WebSSQL — A Query Language for Multimedia Web Documents

Changqing Zhang, Weiyi Meng, Zhongfei Zhang, Zonghuan Wu
Department of Computer Science
State University of New York at Binghamton
Binghamton, NY 13902
Email: {meng,zhongfei}@cs.binghamton.edu Fax: (607) 777-4729

Abstract

In this paper, we describe an SQL-like query language — WebSSQL — for retrieving desired web pages. WebSSQL has several unique features. First, WebSSQL assumes that each web page is a multimedia document consisting of structured data, text data and possibly image data. Second, WebSSQL treats each page as a node in a directed graph composed of many web pages and links among them. Third, WebSSQL is similarity-based meaning that the retrieved web pages will be ranked based on their closeness to a given query. The traditional SQL does not support similarity-based retrieval and ranking. With WebSSQL, users can specify their search needs more precisely, leading to more accurate retrieval of useful information.

1 Introduction

The World Wide Web contains a large, heterogeneous, distributed and hyper-linked collection of documents. As more and more web pages are added to the Web, how to effectively extract the right information becomes a serious problem. To help users find information on the Web, many search engines such as HotBot and AltaVista have been created.

Most existing search engines are strictly keywords based. A typical user query submitted to such a search engine consists of one or more words with possibly some Boolean operators. The search result is a list of web pages (or their URLs) in descending similarities (ranking scores) with the query. A common problem with these search engines is that often too many web pages are retrieved for each query and most of them are not relevant to the query. A main cause for this problem is that user queries are not sufficiently precise. One possible solution to this problem is to enrich the queries so that user desired information can be expressed more precisely.

A careful examination of web pages reveals that in addition to words that appear in each web page, there is also other related information that could be used to more precisely describe a user's information needs. Such information includes (1) well defined (structured) information about each web page such as its URL and title; (2) metadata associated with each web page such as its size and the time it is last modified; (3) images in a web page; and (4) the links that connect different web pages and images. If all of the above information is available, then in addition to using keywords, a user query can also restrict the search to those web pages whose URLs contain a specific string (say Boeing), are modified recently (say within 3 months), contain an image of an airplane, and are linked to a page that matches a given description well. Clearly, these additional pieces of information, if available, can be really useful to describe users' information needs. Unfortunately, to a large extent, most existing search engines do not make use of such information. We believe that advanced applications such as digital libraries for hyperlinked multimedia documents can benefit from more advanced query languages.

In this paper, we propose a new query language — WebSSQL — for specifying queries against hyperlinked multimedia web documents. WebSSQL has several unique features. First, WebSSQL assumes that each web page is a multimedia document consisting of structured data, text data and possibly image data instead of purely text object. Second, WebSSQL treats each page as a node in a directed graph composed of many web pages and links among them instead of treating each page as an isolated object. Third, WebSSQL is similarity-based meaning that the retrieved web pages will be ranked based on their closeness to a given query.

Recently, there has been active research in developing data models and query languages for structured and semi-structured documents (e.g., HTML/XML pages) [1, 16, 20]. These models and query languages explore internal structures of documents and are mainly interested in finding rather specific information within documents. In contrast, we are primarily interested in finding information at a higher level, namely, documents and images. W3QL [9] is an SQL-like query language designed for the Web. It emphasizes more on utilizing Unix utilities and supporting dynamic view maintenance. No database tables are utilized by W3QL. The work that is closest to our work is WebSQL [14] which also uses database tables to support the specification of queries. In fact, the information in the tables used in WebSQL is nearly identical to that in two of our three database tables. The main difference between WebSQL and WebSSQL is that the latter supports similarity-based retrieval while the former does not. Full-text similarity is the basis for most current search engines on the Web and we believe that supporting this capability is critical. As a result, unlike WebSQL and W3QL, our approach can rank retrieved documents (or images) based on how well they match the user query like most current search engines do. Other differences between WebSQL and WebSSQL are (1) WebSQL does not maintain any web page database by itself while our approach is based on pre-fetched data (like the indexes on documents in most search engines); (2) WebSSQL treats images as an important type of data while WebSQL does not consider images. WebSSQL not only allows users to retrieve images but also gives users the ability to specify conditions on images. A method that combines textual and visual information for retrieving images on the Web is reported in [5]. However, no query language is proposed in this paper.

The rest of this paper is organized as follows. In Section 2, we describe the data model based on which WebSSQL is defined. In Section 3, we introduce two types of basic operators that are essential to our query language. In Section 4, we present WebSSQL. Both the syntax and the semantics of WebSSQL will be discussed. In Section 5, we describe the status of the implementation. We conclude the paper in Section 6.

2 Data Model

2.1 Web as a Labeled Directed Graph

The World Wide Web can be modeled as a gigantic labeled directed graph $G(V, E)$. A vertex in V represents a web object (a document or an image) and is identified by the URL of the object. A directed edge $e \in E$ from vertex v_1 to vertex v_2 represents a hyperlink from the web object corresponding to v_1 to the object corresponding to v_2. The label on e is the **anchor** information associated with the hyperlink.

In principle, web objects can also be of non-text and non-image types such as audio/video clips and other application documents like postscript files or Microsoft Word documents. In this paper, we will focus only on text documents (HTML or plain text) and images but our framework is extensible to cover other types of objects. HTML pages have internal structures as defined by various HTML tags. These tags can often convey rich semantics about web pages and they are extensively explored by semi-structured data model (for example, see [1, 16, 20]). In our model, only very limited internal structural information (e.g., title) will be used as our current focus is on the retrieval of web objects rather than information inside them.

Hyperlinks provide relationships among otherwise isolated web objects. An HTML document may have zero or more out-going links to other web objects and may be linked to by many other web pages. Plain text documents and images usually do not link to other web objects. Some newer search engines such as Google [17] have made use of the link information to obtain better retrieval effectiveness.

Usually, when the author of web page p adds a hyperlink to another web page q, the author also includes in the corresponding *anchor* tag a description of q in addition to the URL. These descriptions of q from the authors of other web pages pointing to q have the potential of being very important for the retrieval of q because they includes the perception of these authors about the contents of q. Anchor information is used by several search engines [3, 6, 13]. The anchor information is captured by the label in the graph definition.

Webpages: url, title, text, size, type, server_type, fetch_time, last_modified

Images: url, title, image, description, color, texture, size, type, server_type, fetch_time, last_modified

Links: url, child_url, label

Figure 1: Table Representation of the Web

2.2 Table Representation of the Web Graph

In order to achieve reasonable retrieval efficiency for each user query, we choose to follow the approach of most search engines by preprocessing the web database and creating various indexes in advance before any query is processed. An indexing robot is used to fetch web pages and images of interest to a local site. From the fetched data, three tables are created to store information about each web object and the link relationships between different web objects (see Figure 1). These tables provide the basis for WebSSQL. In other words, WebSSQL is a query language for specifying queries against the three tables.

Table **Webpages** contains one record for each text type web object. Most data in each record are provided by the HTTP protocol employed by the robot. The url of each page serves as the identity of the page. The title is extracted from the title tag of the page. The size indicates the number of bytes in the page. Its type can be html, plain text, etc. The fetch_time and last_modified indicate the dates and times when the page is fetched and last modified, respectively. The "text" of a web page *logically* represents the contents of the page. It may be understood as a vector representation of the web [25, 28] although physically the web page itself is not stored in the record. When a page is fetched from the Web, needed index information will be extracted but the page will not be kept. During retrieval, web pages desired by a user can be fetched using their URLs. Table **Images** contains one record for each image. The description is a text description of the image. The description of each image can be obtained either manually or automatically using texts associated with/related to the image such as caption. Field "image" logically stores the image (an image file id) and the image itself is not stored in the record. For a given image, fields "color" and "texture" store information concerning different features of the image. Field "type" indicates the format (e.g., gif, jpg) the image is in. Again, when an image is fetched from the Web, needed feature information will be extracted but the image itself will not be kept. Table **Links** stores the link structure of the labeled directed graph. Each record represents a directed edge from the web object corresponding to "url" to the web object corresponding to "child_url" with label "label" (i.e., the anchor text). In our modelling, when an image appears in a web page, we treat the image as a child of the web page. As a result, there would be a directed edge from the web page to the image.

3 Two Basic Types of Operations

WebSSQL extends the traditional SQL in primarily two aspects. The first is the capability to deal with the link structures of the Web and the second is to support similarity-based retrieval of information. In Section 3.1, a set of link-related operators are introduced to cover the first aspect. In Section 3.2, a special operator (**similar_to**) is proposed to address the second aspect.

3.1 Link Operations

In [14], the authors proposed a rich set of notations to denote different types of links. We decide to adopt some of these link notations in WebSSQL with some modifications and extensions to suit our environment. Among different types of links, "=" is used to denote an empty link and $\rightarrow$ is used to denote a real hypertext link. As an example, if o_i and o_j are two web objects, then $o_i \rightarrow o_j$ indicates that o_i has a hyperlink pointing to o_j.

We extend $\rightarrow$ by permitting two optional parameters. The first is the *distance* parameter used to indicate the number of links in a directed path connecting two web objects. The fact that o_j is at least one click away but no more than three clicks away from o_i by following at least one path can be denoted by $o_i \rightarrow(3)$ o_j. Without the extension, the same condition can be expressed as $(\rightarrow)(= \mid \rightarrow)(= \mid \rightarrow)$, where $\mid$ is the *alternation* operation. Note that there may be multiple directed paths from one object to another object and these paths may have different lengths (i.e., number of links). In general, "$o_i \rightarrow$ (n) o_j" means that the shortest path from o_i to o_j contains at least one but no more than n links. The second parameter is the *quantifier* parameter. One of the following two values can be used: (1) *all*, representing the universal quantifier and (2) *some*, representing the existential quantifier. As an example, the predicate that every URL of a web object within two clicks away from o_i contains "binghamton.edu" can be specified as "$o_i \rightarrow$(2, all) o such that (o.url contains 'binghamton.edu')". (The discussion on this type of predicates will be provided in Section 4.) When "all" is replaced by "some" in this example, the predicate is for specifying that at least one URL of a web object within two clicks away from o_i contains "binghamton.edu".

3.2 The similar_to Operator

Several fields in the three tables described in Section 2.2, such as "text" and "color", can support both binary (true or false) and similarity-based predicates. First, let us see an example for specifying a binary predicate. For a given object o, predicate *o.text contains 'solar system'* is a binary predicate because it will be evaluated to either true or false. Often, using only binary predicates against text and image objects are not sufficient to express users' information needs. Users often need to find documents that are *similar to* a text description or find images whose color distribution is similar to that of a known image. To accommodate these practical needs, we propose to incorporate the **similar_to** operation into WebSSQL. In Section 3.2.1, we discuss different variations of the similar_to operation against text objects. In Section 3.2.2, we discuss this operation in the context of comparing image objects. In WebSSQL, similar_to is an overloaded operator. It has multiple interpretations and implementations. The correct interpretation and subsequently the invocation of the correct implementation of this operator in a particular query is determined based on the parameters used in the operator and the type of data against which the operator is used.

3.2.1 similar_to for Text

When similar_to is used for text data, it has the following general syntax:

$$text\_attribute\ similar\_to[(n) \mid (DomainList)$$

$$\mid (n, DomainList)]\ text\_constant \qquad (1)$$

where text_attribute indicates any attribute of text type such as *text*, *title* and *description*, and text_constant indicates a given constant text used for comparison. In WebSSQL, text_constant is denoted by words in double quotes. The parameters are optional. When parameters are absent, similar_to returns all values under the text_attribute that have a positive similarity with the text_constant. The corresponding similarity will also be returned. As an example, consider the following WebSSQL query (see Section 4 for the full syntax of WebSSQL):

```
Q1: select url
    from Webpages
    where text similar_to "digital library"
```

This query returns all web pages (actually their URLs) whose texts have positive similarities with "digital library" based on some similarity function. In the context of the vector space model [24], the texts of all web pages form a document collection and the text_constant "digital library" is a query. Each document is logically represented as a vector of *terms* with *weights* [24], where a term is essentially a content word and the dimension of the vector is the number of all distinct terms in the collection. When a term appears in a document, the component of the document vector corresponding to the term, which is the *term weight*, is positive; if it is absent, the corresponding term weight is zero. The weight of a term usually depends on the number of occurrences of the term in the document (relative to the total number of occurrences of all terms in the document) [24, 28]. This is the *term frequency weight.* The weight of a term may also depend on the number of documents having the term relative to the total number of documents in the database. The weight of a term based on such information is called the *inverse document frequency weight* [24, 28]. A query is similarly transformed into a vector with weights except only the term frequency information is used. The similarity between a query and a document can be measured by the dot product of their respective vectors. Often, the dot product is divided by the product of the *norms* of the two vectors. This is to normalize the similarity between 0 and 1. The similarity function with such a normalization is known as the *Cosine* function [24, 28]. Back to the above WebSSQL query (i.e., Q1). As a default, the returned web pages will be listed in descending order of the similarity values.

The two parameters of similar_to can be used separately or together. The first parameter, n, limits the web pages to be returned to those whose similarities with the text_constant are among the n largest. For example, if in the above query, similar_to is replaced by similar_to(5), then only the web pages that have the 5 largest similarities with "digital library" will be returned. Note that if all similarities are unique, then similar_to(n) returns no more than n web pages. Otherwise, more than n web pages may be returned. The second parameter, DomainList, is a list of domain names used to limit the web pages to be considered. A full domain name is simply the part of a URL after the protocol (e.g., http or ftp) and before the file path specification. For example, "www.binghamton.edu" is a domain name. *Partial domain names* are also allowed. Each domain name consists of several strings separated by a ".". A partial domain name is obtained when one or more leading strings together with their dots are removed from a full domain name. For example, "binghamton.edu" and "edu" are partial domain names of "www.binghamton.edu". A web page is said to be in a domain if its URL contains the domain name. In the following WebSSQL query, all web pages associated with Binghamton University or SUNY Buffalo that have a positive similarity with "database research" will be returned.

```
Q2: select url
    from Webpages
    where text similar_to(binghamton.edu,
          buffalo.edu) "database research"
```

When both parameters of similar_to are present, then the query can be interpreted as finding the web pages in specified domains whose similarities with text_constant are among the n largest. For example, the following query returns all web pages associated with Binghamton University or SUNY Buffalo that have the 5 largest similarities with "database research".

```
Q3: select url
    from Webpages
    where text similar_to(5, binghamton.edu,
          buffalo.edu) "database research"
```

3.2.2 similar_to for Imagery

We also use the similar_to operator to support similarity-based retrieval of images in WebSSQL. The **Images** table contains two types of information for each image. The first type concerns the metadata of the image, such as *url, title, size, type, fetch_time*, etc. The second type of information indicates the content features of the image, such as *description, color* and *texture*. The similar_to operator can be applied to image data of the second type as described below.

Description. This is a textual description of the image content. If an image comes with a caption and/or any collateral text, this is the field where the information is stored. If the image does not have any caption-like accompanying text, a manual annotation with a textual description about the image content may be applied, and the description is stored in this field (see Section 5 for more discussion on this). This field can be treated the same way as the *text* field in table Webpages (see Section 3.2.1). An example of querying this field is:

```
Q4: select i.url
    from Images i
    where i.description similar_to(10,
          binghamton.edu) "sky"
```

This query looks for all the urls of those images from the Binghamton University domain whose descriptions have the 10 largest similarities with "sky".

Color. This field stores image color features that may be used in conducting color-based similarity computation and matching. There are many proposed color features together with their corresponding indexing techniques and similarity matching algorithms existing in the literature. Examples of these techniques include color histograms [26], coherent color vectors [18], Correlograms [8], spatial histograms [21], and geometric histograms [22]. In principle, any of these techniques could be used to implement the color similarity matching function in WebSSQL. Suppose HVC color histogram [26] is used to implement the color similarity matching function in WebSSQL. An example query having a condition against the color field is below:

```
Q5: select i.url
    from Images i
    where i.color similar_to(10,
          binghamton.edu) foo.gif
```

This query is for finding the urls of those images from the Binghamton University domain whose color HVC histograms are among the 10 best matches to the HVC histogram of a given image file `foo.gif`.

In order to further allow users to describe the color features verbally, we can conduct a "principal component" analysis of the HVC color histogram vectors for each image and generate a *color description vector* for each image. A color description vector is a "qualitative" histogram vector and each component in the vector is of the format (c, p), where c is one of the eight qualitative color descriptions (*red, green, blue, white, black, yellow, purple, complex*) with *complex* denoting a complex color which cannot be simply described by any of the other seven colors, and p denotes the percentage of the given color among all colors of the image (in terms of color pixels). In WebSSQL, queries are allowed to specify only intended color components in the color description vector. For example, the following WebSSQL query is to find those images whose color features are among the 10 best matches with images that have 50% red color and 30% yellow color (and rest 20% may be arbitrary colors and thus are not specified).

```
Q6: select i.url
    from Images i
    where i.color similar_to(10)
          (red 0.5, yellow 0.3)
```

Texture. This field stores image texture features, if there are any, that can be used to conduct texture-based similarity computation and matching. Texture features are also widely used in image retrieval and there are many proposed techniques. Examples of texture-level feature extraction and matching techniques include non-parametric statistical testing [19], fractal texture transform [12], Gabor Wavelet transform [11], and color distribution moments [27], as well as Gaussian functions [23]. We plan to use Gaussian functions [23] to implement the texture similarity function [23]. Like color attributes, an image file with preferred texture needs to be provided in a query if a texture similarity is required. The following WebSSQL query is to find those images whose texture features are among the 10 best matches with the texture of a given image file `foo.jpg`.

```
Q7: select i.url
    from Images i
    where i.texture similar_to(10,
          binghamton.edu) foo.jpg
```

4 WebSSQL

WebSSQL is an SQL-like query language for a multimedia web search engine under development. Like SQL, WebSSQL has a basic three-clause structure. The *select-clause* indicates what is to be retrieved. Based on our objective, either web pages or images can be selected. In traditional SQL, "select" does not remove duplicate from the result unless the variation *select distinct* is used. In WebSSQL, select will remove duplicate so only unique web objects will be returned. In addition, if two copies of the same web object with different similarity values exist, then only the one with the larger similarity value will be returned. The *from-clause* lists the database tables or table variables that are to be involved in the query. The *where-clause* specifies the conditions to be satisfied by returned web pages (or images). Each web object in the result has an associated similarity computed by the where-clause. As a special case, when no similar_to operator is used, each object in the result will have a similarity of 1. As a default, the objects in the result will be listed in descending similarity values. In addition to the above three clauses, a fourth clause, *with-size*, is used in WebSSQL to indicate the maximum number of web objects to be returned. For a search engine of a large size, a large number web objects may satisfy a query to some extent (i.e., having a positive similarity). The user may choose to retrieve only the desired number of top ranked web objects (i.e., those with the highest similarities). When this clause is absent, all web objects with positive similarities will be returned.

We now use a number of examples of WebSSQL queries to illustrate the syntax and semantics of WebSSQL. The full syntax of WebSSQL will be provided in Section 4.1.

Example 4.1 Find all web pages that are similar to "web query language" and were last modified after May 1, 1999. The WebSSQL query is as follows:

```
select p.url
from Webpages p
where p.text similar_to "web query language"
      and p.last_modified > May-01-99
```

This query can be interpreted as follows. First, each condition can be evaluated independently. When a condition is evaluated, a similarity value will be assigned to each involved web object. If the condition does not involve the similar_to operator, then the similarity will either be 1, indicating that the condition is satisfied, or 0, indicating the condition is not satisfied. If the condition involves the similar_to operator, then a similarity function will be invoked to compute a similarity between 0 and 1 (see Section 3). Next, if a web object o satisfies the two conditions with similarities s_1 and s_2 respectively, then the overall/combined similarity of o is the minimum of the two similarities (i.e., $\min\{s_1, s_2\}$). Finally, the web pages with positive combined similarities are displayed with the ones with the largest similarities displayed first. ■

In general, the where-clause of a WebSSQL query may contain multiple conditions. Without loss of generality, we assume that the conditions are in *disjunctive normal form*, that is, conditions are first ANDed and then ORed. As an example, if A, B, C and D represent conditions, then "(A and B) or (C and D)" is in disjunctive normal form and "(A and B)" and "(C and D)" are called *conjunct subclauses*. In WebSSQL, we use the following rule to combine similarities:

Rule 1: Suppose the where-clause of a WebSSQL query has n conjunct subclauses $\{C_1, ..., C_n\}$ and subclause C_i contains k_i conditions $\{C_{i,1}, ..., C_{i,k_i}\}$, $1 \leq i \leq n$. If a web object o receives a similarity value $s_{i,j}$, $0 \leq s_{i,j} \leq 1$, due to condition $C_{i,j}$, $1 \leq i \leq n$, $1 \leq j \leq k_i$, then the combined similarity of o is defined to be

$$csim(o) = \max\{\min_{1 \leq j \leq k_1}\{s_{1,j}\}, ..., \min_{1 \leq j \leq k_n}\{s_{n,j}\}\} \quad (2)$$

The rationale behind the above rule is as follows. If a web object must satisfy several conditions at the same time, then the overall satisfaction of the object to these conditions should be based on its satisfaction to the most strict condition (i.e., the one yielding the smallest similarity for the object). As a special case, if one condition is not satisfied by the object (i.e., the corresponding similarity is zero), then overall the object does not satisfy these conditions. On the other hand, if a web object only needs to satisfy at least one of several conditions, then the overall satisfaction of the object to these conditions should be based on its satisfaction to the least strict condition (i.e., the one yielding the largest similarity for the object).

Example 4.2 Find each web page in the Binghamton University domain that has the 5 largest similarities with "computer science", whose title contains "department" and the web page has a child page whose text is similar to "digital library". A WebSSQL query for this statement is as follows:

```
select p.url
from Webpages p
where p.text similar_to(5, binghamton.edu)
      "computer science"
      and p.title contains "department"
      and p -->(some) p1 such that (p1.text
      similar_to "digital library")
```

Recall that p →(some) p1 specifies that page p has a child page p1. In this example, the parameter *some* is not necessary. The last predicate, called a *such-that predicate*, in the above query should be interpreted as follows. Note that p may have many child pages. For each child page p1 of p, the condition in the parentheses following the *such that* is evaluated, resulting in a similarity between p1.text and "digital library" to be computed in our example. One of the children, say p′, will have the largest similarity and this p′ represents the best child of p as far as matching "digital library" is concerned. This largest similarity achieved by p′ will be assigned to p after this predicate is evaluated. This query illustrates that descendant pages can be used as contextual information to qualify a page. ■

In general, we have the following rule for assigning similarities to a page when conditions on its descendant pages are involved:

Rule 2: Consider a query with a such-that predicate of the format "p →(n,x) p^c such that (condition-on-p^c)", where x is either *some*, *all* or absent. Suppose p has m descendants that satisfy p →(n) p^c and these descendants are p^c_1, ..., p^c_m. If p^c_i receives a similarity value s_i, $0 \leq s_i \leq 1$, due to the condition on p^c_i, $1 \leq i \leq m$, then the similarity assigned to p is $\max\{s_i\}$ if x = some or x is absent, or $\min\{s_i\}$ if x = all.

The reason for the above similarity assignment to p is because *some* has the semantics of *or* and *all* has the semantics of *and*. In Rule 1, max is used to combine similarities for *or* while min is used for *and*.

Note that the query in Example 2 is equivalent to the following query:

```
select p.url
from Webpages p, Webpages p1
where p.text similar_to(5, binghamton.edu)
      "computer science"
      and p.title contains "department"
      and p --> p1 and p1.text similar_to
      "digital library"
```

The reason is as follows. Consider a given page p. Based on Rule 1, for a given page p1, the combined similarity of p will be the minimum of the four similarities associated with the four conditions in the new query. For different p1, the combined similarity of p may be different. Since the first two conditions are independent of p1, it can be seen that if p1 is a child page of p and p1.text has the largest similarity with "digital library", then the combined similarity of p will reach the maximum. According to the semantics of the select clause of WebSSQL (for multiple copies of the same object, the one with the largest similarity will be returned), this is equivalent to evaluating p →(some) p1 such that (p1.text similar_to "digital library") based on the most similar p1. From this example, we can see that the use of a such-that predicate can improve the readability of the query. The above query is also equivalent to:

```
select p.url
from Webpages p, Links l, Webpages p1
where p.text similar_to(5, binghamton.edu)
      "computer science"
      and p.title contains "department"
      and p.url = l.url
      and l.child_url = p1.url and
      p1.text similar_to "digital library"
```

Now we see that link operations like → can simplify query specification substantially.

Example 4.3 Find each web page in the Binghamton University domain that has the 5 largest similarities with "computer science", whose title contains "department" and that has an image whose color histogram is similar to that of a given image bulogo.gif. A WebSSQL query for this statement is as follows:

```
select p.url
from Webpages p
where p.text similar_to(5, binghamton.edu)
      "computer science"
      and p.title contains "department"
      and p -->(some) p1 such that
      (p1.color similar_to bulogo.gif)
```

This query is very similar to that in Example 4.2 except that the condition on child pages has been changed to that on contained images. Recall that in Section 2 we mentioned that in our modelling method, an image contained in a web page is treated as a child object of the web page. The such-that predicate is used because a page may contain multiple images and we are interested in the one whose color distribution matches a given image the best. ■

4.1 WebSSQL Syntax

The above examples illustrated some of the major capabilities of the WebSSQL query language. We now provide the syntax of WebSSQL. We focus on the parts that are extensions to the SQL language.

Query := SELECT TargetAttr FROM TableList [WHERE Conditions] [WITH-SIZE Size];
TargetAttr := url | TableVar.url
TableVar := TableName | TableVariable
TableList := TableDef {, TableDef }
TableDef := TableName | TableName TableVariable
Conditions := BoolTerm { OR BoolTerm }
BoolTerm := Attribute = Attribute
  | Attribute = "StringConstant"
  | Attribuet CONTAINS "TextConstant"
  | SimilarityCondition
  | TableVar LinkExp TableVar such that (Conditions)
  | OtherSQLStandardCondition
  | BoolTerm { AND BoolTerm }
  | (Conditions)
TextConstant := Term {, TextConstant }
SimilarityCondition := TableVar.TextAttr SIMILAR_TO [(SimilarToParameters)] "TextConstant"
  | TableVar.Color SIMILAR_TO [(SimilarToParameters)] ColorConstant
  | TableVar.Texture SIMILAR_TO [(SimilarToParameters)] TextureConstant
SimilarToParameters := Integer | DomainList | Integer, DomainList
DomainList := DomainName {, DomainList}
ColorConstant := ImageFile | ColorDescriptionVector
ColorDescriptionVector := (Color Percentage {, Color Percentage})
Color := red | green | blue | white | black | yellow | purple | complex
Percentage := PositiveDecimalNumber
TextureConstant := ImageFile | TextureConstant
LinkExp := → [(LinkParameters)] | =
LinkParameter := integer | Quantifier | integer, Quantifier
Quantifier := ALL | SOME
Size := PositiveInteger

5 Status of the Implementation

We have been implementing a search engine based on WebSSQL. We also use WebSSQL to name this search engine. In this section, we report the status of the implementation. In addition to reporting the aspects that have been implemented, we will also provide some thoughts on certain aspects that have not been implemented yet. The software component architecture of WebSSQL is outlined in Section 5.1. In Section 5.2, we briefly describe how we prepare the data for the search engine. In Section 5.3, a simply query processing strategy is sketched.

5.1 Software Component Architecture

The WebSSQL search engine is built up with the following key software components (see Figure 2):

Query parser and decomposer: It takes the query from a user and parses it first. During parsing, it performs the WebSSQL syntax check, and returns all the errors back to the user. If the query has correct syntax, it decomposes the query into multiple subqueries (see Section 5.3 for more detail).

Similarity computation server: All the similarity based subqueries from the query decomposer are forwarded to this component. The corresponding similarity or similarities are computed. As a result, a number of webpage or image URLs or ids are returned to the *result generator* to be combined or to the SQL server for further SQL computation.

SQL query server: The SQL related subqueries from the query decomposer are directed to this component. Some intermediate results from the similarity computation may also be transferred into the SQL server. These inputs are combined as standard SQL queries and executed by the relational database server.

Result generator: The results from the similarity server and SQL server are then combined together as discussed in the previous section. A list of webpages/images are formulated into an HTML page output.

User interface: It's a web browser based interface and forms are provided to end-users to enter the queries.

Index databases: A set of indexing files for the texts, image colors, and textures is maintained.

Relational database: The three tables, namely Webpages, Images and Links, are stored in a relational database.

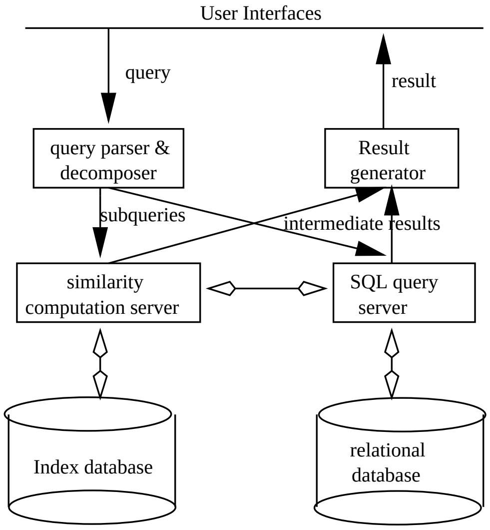

Figure 2: Software Components of WebSSQL

5.2 Data Preparation

A *robot* is implemented to fetch web pages and images from user specified web site(s). For each web page fetched, a record (tuple) in table Webpages is created. Most values in the record, except *text*, are provided by the HTTP protocol used. The web page itself is temporarily stored as a local file to be further processed. Only the id of the file is actually stored under the *text* field. Similarly, for each image fetched, a record in table Images is created. In the current implementation, fields *color* and *texture* are not used and the values in field *description* are manually provided. The image itself is not stored in the table and the value in field *image* contains the id of the image file. When a link from page with url u1 and anchor information a1 to a web object with url u2 is found by the robot, a record (u1, u2, a1) is created in the Links table. In summary, three types of data, namely structured data (i.e., the three tables), texts and images, are produced by the robot. In our current implementation, the three tables are stored in a commercial relational database (Sybase). Texts and images are temporarily stored as files for further processing.

The set of text files are treated as a document collection. They are indexed based on the terms in them as in traditional information retrieval [25]. Conceptually, each web page will be represented as a vector of weights $(w_1, w_2, ..., w_k)$, where w_i is the weight or significance of the ith term in representing the content of the page (see Section 3.2). In order to facilitate efficient query processing, an inverted file index is usually created. For a given term, such an index can be used to find the weights of those documents containing the term quickly. After the creation of document vectors or the inverted file index, the documents themselves no longer need to be kept.

We are currently working on the automated indexing of images. The image processing communities have developed many techniques to automatically extract image features such as color histograms and textures. We are implementing some of these algorithms. Automatically generating collateral text for images is a very challenging problem. In the Web environment, images are not isolated objects. They are often referenced/linked by other web objects or the same web object. This environment thus provides us good opportunities to obtain text descriptions for images. We have recently re-implemented our robot to extract collateral text for images. For a given image, the collateral text could come from up to six different sources such as *alternate text* associated with the image, the *anchor text* associated with a link pointing to the image, and the name of the image file. We are currently evaluating the effectiveness of our algorithm.

5.3 Query Processing

We have implemented a simpler version of WebSSQL. Only one similar_to operation is allowed for text and one similar_to is allowed for image (against the description field). Our discussion on query processing is based on this version of WebSSQL. Each user query will be processed in three steps. First, the query is decomposed into a number of subqueries. Second, the subqueries are processed in a certain efficient way. Finally, the results of the subqueries are assembled to produce the final result. These three steps are sketched below (see [29] for more details).

5.3.1 Query Decomposition

When a user query is received, it is first decomposed by a query decomposition algorithm. Consider a query q for retrieving web pages (for queries for retrieving images, the discussion is similar). In general, q could be decomposed into up to three subqueries, q1, q2, and q3, depending on the number of different types of conditions involved. Subquery q1 is a standard SQL query against the tables in the relational database. This subquery is generated only when some data stored in the three database tables are referenced in q. Subquery q2 is a query against the text documents and is generated only when the "text similar_to" condition appears in the where clause of q. Finally, subquery q3 is a query against the description field of the Images table and is generated only when the "Images.description similar_to" condition is present. In addition, if "text similar_to" is in q, then Webpages.file_id is added to the select clause of q1. This is to enable the matching of web page records in Webpages and their corresponding text indexes. If "Images.description similar_to" is present in q, then Images.image_id and Images.description are added to the select clause of q1. The former is used to determine which images appear in which web pages and the latter is to retrieve the descriptions of appropriate images to feed into the subquery q3.

5.3.2 Subquery Processing

The processing of q1 is done by the employed relational database system. After q1 is processed, a set of quadruplets (Webpages.url, Webpages.file_id, Images.image_id, Images.description) will be obtained. For each quadruplet, a triplet (Webpages.url, Webpages.file_id, Images.image_id) are extracted and the set of triplets is used by the next step for assembling the final result. The fourth component of each quadruplet (i.e., Images.description) will be extracted and be used by subquery q3. Subquery q2 is a query against a collection of web pages. This is exactly the kind of query seen in most search engines and in traditional document retrieval systems [24]. The standard approach for evaluating such a query is to use the inverted file index for the document collection [24, 28]. The result of evaluating q2 is a set of pairs (Webpages.file_id, similarity), where similarity > 0 is the similarity of the web page identified by the file id with the query (the description following "text similar_to"). This set of pairs will be used by the next step for assembling the final result.

Subquery q3 is processed after subquery q1 has been processed. A benefit of this evaluation order is that only image descriptions whose corresponding image ids are returned by q1 need to be used to process q3. Note that there is no inverted file index for the descriptions of images. As a result, the similarities between the image descriptions returned by q1 and the image description in the query have to be computed one by one. The evaluation of q3 produces a set of pairs (Images.image_id, similarity), where similarity > 0 is the similarity of the description of the image identified by the image id with the image description in the user query. Again, this set of pairs will be used by the next step for assembling the final result.

5.3.3 Result Assembling

As described above, the result of evaluating q1 is a set of triplets (Webpages.url, Webpages.file_id, Images.image_id), the result of evaluating q2 is a set of pairs (Webpages.file_id, similarity), and the result of evaluating q3 is a set of pairs (Images.image_id, similarity). We now discuss how to generate the final result to the user query from these triplets and pairs. The result assembling is accomplished by the following algorithm.

1. Sort the triplet file based on the url field. Sort the two pair files based on the file_id and the image_id fields, respectively.

2. For each triplet, say (url_1, w_id_1, i_id_1), do

   - Use w_id_1 to find a pair in (Webpages.file_id, similarity) such that w_id_1 = Webpages.file_id. Note that at most one such pair can be found as file id is unique for web pages. It is possible that no such pair can be found. This corresponds to the case where the web page with the file id has a similarity of zero with the text query in q2. In this case, discard the triplet and continue with the next triplet. Without loss of generality, suppose there is one matching pair and it is (w_id_1, sim_{11}).
   - Use i_id_1 to find a pair in (Images.image_id, sim-
   ilarity) such that i_id_1 = Images.image_id. Again, at most one such pair can be found. If no such pair is found, discard the triplet and continue with the next triplet. Again, suppose there is one matching pair and it is (i_id_1, sim_{12}).
   - Let $\text{sim}_1 = \min\{sim_{11}, sim_{12}\}$. Note that sim_1 is the combined similarity of the web page based on the text of the page and the image identified by i_id_1. Return pair (url_1, sim_1).

3. If a url appears in several returned pairs (url, sim), keep the pair with the largest sim and discard other pairs for the url. Note that several pairs with the same url may be returned from step 2 if the corresponding web page contains multiple images that satisfy the relevant conditions in the query.

4. Display the remaining urls in descending sim values.

6 Conclusions

In this paper, we described an SQL-like query language — WebSSQL — for retrieving desired web pages. WebSSQL has several unique features. First, WebSSQL assumes that each web page is a multimedia document consisting of structured data, text data and possibly image data instead of purely text object. Second, WebSSQL treats each page as a node in a directed graph composed of many web pages and links among them instead of treating each page as an isolated object. Third, WebSSQL is similarity-based meaning that the retrieved web pages will be ranked based on their closeness to a given query. With WebSSQL, users can specify their search needs much more precisely. As a result, more useful information and much less irrelevant results can be returned.

A prototype search engine based on a simplified version of WebSSQL has been implemented. We are currently implementing the second version of the search engine. The efforts are being focused on the automated indexing of image features (collateral text, color histograms and texture) as well as on the development of efficient query processing strategies. We are also investigating how to develop a good user interface that will be, on the one hand, easy to use as most users will not be able to use WebSSQL directly, and on the other hand, can support most of the key features of WebSSQL.

Acknowledgement: This work is supported in part by NSF grant IIS-9902872.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. *The Lorel Query Language*

for Semistructured Data. International Journal on Digital Libraries, 1:1, pp. 68-88, April 1997.

[2] Gustavo O. Arocena, et al. *Applications of a Web Query Language.*

[3] S. Brin, and L. Page. *The Anatomy of a Large-Scale Hypertextual Web Search Engine.* WWW7 Conference, 1998.

[4] C. Buckley, G. Salton, and J. Allan. *Automatic Retrieval with Locality Information Using Smart.* First TREC Conference, Gaithersburg, MD, pp. 59-72, 1993.

[5] M. La Cascia, S. Sethi, and S. Sclaroff. *Combining Textual and Visual for Content-based Image Retrieval on the World Wide Web.* IEEE Workshop on Content-based Access of Images and Video Libraries, June 1998.

[6] M. Cutler, Y. Shih, and W. Meng. *Using the Structures of HTML Documents to Improve Retrieval.* USENIX Symposium on Internet Technologies and Systems (NSITS'97), Monterey, California, 1997, pp. 241-251.

[7] Daniela Florescu, et al. *Database Technique for the World-Wide Web: A Survey.* ACM SIGMOD Record, 27:3, September 1998, pp. 59-74.

[8] J. Huang, S. Kumar, M. Mitra, W. Zhu, and R. Zabih. *Image Indexing Using Color Correlograms.* IEEE International Conference on Computer Vision and Pattern Recognition, 1997.

[9] D. Konopnicki, and O. Shmueli. *W3QS: A Query System for the World Wide Web.* Very Large Data Bases Conference, 1995.

[10] S. Lawrence, and C. Lee Giles. *Accessibility of Information on the Web.* Nature, 400, July 1999, pp. 107-109.

[11] B. Manjunath, and W. Ma. *Texture Features for Browsing and Retrieval of Image Data.* IEEE Trans. PAMI, 18, pp. 837-842, 1996.

[12] J. Marie-Julie, and H. Essafi. *Image Database Indexing and Retrieval Using the Fractal Transform.* Proc. IEEE Multimedia Systems, 1997.

[13] O. A. McBryan. *GENVL and WWWW: Tools for Taming the Web.* First WWW Conference, Geneva, May 1994.

[14] A. Mendelzon, G. Mihaila, and T. Milo. *Querying the World Wide Web.* International Journal on Digital Libraries, 1:1, pp. 54-67, April 1997.

[15] Mendelzon, et al. *Finding Regular Simple Paths in Graph Databases.*

[16] G. Navarro, and R. Baeza-Yates. *A Model to Query Documents by Contents and Structure.* ACM SIGIR Conference, pp. 93-101, 1995.

[17] L. Page, S. Brin, R. Motwani, and Terry Winograd. *The PageRank Citation Ranking: Bring Order to the Web.* Technical Report, Stanford University, 1998.

[18] G. Pass, and R. Zabih. *Histogram Refinement for Content-based Image Retrieval.* IEEE Workshop on Applications of Computer Vision, 1996.

[19] J. Puzicha, T. Hofmann, and J. Buhmann. *Non-parametric Similarity Measures for Unsupervised Texture Segmentation and Image Retrieval.* IEEE International Conference on Computer Vision and Pattern Recognition, 1997.

[20] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. *Querying Semistructured Heterogeneous Information.* DOOD International Conference, pp. 319-344, 1995.

[21] A. Rao, R. Srihari, and Z. Zhang. *Spatial Color Histograms for Content-based Image Retrieval.* IEEE International Conference on Tools with Artificial Intelligence, 1999.

[22] A. Rao, R. Srihari, and Z. Zhang. *Geometric Histograms: A Distribution of Geometric Configurations of Color Subsets.* SPIE International Conference on Internet Imaging, 2000.

[23] S. Ravela, and R. Manmatha. *Retrieving Images by Appearance.* IEEE International Conference on Computer Vision and Pattern Recognition, 1997.

[24] G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* New York: McCraw-Hill, 1983.

[25] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison Wesley, 1989.

[26] M. Swain, and D. Ballard. *Color indexing.* International Journal of Computer Vision, Kluwer, 7, pp. 11-32, 1991.

[27] M. Swain, C. Frankel, and M. Lu. *View-Based Techniques for Searching for Objects and Textures.* Proc. Asian Conference on Computer Vision, IEEE, 1995.

[28] C. Yu, and W. Meng. *Principles of Database Query Processing for Advanced Applications.* Morgan Kaufmann, San Francisco, 1998.

[29] C. Zhang. *WebSSQL: Similarity-based SQL for Searching the Web.* Master Thesis, dept. of Computer Science, SUNY at Binghamton, Fall 1998.

[30] Z. Zhang. *Identifying Human Faces in General Appearances.* IEEE International Conference on Systems, Man, and Cybernetics, 1998.

[31] Z. Zhang, R. Srihari, and A. Rao. *Face Detection and its Applications in Intelligent and Focused Image Retrieval.* IEEE International Conference on Tools with Artificial Intelligence, 1999.