UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Tarcísio Eduardo Moreira Crocomo

WF-QL: UMA PROPOSTA DE LINGUAGEM DE CONSULTA POR SIMILARIDADE PARA FORMULÁRIOS WEB

Florianópolis

2013

Tarcísio Eduardo Moreira Crocomo

WF-QL: UMA PROPOSTA DE LINGUAGEM DE CONSULTA POR SIMILARIDADE PARA FORMULÁRIOS WEB

Trabalho de Conclusão de Curso submetido ao Curso de Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. Olinto José Varela Furtado
Coorientador: Prof. Dr. Ronaldo dos Santos Mello

Florianópolis

2013

Tarcísio Eduardo Moreira Crocomo

WF-QL: UMA PROPOSTA DE LINGUAGEM DE CONSULTA POR SIMILARIDADE PARA FORMULÁRIOS WEB

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovado em sua forma final pelo Curso de Ciências da Computação.

Florianópolis, 27 de maio 2013.

Prof. Dr. Vitório Bruno Mazzola
Coordenador do Curso

Banca Examinadora:

Prof. Dr. Olinto José Varela Furtado
Orientador

Prof. Dr. Ronaldo dos Santos Mello
Coorientador

Prof. Dr. Ricardo Azambuja Silveira

Profa. Dra. Carina Friedrich Dorneles

AGRADECIMENTOS

À minha família, por razões que não precisam ser escritas.

Aos meus amigos todos, por razões de descrição igualmente desnecessária.

Aos meus amigos que desenvolveram seus Trabalhos de Conclusão de Curso ao mesmo tempo, por compartilharem do momento.

Ao professor Olinto, por ter me apresentado a área de Linguagens Formais, pela qual me apaixonei e por ter lembrado de que comentei que gostaria de desenvolver algo na área.

Ao professor Ronaldo, pela proposta do tema e pelo desenvolvimento de toda a base necessária.

E à Thaís Bardini Idalino, sem a qual todos os prazos de entrega de todas as disciplinas do projeto teriam sido perdidos.

RESUMO

A Web possui um vasto conjunto de informações. Já existem diversas iniciativas no sentido de indexar e facilitar a utilização de todos estes dados. Porém, as formas comuns de indexação tem diversos limites em relação a como acessar estas informações. Ao conjunto dessas informações de acesso limitado dá-se o nome de *Deep Web*. Por causa disso, fazem-se necessárias formas diferentes de indexação para formas diferentes de se acessar dados. Um dos pontos de acesso para informações não normalmente acessíveis são formulários Web utilizados para gerar páginas dinâmicas com conteúdo de bancos de dados que não são diretamente acessíveis.

O projeto WF-SIM é uma iniciativa de indexar esses formulários que agem como pontos de entrada e permitir buscas, principalmente através do uso de similaridade. Este trabalho propõe uma linguagem para expressar de forma mais direta as buscas a serem feitas sobre estes formulários, a fim de permitir seu uso para a extração dos dados escondidos, procurando prover uma forma mais expressiva para este domínio, com melhor usabilidade comparada à propostas anteriores.

Palavras-chave: Deep Web, Formulários Web, Linguagem de Consulta, WF-SIM, WF-QL

ABSTRACT

The Web hosts a vast data set. There are, already, efforts to index and ease the use of this information. The common indexing approaches, though, are limited on how they can access this data. The set composed by limited-access data is called the *Deep Web*. Because of that, different approaches are needed to index different ways of accessing informatin. One of the entrances to not normally acessible data are Web forms used to generate dynamic pages with hidden, not directly accessible database content.

The WF-SIM project is an initiative to index these forms that act as entry points and allow searching, especially through similarity. This work proposes a language to express in a more straightforward way the queries to be applied on these forms, allowing their use to extract hidden data, looking forward to provide a more expressive model for this domain, with better usability compared to previous proposals.

Keywords: Deep Web, Web Forms, Query Languages, WF-SIM, WF-QL

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A World Wide Web hoje é composta por uma quantidade muito grande de informações. Como forma de trabalhar melhor com essas informações, foram criados diversos serviços, como por exemplo, *engines* de busca. Serviços desse tipo permitem a indexação de páginas e busca rápida por informações desejadas, possibilitando lidar de forma mais aceitável com essa quantidade de dados. Ao conjunto de páginas indexáveis pelas *engines* supracitadas dá-se o nome de *Visible* ou *Surface Web* (BARKER, 2004).

De forma contrária à definição de *Surface Web*, tem-se a *Deep Web*, que é composta por páginas que, por diferentes causas, saem do escopo de indexação das *engines* padrão de busca. Estimava-se, já em 2001, que a quantidade de informação disponível nessa parte da *Web* era de 400 a 550 vezes maior do que a informação facilmente indexável (BERGMAN, 2001).

Um dos motivos para algumas informações não estarem visíveis na *Surface Web* é por estarem contidas em bancos de dados consultados apenas pela submissão de formulários devidamente preenchidos. Para auxiliar na busca desse tipo de informação, foram criados *crawlers*, ou seja, programas que navegam pela Web e registram os endereços das páginas encontradas. *Crawlers* específicos para essa tarefa são programados para encontrar formulários Web que, quando preenchidos e submetidos, gerem páginas contendo informações que não possam ser encontradas por uma busca feita com *engines* comuns (BARBOSA; FREIRE, 2007). Estes formulários são denominados também como ”portas de entrada”, ou *entry points*.

Devido ao grande volume de dados encontrado na *Deep Web* e a falta de estrutura padronizada destes dados, trabalhar com eles é uma tarefa com alto nível de complexidade. Por isso, esforços estão sendo feitos no sentido de descobrir, extrair, integrar, catalogar e consultar esses dados de forma mais eficaz e eficiente. Estes esforços também envolvem definir linguagens de consulta que permitam expressar buscas específicas envolvendo dados referentes à *Deep Web* (SHESTAKOV; BHOWMICK; LIM, 2005), (BARBOSA; FREIRE, 2007), (NTOULAS; ZERFOS; CHO, 2005), (GONÇALVES et al., 2011). Os trabalhos já existentes limitam-se a buscas exatas por palavras-chave indexando termos presentes em formulários, não provendo ainda tão boa expressividade para buscas.

Como os formulários que agem como *entry points* estão localizados na *Surface Web* e são, portanto, facilmente indexáveis, estes passam a ser bons pontos de partida para buscas dentre os dados escondidos.As informações sobre estes formulários, portanto, são metadados (rótulos de atributos) e dados (valores de atributos), que permitem expressar buscas a dados situados

na *Deep Web*.

O projeto WF-SIM (GONÇALVES et al., 2011), do Grupo de Banco de Dados da Universidade Federal de Santa Catarina, possui um repositório com um grande volume de formulários relacionados a diversos domínios de interesse, como por exemplo: veículos, viagens aéreas, livros, etc. O projeto também provê formas de se aplicar comparações por similaridade nos metadados e dados dos formulários, no seguinte sentido: em um domínio específico, como venda de veículos, diversos formulários onde a marca de veículos pela qual deseja-se procurar é representada por um campo marcado como “brand”, “make”, “made by”, “car make”, entre outras variações. Isso dificulta a criação de uma cláusula de consulta que cubra todas as possibilidades, e consequentemente torna mais complexa a extração de dados utilizando todos os formulários disponíveis. A aplicação de métricas de similaridade nesses metadados permite cobrir mais possibilidades de forma mais sucinta e genérica, ajudando a sanar esse empecilho.

O objetivo deste trabalho é a definição de uma linguagem, e, em seguida, a descrição da análise léxica, sintática, e da síntese de consultas SQL a serem utilizadas sobre este banco de dados gerado pelo WF-SIM. Com isso, espera-se permitir a expressão de consultas diferenciadas, que inclusive invoquem métricas de similaridade sobre as informações dos formulários. Assim, possibilita-se uma busca mais concisa e com resultado mais completo do que uma busca por simples casamento exato de termos presentes em formulários. Isso possibilita encontrar um volume maior de informações relacionadas ao critério da busca dentro do conjunto de informações geradas por estes formulários.

Algumas propostas e implementações já existem para resolver o problema de comparação de dados por similaridade, e buscas envolvendo dados da *Deep Web*. Mais adiante essas propostas serão discutidas e comparadas, a fim de demonstrar quais são suas limitações em cobrir os pontos que aqui foram apresentados completamente, e discute-se como a proposta deste trabalho pode melhorar estes pontos.

Este trabalho está organizado em mais 5 capítulos. O Capítulo 2 apresenta uma fundamentação teórica sobre o conceito de Deep Web e similaridade de dados, e apresenta o projeto WF-SIM, que dá a base para a execução deste trabalho. O Capítulo 3 explora trabalhos correlatos a fim de demonstrar ideias a serem aproveitadas e adaptadas, e ilustrar o objetivo deste trabalho como um avanço no assunto. O Capítulo 4 apresenta a proposta e a especificação da linguagem, o Capítulo 5 apresenta a descrição de um processador que permite o uso da linguagem, o Capítulo 6 mostra o resultado da aplicação de consultas, e, por fim, o Capítulo 7 traz considerações finais sobre o trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta as definições necessárias para a execução deste trabalho, a abordagem utilizada para lidar com os formulários, uma base sobre similaridade de dados e o projeto WF-SIM. Estas apresentações trazem a base que permite a definição da linguagem e as buscas pelos formulários.

2.1 DEEP WEB

A separação do conteúdo da *Web* entre *Surface* e *Deep Web* foi popularizada e assim nomeada por Bergman, em artigo publicado em 2001 (BERGMAN, 2001). Bergman, em seu artigo, aponta que, para ser indexada por *crawlers* de *engines* de busca comuns, uma página necessita possuir uma referência estática e estar referenciada por links em outras páginas. Assim, definiu que toda página que não obedeça a essas duas regras faz parte da *Deep Web*.

Bergman cita outro artigo publicado em 1994 por Jill Ellsworth, que utilizava o termo *Invisible Web* para se referir a páginas que não haviam sido registradas em *engines* de busca. O motivo, segundo Bergman, de utilizar-se de outro termo, é que as páginas não estão de fato "invisíveis", podendo ser acessadas por qualquer pessoa que esteja interessada em acessá-las diretamente. O que realmente ocorre, segundo ele, é apenas que estas páginas não podem ser indexadas pelos meios mais comuns de busca e portanto não farão parte de resultados dos sistemas de busca automática mais utilizados.

Já em 2001, Bergman estimava que os 60 maiores sites da *Deep Web* continham em torno de 750 terabytes de informação. Com isso, já nesta época, o conteúdo da *Deep Web* excedia em 40 vezes o tamanho estimado para a *Surface Web*.

Um exemplo de informação contida na Deep Web são páginas acessíveis somente por túneis de redirecionamento entre diversos *proxies*, gerando conexões anônimas. Estes dados são ainda mais difíceis de serem buscados a partir da *Surface Web*, pois os pontos de acesso e sua estrutura não são tão claros quanto os formulários supracitados. Este conteúdo, porém, não faz parte do escopo deste trabalho.

Outro exemplo comum e direto de conteúdo não indexável é conteúdo apresentado em páginas dinâmicas geradas com base no preenchimento de formulários. Nestes casos, a *engine* de busca indexa apenas o formulário, mas não tem meios de preenchê-lo e submetê-lo, o que cria uma limitação (SHESTAKOV; BHOWMICK; LIM, 2005). Este tipo de conteúdo é a moti-

vação por trás do projeto WF-SIM e deste trabalho. A estrutura destas relações está exemplificada na figura 1.

Para se obter acesso a esse conteúdo, é necessário descobrir formulários que atuem como pontos de entrada (*entry points*) para as páginas não indexáveis. Após coletar todos estes dados, passa a se fazer necessária uma forma de facilitar o acesso a eles, de uma forma organizada e expressiva. Isto deve levar em conta a questão da heterogeneidade de formulários que levam a dados que na realidade possuem uma relação semântica. Este trabalho se dedica a propôr uma linguagem que possibilite tal de acesso.

Para a indexação destes formulários, seus metadados são modelados e relacionados da seguinte forma (MELLO; PINNAMANENI; FREIRE, 2010):

- São denominados *atributos* os campos do formulário a serem preenchidos. Cada atributo é composto por um *rótulo* e um conjunto de *valores*.

- O *rótulo* é o texto que explicita o valor semântico de um atributo.

- Os *valores* são as formas predefinidas de se preencher um *atributo*, quando os valores têm de ser escolhidos a partir de um conjunto previamente oferecido.

- Atributos podem se relacionar por meio de *restrições*, ou seja, dado um valor em específico para um atributo, restringe-se o preenchimento de outros atributos, ditos *dependentes*, para um subconjunto de valores, de forma a impedir preenchimentos com semântica inválida.

A figura 2 mostra um exemplo de formulário e seus atributos.

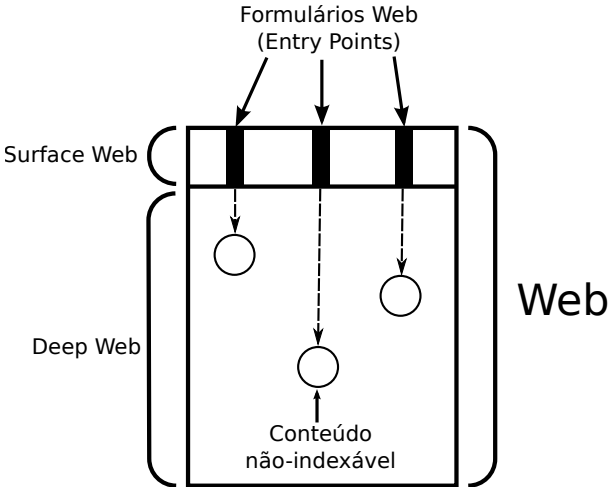

Figura 1: Diagrama mostrando como os formulários web formam pontos de entrada para informações escondidas.

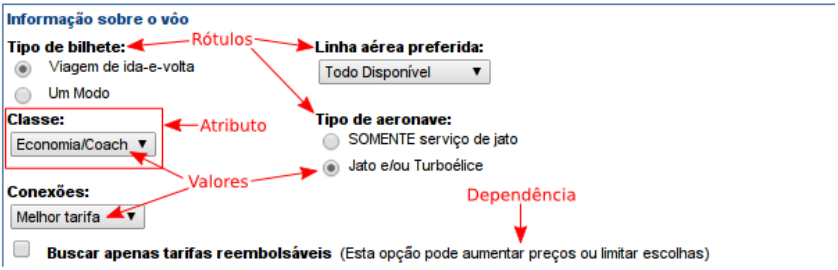

Figura 2: Exemplo de Formulário Web com atributos, rótulos, valores e dependência. (Retirado de *http://hotels.thebostonguide.com/airlines/search.jsp?cid=78677*)

2.2 SIMILARIDADE DE DADOS

Para encontrar dados específicos dentro de um conjunto maior, é necessário definir um critério que determina a especificidade desejada. Isto pode ser feito de diversas formas, como comparação literal, comparação de valo-

res numéricos, dentre outras. Porém, para certas aplicações e certos tipos de dados, faz-se necessário um critério de *similaridade*, ou seja, determinar se, mesmo diferentes em alguns detalhes, dois dados representam o mesmo objeto no mundo real (DORNELES; GONÇALVES; MELLO, 2011).

Para alguns tipos de dado, isso se torna uma tarefa não-trivial, dada uma complexidade em estrutura, ou mesmo na própria definição de similaridade a ser usada para definir se o mesmo objeto está sendo representado. Desta forma, pode-se haver similaridade estrutural ou por conteúdo.

Neste trabalho, como o foco é prover uma forma de buscar formulários com base em seus atributos e valores textuais, a ideia de similaridade a ser usada é por conteúdo. Em específico, similaridade textual, ou seja, dados dois (ou mais) textos, no caso os rótulos e valores de atributos no formulário, determinar se eles são similares o suficiente para afirmar que representam um valor semanticamente semelhante. Isto pode ser feito com funções de similaridade ou com funções de distância.

Uma função de similaridade *fs(v1,v2)* estabelece um *score* no intervalo [0,1]. Quanto mais próximo de 1 o resultado, mais similares os valores são considerados. Já uma função de distância *fd(v1,v2)* retorna um valor no intervalo [0,∞[, e o resultado é considerado mais similar quanto mais próximo a zero estiver.

Em ambos os casos, o resultado pode ser convertido para um valor booleano (similar ou não), se ultrapassar um valor predefinido denominado *threshold*, da seguinte forma para para funções de similaridade:

$$fst(fs,th,v1,v2) = \begin{cases} 0 & fs(v1,v2) < th \\ 1 & fs(v1,v2) \geq th \end{cases} \tag{2.1}$$

Ou, para funções de distância:

$$fdt(fd,th,v1,v2) = \begin{cases} 0 & fd(v1,v2) > th \\ 1 & fd(v1,v2) \leq th \end{cases} \tag{2.2}$$

Desta forma, definido um threshold, pode-se utilizar a função de similaridade como um comparador booleano, da mesma forma que o c̄omum, e assim utilizá-lo como critério em uma busca e parte de uma operação booleana mais complexa.

Outra estratégia, também implementada pelo WF-SIM, é a criação de *clusters*, que agem como *caches* para funções de similaridade. Neste caso, os dados são previamente analisado e agrupados em grupos considerados similares. Isso torna a consulta mais rápida, porém remove a possibilidade de ajuste do threshold da pesquisa, pois os clusters terão um threshold fixo, que será o que foi utilizado na co

A WF-QL provê a sintaxe necessária para o uso de diferentes funções de similaridade e ajuste do threshold, prevendo a implementação de adições ao SGBD MySQL, como *stored procedures* que permitam a aplicação destas.

2.3 O PROJETO WF-SIM

Inicialmente, para se trabalhar com dados em páginas dinâmicas geradas por formulários, deve-se coletar metadados sobre estes. Este trabalho pode ser automatizado utilizando *crawlers*, pois os formulários estão situados na *Surface Web*. Desta forma, pode-se proceder a indexação como uma indexação comum de busca, apenas especializando os *crawlers* para focarem páginas que contenham formulários.

Já existem projetos nesse sentido, como o DeepPeep, da Universidade de Utah (BARBOSA; FREIRE, 2007), o projeto DEQUE (SHESTAKOV; BHOWMICK; LIM, 2005), que também propõe uma linguagem de consulta e o projeto apresentado no artigo *Downloading Hidden Web Content*, da UCLA (NTOULAS; ZERFOS; CHO, 2005).

O projeto WF-SIM, do Grupo de Bancos de Dados da UFSC, propõe um passo além da simples indexação e busca de formulários web. O foco do projeto é o uso de métricas de similaridade como critério de consulta dentre os formulários indexados. A motivação para isso vem diretamente do fato de que diversos formulários que tratam do mesmo tipo de informação, ditos "no mesmo domínio", podem possuir campos e valores similares, mas não exatamente iguais entre si (GONÇALVES et al., 2011).

Isso pode ser ilustrado pelo seguinte exemplo: dois formulários de páginas de revendas de automóveis que possuam estrutura semelhante, pedindo como entrada, dentre outras informações, o nome do fabricante do qual se deseja encontrar automóveis a venda. Porém, um dos formulários possui o campo referente a essa informação rotulado como "Make" enquanto o outro rotula o campo como "Brand". Uma busca por comparação exata não consideraria os dois formulários, a não ser que o critério fosse especificado para ambas as possibilidades, e possivelmente outras, como "Car Make" ou "Made by".

A introdução de uma métrica de similaridade neste caso simplifica a expressão da busca. A forma como as informações são indexadas, no projeto WF-SIM, inclui um índice baseado em similaridade, auxiliado por uma estratégia de clusterização de atributos similares presentes em formulários diferentes. Dessa forma, campos semelhantes são previamente indexados, e consultas podem ser feitas com base nessas relações de similaridade.

Outra possibilidade a ser considerada é o uso de métricas algorítmicas

de similaridade textual. Isto pode ser utilizado para encontrar, por exemplo, rótulos ou valores grafados de forma semelhante, porém com erros de digitação, ou diferenciados por abreviações, como por exemplo “Rio Grande”, “R. Grande” ou “Rio Grnde”.

A arquitetura do projeto WF-SIM é mostrada na figura 3. No momento, conforme é possível verificar, a entrada para uma busca por formulários é feita por um formulário *template*, representando o conjunto de atributos que se deseja que os resultados possuam.

Nessa arquitetura, pode-se posicionar a linguagem WF-QL aqui proposta como um novo modelo de entrada para o módulo de busca, além do formulário *template* atualmente utilizado. Isso permite uma expressividade maior nas consultas, e também dá a base para o uso, por exemplo, de consultas embutidas em código, tal qual SQL em um SGBD comum.

A adição deste trabalho ao projeto WF-SIM é de uma linguagem formalmente descrita para expressar consultas dentre o conjunto de formulários indexados, incluindo o uso de comparações por similaridade dentre rótulos e valores, expressão de dependências entre campos, e todas as outras possibilidades de busca que o WF-SIM ou um modelo de dados de formulários Web proporciona, de forma a facilitar a manipulação desta informação.

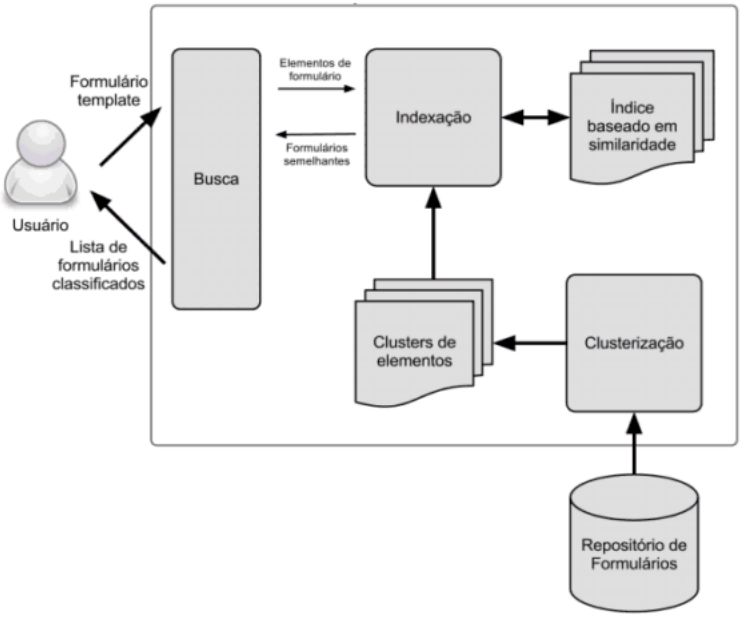

Figura 3: Diagrama da arquitetura do projeto WF-SIM

3 TRABALHOS CORRELATOS

Este capítulo apresenta trabalhos correlatos à proposta e procura mostrar semelhanças e diferenças entre a proposta e os trabalhos relacionados. Procura mostrar também como as limitações desses trabalhos quanto ao nosso objetivo, motivando o desenvolvimento de uma linguagem que trabalhe com todos os requisitos necessários para buscas por similaridade em formulários Web.

3.1 PGSIMILAR: UMA FERRAMENTA OPEN SOURCE PARA SUPORTE A CONSULTAS POR SIMILARIDADE NO POSTGRESQL

Este trabalho propõe uma extensão ao PostgreSQL permitindo consultas por similaridade, inclusive com a escolha da métrica de similaridade e alteração dos parâmetros (BORGES; DORNELES, 2006). O foco da extensão é aplicar similaridade a dados relacionais, sem visar a aplicação específica a dados de formulários Web.

Um exemplo de consulta com o PgSimilar utilizando a função de Levenshtein para encontrar cidades com nome similar a "Rio Grande"é mostrado a seguir:

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

Esta consulta utiliza o operador binário de similaridade pelo método de Levenshtein (≅). A notação utilizada para os operadores binários acaba por ser pouco intuitiva, necessitando a consulta a uma tabela que relaciona funções a operadores.

A seguir, uma consulta utilizando a função de Hamming para encontrar cidades com similaridade acima de um limite (o operador binário de Hamming não foi implementado):

```
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;
```

Como citado anteriormente, no escopo deste trabalho, essa extensão do PostgreSQL é limitada por não ser projetada para trabalhar diretamente sobre dados de formulários Web, não definindo uma sintaxe e semântica que permita expressividade e facilite a descrição de consultas aplicadas a este tipo de dados em específico. Porém, serve como base para a ideia da aplicação da similaridade nas consultas SQL geradas a partir de consultas em WF-QL, com a implementação de uma extensão similar para o SGBD MySQL.

3.2 DEQUE: QUERYING THE DEEP WEB

Este trabalho propõe um sistema de consulta genérico dentre resultados retornados por um formulário. Desta forma, dado um formulário previamente conhecido, ele permite expressar, em uma linguagem abrangente, denominada DEQUEL, consultas, permitindo a seleção de resultados de uma consulta em um formulário, expressa em uma cláusula *where* que determina os valores a serem utilizados em cada atributo do formulário (SHESTAKOV; BHOWMICK; LIM, 2005). A linguagem também permite o uso de resultados de um formulário para o preenchimento de outro, modelando relações entre domínios diferentes de informação.

Uma consulta, em DEQUEL, procurando os trabalhos de pesquisadores, cujas informações estão previamente contidas em uma lista *researcher*, no formulário *pubmed*, que tenham sido publicados em 2002.

```
SELECT authors, work, published, pubmed.TEXT
FROM pubmed, researcher
WHERE pubmed.db = "PubMed" AND pubmed.TEXT = {reseacher.name,all}
CONDITION published = (text contains "2002")
```

É importante ressaltar que a busca é feita simplesmente sobre "pubmed"pois esse é o nome que foi utilizado ao se registrar o formulário encontrado, à época da escrita do artigo, em *http://www.ncbi.nlm.nih.gov/entrez/query.fcgi* no banco de dados no qual a busca é realizada. A seleção de *authors, work, published* e *pubmed* denota os campos do resultado que devem estar no retorno.

A seguir, uma consulta procurando os 3 "melhores"vôos de Cingapura a Londres nas datas de 28 de outubro e 14 de novembro de 2002 e 24 de janeiro de 2003, com assentos livres nas classes executiva ou econômica. O formulário utilizado para a consulta, à época da escrita do artigo, estava disponível em *http://www.amadeus.net/home/index.http*.

```
SELECT FIRST(3) flight, depart, arrive, stops_aircraft, duration,
```

```
business_seat, economy_seat, amadeus.D_Month, amadeus.D_Day AS
Seatsaval
FROM amadeus
WHERE amadeus.D_City = "Singapore" AND amadeus.A_City = "London" AN
(amadeus.D_Month, amadeus.D_Day) = ("October", "28" "November",
"14", "January 2003", "24")
CONDITION business_seat = (text contains "Yes") OR
economy_seat = (text equal "Yes")
```

Os atributos apresentados na cláusula *select* se referem ao resultado da consulta no formulário. O preenchimento do formulário para a efetivação da busca é feito com os parâmetros inseridos na consulta dentro da cláusula *where*.

Este trabalho se diferencia na questão do objeto das consultas. Enquanto a linguagem DEQUEL se propõe a fazer buscas dentro do retorno de formulários específicos, WF-QL pretende permitir a busca dentre formulários, sem a intenção de utilizá-los diretamente. DEQUEL não provê, também, operações de similaridade nas consultas.

3.3 XSIMILARITY: UMA FERRAMENTA PARA CONSULTAS POR SIMILARIDADE EMBUTIDAS NA LINGUAGEM XQUERY

Este trabalho se propõe a estender a linguagem XQuery para permitir consultas em bases de dados XML utilizando similaridade (SILVA; BORGES; GALANTE, 2008). A extensão é semelhante ao que o PGSimilar adiciona ao PostgreSQL. Relaciona-se ao nosso trabalho por implementar consultas por similaridade, mas distancia-se por trabalhar com XML e não se propôr a manter semelhança com SQL, uma vez que estende a XQuery.

Uma consulta utilizando o XSimilarity se apresenta da seguinte forma:

```
declare namespace simi = "java:xsimilarity.Similaridade";
for $b in doc("dblp2.xml")/dblp/incollection
return
if (some $a in $b/author satisfies simi:mongeElkan($a,
"Leonidas Kavvbos") > 0.7)
then
($b/title,$b/author)
else()
```

Aqui foi utilizada a função de similaridade Monge Elkan para se recuperar da base de dados uma entrada relacionada a um livro, com uma diferença entre o nome consultado (Leonidas Kavvbos) e o nome real do autor

registrado no banco de dados (Leonidas Koveos). O artigo demonstra claramente que o resultado foi obtido mesmo com o nome incorreto do autor.

O trabalho também possui uma interface para auxílio à construção da chamada da função de similaridade, definindo qual função será usada e qual a tolerância. É semelhante à proposta de uma interface de construção das consultas, mas limitada somente à chamada das funções de similaridade, não auxiliando na construção de consultas complexas.

3.4 INDEXING WEB FORM CONSTRAINTS

A ideia principal deste trabalho é dar uma definição formal de formulários Web, com seus componentes, restrições, representação e formas de indexação (MELLO; PINNAMANENI; FREIRE, 2010). Possui diversas demonstrações dos métodos propostos e mostra resultados de performances de consultas sobre os índices propostos.

Este trabalho dá a base para a definição de formulário apresentada na fundamentação teórica. Com isso, se correlaciona com este trabalho na base teórica necessária para a execução deste.

O trabalho não cobre a definição de uma linguagem ou a extensão de alguma linguagem preexistente, e não contempla em detalhes a ideia da busca por similaridade dentro dos formulários, mas forma a base teórica da estrutura dos dados que são o objeto de trabalho desse projeto. Com isso, provê uma base formal fundamental para a definição da forma como a linguagem trabalha com os formulários.

Pode-se concluir com base na análise dos trabalhos correlatos que foram encontrados, que nenhum deles possui todas as características desejadas nesse trabalho, ou seja, uma linguagem com foco em dados da Deep Web, fazer a busca sobre formulários, possuir estrutura semelhante à uma linguagem conhecida de consultas (SQL) e trabalhar com similaridade.

A Tabela 1 a seguir sumariza as limitações dos trabalhos relacionados, destacando as necessidades que a linguagem de consulta aqui proposta deseja. O trabalho *Indexing Web Form Constraints* não está relacionado por não se tratar de uma proposta de linguagem ou extensão de uma linguagem preexistente.

Tabela 1: Tabela comparativa entre os trabalhos avaliados e este trabalho.

	Trabalha diretamente com formulários	Proximidade sintática ao SQL	Provê operadores de similaridade	Sintaxe específica para o domínio de Deep Web
PgSimilar		✓	✓	
DEQUEL	✓[1]	✓		✓
Xsimilarity			✓	
Este trabalho	✓[2]	✓	✓	✓

1: Trabalha com busca dentre os resultados de um formulário.

2: Trabalha com busca dentre formulários.

4 PROPOSTA

Este trabalho apresenta WF-QL, uma linguagem de consulta por similaridade para formulários Web, bem como um processador de consultas para esta linguagem. Esta linguagem permite a expressão de consultas envolvendo diferentes domínios, comparação exata e por similaridade de rótulos e valores e a definição das funções de similaridade a serem utilizadas, bem como o valor limite de similaridade esperado pela consulta.

A base desta proposta foi a gramática que descreve a cláusula *select* na linguagem SQL. O motivo desta escolha é a alta difusão do SQL e de linguagens que seguem o mesmo estilo sintático, tornando mais amigável a formulação de consultas, facilitando a familiaridade. Porém, algumas modificações sintáticas e semânticas foram propostas, de forma a adaptar a linguagem ao domínio específico de buscas por formulários Web. Outra justificativa para a escolha é que, pelo fato das informações sobre os formulários Web estarem registradas em um banco de dados relacional, a semelhança com SQL facilita o mapeamento das consultas para instruções SQL a serem executadas.

Para fazer uso da linguagem, o único conhecimento esperado do usuário, além da sintaxe, são os domínios contemplados no banco de dados a ser consultado. Para resolver isto, a linguagem provê uma construção (*select domains*) que retorna os domínios registrados no momento.

4.1 GRAMÁTICA DA WF-QL

A construção de uma consulta em WF-QL é baseada na seguinte produção:

```
query ::= select (from (where)?)?
```

A construção *select* tem a seguinte definição:

```
select ::= SELECT ((selectable (COMMA selectable)*) | STAR)

selectable ::= TITLE | URL | LABELS | VALUES | ATTRIBUTES
```

Isso permite a especificação do conteúdo do resultado da consulta, utilizando-se uma lista de *selectables*, ou seja, os tokens **, title, url, labels, values, attributes*, respectivamente. É interessante notar a detecção, já na sintaxe, da duplicação do símbolo *, que é semanticamente incoerente e fica, conforme a especificação acima, impossibilitada. A interpretação semântica desses tokens é a seguinte:

- *Title*: título da página Web que contém o formulário.
- *URL*: URL da página que contém o formulário.
- *Labels*: Rótulos que nomeiam os campos do formulário.
- *Values*: Valores que os campos predefinidos do formulário podem tomar.
- *Attributes*: Rótulos relacionados aos *values* que se se referem ao campo denotado por eles.

Construções dadas pela produção *select* seguirão portanto o seguinte estilo:

```
select *
select attributes
select title, labels
```

A construção *from* segue a seguinte forma:

```
from ::= FROM (domain (COMMA domain)*)

domain ::= (ID ((AS)? ID)?)
```

Dessa forma é possível a definição dos domínios a serem pesquisados, que são dados por identificadores referentes aos domínios registrados no banco de dados sobre o qual a consulta está sendo feita. Também é possível a renomeação do domínio no escopo da consulta. Isso pode ser feito com a keyword *as* ou mesmo a omitindo, seguindo o mesmo estilo da linguagem SQL.

Dessa forma, a cláusula *from* produz o seguinte tipo de sentença:

```
from airfare
from books
from airfare, biology
from books b
from books as b, hotels as h
```

Por fim, a cláusula *where* é definida da seguinte forma:

```
where ::= WHERE criteria
```

Onde a construção *criteria* é definida por uma gramática de operadores que permite o uso de operações lógicas como *and*, *or* e *not* entre as operações básicas de comparação fornecidas pela linguagem, dadas pelas seguintes produções:

```
criteria ::= compoundCritOr

compoundCritOr ::= compoundCritAnd (OR compoundCritAnd)*

compoundCritAnd ::= notCrit (AND notCrit)*

notCrit ::= (NOT)? booleanPrimary

booleanPrimary ::= labelCriteria | valueInFieldCriteria
                 | enablesCriteria

labelCriteria ::= LABELS (CONTAIN)? (similar)? labelList

labelList ::= label | RPAREN label (COMMA label)* LPAREN

label ::= STRING

valueInFieldCriteria ::= label (relational | similar)
                         (value | value_list)

relational ::= LT | GT | LTE | GTE | EQ | NEQ

valueList ::= RPAREN value (COMMA value)* LPAREN

value ::= STRING

similar ::= SIMILAR TO (LT (WITH function)? (THRESHOLD
                                              THRESHOLD_VALUE)?
                        GT)?

enablesCriteria ::= valueInFieldCriteria ENABLES
                    valueInFieldCriteria
```

Essas produções permitem as três seguintes verificações básicas: existência de um campo com um rótulo específico no formulário, possibilidade da associação de um valor específico a um campo e dependência entre campos. Também há a definição da sintaxe usada para buscas por similaridade, incluindo a possibilidade do uso de uma função de similaridade e um valor de limiar que não o padrão para a comparação.

Com as três cláusulas (*select, from* e *where*) é possível expressar filtros conforme exemplificado a seguir:

Exemplo 1: Uma consulta no domínio de viagens aéreas, procurando as URLs de formulários que contenham campos relativos a destino e origem da viagem, e que permitam que a origem seja "London", de forma a habilitar que o destino seja "Paris".

```
select url from airfare where labels contain similar to
("to", "from") and
"from" = "London" enables "to" = "Paris"
```

Exemplo 2: Uma busca por formulários referentes a livros, que possuam um campo referente a preço, que permitam que o valor seja menor que 25.

```
select * from books where labels contain similar to "price"
and "price" < 25
```

Exemplo 3: Uma consulta no domínio de venda de automóveis que possua um campo referente ao fabricante e que aceite os valores "Toyota" e "Ford" para esse campo:

```
select * from auto where labels contain similar to "make" and
"make" = ("Toyota", "Ford")
```

Para a obtenção dos domínios disponíveis para pesquisa, existe a construção dada pela seguinte produção:

```
select ::= SELECT DOMAINS
```

Esta produção permite o comando

```
select domains
```

que retorna a lista dos domínios disponíveis no momento no banco de dados.

Há também a construção *set*, que permite redefinir a função de similaridade e o valor padrão do limiar aceitável para o resultado. Esta construção é dada pela seguinte gramática:

```
set ::= SET DEFAULT (function)? (THRESHOLD threshold_value)?

function ::= ID

threshold_value ::= DOUBLE
```

Assim é possível a redefinição desses parâmetros conforme mostrado a seguir:

```
set default
set default levenshtein
set default levenshtein threshold 75
```

Importante notar que as keywords *set default* por si só permitem o retorno à definição padrão do sistema.

A função de similaridade, bem como o *threshold*, também podem ser definidos apenas para uma consulta, com base na seguinte construção gramatical:

```
similar ::= SIMILAR TO (LT (WITH function)? (THRESHOLD
                                                THRESHOLD_VALUE)?
                          GT)?
```

Gerando consultas como a seguinte, que usa especificamente a função de Levenshtein com *threshold* de 85%.

```
select url from airfare where labels contain similar to <with
levenshtein threshold 85>
("to", "from") and "from" = "London" enables
                            "to" = "São Paulo"
```

4.2 BANCO DE DADOS

O banco de dados contendo as informações a serem buscadas pelo processador da WF-QL é um dos bancos de dados do projeto WF-SIM. Sua estrutura pode ser encontrada no Anexo B.

Para a execução da proposta, a estrutura presente do banco foi analisada, e concluiu-se que alguns dos relacionamentos necessários para as consultas a serem expressas na WF-QL estavam dispostos de forma a se tornarem consultas ineficientes em SQL. Também muitos dos dados fornecidos pelo banco não eram úteis às consultas, como informações posicionais dos elementos da página web, por exemplo.

Por isso, tomou-se a decisão de reprojetar o banco de dados de forma a obter uma estruturação dos dados que contribuísse com as consultas a serem feitas, e somente com os metadados realmente úteis para estas.

4.2.1 Reprojeto do Banco de Dados

O primeiro passo para a reprojeto do Banco de Dados foi a criação de um novo esquema com os dados desejados e as relações a serem consultadas devidamente colocadas. Foram extraídos os campos relevantes e dispostos em novas tabelas, removendo-se as tabelas de relacionamento existentes no banco original, preferindo-se então permitir referências nulas em caso de um relacionamento inexistente.

Os seguintes campos foram extraídos pois eram estritamente necessários para a execução do projeto:

Da tabela *webform*:

- *id*

Da tabela *webpage*:

- *url*
- *title*

Da tabela *domain*:

- *id*
- *domain_name*

Da tabela *webformfield*:

- *id*
- *webform_id*

Da tabela *webformfield2label*:

- *label_id*
- *webformfield_id*

Da tabela *cluster*:

- *id*
- *cluster_name*

Da tabela *textlabel*:

- *id*
- *text_content*
- *cluster_id*

Da tabela *webformfieldvalue*:

- *id*
- *webformfield_id*
- *text_content*
- *value*
- *isdefault*

Da tabela *webformfielddependedency*:

- *id*
- *parent_id*
- *child_id*

Da tabela *valuedependency*:

- *id*
- *parent_id*
- *child_id*

A referência de *webform* para *webpage* foi simplificada trazendo as informações relevantes de *webpage* (*url* e título) para a tabela *Webforms*.

Por fim, fez-se a reestruturação de alguns desses campos, tornando alguns dos relacionamentos representados por tabelas para *foreign keys* diretas.

- A tabela *webformfield2label* foi substituida por *Attributes.label* referenciando *Labels* diretamente.
- A tabela *domain2webform* foi substituida por *Webforms.domain* referenciando *Domains* diretamente.

Foi também feita a renomeação de *field* para *attribute*, a fim de manter a nomenclatura utilizada na proposta da linguagem.

As expressões SQL utilizadas para a criação das novas tabelas com base no Banco de Dados original podem ser encontradas no Anexo D. Restrições de chaves primárias e estrangeiras foram todas aplicadas pela interface MySQL Workbench.

5 PROCESSADOR DA WF-QL

5.1 ANÁLISE

Para a criação dos analisadores léxico e sintático da linguagem WF-QL, escolheu-se a ferramenta GALS (Gerador de Analisadores Léxicos e Sintáticos), também desenvolvida na UFSC (GESSER, 2002).

O GALS provê um ambiente para a definição das especificações léxicas e sintáticas para os analisadores, bem como a definição das ações semânticas a serem tomadas de acordo com a AST (*Abstract Syntax Tree*).

5.1.1 Especificação léxica

A especificação léxica é feita fornecendo expressões regulares que definam os *tokens*. Com isso, o GALS pode gerar o código necessário para fazer a análise do texto fornecido e transformá-lo em um formato que o analisador sintático possa utilizar para fazer seu processamento.

O GALS gera, com base na especificação dada, uma classe Java *Lexico*, que é a interface para o analisador léxico, e é um dos argumentos para o método *parse* da API do analisador sintático. Desta forma, o analisador sintático pode consumir *tokens* até cumprir uma de suas regras sintáticas, até o fim do código a ser analisado, ou até um erro acontecer.

As expressões regulares que definem cada token para a WF-QL podem ser encontradas no Anexo E.

5.1.2 Análise Sintática

Para a análise sintática, foi necessário, antes de tudo, converter a gramática para o formato reconhecido pelo GALS (BNF, ou *Backus-Naur Form*). Desta forma, foi necessário criar produções explícitas para produções que, durante a especificação da gramática, feita em EBNF (*Extended Backus-Naur Form*), utilizavam da facilidade desta extensão em expressar partes opcionais de uma produção, da seguinte forma:

```
set ::= SET DEFAULT (function)? (THRESHOLD threshold_value)?
```

Neste exemplo, partes destacadas por parênteses e seguidas de uma interrogação (?) são consideradas opcionais. O GALS não aceita essa notação, criando a necessidade de criar produções que reflitam essa semântica.

A própria expansão destas produções mostrou não-fatorações na gramática. Não-fatorações acontecem quando o mesmo não-terminal pode ser utilizado em duas produções que possuem uma mesma subsentença à esquerda, como no seguinte exemplo:

```
set ::= SET DEFAULT function THRESHOLD threshold_value
      | SET DEFAULT function
      | SET DEFAULT THRESHOLD threshold_value
      | SET DEFAULT
```

Como o tipo de *parser* escolhido foi o LL(1), foi necessária a fatoração da gramática, criando símbolos novos para permitir a eliminação das produções não-fatoradas. Neste caso, por exemplo, o não-terminal *set* passou a ser resolvido da seguinte forma:

```
set ::= SET DEFAULT setAux ;

setAux ::= function thresholdDef | thresholdDef ;

thresholdDef ::= THRESHOLD thresholdValue | & ;
```

O processo de fatoração, a partir deste ponto, foi feito em toda a gramática. Com isso, foi possível usar o GALS para gerar o analisador sintático e começar a implementação das ações semânticas para gerar código SQL a ser utilizado com o Banco de Dados reprojetado. A gramática devidamente fatorada e no formato aceito pelo GALS pode ser encontrada no Anexo F.

5.2 SÍNTESE

5.2.1 Geração de código SQL

O GALS, após processar as especificações léxica e sintática da linguagem, gera código (C++, Java ou Delphi, no caso, optou-se pelo Java, linguagem utilizada no projeto WF-SIM) para um analisador léxico, um sintático e um semântico.

Esses códigos são devidamente organizados em classes, permitindo o uso da orientação a objetos no projeto do processador. Especificamente, o GALS gera as classes *Lexico*, *Sintatico* e *Semantico*. A classe *Lexico* recebe o input (algum objeto que implemente a *interface Reader*, provida em java.io.Reader) e gera *tokens*. É utilizada, assim, diretamente pela classe Sintatico via o método *parse*. Esse método recebe uma instância de *Lexico* e

uma instância de *Semantico*, e conforme aceita *tokens* que estejam de acordo com a especificação das produções da gramática, quando aplicável, aciona o método *executeAction* do objeto da classe *Semantico*, fornecendo o token lido e o número da ação a ser executada.

O que define o momento em que *Sintatico* acionará o método *executeAction* são as *ações semânticas* definidas na gramática, como na seguinte forma:

```
<nao_terminal> ::= <simbolo> \#1
```

A marcação *#1* significa que, durante a avaliação da produção, após a aceitação de <*simbolo*>, *Semantico* será acionado para executar a ação de número 1 com o último *token* reconhecido.

A assinatura de *executeAction* é a seguinte:

```
public void executeAction(int action, Token token);
```

O GALS gera apenas um corpo simples para *executeAction*, que imprime na tela a ação chamada e o *token* fornecido. Deve-se então reimplementar o método, de forma que ele execute ações significativas.

A implementação para selecionar a ação a ser executada é bem simples. como *action* é um *int*, pode-se utilizar uma estrutura *switch/case*. Como boa prática de programação, cada *case* apenas chama um outro método, implementado em *Semantico*, que cuida da real execução da ação semântica.

Como estratégia para gerar as *queries* SQL, criou-se uma classe *Query*, dedicada a receber as informações sobre a *query* dada em WF-QL (o que foi selecionado, quais domínios foram solicitados e quais as restrições dadas pela cláusula *where*). Desta forma, as ações executadas por *Semantico* compõem o objeto *Query* com as informações necessárias. Após o *parse*, pode-se obter o objeto *Query* através do método *query* de *Semantico*.

Query possui uma lista de domínios, uma lista de selecionáveis (as informações selecionáveis dos formulários conforme descrito na proposta da gramática e definido na produção *selectable*) e uma estrutura de árvore com nodos que implementam uma interface *CriteriaTreeNode* capaz de gerar o SQL necessário para as operações de conjuntos entre seus filhos, adequada à implementação da operação booleana, ou seja:

- *And*: Intersecção
- *Or*: União
- *Not*: Complemento

A própria definição da cláusula *where* na gramática como uma gramática de operadores cria a base para a criação de uma estrutura de árvore,

inclusive especificando a precedência dos operadores, com *not* avaliado primeiro, seguido de *and* e por fim *or*, com parênteses possibilitando a descrição de operações com precedência alterada.

Os critérios de seleção, por sua vez, são implementações de uma mesma interface *Criterion*. Para cada um dos possíveis critérios, deve existir uma implementação especializada capaz de agrupar metadados do critério e ao fim, gerar o SQL necessário para aquela operação.

Por fim, *Query* disponibiliza o método *generateSQL*, que computa e retorna a *query* SQL, composta de acordo com as informações obtidas durante o *parsing*, utilizando *subqueries* para garantir as restrições dadas e operações de *join* para modelar as operações Booleanas sobre as restrições.

As operações de similaridade podem ser geradas utilizando *stored procedures* do MySQL que provenham tais funções. As ações semânticas prevêem este uso, coletando o nome da função e o threshold a serem usados. Outra possibilidade, utilizada nas consultas mostradas no próximo capítulo, é consultar o cluster já provido no banco de dados do WF-SIM.

A estrutura do processador é descrita pelo diagrama apresentado na figura 4.

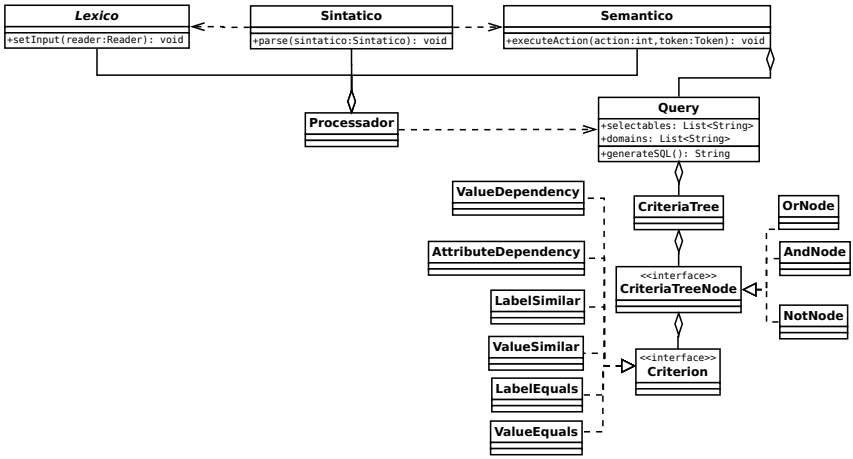

Figura 4: Diagrama de classes do processador da WF-QL.

5.2.2 Ações Semânticas

As ações semânticas necessárias serão aqui descritas de acordo com sua posição na gramática, com suas operações necessárias.

```
<statement> ::= <set> #5 ;

<set> ::= SET DEFAULT <setAux> ;

<setAux> ::= <function> #1 <thresholdDef> | #2 <thresholdDef> ;

<thresholdDef> ::= THRESHOLD <thresholdValue> #3| î #4 ;
```

- #1: Coletar o nome da função a ser utilizada.
- #2: Manter a função atual.
- #3: Coletar o valor novo de threshold.
- #4: Manter o valor atual de threshold.
- #5: Submeter as mudanças para o processador, para uso em ações subsequentes.

```
<statement> ::= <query> ;

<query> ::= <select> <from> <where> ;

<select> ::= SELECT <selectList> ;

<selectList> ::= STAR #6 | <selectable> #7 <selectListAux> ;

<selectListAux> ::= COMMA <selectList> | î ;

<selectable> ::= TITLE | URL | LABELS | VALUES | ATTRIBUTES ;

<from> ::= FROM <fromList> ;

<fromList> ::= <domain> <fromListAux> ;

<fromListAux> ::= COMMA <fromList> | î ;

<domain> ::= ID #8 <domainAux> ;
```

```
<domainAux> ::= ID #9 | AS ID #9 | î ;

<where> ::= WHERE #11 <criteria> #28 | î #12 ;
```

- #6: Define que todos os selecionáveis serão utilizados.
- #7: Coleta selecionáveis para a lista em *Query*.
- #8: Coleta o nome de um domínio a ser utilizado por #10.
- #9: Define um sinônimo para o domínio a ser utilizado.
- #10: Submete o domínio e o seu possível sinônimo para *Query*, verificando possíveis conflitos de sinônimos.
- #11: Inicia a criação da árvore de operadores.
- #12: Cria a árvore de operadores como vazia.
- #28: Termina a construção da árvore.

```
<criteria> ::= <compoundCritOr> ;

<compoundCritOr> ::= <compoundCritAnd> <compoundCritOrAux> ;

<compoundCritOrAux> ::= OR <compoundCritOr> #13 | î ;

<compoundCritAnd> ::= <notCrit> <compoundCritAndAux> ;

<compoundCritAndAux> ::= AND <compoundCritAnd> #13 | î ;

<notCrit> ::= <booleanPrimary> #16 | NOT <booleanPrimary> #15 ;
```

- #13: Cria um nodo OR.
- #14: Cria um nodo AND.
- #15: Cria um nodo sem operação para conter o resultado de <*booleanPrimary*>.
- #16: Cria um nodo NOT contendo o resultado de <*booleanPrimary*>.

```
<booleanPrimary> ::= LPAREN  <criteria> RPAREN
                   | <labelCriteria> #27
                   | <valueInFieldCriteria> <enablesCriteria> ;
```

```
<labelCriteria> ::= LABELS #17 <labelCriteriaAux> ;

<labelCriteriaAux> ::= <labelCriteriaFac>
                     | CONTAIN <labelCriteriaFac> ;

<labelCriteriaFac> ::= #18 <labelList>
                     | <similar> #19 <labelList> ;

<labelList> ::= <label> #20 | LPAREN <labelListAux> RPAREN ;

<labelListAux> ::= <label> #20 <labelListAuxFac> ;

<labelListAuxFac> ::= COMMA <labelListAux> | î ;

<label> ::= STRING ;

<valueInFieldCriteria> ::= <label> #21 <valueInFieldCriteriaFac> ;

<valueInFieldCriteriaFac> ::= <relational> #22 <valueFac>
                            | #23 <similar> <valueFac> ;

<valueFac> ::= <value> #24 | <valueList> ;

<relational> ::= LT | GT | LTE | GTE | EQ | NEQ ;

<valueList> ::= LPAREN <valueListAux> RPAREN ;

<valueListAux> ::= <value> #24 <valueListAuxFac> ;

<valueListAuxFac> ::= COMMA <valueListAux> | î ;

<value> ::= STRING ;

<similar> ::= SIMILAR TO <similarFac> ;

<similarFac> ::= LT #26 <similarConfigFac> GT | î ;

<similarConfigFac> ::= WITH <function> #1 <thresholdDef>
                     | #2 THRESHOLD <thresholdValue> #3 ;
```

```
<enablesCriteria> ::= ENABLES <valueInFieldCriteria> #28 | î #29 ;
```

- Para o critério <labelCriteria>:
  - #17: Inicia a coleção de dados para o critério.
  - #18: Define que o critério será de comparação exata e define qual o comparador.
  - #19: Define que o critério será de comparação por similaridade.
  - #20: Coleta os nomes de labels a serem buscados.
- Para o critério <valueInFieldCriteria>:
  - #21: Inicia a coleção de dados para o critério e coleta o rótulo a ser utilizado.
  - #22: Define que a comparação será exata ou numérica e coleta o operador.
  - #23: Define que a comparação será por similaridade.
  - #24: Coleta os valores a serem utilizados.
- Para o operador de similaridade:
  - #26: Inicia a coleção de parâmetros para a comparação (função e *threshold*)
- Ao final da definição do critério:
  - #27: Submete os dados coletados para a árvore como um critério de rótulo, por similaridade ou não.
  - #28: Submete os dados como um critério de habilitação, composto por dois critérios de valor em atributo.
  - #29: Submete os dados como um critério de valor em atributo.

6 RESULTADOS

Este capítulo apresenta exemplos da WF-QL, suas expressões em SQL para o banco de dados reprojetado como descrito no capítulo 5, e os resultados obtidos com a execução das consultas.

Estas consultas foram geradas utilizando a estratégia de clusterização já existente no WF-SIM, portanto não aplicam funções específicas de similaridade.

Como primeiro exemplo, uma consulta sem critérios de seleção, buscando todos os webforms em *airfare*:

```
select url from airfare
```

Expressa em SQL:

```
select
    w.url
from
    Webforms w,
    Domains d
where
    w.domain = d.id
        and d.domain_name = 'airfare'
```

Resultando em:

http://64.201.186.248/cheaptickets/index/default.asp?product=abc
http://a-la-carte-vacations.com/airline-tickets/mexico/tuxtla-gutierrez-chiapas.html
http://a-zvacations.com/world/north-america/united-states/nevada/las-vegas/index.html
http://a-zvacations.com/world/north-america/united-states/nevada/las-vegas/index.html
http://abs.travel/airlines/american.shtml
http://aero-california.us/
http://aerohondurasairlines.com/
http://afaf.com/

Tabela 2: Parte dos resultados da consulta exemplo 1.

A seguinte consulta, buscando todos os formulários em airfare que contenham um campo de rótulo similar a *children*:

```
select url from airfare where labels contain similar to "children"
```

Expressa em SQL assume esta forma:

```
select
    url
from
    Domains d,
    Webforms w
        join
    (select
        a.webform
    from
        Attributes a, Labels l, Clusters c
    where
        a.label = l.id and l.cluster = c.id
            and c.cluster_name = 'children') a
    on w.id = a.webform
where
    w.domain = d.id
        and d.domain_name = 'airfare'
```

Fornecendo os seguintes resultados:

http://airfare-cheap-flights.com/
http://airfares.sentienttechnology.com/default.aspx?aid=10&agency=atn
http://booking.buraqair.com/requirements.asp
http://discount-airline.hit.bg/
http://flyforless.ca/
http://gigablast.com/get?q=flights&c=main&rtq=0&d=20531273469
...

Tabela 3: Parte dos resultados da consulta exemplo 2.

A mesma consulta com a operação de comparação exata:

```
select url from airfare where labels contain "children"
```

Em SQL é expressa da seguinte forma:

```
select
    url
from
    Domains d,
    Webforms w
        join
    (select
        a.webform
    from
        Attributes a, Labels l
    where
        a.label = l.id and l.text = 'children') a
    on w.id = a.webform
where
    w.domain = d.id
        and d.domain_name = 'airfare'
```

Fornecendo os seguintes resultados:

http://booking.buraqair.com/requirements.asp
http://gigablast.com/get?q=flights&c=main&rtq=0&d=20531273469
http://hotels.thebostonguide.com/airlines/search.jsp?cid=78677
http://hotels.thecharlestonguide.com/airlines/search.jsp?cid=81193
http://reservations.hoteldiscounts.cc/nexres/air/destinations.cgi?src=10016137&src
http://specials.worldsbestdeals.com/airlines/search.jsp?cid=46742
...

Tabela 4: Parte dos resultados da consulta exemplo 3.

Utilizando a busca por valor de atributo, onde um campo rotulado "children" admite o valor numérico 2:

```
select url from airfare where "children" = 2
```

Consegue-se a expressão em SQL:

```
select
    url
from
    Domains d,
    Webforms w
        join
    (select
        a.webform
    from
        Attributes a
    join (select
            v.attribute
        from
            final_bd_tcc.Values v
        where
            v.text_content = '2') v
        on v.attribute = a.id, Labels l
    where
        a.label = l.id and l.text = 'children') a
    on w.id = a.webform
where
    w.domain = d.id
        and d.domain_name = 'airfare'
```

Resultando em:

http://specials.worldsbestdeals.com/airlines/search.jsp?cid=46742
http://travelhero.com/
http://www.airticketsforcheap.net/
http://www.asia.com/asia
http://www.ata.com/home.html
http://www.cancunwired.com/
...

7 CONCLUSÕES

Este trabalho apresenta a proposta da linguagem WF-QL, que tem o objetivo de dar suporte às ideias discutidas previamente, e provê uma possível implementação desta.

Após todo o processo de desenvolvimento e refinamento da gramática e da tentativa de implementação utilizando somente o banco de dados suprido pelo projeto WF-SIM, nota-se claramente que para uma implementação eficiente e completa das ideias propostas seriam necessários outros recursos, como por exemplo *stored procedures* que possibilitem a aplicação de funções específicas de similaridade e uma representação direta dos formulários como uma estrutura contendo os selecionáveis, de forma a permitir uma implementação mais direta da seleção. Ainda assim, a viabilidade do desenvolvimento e a expressividade da linguagem se mostrou coerente às expectativas, demonstrando esta como uma proposta promissora para refinamentos e extensões futuras.

7.1 TRABALHOS FUTUROS

Para a viabilização do uso da linguagem como ferramenta de trabalho, são necessários esforços para a otimização do processo de busca. Isso poderia ser feito buscando otimizar as consultas SQL geradas. É necessária também a implementação de *stored procedures* para o SGBD MySQL, para possibilitar o uso das funções de similaridade, aceitando como parâmetro o *threshold*.

Habilitar a parte que permite a definição de parâmetros para as operações de similaridade também envolveria a criação de um processador ativo, ou a utilização de alguma extensão do SGBD MySQL que permita o uso de consultas por similaridade. Por questões de eficiência e de extensibilidade, o processador ativo seria uma opção mais interessante.

Outro trabalho necessário é a validação da linguagem com um estudo da usabilidade, para validar o design da linguagem como familiar e amigável. Também faz-se necessária a análise de revocação/precisão dos resultados obtidos para validar o processo de geração de consultas SQL.

REFERÊNCIAS BIBLIOGRÁFICAS

BARBOSA, L.; FREIRE, J. An adaptive crawler for locating hidden-web entry points. p. 441–450, 2007.

BARKER, J. *Invisible Web: What it is, Why it exists, How to find it, and its inherent ambiguity*. 2004. <http://www.lib.berkeley.edu/TeachingLib/Guides/Internet/InvisibleWeb.html>. Acessado em 02/04/2013.

BERGMAN, M. K. The deep web: Surfacing hidden value. 2001.

BORGES, E. N.; DORNELES, C. F. PgSimilar: Uma ferramenta open source para suporte a consultas por similaridade no PostgreSQL. *III SESSÃO DE DEMOS / XXI SIMPÓSIO BRASILEIRO DE BANCO DE DADOS - SBBD*, p. 1–6, 2006.

DORNELES, C. F.; GONÇALVES, R.; MELLO, R. d. S. Approximate data instance matching: a survey. *Knowledge and Information Systems*, v. 27, n. 1, p. 1–21, 2011.

GESSER, C. E. Ambiente para geração de analisadores léxicos e sintáticos. *Monografia para obtenção de grau de Bacharel em Ciências da Computação, UFSC*, 2002.

GONÇALVES, R. et al. A similarity search method for web forms. *IADIS International Conference IADIS WWW/Internet*, p. 381–387, 2011.

MELLO, R. d. S.; PINNAMANENI, R.; FREIRE, J. Indexing web form constraints. *JIDM*, v. 1, n. 3, p. 343–358, 2010.

NTOULAS, A.; ZERFOS, P.; CHO, J. Downloading textual hidden web content through keyword queries. p. 100–109, 2005.

SHESTAKOV, D.; BHOWMICK, S.; LIM, E.-P. DEQUE: querying the deep web. *Data Knowl. Eng.*, v. 52, n. 3, p. 273–311, 2005.

SILVA, M. E. V. da; BORGES, E. N.; GALANTE, R. d. M. XSimilarity : Uma ferramenta para consultas por similaridade embutidas na linguagem XQuery. *IV ESCOLA REGIONAL DE BANCOS DE DADOS - ERBD*, 2008.

ANEXO A – Gramática da WF-QL

```
statement = set | query

set ::= SET DEFAULT (function)? (THRESHOLD thresholdValue)?

function ::= ID

thresholdValue ::= DOUBLE

query ::= SELECT DOMAINS | select from (where)?

select ::= SELECT (STAR | (selectable (COMMA selectable)*))

selectable :: TITLE | URL | LABELS | VALUES | ATTRIBUTES

from ::= FROM (domain (COMMA domain)*)

domain ::= (ID ((AS)? ID)?)

where ::= WHERE criteria

criteria ::= compoundCritOr

compoundCritOr ::= compoundCritAnd (OR compoundCritAnd)*

compoundCritAnd ::= notCrit (AND notCrit)*

notCrit ::= (NOT)? booleanPrimary

booleanPrimary ::= labelCriteria | valueInFieldCriteria
                 | enablesCriteria

labelCriteria ::= LABELS (CONTAIN)? (similar)? labelList

labelList ::= label | RPAREN label (COMMA label)* LPAREN

label ::= STRING

valueInFieldCriteria ::= label (relational | similar)
                         (value | value_list)

relational ::= LT | GT | LTE | GTE | EQ | NEQ
```

```
valueList ::= RPAREN value (COMMA value)* LPAREN

value ::= STRING

similar ::= SIMILAR TO (LT (WITH function)? (THRESHOLD
                                                THRESHOLD_VALUE)?
                        GT)?

enablesCriteria ::= valueInFieldCriteria ENABLES valueInFieldCriteria
```

ANEXO B – Estrutura do Banco de Dados original do projeto

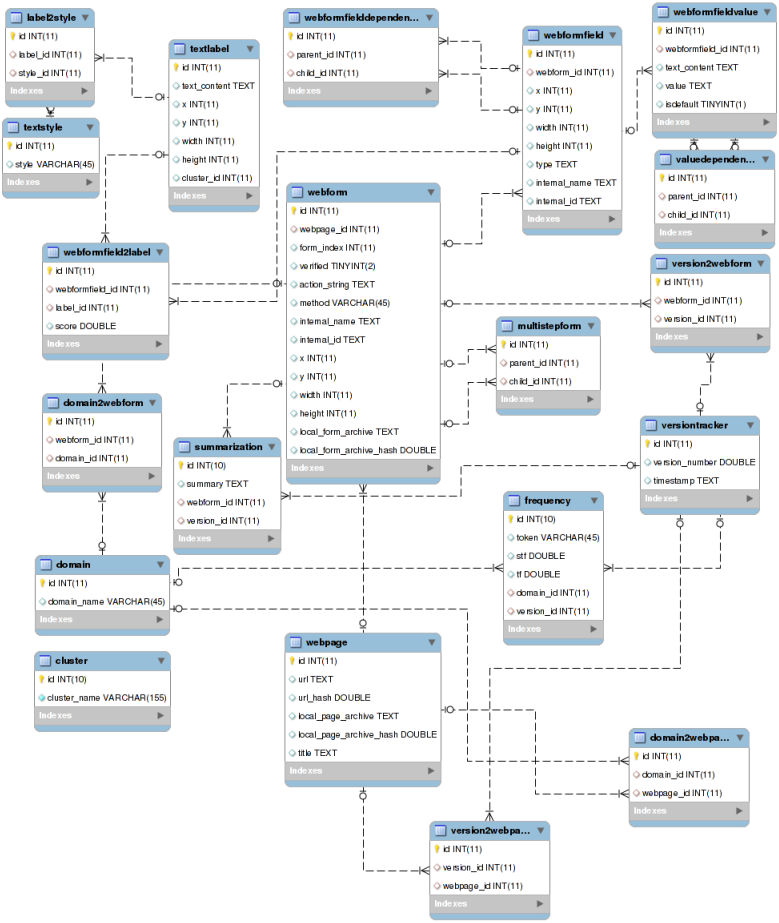

Figura 5: Diagrama ER do Banco de Dados original.

ANEXO C – Estrutura do Banco de Dados reprojetado

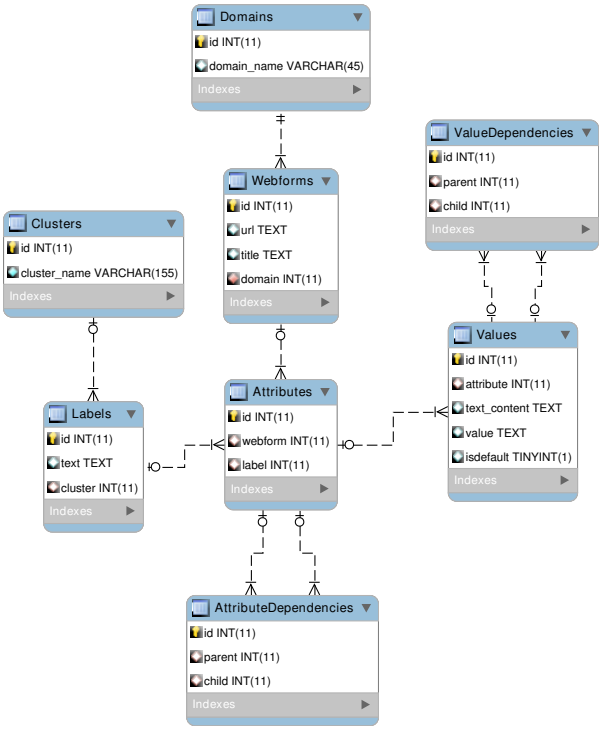

Figura 6: Diagrama ER do Banco de Dados reprojetado.

ANEXO D – Consultas SQL para o reprojeto do Banco de Dados

```
create table refactored.AttributeDependencies
engine = InnoDB as
(select wffd.id,
wffd.parent_id as parent,
wffd.child_id as child
from original.webformfielddependency wffd)

create table refactored.Attributes engine = InnoDB as
(select wff.id,
wff.webform_id as webform,
wff2l.label_id as label
from original.webformfield as wff
left join
original.webformfield2label as wff2l
on
wff2l.webformfield_id = wff.id)

create table refactored.Clusters engine = InnoDB as
(select * from original.cluster)

create table refactored.Domains as
(select d.id,
d.domain_name
from original.domain d)

create table refactored.Labels engine = InnoDB as
(select id,
text_content as text,
cluster_id as cluster
from original.textlabel)

create table refactored.ValueDependencies
engine = InnoDB as
(select wfvd.id,
wfvd.parent_id as parent,
wfvd.child_id as child
from original.valuedependency wfvd)

create table refactored.Values
engine=InnoDB as
(select wffv.id,
```

```
wffv.webformfield_id as attribute,
wffv.text_content,
wffv.value,
wffv.isdefault
from original.webformfieldvalue wffv)

create table refactored.Webforms engine=InnoDB as
(select wf.id,
wp.url,
wp.title,
d.id as domain
from original.webform wf,
original.webpage wp,
original.domain d,
original.domain2webform d2wf
where (wf.id = d2wf.webform_id and
d.id = d2wf.domain_id)
and wf.webpage_id = wp.id)
```

ANEXO E – Especificação Léxica no formato utilizado pelo GALS

Definições regulares:

```
ALPHA : [a-zA-Z]
NUM : [0-9]
```

Tokens:

```
D : {ALPHA}({ALPHA}|NUM)*
INTEGER : \-?{NUM}*
AND = ID : "and"
AS = ID : "as"
ATTRIBUTES = ID : "attributes"
CONTAIN = ID : "contain"
DEFAULT = ID : "default"
ENABLES = ID : "enables"
FROM = ID : "from"
LABEL = ID : "label"
LABELS = ID : "labels"
NOT = ID : "not"
OR = ID : "or"
SELECT = ID : "select"
SET = ID : "set"
SIMILAR = ID : "similar"
THRESHOLD = ID : "threshold"
TITLE = ID : "title"
TO = ID : "to"
URL = ID : "url"
VALUES = ID : "values"
WHERE = ID : "where"
WITH = ID : "with"
COMMA : ,
EQ : =
GT : >
GTE : >=
LPAREN : \(
LT : <
LTE : <=
NEQ : <>
RPAREN : \)
STAR : \*
STRING : \"[^\"]*\"
: [\s\t\n\r]
```

ANEXO F – Especificação sintática em BNF e fatorada para a geração de um parser LL(1).

```
<statement> ::= <set> | <query> ;

<set> ::= SET DEFAULT <setAux> ;

<setAux> ::= <function> <thresholdDef> | <thresholdDef> ;

<thresholdDef> ::= THRESHOLD <thresholdValue> | î ;

<function> ::= ID ;

<thresholdValue> ::= DOUBLE ;

<query> ::= SELECT <selectAux> ;

<selectAux> ::= <selectList> <from> <where> | DOMAINS ;

<selectList> ::= STAR | <selectable> <selectListAux> ;

<selectListAux> ::= COMMA <selectList> | î ;

<selectable> ::= TITLE | URL | LABELS | VALUES | ATTRIBUTES ;

<from> ::= FROM <fromList> ;

<fromList> ::= <domain> <fromListAux> ;

<fromListAux> ::= COMMA <fromList> | î ;

<domain> ::= ID <domainAux> ;

<domainAux> ::= ID | AS ID | î ;

<where> ::= WHERE <criteria> | î ;

<criteria> ::= <compoundCritOr> ;

<compoundCritOr> ::= <compoundCritAnd> <compoundCritOrAux> ;

<compoundCritOrAux> ::= OR <compoundCritOr> | î ;

<compoundCritAnd> ::= <notCrit> <compoundCritAndAux> ;
```

```
<compoundCritAndAux> ::= AND <compoundCritAnd> | î ;

<notCrit> ::= <booleanPrimary> | NOT <booleanPrimary> ;

<booleanPrimary> ::= LPAREN  <criteria> RPAREN | <labelCriteria>
                   | <valueInFieldCriteria> <enablesCriteria> ;

<labelCriteria> ::= LABELS <labelCriteriaAux> ;

<labelCriteriaAux> ::= <labelCriteriaFac>
                     | CONTAIN <labelCriteriaFac> ;

<labelCriteriaFac> ::= <labelList> | <similar> <labelList> ;

<labelList> ::= <label> | LPAREN <labelListAux> RPAREN ;

<labelListAux> ::= <label> <labelListAuxFac> ;

<labelListAuxFac> ::= COMMA <labelListAux> | î ;

<label> ::= STRING ;

<valueInFieldCriteria> ::= <label> <valueInFieldCriteriaFac> ;

<valueInFieldCriteriaFac> ::= <relational> <valueFac>
                            | <similar> <valueFac> ;

<valueFac> ::= <value> | <valueList> ;

<relational> ::= LT | GT | LTE | GTE | EQ | NEQ ;

<valueList> ::= LPAREN <valueListAux> RPAREN ;

<valueListAux> ::= <value> <valueListAuxFac> ;

<valueListAuxFac> ::= COMMA <valueListAux> | î ;

<value> ::= STRING ;

<similar> ::= SIMILAR TO <similarFac> ;
```

```
<similarFac> ::= LT <similarConfigFac> GT | î ;

<similarConfigFac> ::= WITH <function> <thresholdDef>
                     |  THRESHOLD <thresholdValue> ;

<enablesCriteria> ::= ENABLES <valueInFieldCriteria> | î ;
```