso, boa parte dos dados de interesse nas páginas Web está inserida em contexto fraco. Dizemos que um contexto é pobre ou fraco quando ele é formado por delimitadores variáveis, cujo conteúdo muda conforme o dado de interesse. Além de variáveis esses delimitadores podem inclusive estar ausentes. Ao contrário, um contexto é forte quando é formado por delimitadores estáticos que sempre estão presentes junto ao dado de interesse.

Outro desafio para a geração de extratores está relacionado à estruturação dos dados. Em muitos casos, a estrutura dos dados apresenta variações e elementos opcionais. Tipicamente os objetos nas páginas da Web podem ser representados por tipos aninhados e variantes [da Silva et al. 2002].

Neste artigo apresentamos a ferramenta *DESANA* (*Data Extraction and Structural ANALysis*)<sup>1</sup>, que implementa uma nova abordagem para geração semi-automática de extratores onde, com base em exemplos fornecidos por um usuário, padrões de extração são automaticamente inferidos mesmo quando a informação contextual é pobre. Cada padrão de extração é usado para extrair os valores de um atributo. Uma vez extraídos, os valores destes atributos são agrupados para construir objetos complexos. A estrutura para este agrupamento é automaticamente inferida pela ferramenta com base na análise do posicionamento dos valores extraídos nas páginas de origem.

A ferramenta foi desenvolvida para permitir a validação prática e experimental do método de extração proposto em [Oliveira 2006]. Na ferramenta, os exemplos de treinamento selecionados pelo usuário são transformados em padrões de extração por meio de alinhamentos progressivos das strings que os compõem. Diferentemente de outras abordagens que requerem uma interação intensa com o usuário, seja pelo processo de especificação visual do extrator [Baumgartner et al. 2001, Sahuguet and Azavant 2001], seja pela definição de uma estrutura para os dados [Laender et al. 2002], *DESANA* requer apenas a seleção de alguns exemplos de valores dos atributos a serem extraídos. Essa tarefa é enormemente facilitada pela utilização de sua interface gráfica. A montagem dos objetos é automática e consiste na análise de padrões no posicionamento dos valores extraídos na página de origem. Esta análise é realizada pelo algoritmo *HotCycles* [da Silva 2002, Oliveira 2006] que é capaz de gerar estruturas aninhadas multi-nível, representando mais fielmente os objetos contidos nas páginas que as estruturas geradas por métodos completamente automáticos [Crescenzi et al. 2001, Arasu and Garcia-Molina 2003]. Em geral, estes métodos não conseguem distinguir o que é realmente relevante ao usuário refletindo mais a estrutura da página do que a dos objetos de interesse.

O restante do artigo está organizado da seguinte forma. Na Seção 2, apresentamos uma visão geral dos processos de geração de extratores e de extração de dados realizados

---

<sup>1</sup> Este nome é uma referência a Desâna, nome de uma etnia indígena cujos indivíduos habitam o extremo norte no alto Rio Negro, no estado do Amazonas.

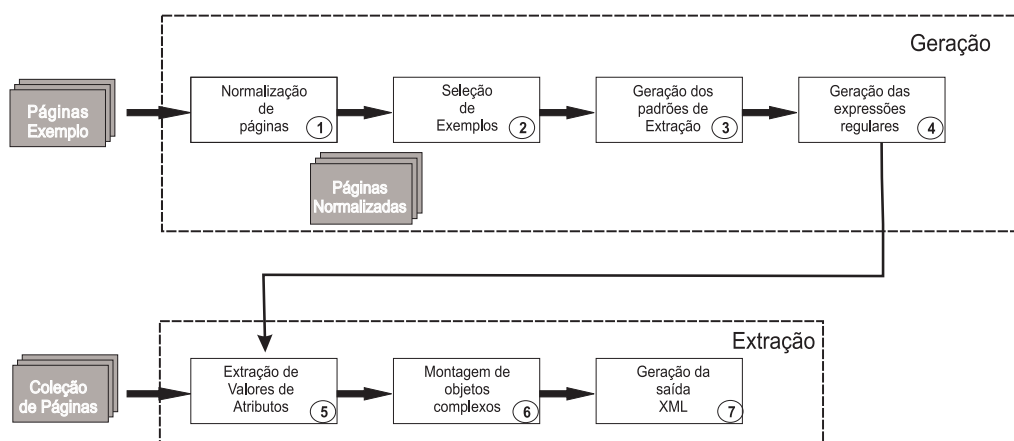


com a ferramenta. Na Seção 3 são brevemente descritos alguns dos principais recursos disponíveis na ferramenta. Finalmente, na Seção 4 apresentamos nossas conclusões e comentários finais.

## 2. Visão Geral da Ferramenta

Nesta seção apresentaremos uma visão geral de nossa ferramenta, descrevendo brevemente as etapas que constituem a abordagem de extração que ela implementa. Omitimos os detalhes dos algoritmos utilizados por considerarmos fora do escopo do artigo e por brevidade na apresentação. Descrições detalhadas destes algoritmos podem ser encontradas em [Oliveira 2006].

O funcionamento da ferramenta DESANA para a geração semi-automática de extratores baseada em exemplos pode ser resumida de acordo com os passos ilustrados na parte superior da Figura 1.



**Figura 1. Visão geral do funcionamento da Ferramenta DESANA**

A ferramenta realiza, no Passo 1, uma normalização prévia do arquivo a ser processado, transformando as páginas HTML em documentos XHTML bem formados. Essa normalização visa simplesmente otimizar o processo de geração de padrões de extração.

No Passo 2, o usuário fornece um conjunto de exemplos de treinamento dos valores dos atributos que deseja extrair da página. Para cada exemplo fornecido são automaticamente identificados um sufixo e um prefixo, os quais são em seguida transformados em listas de delimitadores. Após a identificação dos delimitadores, inicia-se o Passo 3, onde se busca padrões contextuais semelhantes para os exemplos através de alinhamentos progressivo das strings que formam estes exemplos. O objetivo é encontrar uma boa distribuição dos delimitadores que permitirá realizar a geração de um padrão de extração. Cada padrão de extração é gerado para extrair os valores de um atributo.

Depois de sucessivos alinhamentos progressivos, inicia-se o Passo 4, onde transformamos cada padrão de extração gerado em uma expressão regular que é então avaliada para se recuperar os dados nas páginas de interesse.

O Passo 4 encerra o processo de geração propriamente dito. Isso significa que o extrator gerado é constituído basicamente de um conjunto de expressões regulares, um para cada atributo a ser extraído. Depois de gerado, o extrator pode ser aplicado a qualquer

página similar às páginas usadas para seleção de exemplos pelo usuário. A extração dos valores dos atributos é realizada por um interpretador de expressões regulares comum que sequencialmente extrai a informação que casa com a expressão fornecida.

O processo de extração é ilustrado na parte inferior da Figura 1. No Passo 5, as expressões regulares são usadas para extrair os valores de cada atributo das páginas de entrada. Uma vez que todos os valores foram extraídos, inicia-se o Passo 6 onde ocorre a montagem dos objetos complexos de acordo com a estrutura que vai sendo inferida pelo algoritmo HotCycles [da Silva 2002, Oliveira 2006]. O algoritmo procura, examinando o posicionamento dos valores extraídos nas páginas, padrões que identifiquem elementos estruturais como listas ou tuplas e vai, dessa forma, reconstituindo os objetos. Finalmente, no Passo 7 temos os objetos devidamente montados e um arquivo XML com o resultado da extração é gerado, finalizando o processo de extração.

A ferramenta DESANA permite que execução sucessiva e cíclica dos processos de geração de extratores e extrações, de forma que é possível refinar iterativamente os extratores gerados. DESANA é totalmente desenvolvida em Java e utiliza somente componentes públicos.

### 3. Recursos da Ferramenta

Nesta seção apresentamos uma breve descrição dos principais recursos da ferramenta.

#### Interface Gráfica

Toda a operação da ferramenta se dá através de uma interface gráfica com a qual o usuário interage para realizar a geração dos extratores e possivelmente também a extração dos dados usando esse extrator. A Figura 2 ilustra uma seção de geração de extratores.

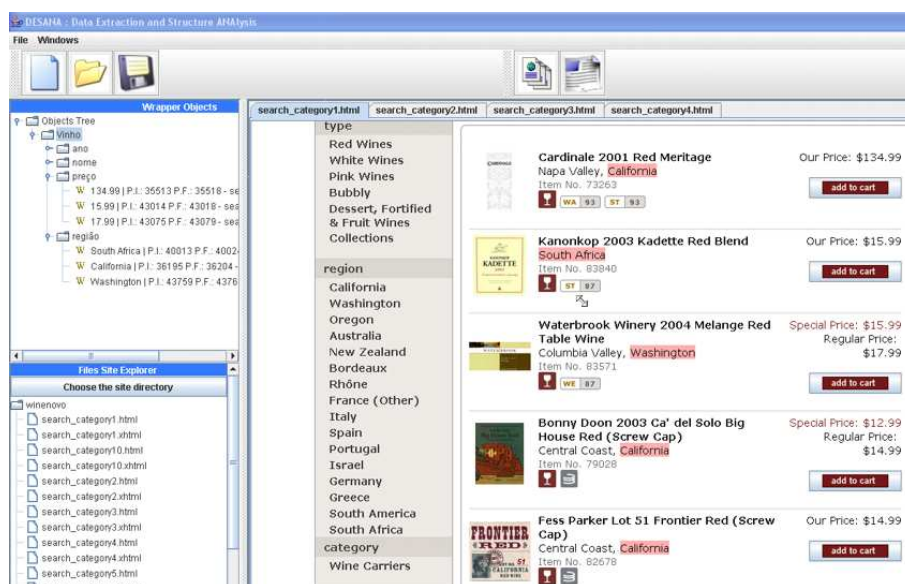


Figura 2. Interface Gráfica da Ferramenta

#### Páginas para seleção de exemplos

A interface gráfica da ferramenta inclui um browser HTML, o qual foi modificado para permitir a seleção de exemplos pelos usuários através de uma operação simples de copiar-e-colar. As strings selecionadas pelos usuários permanecem marcadas durante toda a

seção de trabalho. O browser permite a abertura de várias páginas através de abas. A possibilidade da seleção de exemplos de várias páginas contribui para aumentar a resiliência dos extratores gerados, já que existe a possibilidade de se ter exemplos com diferentes contextos em páginas diferentes.

### **Refinamento dos Extratores**

Para a geração de extratores, há casos em que o mínimo de dois exemplos é o suficiente para alcançar um bom nível de eficácia. Por outro lado, há casos onde é necessário selecionar vários exemplos que representem a variedade de contextos para os valores de um certo atributo. Para ajudar na tarefa de seleção de exemplos representativos, a DESANA implementa um processo de refinamento dos extratores. Ao selecionar pelo menos dois exemplos, o usuário pode executar a extração para o atributo em questão. Todos os dados extraídos nas páginas de exemplo são marcados no browser com cor diferente possibilitando a seleção de exemplos não extraídos até que se tenha uma quantidade suficiente de exemplos para a extração de todos os valores de um certo atributo. Enfatizamos a capacidade que tem a ferramenta de gerar extratores para valores que ocorrem em contexto fraco.

### **Inferência automática da estrutura de objetos complexos**

Como já foi discutido, a ferramenta DESANA implementa o algoritmo HotCycles [da Silva 2002, Oliveira 2006] que infere, a partir dos valores de atributos extraídos e sua posição nas páginas-alvo, uma estrutura que reflete a organização destes valores nestas páginas. É importante mencionar que o processo de inferência de estrutura ocorre somente durante o processo de extração e não durante o processo de geração dos extratores. Esse aspecto é muito importante, pois confere flexibilidade à ferramenta com respeito a possíveis variações de estrutura, típicas dos objetos implícitos semi-estruturados encontrados na Web.

### **Arquivos de Projeto**

A DESANA implementa o conceito de *Projeto de Extratores*. Isso significa que é possível salvar o ambiente de trabalho corrente com informações de diretórios, páginas-alvo, objetos, atributos e exemplos para posterior alteração ou continuação de uma seção de trabalho.

### **Extrator Stand-alone**

A ferramenta possui um recurso para exportação do extrator gerado para que este possa ser usado sem a ferramenta. A execução do extrator pode ser feita na linha de comando, passando como argumento um diretório onde estão as páginas cujos dados serão extraídos. A completa execução do extrator cria um arquivo XML com os dados extraídos e estruturados.

## **4. Conclusão**

Apresentamos neste artigo a ferramenta *DESANA* para geração semi-automática de extratores. Esta ferramenta permite a geração de extratores para valores de atributos implicitamente presentes em páginas Web a partir da especificação de exemplos destes valores selecionados por um usuário. A geração destes extratores é feita com base em

uma nova técnica de alinhamento progressivo dos exemplos que garante boa eficácia na extração, mesmo quando os valores a serem extraídos estão presentes em um contexto fraco. Quando extraídos, estes valores são agrupados para formar objetos complexos cuja estrutura é automaticamente inferida a partir da análise da disposição dos valores na página de origem. A ferramenta dispõe de diversos recursos que facilitam a especificação de exemplos através de refinamentos sucessivos e testes dos extratores que estão sendo gerados.

Experimentos realizados com 21 coleções de páginas Web reais demonstram a viabilidade do uso da ferramenta DESANA, que com poucos exemplos de treinamento é capaz de alcançar níveis de desempenho superiores a 94% de eficácia. Estes experimentos, discutidos detalhadamente em [Oliveira 2006], tem seus resultados disponíveis em <http://www.dcc.ufam.edu.br/~alti/desana>.

## **Referências**

- Arasu, A. and Garcia-Molina, H. (2003). Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 337–348.
- Baumgartner, R., Flesca, S., and Gottlob, G. (2001). Visual web information extraction with lixto. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 119–128.
- Crescenzi, V., Mecca, G., and Merialdo, P. (2001). Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109–118.
- da Silva, A. S. (2002). *Estratégias Baseadas em Exemplos para Extração de Dados Semi-Estruturados da Web*. Tese de Doutorado, Universidade Federal de Minas Gerais.
- da Silva, A. S., Filha, I. M. R. E., Laender, A. H. F., and Embley, D. W. (2002). Representing and querying semistructured web data using nested tables with structural variants. In *Proceedings of the 21st International Conference on Conceptual Modeling*, pages 135–151.
- Laender, A. H. F., Ribeiro-Neto, B. A., and da Silva, A. S. (2002). DEByE - data extraction by example. *Data and Knowledge Engineering*, 40(2):121–154.
- Liu, L., Pu, C., and Han, W. (2000). XWRAP: An XML-enabled wrapper construction system for web information sources. In *Proceedings of the 16th International Conference on Data Engineering*, pages 611–621.
- Oliveira, D. P. (2006). Geração semi-automática de extratores de dados da web considerando contextos fracos. Dissertação de Mestrado, Universidade Federal do Amazonas.
- Sahuguet, A. and Azavant, F. (2001). Building intelligent web applications using lightweight wrappers. *Data and Knowledge Engineering*, 36(3):283–316.

# FastMapDB: Uma Ferramenta para Visualização em SGBDRs com uma implementação interativa do algoritmo para detecção de agrupamentos *k-medoid*\*

Gabriel de Souza Fedel<sup>1</sup>, Humberto Razente<sup>1</sup>, Agma Juci Machado Traina<sup>1</sup>,  
Caetano Traina Júnior<sup>1</sup>

<sup>1</sup>Departamento de Computação – Universidade de São Paulo  
Av. do Trabalhador São Carlsense, 400, Centro  
Caixa Postal 668, CEP 13560-970, São Carlos - SP, Brasil

`gabriel-fedel@yahoo.com.br, {hlr, agma, caetano}@icmc.usp.br`

**Resumo.** *O aumento da quantidade de informação armazenada nas últimas décadas, e a conseqüente dificuldade em analisá-la, motivou a pesquisa de métodos e ferramentas para auxiliar o ser humano a analisar e retirar informações ocultas nos dados. Este artigo apresenta a incorporação de uma nova funcionalidade na ferramenta FastMapDB: um algoritmo interativo para detecção de agrupamentos (*k-medoid*), que permite ao usuário visualizar graficamente o processo de detecção de agrupamentos e, posteriormente, armazenar as informações referente aos agrupamentos encontrados no banco de dados.*

## 1. Introdução

Motivado pela grande quantidade de dados que é produzida e conseqüentemente armazenada, e pela dificuldade de analisar e extrair informações úteis ocultas nos dados, muitas técnicas e ferramentas foram desenvolvidas para auxiliar o ser humano. Entre elas, temos a técnica para detecção de agrupamentos, uma técnicas de mineração de dados (*Data Mining*), que é utilizada para dividir um conjunto de dados em grupos e identificar distribuições interessantes, descobrindo padrões ocultos nos dados [M. Halkidi and Vazirgiannis, 2001].

Uma outra técnica bastante utilizada para o auxílio da análise de grandes quantidades de dados, é a visualização de informação (*Visualization Information*). Esta permite que o ser humano participe do processo de mineração de dados ativamente, beneficiando-se de uma facilidade do ser humano de analisar representações gráficas.

Este artigo apresenta a implementação de uma técnica interativa para detecção de agrupamentos (*clustering*), o *k-medoid*, dentro da ferramenta FastMapDB, uma ferramenta de visualização de informações em SGBDRs. A técnica *k-medoid* é tradicional em mineração de dados, porém ela é executada como uma única operação, sem interferência do usuário. A nova técnica interativa desenvolvida permite ao usuário interagir, graficamente, com os resultados parciais obtidos a medida que a execução do algoritmo avança. Com essa funcionalidade, é possível visualizar o resultado e/ou os passos do processo de detecção de agrupamentos, e utilizar os recursos, presentes no FastMapDB, para manipular essa visualização e por fim salvar as informações (agrupamentos e *medoids*) referentes ao processo de detecção de agrupamentos.

A estrutura do artigo é a seguinte: a Seção 2 apresenta a Ferramenta FastMapDB, a Seção 3 aborda a técnica de detecção de agrupamentos e o algoritmo *k-medoid*, na Seção

---

\*Projeto financiado pelo CNPq, Capes e Fapesp

4 é exibido a maneira de utilizar o *k-medoid* no FastMapDB, na Seção 5 são exibidos alguns experimentos utilizando esta implementação do *k-medoid* e na Seção 6 apresenta-se a conclusão.

## **2. A Ferramenta *FastMapDB***

A Ferramenta FastMapDB baseia-se no algoritmo FastMap [Faloutsos and Lin, 1995], que permite projetar objetos de um espaço N-dimensional para um espaço Euclidiano, mantendo as distâncias entre os objetos e reduzindo as possíveis distorções que possam ocorrer. Essa ferramenta considera uma base de dados relacional com N atributos como um espaço N-dimensional, e usando o algoritmo FastMap, projeta os objetos desse espaço em um espaço tri-dimensional [Traina et al., 2001]. Dessa maneira, ela permite a mineração dos dados a partir da análise visual gráfica dos dados, aproveitando-se de uma facilidade humana para analisar gráficos.

Para auxiliar essa análise, a ferramenta dispõe de recursos, como os de manipulação da visualização mostrada, como: Grab (para deslocar o mapeamento), Rotate (para rotacionar o mapeamento), Spin (para girar o mapeamento) e Scale (para mudar o tamanho do mapeamento). Outro recurso é a seleção de objetos que permite selecionar os objetos por meio de formas geométricas (plano, ponto, esfera e cubo), e a partir disso recuperar informações do mapeamento que estão na base de dados.

A ferramenta FastMapDB, foi desenvolvida utilizando a linguagem C++, a biblioteca gráfica OpenGL (<http://www.opengl.org>) e ODBC para conexão.

## **3. Técnica de detecção de agrupamentos e o algoritmo *k-medoid***

A técnica de detecção de agrupamentos permite dividir os dados em grupos que tenham algum significado útil. Essa técnica é utilizada em um grande variedade de áreas: psicologia e outras ciências sociais, biologia, estatística, reconhecimento de padrões, recuperação de informação, aprendizado de máquina e mineração de dados [Kaufman and Rousseeuw, 2005]. Basicamente, o que se tenta fazer na busca de agrupamentos é identificar os grupos de objetos, de maneira que objetos pertencentes a um grupo sejam similares (ou próximos) aos outros objetos do mesmo grupo, e objetos de grupos diferentes não sejam similares (ou estejam distantes).

Mineração de dados é uma área voltada para encontrar informações úteis que estejam ocultas em bases de dados. A detecção de agrupamentos é utilizada em mineração de dados, pois possibilita encontrar grupos a partir de dados armazenados numa base de dados, proporcionando a descoberta de estruturas que talvez antes não fossem possíveis de serem compreendidas.

Dentro da área de busca de agrupamentos existem vários algoritmos, entre eles o *k-medoid*. O *k-medoid* é um método particionador, pois divide o conjunto de dados em *k* grupos (onde todos os objetos pertencem a um, e somente um, grupo) [Kaufman and Rousseeuw, 2005]. O *k-medoid* tenta encontrar os grupos a partir de objetos representativos (os *medoids*), fazendo com que os objetos mais próximos de um determinado *medoid* sejam classificados como pertencentes ao agrupamento do respectivo *medoid*. Para isso ele executa os seguintes passos [Kaufman and Rousseeuw, 2005]:

1. *Selecionar os K objetos iniciais* - esses objetos serão candidatos a *medoid*.
2. *Considerar o efeito de trocar um medoid por um objeto não-medoid* - a distância de cada ponto não-*medoid* ao *medoid* mais próximo é calculada, e essa distância é somada sobre todos os objetos. Essa distância representa o custo da configuração corrente. Os custos para cada configuração diferente da atual são calculados.

3. *Selecionar a configuração com menor custo* - caso o novo custo seja diferente da anterior, repita o passo 2.
4. *Associe cada ponto não-medoid com o medoid mais próximo.*

Neste trabalho, o *k-medoid* foi implementado integrando cada ciclo (em que se repetem os passos 2 e 3) com os recursos de visualização da ferramenta FastMapDB, criando uma técnica interativa para detecção de agrupamento.

#### 4. Estendendo o FastMapDB para detectar agrupamentos

Na implementação do algoritmo *k-medoid* dentro do FastMapDB, o usuário é inserido dentro do processo de detecção de agrupamentos. Para isso alguns recursos são disponibilizados para o usuário.

O usuário pode visualizar o resultado do *k-medoid* no mapeamento gerado pelo FastMapDB (como o da Figura 1), podendo distinguir os grupos e seus respectivos *medoids*. Ele pode visualizar o processo de detecção de agrupamentos de maneira a ver o resultado de cada iteração do *k-medoid* (modo passo-a-passo). O valor do *k* (número de agrupamentos) pode ser aleatório ou definido pelo usuário. Nessa implementação, ainda é possível escolher sobre quais objetos o *k-medoid* será executado: os originais (da base de dados) ou os já mapeados pelo FastMap. Por fim, os dados gerados pelo *k-medoid* (grupos e *medoids*) podem ser salvos na base de dados, para posterior utilização.

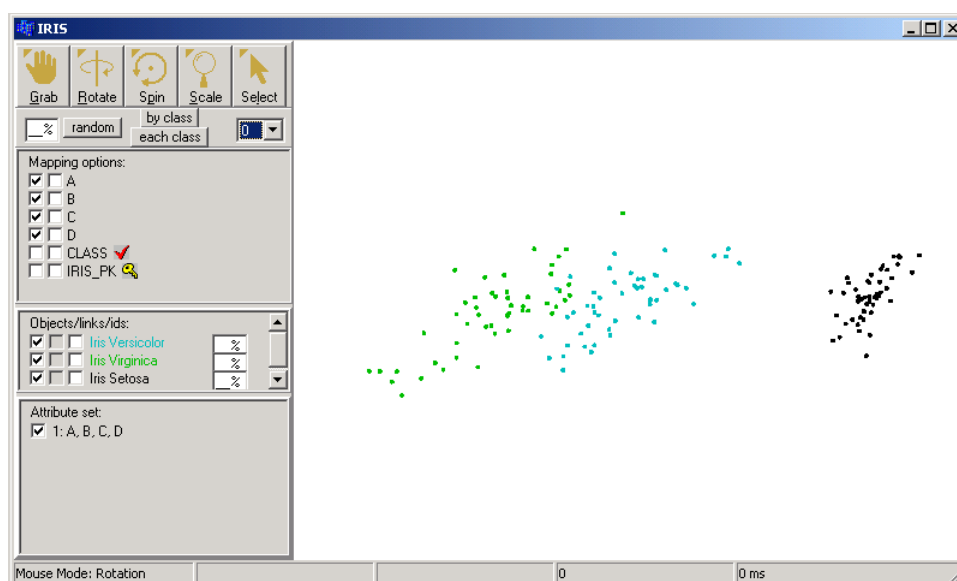


Figura 1: Base de dados IRIS mapeada no FastmapDB

##### 4.1. Utilizando o *k-medoid* no FastmapDB

Para utilizar o *k-medoid* na ferramenta FastmapDB, é necessário que os objetos estejam mapeados, ou seja, o FastMap deve ser executado, pois a execução do *k-medoid* utiliza a visualização dos dados. Tendo os objetos mapeados, basta executar a opção *Cluster*. Nesse momento é aberta uma janela (Figura 2) com os parâmetros que definem a execução do *k-medoid*. Os parâmetros disponíveis são os seguintes:

- *Pontos* - permite escolher qual o domínio dos objetos que serão utilizados no algoritmo *k-medoid*: originais ou os mapeados;
- *K - Número de clusters* - permite definir o número de agrupamentos desejado que será utilizado, podendo ser aleatório ou definido pelo o usuário ;

- *Modo de execução* - Determina como o algoritmo será executado. Pode-se escolher entre o normal (o algoritmo é executado até o número máximo de iterações indicado ou até “convergir”, então o resultado é exibido) ou o passo-a-passo;
- *Iterações*, define o número máximo de iterações que o algoritmo executará, caso ele não convirja.

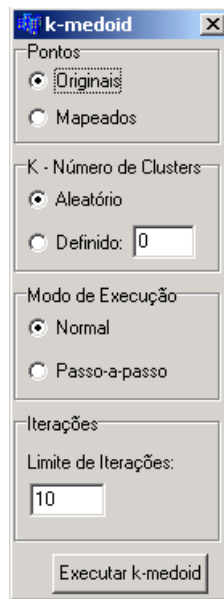


Figura 2: Parâmetros do k-medoid

Após definir os parâmetros que serão utilizados na execução do *k-medoid*, basta clicar em “Executar k-medoid” (na Figura 2). Então na “tela de visualização” do FastMapDB, são exibidos os resultados obtidos pelo *k-medoid*, ou seja, cada conjunto de objetos pertencentes a um agrupamento é exibido com uma cor e os *medoids* recebem uma outra cor (para diferenciá-los dos outros pontos). Um exemplo é mostrado na Figura 3, onde os *medoids* são os pontos pintados de preto.

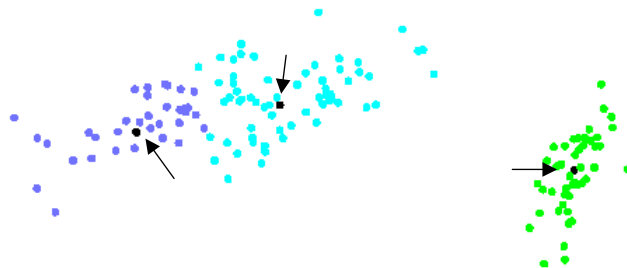


Figura 3: Exemplo de resultado obtido utilizando o k-medoid (com k=3) na base de dados IRIS (as setas indicam os medoids)

O processo de detecção de agrupamentos usando o *k-medoid* é, no FastMapDB, altamente interativo. Caso o *k-medoid* esteja sendo executado no modo passo-a-passo, é possível utilizar as ferramentas de manipulação e seleção de mapeamento disponíveis no FastMapDB. Com isso é possível analisar de forma mais precisa, e passo-a-passo, as informações sobre os agrupamentos geradas pelo *k-medoid*.

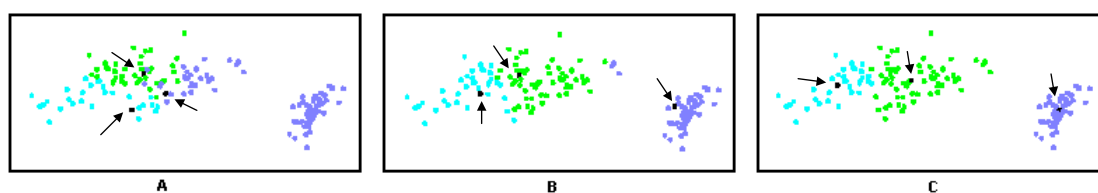
Quando o *k-medoid* chega ao fim, as informações obtidas a partir dele podem ser salvas no Banco de Dados. Para isso é necessário que, antes da execução do *k-medoid*, seja definido um atributo classificador (*Class*). Assim as informações sobre os agrupamentos são salvas nesse atributo, e posteriormente é possível analisá-las e utilizá-las, por exemplo, para gerar uma visualização com o próprio FastMapDB. Uma opção interessante



é utilizar mais de um atributo em diferentes execuções do algoritmo (por exemplo com diferentes valores de  $k$ ) e comparar visualmente os diferentes agrupamentos obtidos.

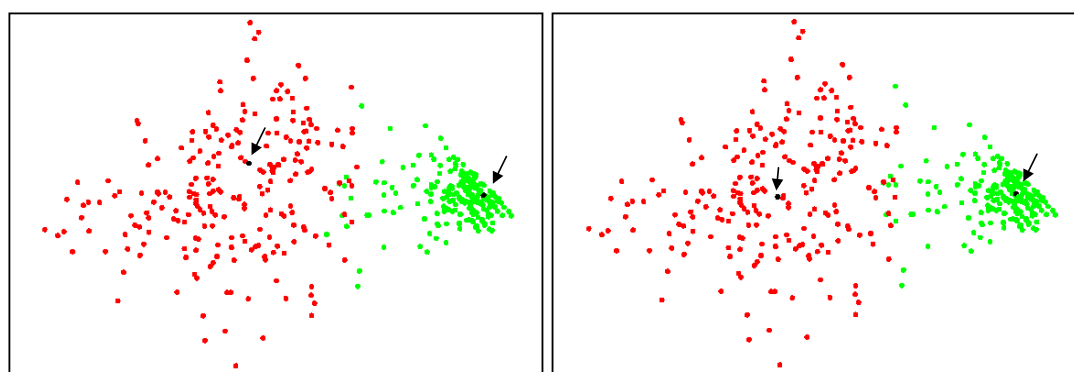
## 5. Experimentos

Para demonstrar a funcionalidade dessa implementação do *k-medoid* foram executados experimentos com duas base de dados. A primeira é a base de dados IRIS [D.J. Newman and Merz, 1998], amplamente conhecida, que contém informações sobre plantas do gênero Iris. A base de dados Iris foi usada nos exemplos anteriores, nas Figuras 1 e 3. Utilizamos a base de dados IRIS para demonstrar a funcionalidade de executar o *k-medoid* passo-a-passo, como vemos na Figura 4. Nesse experimento, os agrupamentos foram encontrados em 3 passos.



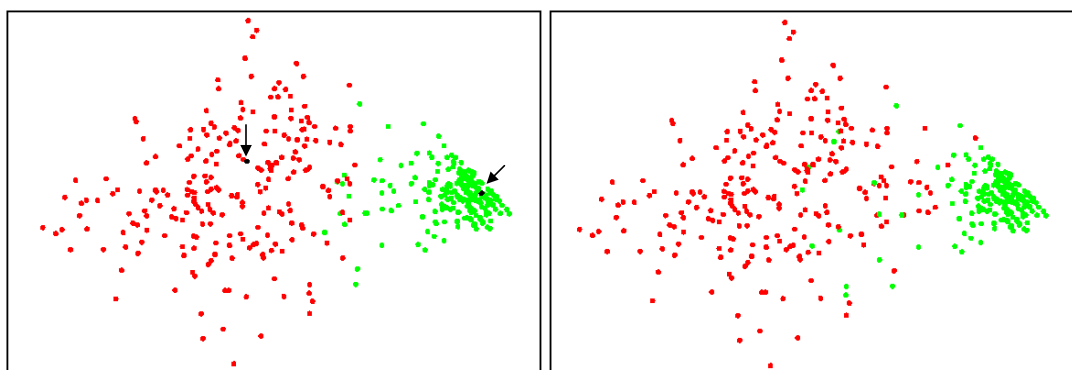
**Figura 4: Execução passo-a-passo do *k-medoid* (com  $k=3$ ) na base de dados IRIS (as setas indicam os medoids)**

A segunda base de dados utilizada é a BreastCancer, também obtida em [D.J. Newman and Merz, 1998], que possui informações sobre tumores localizados na mama. Esta base possui 10 atributos medidos e um atributo classificador que diz se o tumor é maligno ou benigno. Com essa base de dados mostramos a diferença entre executar o *k-medoid* com os dados originais e com os dados mapeados, na Figura 5. Percebe-se que existe uma pequena diferença nas duas execuções, principalmente na “borda” entre os dois agrupamentos. Isso revela distorções geradas pelo mapeamento utilizando o algoritmo FastMap.



**Figura 5: Execução do *k-medoid* na base de dados BreastCancer com os dados originais e mapeados, respectivamente (as setas indicam os medoids)**

Essa base também é utilizada para mostrar na Figura 6 as diferenças entre o resultado quando se executa o *k-medoid* com os dados originais e quando se mapeia os dados utilizando o atributo classificador. Percebemos que os agrupamentos encontrados são similares aos grupos definidos pelo atributo classificador, mostrando a boa qualidade do *k-medoid*.



**Figura 6: Execução do k-medoid na base de dados BreastCancer com os dados originais (as setas indicam os medoids) e dados mapeados com o atributo classificador, respectivamente**

## 6. Conclusão

O FastMapDB, antes da implementação do *k-medoid*, já era uma ferramenta poderosa para mineração de dados. Com essa funcionalidade novas possibilidades de utilização da ferramenta foram inseridas, podendo, por exemplo, usar o *k-medoid* para analisar distorções que o processo de mapeamento causa nos agrupamentos, ou fornecendo subsídios para análises a serem efetuadas posteriormente com outros recursos da ferramenta. Além disso torna-se desnecessária a utilização de outras ferramentas para aplicar o *k-medoid*.

A utilização de uma técnica poderosa como é a detecção de agrupamentos aliada a uma visualização gráfica permitiu desenvolver uma nova técnica interativa que possibilita inserir o usuário no processo de mineração de dados. Dessa maneira o usuário ajuda a superar o limite que algoritmos puramente computacionais as vezes não conseguem ultrapassar.

O *k-medoid* é um algoritmo que consegue resultados muito interessantes mas que podem ser difíceis de analisar. Essa dificuldade é muito reduzida quando se tem uma visualização gráfica de seus resultados, como a agora disponível na nova versão da ferramenta FastMapDB.

## Referências

- D.J. Newman, S. Hettich, C. B. and Merz, C. (1998). UCI repository of machine learning databases [<http://www.ics.uci.edu/mllearn/mlrepository.html>].
- Faloutsos, C. and Lin, K.-I. D. (1995). Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM Int'l Conf. on Data Management (SIGMOD)*, pages 163–174, Zurich, Switzerland. M. Kaufmann.
- Kaufman, L. and Rousseeuw, P. J. (2005). Finding groups in data: An introduction to cluster analysis. pages 68–123. John Wiley and Sons.
- M. Halkidi, Y. B. and Vazirgiannis, M. (2001). Clustering algorithms and validity measures. *SSDBM 2001*, pages 1–2.
- Traina, A. J. M., Traina Jr., C., Barioni, M. C. N., Botelho, E., and Bueno, R. (2001). Visualização de dados em sistemas de bancos de dados relacionais. In *XVI Simpósio Brasileiro de Banco de Dados*, pages 95–109, Rio de Janeiro.

## LABRADOR: Um Sistema para Consultas Baseadas em Palavras-Chave sobre Bancos de Dados Relacionais

Filipe Mesquita<sup>1</sup>, Dinorah de A. Monteiro<sup>1</sup>, Altigran S. da Silva<sup>1</sup>

<sup>1</sup>Departamento de Ciência da Computação  
Universidade Federal do Amazonas  
Manaus – AM – Brasil

{fsm, dam, alti}@dcc.ufam.edu.br

**Resumo.** Este artigo descreve o sistema LABRADOR, desenvolvido para permitir a execução de consultas baseadas em palavras-chave sobre bancos de dados relacionais. O sistema recebe as consultas dos usuários e gera consultas equivalentes em SQL, que podem ser executadas por um SGBD. Este processo utiliza um modelo de redes Bayesianas que estima a probabilidade das consultas geradas satisfazerem a expectativa do usuário considerando a instância do banco de dados. O LABRADOR permite o desenvolvimento rápido e simplificado de aplicações que visam a disponibilização de bancos de dados relacionais na Web, publicando automaticamente qualquer banco de dados, independentemente de recursos específicos dos SGBDs, sem modificar seu esquema.

**Abstract.** This paper describes the LABRADOR system, developed to allow the execution of keyword-based queries over relational databases. The system receives user queries and generates equivalent queries in SQL, which can be executed by a DBMS. This process uses a Bayesian network model that estimates the likelihood of the generated query satisfy the expectation of users considering the current state of the database. The LABRADOR system allows the rapid and simplified development of applications that aim at making available relational databases on the Web by publishing any database, regardless of specific features of the DBMS, with no modifying its schema.

### 1. Introdução

Grande parte das empresas no mundo inteiro utiliza bancos de dados relacionais para armazenar dados cruciais de suas operações de negócio, como estoque de produtos, informações de clientes e fornecedores, etc. Diante da popularização da Web, muitas dessas empresas se sentiram motivadas a publicar seus dados neste veículo como forma de atrair novos clientes e atingir bons níveis de competitividade. Com isso, novos serviços surgiram na Web como as lojas virtuais e bibliotecas digitais.

Por essas e outras razões, muitas empresas necessitam publicar seus bancos de dados na Web de forma que os usuários possam consultar os dados neles armazenados. Para tornar isto possível, é necessário que as interfaces de consulta sejam simples e intuitivas, não exigindo que os usuários possuam conhecimento específico sobre linguagens de consulta e sobre o esquema do bancos de dados.

A solução mais comum para a publicação de bancos de dados na Web é o desenvolvimento de uma aplicação específica que apresenta formulários de consulta contendo

diversos campos de busca, um para cada atributo do banco de dados subjacente. Esta abordagem, embora permita consultas eficientes e precisas, possui desvantagens tanto para desenvolvedores quanto para usuários. O desenvolvimento de tais aplicações é oneroso, pois requer esforço de programação para a construção das interfaces de consulta específicas, que dependem do esquema do banco de dados e dos atributos que se deseja publicar. Do ponto de vista dos usuários, estes muitas vezes precisam procurar, dentre os formulários de consulta disponíveis, aqueles que atendem suas necessidades. Além disso, somente é possível realizar consultas utilizando os formulários e campos previamente definidos.

Por outro lado, os usuários estão familiarizados com as interfaces baseadas em palavras-chave encontradas em máquinas de busca na Web, como *Google* ou *Yahoo!*. Estas interfaces são simples e uniformes para todos os tipos de consulta. Diante disso, uma abordagem interessante seria permitir consultas baseadas em palavras-chave sobre banco de dados, estruturando tais consultas de forma a combinar a eficiência e precisão dos formulários com a simplicidade das interfaces das máquinas de busca na Web.

Neste artigo apresentamos o LABRADOR, um sistema para publicação eficiente de bancos de dados relacionais na Web através de interfaces baseadas em palavras-chaves. A partir de uma consulta baseada em palavras-chave, ou *consulta não-estruturada*, formulada por um usuário, o LABRADOR gera um conjunto de consultas estruturadas candidatas, que são equivalentes àquelas que seriam submetidas através de formulários de consulta. Estas consultas estruturadas são ordenadas segundo a probabilidade de satisfazerem as necessidades do usuário. Dessa forma, o sistema pode submeter ao SGBD a melhor consulta, ou solicitar que o usuário escolha, entre as melhores consultas, a que mais se aproxima das suas necessidades. Depois que a consulta é submetida, os objetos retornados são então apresentados ao usuário na forma de um *ranking*, sendo ordenados segundo a probabilidade de satisfazer a consulta original.

A vantagem de publicar bancos de dados na Web usando o LABRADOR é o esforço mínimo necessário de desenvolvimento, pois a mesma interface é utilizada para múltiplos bancos de dados e a publicação destes bancos é realizada de forma automática. Além disso, nossa abordagem permite que os usuários formulem consultas mesmo não conhecendo o esquema do banco de dados.

Nossa abordagem difere de outras apresentadas na literatura [Agrawal et al. 2002, Bhalotia et al. 2002, Hristidis et al. 2003], pois adotamos uma estratégia não-intrusiva, ou seja, o processamento de consultas é externo ao SGBD e não necessita de nenhuma modificação no banco de dados para a publicação do mesmo, como criação de tabelas, índices e gatilhos. Além disso, utilizamos um modelo de redes Bayesianas para estruturação de consultas, conforme proposto em [Calado et al. 2004].

O artigo está organizado da seguinte maneira. Na Seção 2, explicamos o processo de geração de consultas estruturadas, chamado de estruturação de consultas. Na Seção 3, apresentamos uma visão geral do sistema LABRADOR. Na seção 4, a publicação de bancos de dados é discutida. Finalmente, as conclusões são apresentados na Seção 5.

## **2. Estruturação de Consultas**

O objetivo principal do LABRADOR é derivar automaticamente consultas em SQL, chamadas de *consultas estruturadas*, para execução em um banco de dados relacional, a

partir de consultas escritas pelo usuário usando uma lista de palavras-chave, chamadas de *consultas não-estruturadas*. Este processo é chamado de *estruturação de consultas*.

Mais precisamente, uma consulta não-estruturada é denotada como  $U = \{t_1, t_2, \dots, t_k\}$ , onde cada  $t_i$  é um *termo* ou *palavra-chave* que, supõe-se, ocorrem como parte dos valores dos atributos do banco de dados.

O processo de estruturação de consultas realizado pelo LABRADOR consiste na geração de conjuntos de pares ordenados que representam uma consulta estruturada da forma  $Q = \{\langle A_1 : v_1 \rangle, \dots, \langle A_m : v_m \rangle\}$ ,  $m \geq 1$ , onde cada  $A_j$  é um atributo do esquema de um banco de dados e cada  $v_j \in \{t_1, t_2, \dots, t_k\}$ , assumindo  $v_j$  pertence ao domínio do atributo  $A_j$ . Informalmente, a consulta  $Q$  é composta pela atribuição de cada termo  $t_i$  em  $U$  a um atributo do esquema do banco de dados. Note-se que um mesmo atributo pode ocorrer mais de uma vez numa mesma consulta estruturada, mas cada termo só pode ocorrer uma única vez.

Um exemplo de uma consulta não-estruturada seria  $U = \{\text{"Agatha"}, \text{"Christie"}, \text{"Murder"}\}$ . Uma possível consulta estruturada derivada de  $U$  seria  $Q = \{\langle \text{Author} : \text{"Agatha"} \rangle, \langle \text{Author} : \text{"Christie"} \rangle, \langle \text{Title} : \text{"Murder"} \rangle\}$ .

Dada uma consulta não-estruturada é possível derivar uma grande quantidade de consultas estruturadas a partir de seus termos. Em nosso trabalho utilizamos um modelo de redes Bayesianas [Ribeiro e Muntz 1996] que determina, para cada possível consulta estruturada, a probabilidade desta consulta retornar um resultado que satisfaça a expectativa do usuário ao formular a consulta não-estruturada. Para calcular as probabilidades, toma-se como base a distribuição dos termos usados na consulta não-estruturada formulada pelo usuário ao longo da instância do banco de dados.

Estas probabilidades são usadas para gerar um ranking das consultas estruturadas, de forma que as consultas que melhor representam a consulta do usuário ocupem as primeiras posições do ranking. A partir deste ranking o usuário poderá escolher uma das consultas para ser submetida ao SGBD. Alternativamente, o LABRADOR pode selecionar automaticamente para execução a primeira consulta do ranking. Detalhes da modelagem das redes Bayesianas e dos algoritmos de estruturação são omitidos neste artigo. Estes detalhes são apresentados em [Mesquita et al. 2006].

Consideramos que os resultados da consulta estruturada  $Q$  devem incluir tuplas onde "Agatha" e "Christie" ocorrem como termos dos valores dos atributos `Author`, e onde "Murder" ocorre como um termo para os valores de `Title`. Portanto, uma possível consulta SQL que corresponderia a  $Q$  seria:

```
S1 SELECT * FROM Books WHERE Author LIKE '%agatha%'
      AND Author LIKE '%christie%' AND Title LIKE '%murder%'
```

Na consulta  $S_1$ , os atributos `Author` e `Title` ocorrem na mesma tabela `Books`. Se estes atributos ocorressem em duas tabelas distintas, digamos, `Authors` e `Books`, a consulta SQL abaixo corresponderia à consulta  $Q$ :

```
S2 SELECT * FROM Authors NATURAL JOIN Books
      WHERE Author LIKE '%agatha%' AND Author LIKE '%christie%'
      AND Title LIKE '%murder%'
```

Além disso, ao invés de `LIKE`, o LABRADOR pode utilizar, quando disponível, recursos de *full-text search*, como o operador `CONTAINS`, que é implementado em SGBDs como Oracle e SQL Server.

### 3. O Sistema LABRADOR

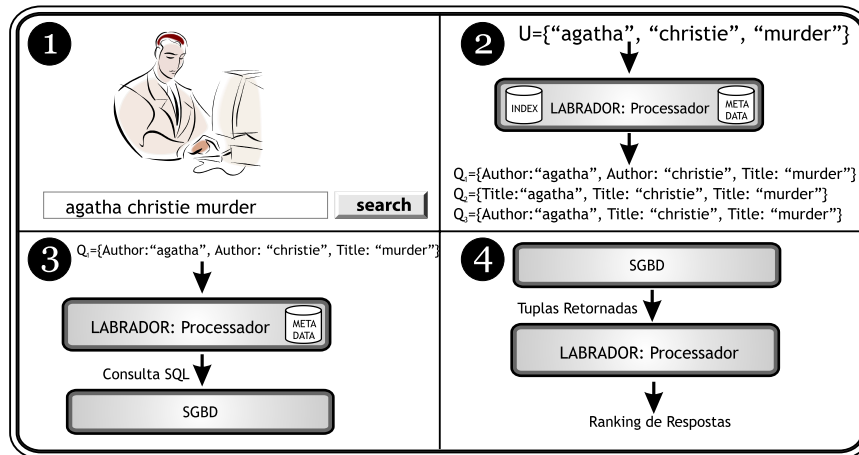


Figura 1. Consulta baseada em palavras-chave usando LABRADOR

O processamento de consultas não-estruturadas no LABRADOR envolve quatro passos, que são ilustrados na Figura 1. No Passo 1, o usuário fornece a consulta não-estruturada. A partir desta consulta, o LABRADOR gera, no Passo 2, consultas estruturadas candidatas através de um modelo de redes Bayesianas para estruturação de consultas, como descrito anteriormente na Seção 2. Neste ponto, uma consulta estruturada pode ser escolhida automaticamente pelo sistema ou pode ser selecionada pelo usuário. No Passo 3, a consulta escolhida é convertida em uma consulta SQL equivalente, que é então submetida ao SGBD. No Passo 4, o conjunto de tuplas é ordenado e apresentado como um ranking. Para isso, é usado outro modelo de redes Bayesianas, que estima a probabilidade de uma tupla satisfazer a consulta do usuário.

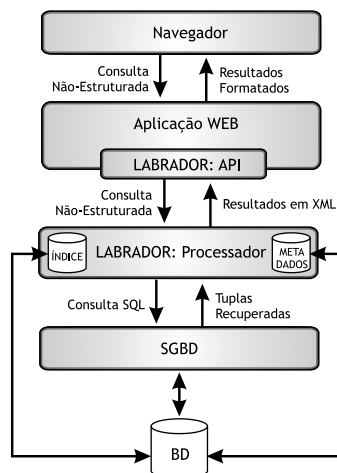


Figura 2. Arquitetura do Sistema LABRADOR

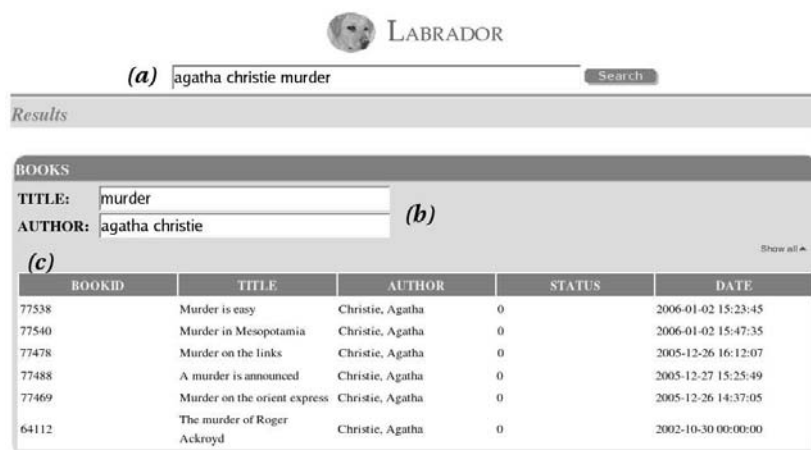
A arquitetura do LABRADOR é ilustrada na Figura 2. O sistema é composto

por dois módulos, um *processador*, implementado em C++, e uma *API (Application Programming Interface)*, implementada em PERL. O processador implementa as funções principais do sistema, como a estruturação de consultas, a classificação de objetos e a comunicação com os SGBDs, realizada através de conectores ODBC. A API é responsável por permitir a comunicação entre a aplicação Web e o processador.

Nesta arquitetura, os termos da consulta não-estruturada são passados da aplicação Web para o processador através da API. Após processar a consulta, o processador retorna o ranking de consultas estruturadas candidatas para a aplicação Web. A consulta escolhida pelo usuário é enviada para o processador, que a converte numa consulta em SQL. Esta é processada pelo SGBD, retornando ao processador o resultado da consulta. Então, o processador classifica as tuplas contidas no conjunto de resultados e as envia para a aplicação Web, através da API.

#### 4. Publicando Bancos de Dados na Web

A publicação de um banco de dados na Web usando o LABRADOR, exige apenas que um administrador escolha quais partes deste banco (tabelas, visões e atributos) devem ser publicadas. Denominamos esse passo de pré-processamento, e, após sua conclusão o sistema estará pronto para processar consultas. No pré-processamento é criado um índice invertido, similar aos usados em sistemas de recuperação de informação, que tem como objetivo acelerar a busca baseada em palavras-chave. Esse índice é mantido no processador do LABRADOR para estruturar consultas sem a necessidade de acessar o banco de dados. É importante ressaltar que o LABRADOR precisa apenas de acesso com permissão de leitura ao banco de dados para realizar o pré-processamento.



The screenshot shows the LABRADOR web application. At the top, there is a search bar with the text "agatha christie murder" and a "Search" button. Below the search bar, the word "Results" is displayed. Underneath, there is a section titled "BOOKS" with two input fields: "TITLE:" containing "murder" and "AUTHOR:" containing "agatha christie". To the right of these fields is a "Show all" link. Below the input fields is a table with the following data:

BOOKID	TITLE	AUTHOR	STATUS	DATE
77538	Murder is easy	Christie, Agatha	0	2006-01-02 15:23:45
77540	Murder in Mesopotamia	Christie, Agatha	0	2006-01-02 15:47:35
77478	Murder on the links	Christie, Agatha	0	2005-12-26 16:12:07
77488	A murder is announced	Christie, Agatha	0	2005-12-27 15:25:49
77469	Murder on the orient express	Christie, Agatha	0	2005-12-26 14:37:05
64112	The murder of Roger Ackroyd	Christie, Agatha	0	2002-10-30 00:00:00

Figura 3. Exemplo de consulta

Uma vez que o banco de dados esteja publicado, pode-se desenvolver aplicações de consulta utilizando os recursos do LABRADOR disponibilizados pela API. A Figura 3 apresenta um exemplo de aplicação Web construída sobre o sistema LABRADOR. Primeiramente, é mostrada uma caixa de texto onde são submetidas as consultas baseadas em palavras-chave (a). A primeira consulta do ranking, gerada pelo processo de estruturação de consultas, é apresentada ao usuário como um formulário HTML (b), onde cada termo da consulta não-estruturada é associado a um campo do formulário, que representa um

atributo. Essa consulta é automaticamente submetida ao SGBD e o seu resultado é mostrado em forma de tabela (c).

É importante mencionar que o LABRADOR não possui uma interface própria de consulta. A forma como as consultas estruturadas e os resultados são apresentadas pode ser definida de acordo com as necessidades dos desenvolvedores da aplicação.

## **5. Conclusões**

Neste artigo apresentamos o LABRADOR, um sistema para publicação de qualquer bancos de dados relacional através de uma interface simples e uniforme, onde o usuário pode formular uma consulta através de uma lista de palavras-chave. A partir de tais consultas, o sistema deriva consultas estruturadas em SQL associando cada palavra-chave a um atributo do esquema do banco de dados.

Como o número de possíveis consultas estruturadas é potencialmente grande, o sistema utiliza um modelo de redes Bayesianas para classificá-las segundo a probabilidade de retornarem tuplas que satisfaçam a necessidade de informação do usuário. Uma dessas consultas é selecionada pelo usuário ou pelo sistema, de forma automática, e é então submetida ao SGBD. O conjunto de tuplas retornado é ordenado pelo LABRADOR através de um segundo modelo de redes Bayesianas, gerando assim um ranking como resposta à consulta original do usuário.

O sistema LABRADOR provê um arcabouço para o desenvolvimento de aplicações que necessitem publicar bancos de dados relacionais na Web. A publicação é realizada pelo LABRADOR de forma automática e independente, sem necessitar de recursos específicos dos SGBDs. Além disso, este processo não faz modificações no esquema do banco de dados, necessitando apenas de acesso para leitura dos dados.

## **Referências**

- Agrawal, S., Chaudhuri, S., and Das, G. (2002). DBXplorer: A system for keyword-based search over relational databases. In *Proceedings of the 18th International Conference on Data Engineering*, páginas 5–16.
- Bhalotia, G., Nakhe, C., Hulgeri, A., Chakrabarti, S., and Sudarshan, S. (2002). Keyword searching and browsing in databases using BANKS. In *Proceedings of the 18th International Conference on Data Engineering*, páginas 431–440.
- Calado, P., da Silva, A. S., Laender, A. H. F., Ribeiro-Neto, B. A., and Vieira, R. C. (2004). A bayesian network approach to searching web databases through keyword-based queries. *Information Processing and Management*, 40(5):773–790.
- Hristidis, V., Gravano, L., and Papakonstantinou, Y. (2003). Efficient IR-style keyword search over relational databases. In *Proceedings of the International Conference on Very Large Data Bases*, páginas 850–861.
- Mesquita, F., da Silva, A. S., de Moura, E. S., Calado, P., and Laender, A. H. F. (2006). LABRADOR: Efficiently publishing relational databases on the web by using keyword-based query interfaces. Em preparação.
- Ribeiro, B. A. N. and Muntz, R. (1996). A belief network model for IR. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, páginas 253–260.



## BioProvider: Uma Ferramenta para o Provimento de Bases de Biosseqüências

Maíra Ferreira de Noronha<sup>1</sup>, Sérgio Lifschitz<sup>1</sup>

<sup>1</sup>Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro  
Rua Marquês de São Vicente, 225, Gávea – 22453-900 – Rio de Janeiro, RJ

maira@inf.puc-rio.br, sergio@inf.puc-rio.br

**Abstract.** *This article briefly describes the BioProvider tool, developed to provide data to biosequence comparison programs without changing the program code. The tool uses buffer management techniques to boost the program performance by reducing the number of disk reads required by the processes. The communication between the provider process and the application processes is done through a device driver that replaces the read and write function calls to the database files. By this means, the application code can remain unchanged.*

**Resumo.** *Este artigo descreve brevemente a ferramenta BioProvider, desenvolvida para o provimento de dados para aplicativos de comparação de biosseqüências sem modificar o código desses aplicativos. A ferramenta realiza gerenciamento de buffer para melhorar o desempenho dos aplicativos, diminuindo o número de leituras feitas ao disco. A comunicação entre o processo provedor de dados e os processos dos aplicativos é feita por meio de um driver, que substitui as chamadas a funções de leitura e escrita de arquivos do banco de dados. Deste modo, o código do aplicativo não precisa ser modificado para usar o BioProvider.*

### 1. Introdução

A área dedicada ao desenvolvimento de ferramentas de Informática para a Biologia Molecular é a Bioinformática. Um dos problemas atuais desta área ocorre devido à inexistência de um Sistema Gerenciador de Bancos de Dados (SGBD) específico para suas aplicações. Uma das características comuns das ferramentas para Biologia Molecular é o acesso a grandes bancos de seqüências e de anotações referentes às mesmas. De modo geral, os bancos de dados usados consistem em arquivos textos ou binários, não sendo usados SGBDs, pois os que existem normalmente não são adequados. Deste modo, as ferramentas não se beneficiam de mecanismos eficientes de armazenamento, acesso a disco e gerenciamento de *buffer* diferentes dos oferecidos pelo próprio sistema operacional.

Uma das tarefas mais importantes na análise dos dados de Biologia Molecular é a de comparação de biosseqüências, sendo a base para várias outras manipulações mais elaboradas. Informações acerca da funcionalidade de genes e proteínas, a localização dos genes nos cromossomos, entre outras informações, são inferidas através de comparações com seqüências conhecidas, armazenadas junto com suas anotações em bancos de dados como o Genbank [Benson et al., 2005] e o SWISS-PROT [SIB, 2006].

As ferramentas da família BLAST [Altschul et al., 1990] são as mais utilizadas pelos pesquisadores atualmente e existem diversos *sites* que disponibilizam os algoritmos para os usuários, podendo ocorrer acessos simultâneos. O desempenho é um fator muito importante para ferramentas de comparação de biosseqüências e pequenas melhoras nestas podem trazer grandes benefícios.

Este trabalho consiste na implementação de uma ferramenta provedora de dados que realiza um gerenciamento de *buffer* eficiente para o BLAST, diminuindo a quantidade de leituras feitas ao disco e tornando, assim, mais rápida sua execução. A utilização do provedor é feita de maneira transparente ao BLAST, pois a comunicação entre ambos é feita através de um *driver* que substitui as chamadas a funções de leitura e escrita de arquivos do banco de dados. A ferramenta pode ser futuramente estendida para prover dados para outros tipos de aplicativos, podendo usar outras estratégias de gerência de *buffer* ou prover dados armazenados em formatos diferentes dos lidos pelos processos clientes, convertendo-os em tempo de execução. O trabalho foi inspirado nas propostas descritas em [Lemos, 2000] e [Mauro et al., 2004].

## **2. Gerenciamento de Buffer para o Blast**

O algoritmo do BLAST consiste em uma heurística para o alinhamento local de biosseqüências, e é usado para comparar seqüências de consulta com bancos de dados de seqüências. Os bancos de dados lidos pelo BLAST possuem um formato específico, sendo formados, de modo geral, por 3 arquivos: um de seqüências, um de anotações associadas às seqüências e um com índices relacionando as seqüências às suas anotações [WU, 2006]. Existem diferentes implementações do BLAST, sendo mais conhecidos o NCBI BLAST [NCBI, 2006] e o WU-BLAST [WU, 2006].

O algoritmo básico do BLAST possui três etapas e seu funcionamento é detalhado em [Altschul et al., 1990]. Na segunda etapa do BLAST, todo o arquivo de seqüências é varrido. Esta é também a etapa na qual há um maior número de leituras feitas do disco para a memória. Já que o BLAST lê diretamente de arquivos do sistema operacional, não utilizando um SGBD, não há um gerenciamento de memória adequado e a leitura das seqüências do banco durante a segunda etapa do algoritmo é feita de maneira ineficiente em algumas situações que são muito comuns. A leitura ineficiente ocorre quando o arquivo de seqüências não pode ser mantido inteiramente na memória e diversos processos são executados ao mesmo tempo, pois não são usadas técnicas de gerenciamento de *buffer* eficientes no aproveitamento de partes do banco de seqüências que já estão em memória para enviar aos processos. Deste modo, a execução de cada processo deverá fazer com que parte do banco de seqüências seja lido novamente do disco para a memória durante a segunda etapa do algoritmo.

A segunda etapa do algoritmo BLAST possui também as seguintes características:

- As seqüências do banco de dados são lidas de maneira seqüencial, logo, é possível prever as próximas páginas que serão necessárias aos processos e trazê-las para o *buffer* antes mesmo de serem requisitadas, realizando o *prefetching*;
- Vários processos podem acessar o banco de dados ao mesmo tempo e, por isso, é desejável que eles compartilhem páginas do *buffer*;

- Não é importante a ordem em que é realizada cada comparação da sequência de consulta com as sequências do banco de dados. Por isso, um processo BLAST pode começar sua comparação por qualquer sequência do banco desde que, ao final da execução da segunda etapa do algoritmo, todas as sequências tenham sido comparadas.

Devido às características descritas, foi sugerida em [Lemos, 2000] uma estratégia de gerenciamento de *buffer* para o BLAST. Nesta estratégia são usadas estruturas de armazenamento de sequências na memória denominadas anéis. Um anel consiste em um *buffer* na memória para o qual o arquivo de sequências do banco vai sendo carregado aos poucos, sendo sua política de atualização semelhante à estratégia FIFO (*First In First Out*). O anel, então, é lido pelos vários processos BLAST de maneira compartilhada, sendo que as atualizações de partes do anel ocorrem quando todos os processos em execução sendo atendidos já tiverem consumido a informação disponível. A grande melhora em relação às políticas de gerenciamento de *buffer* dos sistemas operacionais é que cada processo BLAST, quando iniciado, pode começar a ler o arquivo de sequências a partir do ponto em que este se encontra disponível no anel, já que a ordem de leitura das sequências não é importante. Como o início do arquivo de sequências é carregado novamente para o anel após cada término de sua leitura completa, o novo processo lerá também todas as sequências anteriores à sequência inicialmente considerada, completando a leitura do arquivo.

Esta estratégia de uso eficiente do *buffer* é interessante para ser oferecida junto com o BLAST, pois além de possibilitar o *prefetching*, permite que vários processos compartilhem do mesmo *buffer*, diminuindo assim o tempo gasto em leituras do disco para a memória.

### **3. Implementação do BioProvider**

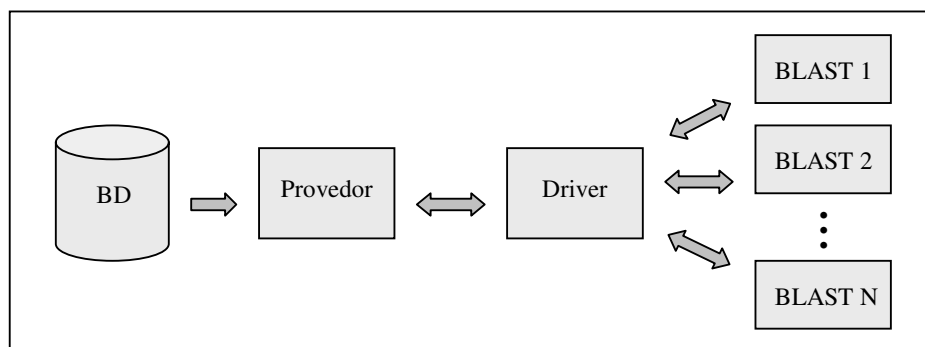
Há dois modos de implementar o gerenciamento de *buffer* para o BLAST: de maneira intrusiva no código e de maneira não-intrusiva. A maneira intrusiva de implementar é através da substituição, no código, de cada chamada às funções de leitura de sequências por outras funções que se comunicam com um processo provedor de dados que irá realizar o gerenciamento de *buffer*. A maneira não-intrusiva de implementar é não modificando o código do BLAST, o que pode ser feito através da utilização de um *driver* que simule o funcionamento dos arquivos do banco de dados e se comunique ao mesmo tempo com o processo provedor. Esta foi a idéia sugerida em [Mauro et al., 2004], que ainda não havia sido implementada.

Neste trabalho, optou-se por implementar o gerenciamento de *buffer* de maneira não-intrusiva através da criação de um *driver*. Deste modo, não é necessário modificar o código do BLAST para acrescentar o gerenciamento de *buffer* e a ferramenta criada pode ser usada para diferentes versões do BLAST com poucas modificações (ou nenhuma) no código da ferramenta.

Um *driver* é um programa que possibilita a comunicação de aplicativos (ou o sistema operacional) com dispositivos de *hardware* e *software*, escondendo a maneira como é realizada a comunicação direta com os dispositivos. Neste trabalho, o *driver* foi implementado para o Linux como um módulo do núcleo (*kernel*). Os arquivos do banco de dados são substituídos por arquivos de dispositivos associados ao *driver* e, ao

executar funções de abertura e leitura destes arquivos, o BLAST chama na realidade as funções implementadas pelo *driver*, que realizam a comunicação com o processo provedor de banco de dados e controlam ao mesmo tempo a concorrência e o bloqueio de processos.

A Figura 1 ilustra o fluxo de informação durante o funcionamento do BioProvider. O processo provedor executa a função de leitura do *driver* e é bloqueado à espera de requisições. Os processos BLAST realizam leituras dos arquivos de dispositivos como se fossem do banco de dados, executando a função de leitura do *driver*, e informando deste modo quais páginas desejam ler do banco de dados. A função de leitura do *driver* bloqueia os processos que realizaram pedidos e acorda o processo provedor, enviando as novas requisições. O processo provedor gerencia o anel de *buffer* e fornece ao *driver* as seqüências desejadas, lidas do banco ou do anel em memória, ao executar a função de escrita do *driver*. Esta função de escrita acorda os processos BLAST cujos pedidos foram atendidos. Em seguida, o processo provedor executa novamente a função de leitura do *driver* e é bloqueado à espera de novas requisições.



**Figura 1 – Fluxo de informação no funcionamento do BioProvider**

O fato de processos BLAST lerem o arquivo de seqüências a partir de posições diferentes, dependendo do conteúdo do anel quando a leitura foi iniciada, gera problemas na interpretação do arquivo de índices. Como este arquivo possui ponteiros para o arquivo de seqüências e o de anotações, os ponteiros poderão estar apontando para posições defasadas do arquivo de seqüências enxergado pelo processo BLAST.

A solução encontrada para resolver o problema descrito é limitar as posições do arquivo de seqüências a partir das quais os processos podem iniciar a leitura, dividindo-o em blocos de  $n$  seqüências. Utilizando-se a ferramenta *formatdb* do BLAST para criar arquivos do banco de dados a partir de arquivos em formato texto, são criadas instâncias de arquivos de índices e anotações para cada possível ordem na qual as seqüências podem ser lidas. Como há apenas  $n$  posições a partir das quais os processos podem começar a ler o arquivo de seqüências, haverá  $n$  instâncias de arquivos de anotações e índices. O processo provedor fornecerá a um processo BLAST os arquivos correspondentes ao bloco a partir do qual o mesmo iniciou a leitura do arquivo de seqüências.

Apesar de aumentar o uso de espaço em disco, a utilização desta estratégia torna a implementação do *driver* independente dos formatos dos arquivos de índices e anotações do banco de dados, já que apenas o formato do arquivo de seqüências deve

ser conhecido pelo processo provedor. O formato deste varia pouco entre as versões do BLAST.

Além de gerenciar o anel em memória, o processo provedor é responsável por decidir qual processo deverá ser atendido a cada momento e a partir de qual posição do anel. Foram implementadas algumas heurísticas para otimizar este atendimento, como dar preferência a processos que estão mais atrasados na leitura do anel ou que estejam pedindo dados de outros arquivos do banco. Um objetivo importante é garantir que todos os processos possam terminar suas execuções, não havendo *starvation*, o que pode ocorrer se novos processos são iniciados a todo momento, bloqueando a atualização do anel. Este problema é resolvido atrasando-se, de tempos em tempos, o início do atendimento de novos processos, para possibilitar a atualização do anel durante o período do atraso e, conseqüentemente, o avanço dos processos já sendo atendidos.

O BioProvider foi implementado inicialmente para o Linux com versões de kernel 2.4 e 2.6 e funciona tanto para o NCBI BLAST quanto para o WU-BLAST 1.4 no acesso a bancos de dados de proteínas, devido à semelhança dos formatos de seus arquivos de seqüências. São incluídas ferramentas para a automatização do pré-processamento do banco de dados e a criação de arquivos de configuração. Alguns parâmetros podem ser especificados, como o tamanho do anel na memória e o número de seqüências por bloco.

#### **4. Resultados Obtidos**

Foram realizados testes para avaliar o desempenho do BioProvider para processos do NCBI BLAST usando o banco de dados de proteínas *nr*, disponível em [NCBI, 2006]. O banco *nr* possui o arquivo de seqüências de proteínas de aproximadamente 1,2 Gigabytes. Os testes foram realizados em um computador Pentium 4 com processador de 3 GHz de velocidade, 512 M RAM e disco rígido SAMSUNG SP0802N. O sistema operacional usado foi o Linux Fedora 4 (*kernel* 2.6). Foram avaliados os tempos de execução de 50 processos BLAST para diferentes configurações de tamanho de memória RAM e tamanho de anel, sendo que cada processo foi iniciado 1 minuto depois do anterior. O banco de dados *nr* foi dividido em 5 blocos com números iguais de seqüências. Foi possível limitar a memória da máquina para diferentes valores modificando-se o arquivo de configuração do programa GRUB (*boot loader*). Foram selecionadas aleatoriamente 50 seqüências pertencentes aos bancos de dados *ecoli.aa*, *swissprot* e *ptaa*, e cada processo BLAST utilizou uma destas como seqüência de consulta. As tabelas a seguir mostram os resultados obtidos.

**Tabela 1 – Resultados da execução normal do BLAST**

Tamanho da memória	256M	512M
Tempo médio de execução	2:40:57	1:11:15

**Tabela 2 – Resultados da execução do BLAST usando o BioProvider**

Tamanho da memória	256M			512M		
Tamanho do anel	25M	50M	100M	25M	50M	100M
Tempo médio de execução	27:55	29:24	42:11	24:05	34:45	25:26
Melhora de desempenho	83%	82%	74%	66%	51%	64%

Verifica-se que o desempenho dos processos usando o BioProvider foi muito melhor, variando entre 51% e 83% de melhora.

## 5. Conclusões

Uma versão inicial do BioProvider foi implementada para usar técnicas de gerenciamento de *buffer* que beneficiam o BLAST no acesso a bancos de dados de proteínas. A ferramenta funciona tanto para o NCBI BLAST quanto para o WU-BLAST 1.4. Seu uso é vantajoso quando o arquivo de seqüências do banco de dados não pode ser mantido inteiramente na memória e diversos processos BLAST são executados ao mesmo tempo. Em testes realizados, verificou-se uma grande melhora no desempenho dos processos BLAST.

A ferramenta pode ser estendida em alguns pontos, como na criação de múltiplos anéis na memória para os quais são alocados processos com velocidades de execução diferentes. Deste modo, o desempenho do BioProvider deverá melhorar, diminuindo o tempo de espera de processos rápidos durante a leitura do anel. A ferramenta pode ser estendida também para prover bancos de dados de nucleotídeos para o BLAST, prover dados para outros tipos de aplicativos, usar outras estratégias de gerência de *buffer* ou prover dados armazenados em formatos diferentes dos lidos pelos processos clientes, convertendo-os em tempo de execução.

## Referências

- Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J. and Wheeler, D. L. (2005 ). "GenBank", In: Nucleic Acids Res. Jan 1;33 (Database issue) :D34-8.
- SIB (2006). "Swiss-Prot and TrEMBL", <http://bo.expasy.org/sprot/>.
- Altschul, S., Gish, Miller, W. W., Myers, E. and Lipman, D. J. (1990). "Basic Local Alignment Search Tool", Journal of Molecular Biology 215, p. 403-410.
- Lemos, M. (2000). "Gerenciamento de Memória para Comparação de Biosseqüências", dissertação de mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.
- Mauro, R. and Lifschitz, S. (2005). "An I/O Device Driver for Bioinformatics Tools: the case for BLAST", Genetics and Molecular Research (GMR), v. 4, n. 3, p. 563-570.
- WU (2006). "WU-BLAST". <http://blast.wustl.edu>.
- NCBI (2006). "NCBI BLAST". <http://www.ncbi.nlm.nih.gov/BLAST/>.