DAVID GROSSMAN
OPHIR FRIEDER
EDITED BY ERIK THOMSEN

Treating text as a relational application is a viable alternative for many data warehouses

Integrating Structured Data and Text

DAVID GROSSMAN [*grossman@iit.edu*] is an assistant professor of computer science and **OPHIR FRIEDER** [*frieder@iit.edu*] is the IITRI professor of computer science at the Information Retrieval Laboratory, Illinois Institute of Technology.

MPERATIVE 4
Realize the value of operational ITinvestments hrough integration and analysis aimed at delivering business advantage.

For a complete list of imperatives for the intelligent enterprise,visit ntelligentEnterprise.com.

VAST VOLUMES OF both structured and unstructured data commonly reside in a data warehouse.Data items such as name, address,and phone number are inherently structured and repetitive in nature.Thus, storing such data in relational platforms is easily accomplished. Many organizations have vast repositories of unstructured data such as email,corporate policy documents, internal system documentation, and procedures for dealing with customers. Users want this data in the warehouse, too.

But incorporating these text documents into a data warehouse typically poses a problem: Previously, accessing structured data and text with a single SQL query was not an option — relational systems were too slow to easily access text. The conventional concern with incorporating text into the warehouse typically stems from the belief that relational databases were never designed for text. Thus,many of today's applications incorporate text into a warehouse via Character Large Objects (CLOBs) or by implementing user-defined data types.

The primary difficulty with storing data in CLOBs is that these structures typically support only string-matching operations.Conventional search queries such as "find documents that describe all types of cars"will not find a document about a "Nissan Sentra" unless the document also lists the word "car." User-defined operators, on the other hand, lack standardization across platforms and applications, limiting query optimization. Query optimizers know quite a bit about in-house functions, but training the optimizer to support newly developed, special purpose operators is a nontrivial task.

Given this reality, our Information Retrieval Lab followed the development of relational systems and took a fresh look at the problem in the early 1990s.After several years of testing in the lab, we deployed numerous applications through our consulting relationships using this technology. Most notably, in 1999, a Web search engine running entirely on top of an Oracle database was deployed for the National Center for Complementary and Alternative Medicine. With this system, users are able to combine structured data and text in a single SQL query, making a separate text search system unnecessary.

HOW IT WORKS

Text is first parsed into a set of tokens or terms.Each term is then stored in a structure called an "inverted index." This structure indicates the documents that contain each term. Consider the following example:

D1: The GDP increased 2 percent this quarter.

D2: The economic slowdown continued this quarter.

For these two documents, a typical search engine would build an inverted index that contains a list of terms and a list of which documents contain each term. A possible inverted index implementation for the previous example is the following:

Frieder, O., A. Chowdhury, D. Grossman, M. C. McCabe, "On the Integration of Structured Data and Text: A Review of the SIRE Architecture," *DELOS Workshop on Information Seeking, Searching, and Querying in Digital Libraries*, Zurich, Switzerland, December 2000.

Grossman, D., D. Holmes, and O. Frieder, "A Parallel DBMS Approach to IR in TREC-3," *Overview of the Third Text Retrieval Conference (TREC-3)*, NIST Special Publication 500-225, April 1995.

Grossman, D. and O. Frieder. *Information Retrieval: Algorithms and Heuristics.* Kluwer Academic Press, 1998.

Grossman, D., D. Holmes, O. Frieder, D. Roberts. "Integrating Structured Data and Text: A Relational Approach." *Journal of the American Society of Information Science*, February 1997.

```
CONTINUED > [D2]
ECONOMIC > [D2]
GDP > [D1]
INCREASED > [D1]
PERCENT > [D1]
QUARTER > [D1] > [D2]
SLOWDOWN > [D2]
THE > [D1] > [D2]
THIS > [D1] > [D2]
TWO > [D1]
```

Typically, you eliminate "stop-words" — words that appear in almost all the documents such as "the" and "this" — from the index because they don't differentiate among the documents. We kept them for illustrative purposes. In our example, the term "quarter" also appears in both documents and thus contains a pointer to a list that contains both D1 and D2.

Essentially, the inverted index is just a many-to-many relationship between terms and documents. One term can be in many documents, and certainly one document can contain many terms. The ER diagram shown in Figure 1 illustrates this relationship.

The "document relation" in the ER diagram contains an entry for each document. This entry is usually a unique document identifier, such as the author, creation date, or title. The "term relation" stores information about each term,such as weights. The weights are ultimately used to rank documents in the order of a computed estimate of their relevance to the query. Numerous weights exist, but for now you only need to know that they can be stored in the term relation.

Rate this column at IntelligentEnterprise.com

Once you have this ER diagram, you can build tables and start running queries. A possible keyword search to find all documents that contain a term is the following:

```
SELECT DOCID
      FROM INDEX
      WHERE TERM = <FILL IN YOUR
FAVORITE KEYWORD>
```

TEXT AS A RELATIONAL APP

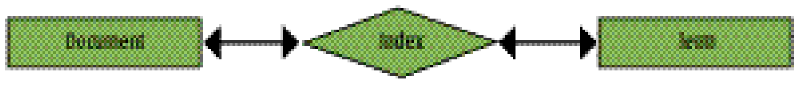
Numerous benefits exist for treating text as a relational application. For starters, you don't need to acquire, install, or integrate a text package into the data warehouse to support access to a few text columns. For example, almost every warehouse has a "comments" column or two that lets users enter whatever unstructured data they feel is relevant to the transactional record. But searching these text columns with a `LIKE` isn't really a good idea because it is typically implemented as a sequential scan.Building your own inverted index in a native OS file system may seem like an efficient alternative, but then you get to write a few thousand lines of code to do all the file manipulation, concurrency control, and access control that already comes with a relational database management system.

Treating text as a relational application also opens the door to parallel processing — something that has eluded the commercial text world because of the inherently sequential nature of the inverted index. The downside, obviously, is that extra overhead happens when you use a relational application, but didn't we go through this argument in the '70s when people were gripping that the relational approach was too slow and the best thing to do was to stick with ISAM files?

MORE NEXT MONTH

In the next column, we'll show how you can implement more complex text functionality (such as relevance ranking) and give some more details, performance statistics, and tuning hints on this approach. The bottom line is that treating text as a relational application is a viable alternative for many data warehouses, and it has been deployed in a number of real-world applications. We suspect that as the need for integration of structured data and text increases, more applications will consider solutions similar to the one discussed here. ie

FIGURE 1 *An ER diagram showing the inverted index.*



REAL-WORLD APPLICATION
THE SIRE APPROACH

A SCALABLE INFORMATION Retrieval Engine (SIRE) was developed using the concepts of integrating structured data and text. The National Institute of Health (NIH) chose the SIRE approach as the basis for searching medical citations for its National Center for Complementary and Alternative Medicine because, in addition to typical search engine functions, it contains all the basic DBMS features, such as concurrency control, recovery, access control, and portability. More important, the medical citations index can now modify a document that has already been indexed for search. Such updates are difficult or impossible for a typical inverted index, but easy with DBMSs.

With SIRE, the system is easily extended to access other structured data in databases at NIH. Back at the lab, we are working to add XML functionality to SIRE and have built a prototype that uses XML-QL(a popular XML query language) that should be ready at the end of 2001.