Keyword Search in Databases

Arvind Hulgeri[*] Gaurav Bhalotia Charuta Nakhe[†] Soumen Chakrabarti
S. Sudarshan
Dept. of Computer Science and Engg., Indian Institute of Technology, Bombay
{aru,bhalotia,soumen,sudarsha}@cse.iitb.ac.in, charuta@pspl.co.in

Abstract

Querying using keywords is easily the most widely used form of querying today. While keyword searching is widely used to search documents on the Web, querying of databases currently relies on complex query languages that are inappropriate for casual end-users, since they are complex and hard to learn. Given the popularity of keyword search, and the increasing use of databases as the back end for data published on the Web, the need for querying databases using keywords is being increasingly felt. One key problem in applying document or web keyword search techniques to databases is that information related to a single answer to a keyword query may be split across multiple tuples in different relations.

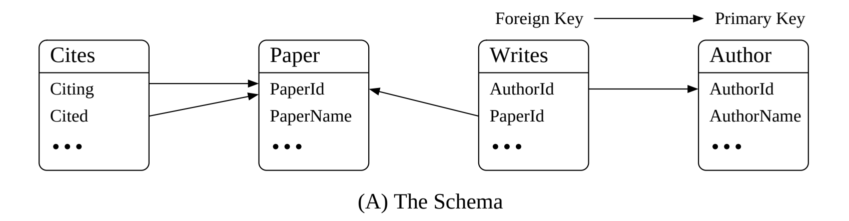
In this paper, we first present a survey of work on keyword querying in databases. We then report on the BANKS system which we have developed. BANKS integrates keyword querying and interactive browsing of databases. By their very nature, keyword queries are imprecise, and we need a model for answering keyword queries. BANKS, like an earlier system called DataSpot, models a database as a graph. In the BANKS model, tuples correspond to nodes, and foreign key and other links between tuples correspond to edges. Answers to a query are modeled as rooted trees connecting tuples that match individual keywords in the query. Answers are ranked using a notion of proximity coupled with a notion of prestige of nodes based on inlinks, the latter being inspired by techniques developed for Web search. We illustrate the power of the model and our prototype through examples.

1 Introduction

Querying relational databases using schema-cognizant languages like SQL and querying document collections by typing arbitrary keywords are the extreme ends of the continuum between structured and unstructured data access. SQL queries have precisely defined semantics, but demand that the data is organized along a strict schema which the user understands. Keyword searches do not require the data to follow a schema, except for the notion of a collection of documents, each being a sequence of delimited tokens. On the other hand, responses to keyword searches are often imprecise.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[*]Supported by an Infosys Fellowship
[†]Current affiliation: Persistent Systems Pvt. Ltd., Pune, India



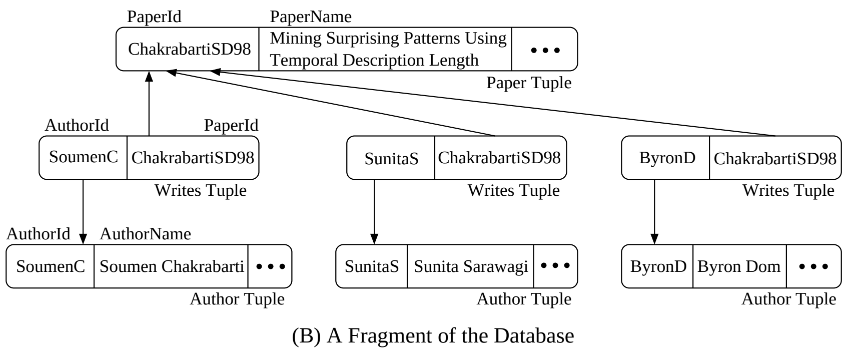


Figure 1: The DBLP Bibliography Databases

Two forces are bridging the gap between these extremes. First, relational databases are increasingly Web enabled: they need to be accessed and manipulated by non-experts who do not know enough about the schema. Second, Web documents are evolving from flat text files through HTML and SGML to XML, adding markups and embedded schema information (albeit irregular) which users would like to exploit to make responses somewhat more precise than with plain keyword matching. Unfortunately, as query languages for relational data evolve to encompass semi-structured data, they are becoming more complex. On the other hand, many naive users need a simple way of extracting information, such as keyword queries.

In relational databases, information needed to answer a keyword query is often split across the tables/tuples, due to normalization. As an example consider a bibliographic database shown in Figure 1. This database contains paper titles, their authors and citations extracted from the DBLP repository. The schema is shown in Figure 1(A). Figure 1(B) shows a fragment of the DBLP database. It depicts partial information—paper title and authors—regarding a paper as stored in the bibliographic database defined above. As we can see the information is distributed across seven tuples linked through foreign key-primary key links. A user looking for this paper may use queries like ”Sunita Temporal” or ”Soumen Sunita”. In keyword based search, we need to identify tuples containing the keywords and ascertain their proximity through links.

Inverted indexing techniques used for document search would help in finding proximity amongst the words in a tuple or a column thereof but will not help in finding proximity between two tuples. This makes inverted index-based keyword search unsuitable for RDBMS. Similar observations hold for text search in XML documents: the markup induces a graph whose nodes contain blocks of text. Different query words may match different blocks or nodes, and the best ‘answer’ may be none of the directly matched nodes.

Web search engines and topic directories have also popularized the browsing paradigm for accessing information, which is virtually non-existent in the relational and semi-structured data domains. Like HTML pages connected via hyperlinks, semi-structured data comprises data entities which are nodes in a graph with labeled edges. The link-based navigation paradigm should therefore be of great use for exploring such databases.

Responses to relational queries are *sets* of tuples, which may be explicitly ordered by a specified list of attributes. Thus the “information unit” is a tuple. For document search, the information unit is a document and the rank of the document in the response list is based on the number of keywords found in the document and their proximity within the document. A suitable information unit is much harder to define when a general graph model is used to represent data entities and relations, because information matching the query may be split across several tables and tuples due to normalization.

Over the last few years, a uniform model has emerged for representing relational databases as a graph with the tuples in the database mapping to nodes and cross references (such as foreign key and other forms of references) between tuples mapping to edges connecting these nodes. Semistructured data maps even more naturally to a graph model, as do HTML pages connected by hyperlinks. The graph model may be used in keyword search as follows. Keywords in a given query *activate* some nodes. The answer to the query is defined to be a subgraph which connects the activated nodes.

In this paper, we present a survey of earlier work on keyword querying of databases, with emphasis on the development of the above model. We then present the model for keyword search used in the BANKS system (BANKS is an acronym for Browsing ANd Keyword Search). BANKS ranks answer subgraphs using a notion of proximity coupled with a notion of prestige of nodes based on inlinks, similar to techniques developed for Web search engines like Google. BANKS proposes meaningful interpretations for matching query tokens not only to text attributes, but also to relation and attribute names. Here we are chiefly concerned with meaningful definitions of responses and their ranking. Efficient query execution strategies for the BANKS model will be described in a separate paper.

BANKS has been developed in Java using servlets and JDBC. It is a generic system and can be used against any relational database supporting JDBC, without any programming. (An XML source adapter is planned.) A demo of BANKS is accessible at `http://www.cse.iitb.ac.in/banks/`.

Organization: We survey earlier work on keyword querying in Section 2. Section 3 outlines the BANKS graph model for representing connectivity information from the database and the model for answering queries and briefly outlines how the BANKS querying model differs from earlier work. Section 4 briefly describes how the query model is implemented. We present an overview of the browsing features of BANKS in Section 5. Section 6 outlines a preliminary evaluation of our system in terms of reasonableness of its answers and feasibility. Section 7 outlines directions for future work. Section 8 concludes the paper.

2 Previous Work

There are a number of commercial and research prototype systems that support keyword search and browsing in relational and semi-structured databases. In this section, we survey several of these systems.

Traditional database query languages and tools are not suitable for applications that require keyword searching. Even languages, such as QBE, that have been targeted at relatively inexperienced users require the user to be aware of the database schema, which is not appropriate for casual users of an information system.

2.1 DataSpot and Mercado Intuifind

The DataSpot system [3] was developed to support database querying using free-form (keyword) queries and navigation for non-technical users seeking information in complex databases such as electronic catalogs. The basis for the DataSpot system is a schema-less representation of data which they call a "Hyperbase". Nodes in the hyperbase view represent data objects, while edges represent associations. There are two types of edges, *simple* and *identification* edges. Simple edges represent inclusion, such as attributes values' inclusion in tuples. Identification edges correspond to references between objects.

The semantics of keyword queries is described in [12], and in more detail in [13]. Given a keyword query, an answer is a connected sub-hyperbase that contains the keywords in the query. Each answer has an associated "location", which intuitively represents the main object of the answer. Answers have a score, which can be computed in one of several ways, but intuitively measures some distance metric on the sub-hyperbase. Several alternatives are suggested, one of which is an edge count on the sub-hyperbase, possibly with weighted edges, with weights being determined by node and edge types. Another alternative is to use a node count. The distance metric used in their preferred approach is based on adding up edge weights, with weights being determined by the types of the edge, the types of the nodes it connects, and the direction of traversal (from parent to child or vice versa; see below).

The system administrator can define a set of nodes as "fact nodes", in which case answers (location nodes) can only come from the set of fact nodes. Refinement queries, which refine answers from earlier queries, can also start with an initial set of locations, in addition to keywords.

To find answers, the system performs weighted best-first search from all source nodes (i.e., keyword nodes, and for refinement queries, location nodes) and each time a fully connected fact node (i.e., a fact node connected to all source nodes) is found, it is output as an answer, along with the minimal-distance paths from that node to all the source nodes. Traversal goes away from source nodes, but edges can be followed regardless of their direction (that is, edges can be traversed forwards or backwards). Edge weights are determined by direction of traversal, as mentioned earlier. When adding edge weights to paths, the edge weight is divided by the number of sources to which the edge is connected, so that when adding up path distances, edge weights of edges leading to multiple sources are not over-counted. Multiple answers may be output, ranked by their score.

Mercado Software, Inc. (`www.mercado.com`) markets an e-catalog search technology called Intuifind which (as far as we can understand) uses the DataSpot technique for keyword search, but also allows "parametric search" on the search results. Intuifind parametric search offers OLAP drill-down type operations based on parameters such as price, manufacturer and category, to allow the user a more structured way of browsing search results.

2.2 EasyAsk

EasyAsk is commercial system that provides natural language search (including keyword search) on data stored in relational databases as well as in text repositories. Our description of EasyAsk is based on white papers on the EasyAsk system, and on its features, available at the EasyAsk web site `www.easyask.com`.

Consider catalog searching, which is a motivating application for EasyAsk (the system can be used with other application domains also). Information about items in catalogs can be split across multiple locations, such as a text description, and at different levels of a product hierarchy in which the item is classified, such as "men's" or "formal". A keyword search on the catalog may contain keywords present in the product description, as well as keywords present in the catalog hierarchy.

To answer queries, the system crawls the data store ahead of time and constructs a contextual dictionary. Based on the dictionary, the system decides that certain keywords correspond to values in catalog attributes, and others correspond to values in text descriptions of products, and generates an appropriate SQL query to answer a given keyword query.

The EasyAsk system supports a wide variety of features such as approximate word matching, word stemming (allowing it to match, for example, hiker with hiking), synonyms (for example, pants, trousers and slacks), and other word associations (for example "hunting" with "waterproof" and "outdoors" in the context of clothes). It also recognizes phrases, and supports comparisons such as "greater than 3 feet" which it translates into appropriate SQL conditions. EasyAsk also mentions that it handles data decoding, whereby users can use normal words, such as "blue", even if the word has been coded in a different way, such as "bl", in the database.

Further details of how EasyAsk handles keyword queries and natural language queries are not publicly available. Ranking is supported by EasyAsk, but as far as we can make out, only on administrator-specified criteria such as price, and not on the quality of match with keywords.

2.3 Proximity Search

Several research prototypes focus on the notion of proximity and efficient ranking or clustering of graph nodes by proximity.

Goldman et al. [7] were early proponents of using a graph distance-based measure in answering proximity queries. They generalized the notion of *near* queries in Information Retrieval (where the goal is to find documents where query tokens occur lexically close to each other) to a graph data model that can model both relational and semi-structured data.

They support queries of the form *find find-set near near-set*; such a query retrieves objects of the specified "find set" (e.g. a specified relation) that have short paths connecting them to objects in the near-set (those that match the specified keywords). The shortest path from objects in the "find set" to each of the objects in the "near-set" is computed using a distance function that adds up edge weights. The inverse of the distance, scaled by the weights of the end point nodes, gives the proximity of the objects. The score of an object in the find set is computed using its proximity to the near objects by either adding up the proximities, or by other means such as $1 - \Pi(1 - p_i)$, where the p_is are the proximities (which range from 0 to 1).

Note that the scores differ from that of DataSpot in that the inverses of distances to the near set objects are added up, instead of the distances themselves. Also, there is no significance to the actual paths, and only the objects in the find set are returned.

Web search provides another natural application where the best response may comprise a graph of connected pages rather than a single page. Li et al. [9] couch this problem in terms of Steiner trees connecting pages that match individual keywords. In their formulation, the graphs are not directed, and unlike DataSpot, pages and edges are not typed.

Proximity search is closely related to clustering. The connections between densely connected clusters in graphs and spectral properties of their adjacency matrices is well-established [8]. Clustering techniques for graphs and hypergraphs can be used to derive notions of distances between categorical data, and thereby to support proximity search between objects with categorical attributes [6].

Many Information Retrieval systems use thesauri and lexical networks to bridge the gap between imprecise user queries and the database by padding the query with synonyms and using *is-a* hierarchies (e.g., horse is a mammal). Such support needs to be hand-crafted into IR systems. In a graph framework, such as that used in DataSpot or BANKS, a network of metadata or linguistic relations can be regarded as simply augmenting the graph being queried with a richer set of connections.

2.4 Other Approaches

Sarda and Jain [14] describe a system called Mragyati to perform keyword search and browsing on databases. They generate SQL queries to retrieve results matching the given keywords. Queries can involve more than one relation, and are generated based on the database foreign-key structure and on the relations/attributes that are matched by the given keywords. Results can be ranked based on user specified criteria, or based on the number of foreign key references to the primary key (if any) of the answer tuples. The system provides mechanisms for handling synonyms and coding mechanisms used to store values in the database. Although supported in the model, the current implementation does not handle queries with paths of length greater than two, presumably because of the extra effort needed to analyze keywords and database connections to generate required queries. The system also generates hyperlinks in the results, to enable browsing.

Maserman and Vossen [10] describe an approach to keyword searching on databases, but their approach is restricted to finding all keywords in a single tuple, with no notion of links and proximity. They generate statements in an SQL extension called Reflective SQL, which is then translated to SQL.

Florescu et al. [5] propose an extension of the XML-QL query language to include keyword search, but their approach requires the use of a complex query language such as XML-QL.

2.5 Browsing

There has been a substantial body of work on browsing of relational databases and object oriented database. Although browsing is not part of keyword searching per se, the results of a keyword search can often be interpreted only as starting points from which the user finds required information by browsing. Work on browsing of databases include Dar et al. [4], Carey et al. [2], and more recently work by Shafer and Agrawal [16], which describes a system for integrated querying and browsing of relational data. Querying is carried out by interaction with form controls, rather than by keyword search. Munroe and Papakonstantinou [11] describe BBQ, an interface for browsing and QBE style querying of XML data.

3 Database and Query Model

In this section we describe how a relational database is modeled as a graph in the BANKS system. First we evaluate various options available and describe the model we adopt informally and then formalize it.

3.1 Informal Model Description

We model each tuple in the database as a node in the directed graph and each foreign key-primary key link as an edge between the corresponding tuples. This can be easily extended to other type of connections; for example, we can extend the model to include edges corresponding to inclusion dependencies, where the values in the referencing column of the referencing table are contained in the referred column of the referred table but the referred column need not be a key of the referred table.

In general, the importance of a link depends upon the type of the link i.e. what relations it connects and on its semantics; for example, in the bibliographic database, the link between the *Paper* table and the *Writes* table is seen as a stronger link than the link between the *Paper* table and the *Cites* table. Conceptually this model is similar the one described in [13] although there are some differences in the details.

To find answers, we need to traverse links backwards and we make this explicit by creating a backward link for each initial link. We model the weight of a backward link generated from a forward link as directly proportional to the indegree of the source node of the backward link (i.e. the referenced node). Since the proximity between the nodes connected by a link is inversely proportional to the link weight, the proximity for a referenced node to its referencing nodes is inversely proportional to the indegree of the referenced node. This notion is formalized in Section 3.2 These definitions are motivated by the intuition that “fans know celebrities better than celebrities know their fans.” As another example, in a students’ course registration database, if there are many students registered for a particular course, the proximity of two students due to the course is less than if there were fewer students registered. A forward edge from a student to a course and a back edge from the course to a student would form a path between each pair of student in the course, and assigning a higher weight to back edges in the case where more students take the course ensures that the paths are longer.

Informally, an answer to a query is a subgraph containing nodes matching the keywords and just by looking at the subgraph it is not apparent as to what information it conveys. We need to identify a node in the graph as a connecting node which connects all the keyword nodes. We consider an answer to be a rooted directed tree containing a directed path from the root to each keyword node. We call the root node an *information node*. The weight of the tree is proportional to the total of its edge weights. We may restrict the information node to be from a selected set of nodes of the graph; for example, we may exclude the nodes corresponding to the tuples from a specified set of relations, in a manner similar to [13].

We incorporate another interesting feature, namely node weights, inspired by prestige rankings such as PageRank in Google [1]. With this feature, nodes that have multiple pointers to them get a higher prestige. In our current implementation we set the node prestige to the indegree of the node. Higher node weight corresponds to higher prestige. E.g., in a bibliography database containing citation information, if the user gives a query *Query Optimization* our technique would give higher prestige to the papers with more citations.

3.2 Formal Database Model

In this section we define the formal graph model for representing the database. It consists of:

Vertices: For each tuple $\mathcal{T}$ in the database, the graph has a corresponding node $u_{\mathcal{T}}$. We will speak interchangeably of a tuple and the corresponding node in the graph.

Edges: For each pair of tuples $\mathcal{T}_1$ and $\mathcal{T}_2$ such that there is a foreign key from $\mathcal{T}_1$ to $\mathcal{T}_2$, the graph contains an edge from $u_{\mathcal{T}_1}$ to $u_{\mathcal{T}_2}$ and a back edge from $u_{\mathcal{T}_2}$ to $u_{\mathcal{T}_1}$ (this can be extended to handle other types of connections).

Edge weights: In our model, the weight of a forward link along a foreign key relationship reflects the strength of the proximity relationship between two tuples and is set to 1 by default. It can be set to any desired value to reflect the importance of the link (low weights correspond to greater proximity).

Let $s(R_1, R_2)$ be similarity from relation R_1 to relation R_2 where R_1 is the referencing relation and R_2 is the referenced relation. The similarity $s(R_1, R_2)$ depends upon the type of the link from relation R_1 to relation R_2 and this is different than the actual edge weights. It is set to infinity if relation R_1 doesn't refer relation R_2. Consider two nodes u and v in the database. Let $R(u)$ and $R(v)$ be the resp. relations they belong to. Further, let $IN_v(u)$ be the indegree of u contributed by the tuples belonging to relation $R(v)$. Note that from node u to node v we may, conceptually, have two edges, one forward edge which depends upon the similarity $s(R(u), R(v))$ and a backward edge which depends upon the similarity $s(R(v), R(u))$ and $IN_v(u)$. In the current implementation the forward edge weight is set to $s(R(u), R(v))$ and the reverse edge weight is set to $[s(R(v), R(u)) * IN_v(u)]$ and the actual edge weight is the minimum of the two as defined below:

$$b(u, v) = min(s(R(u), R(v)), s(R(v), R(u)) * IN_v(u))$$

The weight of a backward link generated from a foreign key relationship is directly proportional to the indegree of the source node (i.e. the referenced node). Since the proximity between the nodes connected by a link is inversely proportional to the link weight, the proximity from a referenced node to its referencing nodes is inversely proportional to the indegree of the referenced node.

Node weights: Each node u in the graph is assigned a weight $N(u)$ which depends upon the prestige of the node. In our current implementation we set the node prestige to the indegree of the node.

3.3 Querying Model

We now present our model for answering keyword queries. Let the query consist of n search terms $t_1, t_2, \ldots, t_n$. The query is (conceptually) answered as follows:

- For each search term t_i in the query we find the set of nodes that are relevant to the search term. Let us call the set S_i. And let $S = \{S_1, S_2, S_3, \ldots, S_n\}$. A node is relevant to a search term if it contains the search term as part of an attribute value. Nodes may also be relevant through metadata (such as column, table or view names). E.g., all tuples belonging to a relation named AUTHOR would be regarded as relevant to the keyword 'author'.

- An answer to a query is a rooted directed tree containing at least one node from each S_i. Note that the tree may also contain nodes not in any of the S_is and is therefore a Steiner tree. The relevance score of an answer tree is computed from the relevance scores of its nodes and its edge weights. (The condition that one node from each S_i must be present can be relaxed to allow answers containing only some of the given keywords.)

- Given an answer with keyword matching nodes $(s_{i,1}, s_{i,2}, \ldots, s_{i,n})$ the relevance of the answer is computed using the edge and node weights as follows:

  $$AnswerRelevance(s_i) = w_n \left[\frac{\sum_j N(s_{i,j})}{N}\right] + w_p \left[\frac{1}{\sum E_k}\right]$$

  where the E_i's are the weights of the edges in the answer tree, N is the maximum node weight sum across all elements of S, and w_n and w_p are weights used to control the relative importance given to node weights and proximity. These combining weights are ad-hoc, but appear to be inescapable in all related systems that we have reviewed [7, 17]. Reasonable choices can be made if sample queries with relevance judgments are provided to 'train' the system [15].

  The minimum Steiner tree problem is a special case of the problem of finding answers of maximum relevance, so the problem of finding the best answers is also NP-Complete. We therefore settle for heuristics to construct answer trees of high relevance. These are discussed in Section 4.

3.4 Relation of BANKS to Earlier Work

BANKS is closely related to DataSpot [3, 12, 13]. In particular, the model of query answers as rooted trees corresponds to the DataSpot model, where the roots are called fact nodes. The details of the underlying graph formalism, however, differ. BANKS currently works on a model where only references, which correspond to equivalence edges in DataSpot, are explicitly represented. Since edges in our model can have attributes such as type and weight, we can model containment (as in DataSpot and in nested XML) simply as edges of a new type. (We are currently working on adding XML support to BANKS.) The BANKS technique of assigning weights to back edges, based on indegrees, has no counterpart in DataSpot, as also the node weight mechanism used in BANKS. The use of node weights based on prestige has proved critical in Web search, and our anecdotal evidence shows their importance in the context of database search as well. BANKS also takes the effect of metadata queries into account, which is not made explicit in DataSpot.

Unlike [9], BANKS (like DataSpot) can exploit the semantically richer set of links available from foreign keys and other constraints in the structured (relational) or semistructured (XML/OEM) setting, which is largely missing in graphs formed by HTML documents.

4 Implementation Approach

Our heuristic solution is based on Dijkstra's single source shortest path algorithm. We assume that the graph fits in memory. This is not unreasonable, even for moderately large databases, because the in-memory node representation need not store any attribute of the corresponding tuple other than the RID. As a result the graphs of even large databases with millions of nodes and edges can fit in modest amounts of memory. We will be looking into external memory based applications as a part of our future work.

Given a set of keywords, first we find, for each keyword term t_i, the set of nodes, S_i, that are relevant to the keyword. In the current implementation, we search only for exact matches, and to facilitate this we build a single index on values from selected string-valued attributes from different tables. The index maps from keywords to (table-name, tuple-id) pairs.

Let $relevantNodes = S_1 \cup S_2 \cup \ldots \cup S_n$ be the relevant nodes for the query. We concurrently run $|relevantNodes|$ copies of the single source shortest path algorithm, one for each node in $relevantNodes$ as source. We run them concurrently by creating an iterator interface to the shortest path algorithm, and creating multiple instances of the iterator.

The important distinction of this approach is that the single source shortest path algorithm traverses the graph edges in reverse direction. The idea is to find a common vertex from which a forward path exists to at least one node in each set S_i. Such paths will define a rooted directed tree with the common vertex as the root and the corresponding keyword nodes as the leaves. The tree thus formed is a connection tree and root of the tree is the information node.

We create a single source shortest path iterator for each keyword node. At each iteration of the algorithm, we need to pick one of the iterators for further expansion. We pick an iterator whose next vertex to be output is at the least distance from the source vertex of the iterator (the distance measure can be extended to include node weights of nodes matching keywords). We keep a list of all the vertices visited for each iterator. Consider a set of iterators containing one iterator each from set S_i. If the intersection of their visited vertex lists is non-empty, then each vertex in the intersection defines a tree rooted at the vertex. A resulting tree is a connection tree only if the root of the tree has more than one child. If the root of a tree has only one child then the tree formed by removing the root node is also present in the result set and is more relevant to the keyword nodes in question.

5 Browsing

The BANKS system provides a rich interface to browse data stored in a relational database and is well integrated with the search facility. The browsing system automatically generates browsable views of database relations, and of query results, by using two mechanisms: foreign key relationships and nesting of data using a mechanism similar to `GROUP BY` in SQL. For every attribute that is a foreign key, a link is created in the display to the referenced tuple. In addition, primary key columns can be browsed backwards, to find referencing tuples, organized by referencing relation (a specific referencing relation can be selected by the user).

Each table displayed comes with a variety of tools for interacting with data. Apart from direct schema browsing we support operations like sorting data on a specified column, restricting the data by a predicate on a column, projecting away a column, taking join with the referenced table by clicking on a foreign key column, data nesting wherein only the distinct values of a column specified are displayed, etc. Controls for these operations can be accessed by clicking on the column names in the table header.

Note that all the hyperlinks are automatically generated by the system and no content programming or user intervention is needed. Each hyperlink is really an SQL query which is executed when a user clicks on the links.

Another important feature are **templates**; templates provide several predefined ways of displaying any data. Templates must be customized by specifying various information (depending on the template); a customized template is given a hyperlink name and is then available to users for browsing.

6 Experience and Performance

We have implemented BANKS using servlets, with JDBC connections to an IBM Universal Database. We have experimented with two datasets: One is the DBLP Bibliography database shown in Figure 1. We converted a dump of DBLP into structured relational format and ran BANKS on this data. There are 124,612 nodes and 319,232 edges in this graph. The other one is a small thesis database. IIT Bombay's database of Masters and Phd dissertations are available in relational format; these are input to BANKS.

A system like BANKS may be evaluated along (at least) two measures: quality of the results and speed. Unfortunately, neither is easy to characterize. There are no agreed-upon benchmarks for evaluating proximity- or prestige-base ranking algorithms in this domain. To work around this, we selected data sets that academics and database researchers are familiar with and can relate to, at least w.r.t. the schema. This makes the discussion of results more meaningful, albeit qualitative. While we can compare the different heuristics with each other on the quality of their answers, it is not clear what is an "optimal" answer to compare against. Eventually, user experience counts, which is what led us to releasing the search facility on a Web site.

We give a few examples of queries on the bibliographic database. For the query "Mohan", C. Mohan came out at the top of the ranking, with Mohan Ahuja and Mohan Kamat following. This was due to the prestige conferred by the *writes* relation which had many tuples for these authors. The query "transaction" returned Jim Gray's classic paper and the book by Gray and Reuter as the top two answers.

The query “soumen sunita” returned only one answer: a tree containing a node corresponding to the paper “Mining Surprising Patterns Using Temporal Description Length” as the information node. This paper has Soumen and Sunita as co-authors and the tree has the two corresponding *author* tuples as leaf nodes and two *writes* tuples – connecting the two author tuples to the paper tuple – as intermediate nodes.

The query “sunita olap” returned some interesting results – the part of DBLP we had loaded had no paper by “sunita” with “olap” in its title, but the system found several papers with “olap” in the title that cited or were cited by papers authored by “sunita”. Several of the resultant papers by “sunita” were on OLAP even though the word was absent in the title. A useful extension would be to differentiate between papers that cite papers by “sunita” and papers cited by papers by “sunita”. We could, for instance, provide a way to select an answer corresponding to one of these forms (at the level of the database schema) and ask for more answers of that form.

The BANKS system supports metadata queries. For example, the query “thesis sudarshan” returns all thesis’ advised by Sudarshan (in addition to thesis’ written by a Sudarshan, if there had been any).

Currently loading the DBLP database takes 2 minutes, and about 100 MB of memory, but we expect this to decrease greatly with a better tuned Java or C implementation. Once the database graph is loaded, queries usually take a second or so to get the first answer, and a few seconds to get all answers (up to some relevance cutoff) on the DBLP database. Our prototype implementation clearly demonstrates the feasibility of using a system such as BANKS for moderately large databases.

7 Extensions and Future Work

Several extensions are possible to our model. A keyword in a tuple/relation-name may not be exactly equal to a search term but instead be a synonym to it. The BANKS model is easily extendible to allow scaling down of node weights to account for approximate match or synonyms. Synonyms are particularly useful in the context of matching metadata. Performance issues caused by metadata keywords matching large numbers of nodes are being addressed in BANKS.

The model can be easily extended to support predicate of the form *attr:[op]value*, e.g. **“author:Sudarshan, year:>2000”**. We want to allow user-defined rankings for temporal and other ordered domains. E.g., one may search for `concurrency recent` requiring a paper about concurrency published “recently”.

For a given set of keywords, there may not exist a reasonable cost tree that includes one node corresponding to each keyword. In such a situation a tree including only some of the keywords may be useful. In our current implementation we set the node prestige to the indegree of the node. We can incorporate a full-fledged eigen-analysis as in Google/PageRank so as to facilitate prestige transfer (a form of spreading activation), wherein nodes pointed to by heavy nodes become heavier. We are also considering several additional functionalities in the user interface such as ways of getting answers with a tree structure (form) similar to a chosen answer, and of getting summaries as answers. We plan to apply our current and later proximity algorithms to XML/OEM.

8 Conclusions

Many Web sites are becoming database-centric, and manually creating interfaces for browsing and querying data is time consuming. Systems supporting keyword search on relational data reduce the effort involved in publishing relational data on the Web and making it searchable. Examples of the types of data that could be published with keyword search support include organizational data, bibliographic data and product catalogs. We surveyed several approaches to keyword search on databases. We have developed BANKS, an integrated browsing and keyword querying system for relational databases. BANKS has many useful features which allow users with no knowledge of database systems or schema to query and browse relational databases with ease.

Acknowledgments: We wish to thank B. Aditya, Urmila Kelkar, Megha Meshram and Parag for implementing some parts of the BANKS system and for their help with loading the databases.

References

[1] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7), 1998.

[2] Michael J. Carey, Laura M. Haas, Vivekananda Maganty, and John H. Williams. Pesto : An integrated query/browser for object databases. In *Procs. of the International Conf. on Very Large Databases*, pages 203–214, 1996.

[3] Shaul Dar, Gadi Entin, Shai Geva, and Eran Palmon. DTL's DataSpot: Database exploration using plain language. In *Procs. of the International Conf. on Very Large Databases*, pages 645–649, 1998.

[4] Shaul Dar, Narain H. Gehani, H. V. Jagadish, and J. Srinivasan. Queries in an object-oriented graphical interface. *Journal of Visual Languages and Computing*, 6(1):27–52, 1995.

[5] Daniela Florescu, Donald Kossmann, and Ioana Manolescu. Integrating keyword search into xml query processing. *WWW9/Computer Networks*, 33(1-6):119–135, 2000.

[6] David Gibson, Jon M. Kleinberg, and Prabhakar Raghavan. Clustering categorical data: An approach based on dynamical systems. In *Procs. of the International Conf. on Very Large Databases*, pages 311–322, 1998.

[7] Roy Goldman, Narayanan Shivakumar, Suresh Venkatasubramanian, and Hector Garcia-Molina. Proximity search in databases. In *Procs. of the International Conf. on Very Large Databases*, pages 26–37, 1998.

[8] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *JACM*, 46(5):604–632, 1999.

[9] Wen-Syan Li, K Selcuk Candan, Quoc Vu, and Divyakant Agrawal. Retrieving and organizing web pages by "information unit". In *World-wide Web Conference*, 10, pages 230–244, 2001.

[10] Ute Masermann and Gottfried Vossen. Design and Implementation of a novel approach to Keyword Searching i n Relational Databases. In *Current Issues in databases and information systems*, pages 171–184, September 2000.

[11] Kevin D. Munroe and Yannis Papakonstantinou. BBQ: A visual interface for integrated browsing and querying of xml. In *Visual Database Systems*, May 2000.

[12] Eran Palmon. Associative search method for heterogeneous databases with an integration mechanism configured to combine schema-free data models such as a hyperbase. United States Patent Number 5,740,421, Granted April 14, 1998, filed in 1995. Available at `www.uspto.gov`, 1998.

[13] Eran Palmon and Shai Geva. Associative search method with navigation for heterogeneous databases including an integration mechanism configured to combine schema-free data models such as a hyperbase. United States Patent Number 5,819,264, granted October 6, 1998, filed in 1995. Available at `www.uspto.gov`, 1998.

[14] N. L. Sarda and Ankur Jain. Mragyati: A system for keyword-based searching in databases. Submitted for publication. Contact: nls@cse.iitb.ac.in., 2001.

[15] Hinrich Schütze, David A. Hull, and Jan O. Pedersen. A comparison of classifiers and document representations for the routing problem. In *ACM SIGIR'95, (Special Issue of the SIGIR Forum)*, pages 229–237, 1995.

[16] John C. Shafer and Rakesh Agrawal. Continuous querying in database-centric web applications. *WWW9/Computer Networks*, 33(1-6):519–531, 2000.

[17] Ron Weiss, Bienvenido Vélez, Mark A. Sheldon, Chanathip Nemprempre, Peter Szilagyi, Andrzej Duda, and David K. Gifford. Hypursuit: A hierarchical network search engine that exploits content-link hypertext clustering. In *Proc. of ACM Hypertext*, pages 180–193, 1996.