BANKS: Browsing and Keyword Searching in Relational Databases*

B. Aditya Gaurav Bhalotia † Soumen Chakrabarti Arvind Hulgeri
Charuta Nakhe ‡ Parag S. Sudarshan

Computer Science and Engg. Dept., I.I.T. Bombay
{baditya,soumen,aru,parag,sudarsha}@cse.iitb.ac.in
bhalotia@eecs.berkeley.edu charuta@pspl.co.in

Abstract

The BANKS system enables keyword-based search on databases, together with data and schema browsing. BANKS enables users to extract information in a simple manner without any knowledge of the schema or any need for writing complex queries. A user can get information by typing a few keywords, following hyperlinks, and interacting with controls on the displayed results. Extensive support for answer ranking forms a critical part of the BANKS system.

1 Introduction

Relational databases store large amounts of data which are queried using structured query languages. A user needs to know the underlying schema and the query language in order to make meaningful ad hoc queries on the data. This is a substantial barrier for casual users, such as users of Web-based information systems. HTML forms can be provided for predefined queries. For example, a university Web site may provide a form interface to search for faculty and students. Searching for departments would require yet another form, as would searching for courses offered. However, creating an interface for each such task is laborious, and is also confusing to users since they must first expend effort finding which form to use. Furthermore, they are not suitable for ad hoc querying or exploratory browsing.

Search engines on the Web have popularized an alternative unstructured querying and browsing paradigm that is simple and user-friendly. Users type in keywords and then follow hyperlinks to navigate from one document to the other. No knowledge of schema is needed. Keyword search can provide a very simple and easy-to-use mechanism for casual users to get information from databases.

Unfortunately, keyword search techniques used for locating information from collections of (Web) documents cannot be used directly on data stored in databases. In relational databases, information needed to answer a keyword query is often split across the tables/tuples, due to normalization. A keyword search system for databases must therefore take into account the fact that an answer to a keyword query can consist of multiple linked tuples. One possible approach to keyword search on databases is to create artificial documents that collect related information from multiple tuples, and provide keyword search on these documents. This results in duplication of data, and it is not feasible to create documents corresponding to every meaningful combination of data. It is therefore best to provide support for keyword querying directly on databases.

In this paper we outline key features of the BANKS system (BANKS is an acronym for Browsing ANd Keyword Searching) which we have developed to address the above problems. The BANKS system enables keyword search together with data and schema browsing on relational databases. BANKS allows a user to get information by typing a few keywords, following hyperlinks, and interacting with controls on the displayed results; absolutely no programming is required, and the query 'language' is as simple as in Web search engines.

There may be multiple answers to a query, and a mechanism for ranking answers based on a relevance score is critical. The keyword search component of

*Partly supported by an IBM Faculty Fellowship grant and an Infosys Ph.D. Fellowship.

† Current affiliation: Univ. California, Berkeley

‡ Current affiliation: Persistent Systems, Pune, India

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

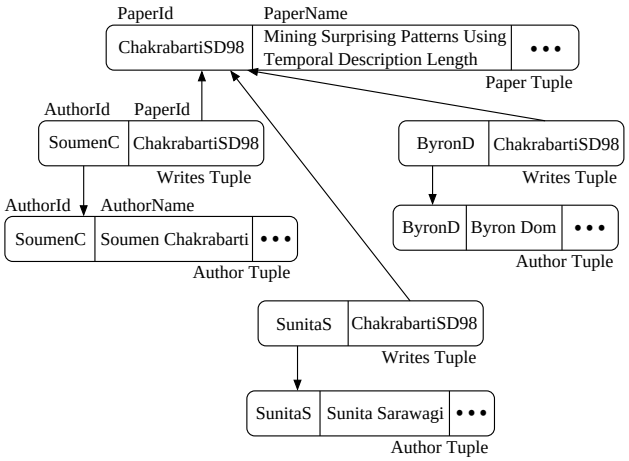

Figure 1: A Fragment of the DBLP Database

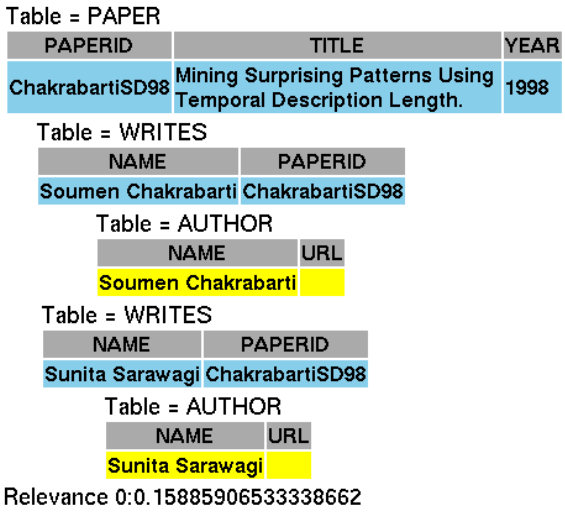

Figure 2: Result of query "soumen sunita"

BANKS, including relevance score computation, is described in more detail in Section 2.

BANKS provides a rich interface to browse data, with automatic generation of hyperlinks. The browsing component of BANKS is described in more detail in Section 3. The BANKS system is developed in Java using servlets and JDBC, and can be run on any database, without any programming. We are also developing a version of BANKS that handles XML data.

The greatest value of BANKS lies in almost zero-effort Web publishing of relational data which would otherwise remain invisible to the Web [2]. For example, BANKS may be used to publish organizational data, bibliographic data, and electronic catalogs.

A demo of the BANKS system is accessible over the Web at the URL:

`http://www.cse.iitb.ac.in/banks/`

2 Keyword search

Consider a fragment of a bibliographic database shown in Figure 1. This database contains paper titles, their authors and citations extracted from the DBLP repository. As we can see, due to normalization, information about a single paper is distributed across seven tuples related through foreign key references. A user looking for this paper may use queries like "sunita temporal" or "soumen sunita".

The BANKS system models the database as a directed graph, with each tuple in the database corresponding to a node in the graph. Each foreign-key–primary-key link is modeled as a directed edge between the corresponding tuples. (This can be easily extended to other type of connections.)

A keyword query in BANKS consists of $n \geq 1$ search terms $t_1, t_2, \ldots, t_n$. The first step is to locate the set of nodes matching search terms; a node matches a search term if it contains the search term as part of an attribute value or metadata (such as column, table or view names). Let S_i denote the set of nodes matching t_i; a node may match more than one search term, so the S_i's may overlap. Intuitively, an answer to a query is a subgraph connecting some set of nodes A that "cover" the keywords, i.e., each keyword must match one of the nodes in A. Just by looking at a subgraph it may not be apparent what information it conveys. We wish to also identify a "central" node in the subgraph, that connects all the keyword nodes, and strongly reflects the relationship amongst them.

An answer to a query is therefore modeled as a rooted directed tree containing at least one node from each S_i; edges are directed away from the root. The motivation for directionality is outlined later in this section. Note that the tree may also contain nodes not in any S_i and is therefore a Steiner tree. Figure 2 shows a sample result of a query containing the keywords *soumen* and *sunita* executed on the bibliographic database. Indentation is used to depict the tree structure, and nodes containing keywords are distinguished by their color.

2.1 Answer Relevance

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 a 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. The link between *Paper* and *Cites* tables can correspondingly be given a higher weight. The weight of a tree is proportional to the total of its edge weights, and the relevance of a tree is inversely related to its weight.

The example in Figure 1 illustrates that some links point toward the root of the tree, instead of away from the root as required by our model. For instance, the *Writes* relation has foreign keys to the *Paper* and *Author* relations, whereas we require paths from *Paper*

to *Author*, traversing a foreign key edge in the opposite direction. However, we cannot simply regard the edges as undirected. Ignoring directionality would cause problems because of "hubs" that are connected to a large numbers of nodes. For example, in a university database a department with a large number of faculty and students would act as a hub. As a result, many nodes would be within a short distance of many other nodes, reducing the effectiveness of tree-weight based scoring mechanism.

To solve the problem, we create for each edge (u, v) a *backward edge* (v, u); in the example from Figure 1, the backward edges ensure that there is a directed tree rooted at the paper, with a path to each leaf. We set the weight of (v, u) to the weight of (u, v) multiplied by a function of the number of links to v from the nodes of the same type as u. Experiments with different functions indicated that the function $\log(1+x)$, where x is the number of inlinks, provided good results [3]. (If there was already an edge from v to u, we set the edge weight to the lower of the original edge weight and the weight computed above.)

BANKS incorporates another interesting feature, namely node weights, inspired by prestige rankings such as PageRank in Google [4]. With this feature, nodes that have multiple pointers to them get a higher node weight (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. As another example, in a TPCD database storing information about parts, suppliers, customers and orders, the orders information contains references to parts, suppliers and customers. As a result, if a query matches two parts (or suppliers, or customers) the one with more orders would get a higher prestige.

In the current implementation we set the node weight to a function of the in-degree of the node. We experimented with different functions, and got good results with the function $\log(1 + x)$, where x is the in-degree. Our use of logarithms in edge and node weights is similar to term weighting schemes in information retrieval.

Node weights and tree weights are combined to get an overall relevance score. We experimented with additive and multiplicative combinations, and found that both worked well when the relative weights for the two scores were appropriately chosen. Details of the search algorithm and the relevance computation, along with a preliminary performance study can be found in [3].

Although a few other systems implement keyword search on databases (e.g., [5, 1, 6]) BANKS differs from all prior work in several ways: notably, in the techniques for edge weight computation and prestige based ranking, and the use of an in-memory graph structure for very efficient search while keeping the bulk of the data disk resident. The connections of BANKS to related work are described in more detail in [3].

2.2 Extensions

The BANKS system supports iterative refinement of queries:

- If multiple nodes match a keyword, the user can select one or more nodes as being relevant and ignore others; as an example, two authors in the DBLP database match the keyword "sudarshan", and the user can choose one of them and execute a query matching it with other keywords.
- Users can request more answers similar to one of the displayed answers; similarity can be defined on the basis of the answer tree structure.

Other refinements to tune node and edge weights are also under development.

Instead of displaying trees consisting of explicit tuples, system developers can specify answer formats based on the type of the root of the answer trees. For instance, one can specify author, conference/journal and year be displayed whenever the root node is from the *paper* relation. We are currently working on implementing answer formatting, and on supporting negation and disjunction in queries.

3 Browsing

The BANKS system provides a rich interface to browse data stored in a relational database. The system automatically generates browsable views of database relations and query results; no content programming or user intervention is required.

Every displayed foreign key attribute value becomes a hyperlink to the referenced tuple. In addition, primary key columns can be browsed backwards, to find referencing tuples, organized by referencing relations (users can select a specific referencing relation).

Each table displayed comes with a variety of tools for interacting with data.

- Columns can be projected away (dropped), and selections can be imposed on any column.
- For foreign key columns, clicking on "join" results in the referenced table being joined in, and its columns also displayed. This eliminates the need for explicitly writing join queries for the normal case of foreign key join. The join feature can also be used in the other direction, from a primary key to a referencing foreign key.
- Results can be grouped by a column; this results in only the distinct values for that column being displayed. The user can click on any of the values to see the tuples associated with that value.
- Tuples in the displayed table can be sorted by a specified column.

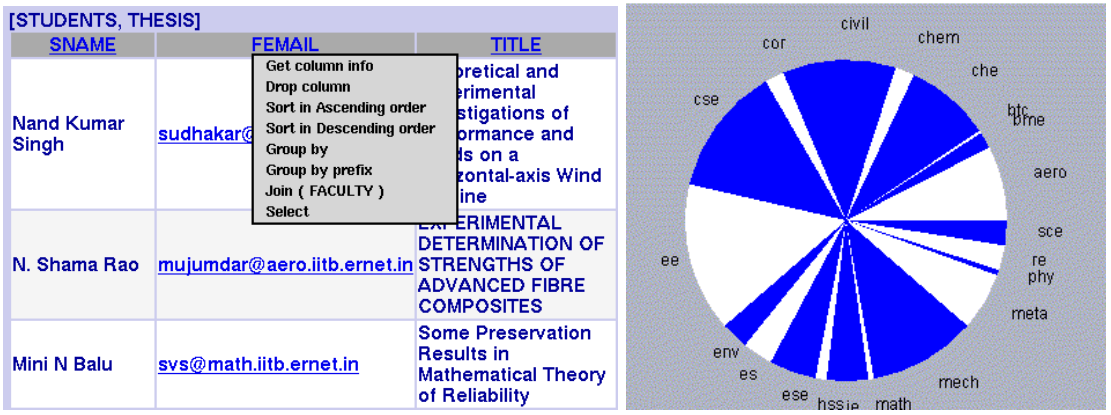

Figure 3: Browsing Examples: (a) Sample browsing session (b) Pie chart

Controls for these operations can be accessed by clicking on the column names in the table header. In addition, displayed data is paginated, and schema browsing is supported.

Figure 3(a) shows the result of browsing the thesis database starting with the *student* relation, using a pop-up menu on the roll number attribute to effect a join with the *thesis* relation and dropping several columns. The join is made possible since the *thesis* relation has a foreign-key attribute referencing the *student* relation. A sample pop-up menu is shown for the *femail* attribute which references the *faculty* table.

Hyperlinks in the displayed data are automatically generated by the system. Each hyperlink corresponds to an SQL query that is executed when a user clicks on the links. Thus, all the pages in the system are generated on the fly by executing corresponding queries against the underlying relational database. No precomputation is required.

BANKS *templates* provide several predefined ways of displaying data. Template instances are customized, stored in the database, and given a hyperlink name, which is used to access the template. The BANKS system currently provides four types of templates:

- Cross-tabs (similar to OLAP cross-tabs), with drill-down facilities.
- The group-by template provides a hierarchical view of data, by specifying a sequence of grouping attributes.
- Folder-tree views, which provide another hierarchical view of data.
- The graphical interface template permits information to be displayed in *bar chart*, *line chart* or *pie chart* format. Hyperlinks are provided on the graphical data via HTML image maps, to allow drill down on the data. Figure 3(b) shows an example pie chart generated by BANKS.

Another interesting feature of templates is that they can be composed together in a hyperlinked, visual manner. Several example templates are available on the BANKS web site.

4 Conclusions

To summarize, we have developed an integrated system for keyword searching and browsing of databases. The system has many useful features which allow casual users to access database information in an intuitive manner. BANKS enables almost effortless Web publishing of relational and XML data that would otherwise remain (at least partially) invisible to the Web.

We have also developed a prototype version of BANKS that works on XML data, supporting keyword searching and scalable browsing of large XML data sets. We are working on integrating XML search/browsing with the rest of the BANKS system.

References

[1] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A system for keyword-based search over relational databases. In *Procs. ICDE*, Feb. 2002.

[2] Peter Bailey, Nick Craswell, and David Hawking. Dark matter on the Web. In *Poster Proceedings, 9th World-Wide Web Conference*, 2000.

[3] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *Procs. ICDE*, Feb. 2002.

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

[5] Shaul Dar, Gadi Entin, Shai Geva, and Eran Palmon. DTL's DataSpot: Database exploration using plain language. In *Procs. VLDB*, 1998.

[6] Vagelis Hristidis and Yannis Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Procs. VLDB*, Aug. 2002.