DbSurfer: A Search and Navigation Tool for Relational Databases

Richard Wheeldon, Mark Levene and Kevin Keenoy

School of Computer Science and Information Systems
Birkbeck University of London
Malet St, London
WC1E 7HX, United Kingdom
{richard,mark,kevin}@dcs.bbk.ac.uk

Abstract. We present a new application for keyword search within relational databases, which uses a novel algorithm to solve the join discovery problem by finding Memex-like trails through the graph of foreign key dependencies. It differs from previous efforts in the algorithms used, in the presentation mechanism and in the use of primary-key only database queries at query-time to maintain a fast response for users.
Keywords: Relational Databases, Hidden Web, Search, Navigation, Memex, Trails, DbSurfer, Join Discovery, XML

1 Introduction

> "Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation)."
>
> E. F. Codd [4]

We consider that for many users of modern systems, being protected from the internal structures of pointers and hashes is insufficient. They also need to be spared the requirement of knowing the logical structures of a company or of its databases. For example, customers searching for information on a particular product should not be expected to know the address at which the relevant data is held. But neither should they be expected to know part numbers or table names in order to access this data, as required when using SQL.

In 1945, Vannevar Bush envisaged a future machine called Memex which would help the user build a "web of trails". His seminal paper "As We May Think" [3] first suggested the concept of a trail as a sequence of connected pages. This concept is now well established in the hypertext community.

We have previously developed tools which automate trail discovery for providing users with navigational assistance and search facilities within web sites and

Javadocs [12]. These tools have been shown to enable users to find information in less time, with fewer clicks, and with a higher degree of satisfaction [9].

Building on this work we have developed a tool called *DbSurfer*, which provides an interface for extracting data from relational databases. This data is extracted in the form of an inverted index and a graph, which can together be used to construct trails of information, allowing free text search on the contents. The free text search and database navigation facilities can be used directly, or can be used as the foundation for a customized interface.

Recent work at Microsoft Research [1], the Indian Institute of Technology[7,11] and the University of California [6] has resulted in several systems similar in many ways to our own. However, the system we describe here differs greatly in the design of the algorithms and in the style of the returned results. Our system also offers the opportunity for integrating both web site and database content with a common interface and for searching both simultaneously.

The rest of the paper is organized as follows. In Section 2 we describe our methods of indexing relational databases for keyword search. In Section 3 we describe the algorithms for extending this to compute joins by building trails, and give examples of the results achieved. Section 4 gives an overview of the system's architecture and Section 5 discusses the query options and syntax available to users of DbSurfer. An extended version of this paper [14] gives further details on the algorithm architecture. Further discussion of the metrics used and the alternative applications of the trail metaphor can be found in the first author's thesis [12].

2 Indexing a Relational Database

A single relation (or table) is a set of rows each of which can be addressed by some primary key. To index these rows we extract the data from each row in turn and construct a *virtual document* or web page, which is indexed by our parser. Since a relational database can be viewed as a special case of a more general model of semistructured data and XML, it might not be suprising that we can handle XML data using DbSurfer. Indeed that is all DbSurfer does! The virtual documents are XML representations of relational tuples, compatible with the emerging SQL/XML standard [5]. By combining the database reader with our web crawler, XML documents discovered on web sites can be automatically recognized as such and indexed in the same way, as can XML documents stored in the database, thus increasing coverage. The entries in the posting lists provide references to a servlet which produces a customized page for each row entry by rebuilding the virtual document and applying an XLST stylesheet. The textual content of each document is extracted and stored in an inverted file, such that the posting lists contain normalized *tf.idf* entries as prescibed by Salton [10]. Attribute names are also indexed as individual keywords so that, for example,

the query "Anatomy of a search engine author" should return trails from the Google anatomy paper [2] to the entries for Sergey Brin and Larry Page.

Answers to users' queries may be spread over several tables, which must be joined together. We can answer such queries with the help of a *link graph*. We have shown how we can create an inverted file containing URLs, some of which reference traditional web pages and some of which reference servlets which return customized views of database content. All these URLs are assigned a separate 32-bit number which identifies them. It is these numbers which are stored in the inverted file, and it is these numbers which are stored in the link graph. The link graph is constructed by examining the foreign key constraints of the database (either by accessing the data dictionary table or via the JDBC APIs) and the data entries themselves. Each matching set of (*table*, *attribute*) pairs where there is a recognized referential constraint generates a bi-directional link.

3 Computing Joins with Trails

Given a suitable link graph, we can utilise our *navigation engine* approach to construct join sequences as trails. The navigation engine works in 4 stages. The first stage is to calculate scores for each of the nodes matching one or more of the keywords in the query, and isolate a small number of these for future expansion. The second stage is to construct the trails using the *Best Trail* algorithm [13,12]. The third stage involves filtering the trails to remove redundant information. In the fourth and final stage, the navigation engine computes small summaries of each page or row and formats the results for display in a web browser.

Selection of *starting points* is done by combining the $tf.idf$ scores for each node with a ranking metric called *potential gain*, which rates the navigation potential of a node in a graph based upon the number of trails available from it. The Best Trail algorithm takes the set of starting nodes as input and builds a set of navigation trees, using each starting point as the root node. Two series of iterations are employed for each tree using two different methods of probabilistic node selection. Once a sufficient number of nodes have been expanded, the highest ranked trail from each tree is selected. The subsequent set of trails is then filtered and sorted. With appropriate choice of parameters, the Best Trail algorithm can emulate the simpler best-first algorithm.

Trails are scored according to two simple metrics: the sum of the unique scores of the nodes in the trail divided by the length plus a constant, and the weighted sum of node scores, where weights are determined by the position in the trail, and the number of repetitions of that node. These functions encourage non-trivial trails, whilst discouraging redundant nodes.

Filtering takes place using a greedy algorithm and removes any sequences of redundant nodes which may be present in the trail. Redundant nodes are nodes

which are either deemed to be of no relevance to the query or replicate content found in other nodes.

Once they have been filtered and sorted, the trails are returned to the user and presented in our *NavSearch* interface, the two main elements of which are a *navigation tool bar* comprising of a sequence of URLs (the "best trail") and a *navigation tree window* with the details of all the trails. Figure 1 shows how trails would be presented in the navigation tree window as a response to the question "vannevar bush" on a database generated from the DBLP data set. The content of any row can be examined in an adjacent frame by clicking on any likely looking entry or by examining the summary data in the enhanced tooltips.

As a preliminary evaluation of DbSurfer's performance, we ran two experiments on the DBLP corpus. In the first experiment, we selected 20 papers from the DBLP corpus, and constructed 20 queries by taking the surname of the first author and 1, 2 or 3 significant keywords with which a user might expect to identify that paper. We submitted these queries to DbSurfer for evaluation. We also submitted them to BANKS (Browsing ANd Keyword Search in relational databases) [7] and CiteSeer [8] for comparison. The key result found was that DbSurfer performed well (and outperformed BANKS and Citeseer) in finding requested references. The second experiment provided a closer analysis of the times taken is computing the results. Computing scores for nodes takes around 50% of the total processing time, with the trail finding taking around 30%, computing the text summaries around 15% and filtering redundant information around 2%, with the remainder being taken up by system overhead, XML transformation and presentation. Increasing the number of keywords causes a limited increase in the time to compute page scores, but this impact is dwarfed by other factors. One other interesting result is that as the number of keywords increases so does the fraction of nodes in the returned trails which are distinct within the entire trailset. Only extensive user testing will confirm whether this is a positive feature.

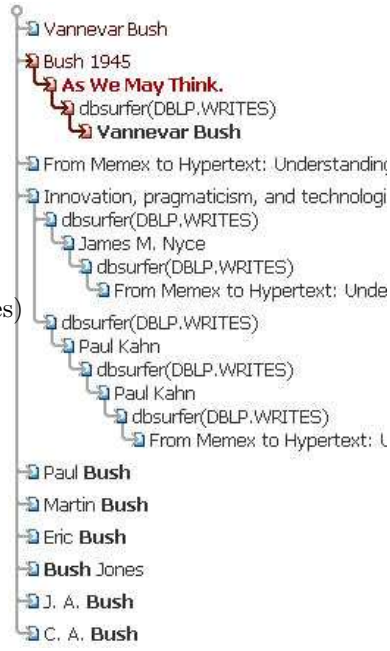

Fig. 1. Trails given by DbSurfer for the query "vannevar bush".

4 Architecture

Conventional web search engines usually use an architecture pattern comprising three components - a robot or crawler, an indexer and a query engine. We extend this design by augmenting the information retrieval engine with our trail finding system. and combining the crawler with the database reader. A key difference betweeen the DbSurfer and a conventional search engine is that a search engine traditionally returns links to pages which are logically and physically separated from the pages of the servers performing the query operations, whereas the links returned by the DbSurfer refer mostly to the row display servlet we have described.

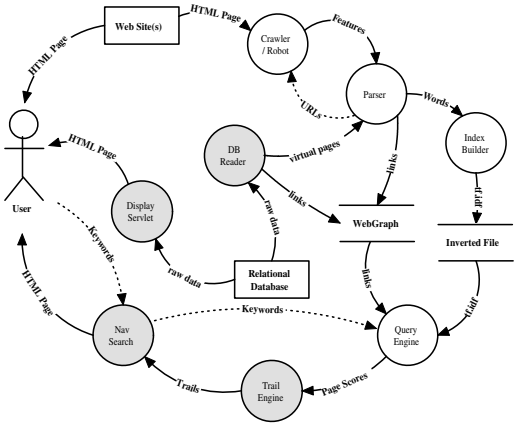

Fig. 2. Architecture of DbSurfer.

Figure 2 shows the basic architecture. The database content is retrieved by the DB Reader when the index is built and by the display servlet when examining the constructed trails. The DB Reader selects all the accessible tables and views, and asks the administrator which of these should be indexed. The program then extracts the referential constraints for all of the selected tables and generates a lookup table. This is kept separate from the main index and is used by both the DB Reader and the display servlet.

5 Query Expressiveness

We have extended the search engine style query syntax to support an attribute container operation using the "=" sign. The construct $x = y$ means that an attribute y must be contained in an XML tag x. For example, the query "Simon" might return publications relating to Simon's probabilistic model as well as articles by authors named Simon. The query `author=simon` would restrict the returned entries to those contained in an XML attribute ⟨`author`⟩, which translates to those in the author table. i.e. publications written by authors named Simon. The search engine query operations such as `+`, `-` and `link:` still remain supported with this extension. By default, we provide trails which answer disjunctive queries, with preference for results containing as many keywords as possible (conjunctive).

6 Concluding Remarks

We have presented DbSurfer - a system for keyword search and navigation through relational databases. DbSurfer's unique feature is a novel join discovery algorithm which discovers Memex-like trails though the graph of foreign-to-primary key dependencies. DbSurfer allows queries to be answered efficiently, providing relevant results without relying on a translation to SQL. We hope that the work will be continued by improving the user interface, allowing effective handling of numerical queries and addressing the security implications of the current architecture.

References

1. Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proceedings of IEEE International Conference on Data Engineering*, pages 5–16, 2002.
2. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of International World Wide Web Conference*, pages 107–117, Brisbane, 1998.
3. Vannevar Bush. As we may think. *Atlantic Monthly*, 76:101–108, 1945.
4. E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.
5. Andrew Eisenberg and Jim Melton. SQL/XML is making good progress. *SIGMOD Record*, 31(2):101–108, 2002.
6. Vagelis Hristidis and Yannis Papakonstantinou. Discover: Keyword search in relational databases. In *Proceedings of the 28th VLDB Conference*, Hong Kong, 2002.
7. Arvind Hulgeri, Gaurav Bhaltoia, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword search in databases. *Bulletin of the Technical Committee on Data Engineering. Special Issue on Imprecise Queries*, 24(3):22–32, 2001.
8. Steve Lawrence, Kurt Bollacker, and C. Lee Giles. Indexing and retrieval of scientific literature. In *Eighth International Conference on Information and Knowledge Management, CIKM 99*, pages 139–146, Kansas City, Missouri, November 1999.
9. Mazlita Mat-Hassan and Mark Levene. Can navigational assistance improve search experience: A user study. *First Monday*, 6(9), 2001.
10. Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24:513–523, 1998.
11. N. L. Sarda and Ankur Jain. Mragyati : A system for keyword-based searching in databases. *Computing Research Repository*, cs.DB/0110052, 2001.
12. Richard Wheeldon. *A Web of Trails*. PhD thesis, Birkbeck University of London, October 2003.
13. Richard Wheeldon and Mark Levene. The best trail algorithm for adaptive navigation in the world-wide-web. In *Proceedings of 1st Latin American Web Congress*, Santiago, Chile, November 2003.
14. Richard Wheeldon, Mark Levene, and Kevin Keenoy. Search and navigation in relational databases. *Computing Research Repository*, cs.DB/0307073, July 2003.