Enhancing XML search with XQuery 1.0 and XPath 2.0 Full-Text

P. Case

Powerful queries of character strings, numbers, dates, and nodes are familiar to users of relational database systems. Full-text database search systems feature queries that (1) use logical, proximity, and starts-with operators, (2) offer user control of case and diacritics, stemming, and wildcards, and (3) support thesauruses, taxonomies, and ontologies. Two emerging standards, XQuery 1.0 and XQ/XPFT (XQuery 1.0 and XPath 2.0 Full-Text), combine the search capabilities of the aforementioned systems and promise to change the face of full-text searching. This paper explores the expected benefits of these standards in searching relational data and full-text documents in XML from an end user point of view and describes how these benefits would apply to a search system used at the Library of Congress. These standards have the potential to solve many real-life full-text search system problems and to restore the end-user control necessary to enable and facilitate complex searching.

INTRODUCTION

XML promises many things to many people. Some expect XML to create state-of-the-art tools for document composition. Others expect it to enable the production of multiple displays from a single source document. Still others expect it to facilitate the interchange of data and documents through Web services. For full-text searching, the promise lies in search systems that are more powerful and composable (i.e., have components that can be selected and assembled in various combinations). Two forthcoming standards from the World Wide Web Consortium (W3C**), *XQuery*[1] and *XQuery and XPath Full-Text*[2] (XQ/XPFT), may represent the first steps in merging and improving search technologies for relational databases and full-text documents. Once relational data and full-text documents are both stored in XML, end users can profit from being able to search them together with a single search engine, exploiting the power of both relational database querying and full-text searching.

Currently, more and more full-text search functionality is being exercised by the use of scoring and categorization algorithms. Scoring algorithms produce scores for search results that predict the relevance of those results to the full-text search conditions. Categorization algorithms cluster search results into categories and display them by category on the search results page. These and similar algorithms provide some convenience, but also diminish user control over the process of obtaining and displaying search results. In contrast, the intent of this paper is to urge researchers and vendors to think about the needs of end users, especially expert users, and to consider researching and implementing XQ/XPFT, thereby restoring the degree of end user control that is necessary for complex searching.

XQ/XPFT offers full-text search functionalities which are becoming less common in full-text search systems today, such as logical, proximity, and starts-with operators; user control of case and diacritics; stemming and wildcards; and support for thesauruses, taxonomies, and ontologies. The depth and breadth of the available functionalities are easily discovered by browsing the *XQuery and XPath Full-Text Use Cases*.[3] These functionalities have been implemented in the GalaTex XML Full-Text search engine prototype,[4] a conformant implementation of XQ/XPFT.

In January 2005, the Pew Internet and American Life Project released a report on online activites which was subtitled: *Internet Searchers are confident, satisfied, and trusting, but they are also unaware and naive*.[5] It noted that only 7 percent use more than three search engines on a regular basis and that those users are less satisfied with open Web searching. For end users of full-text database search systems, the number may be higher. Though 7 percent is a small percentage of users, these users are doing a different kind of search and have higher expectations. They often must find not just some results, but all the appropriate results. Their work may be critical, for example, Congressional staff giving their member of Congress all the bills on cloning introduced in the latest session. It is critical that these advanced users find the results they need.

This paper explores the expected benefits of using XQuery and XQ/XPFT to search relational data and full-text documents from an end user's point of view. These benefits may be advantageous to all end users of full-text search systems, but the emphasis here is on one type of full-text search system, namely, full-text database search systems. Full-text databases are defined as databases primarily containing full-text documents that often contain data as well, in forms including names, titles, dates, and numbers. This data may be treated as document metadata or as related data. The documents and data are homogenous in format and content and are currently stored and searched by proprietary software. The majority of the vendors of full-text search-engine software have not participated in standardization efforts, such as the efforts involving SQL (Structured Query Language) by relational database vendors. Examples of full-text database search system implementations include the databases of LexisNexis[6] (containing legal, tax, and regulatory information), Dialog[7] (providing information services in such fields as business, science, engineering, finance, and law), and FirstGov[8] (the official United States gateway to all government information, including full-text databases available to the public). Full-text database search differs from enterprise search and other types of full-text searching.

This paper focuses narrowly on the expected benefits of the XQ/XPFT standards to full-text database searching and on the real-life full-text search system problems they have the potential to solve. It illustrates the utility of these standards by specific examples from a full-text database search system used at the Library of Congress. The views expressed in this paper are those of the author and do not necessarily represent those of the Library of Congress.

DISTINCTIONS BETWEEN RELATIONAL DATA AND FULL-TEXT DOCUMENT SEARCHING

Queries in relational databases run against structured data, such as character strings, dates, numbers, and nodes. For example, a character string query on *man* may find the character string *semantics*; a date range search on the range August 22, 1951 through October 28, 2005 may find data containing the initial date, ending date, and the dates in between; a node search may be run to determine whether a node of that name exists. Relational database systems have capabilities for finding and transforming character strings, dates, numbers, and nodes; these capabilities would be useful but are not currently available in full-text database search systems. The dominant markets for relational database querying are automated business transactions and content management. These markets have encouraged standardization and precision in search functionality, but have resulted in a decrease in the focus on full-text search in relational databases.

Full-text searches are performed on text that has been tokenized, that is, broken into a sequence of words, units of punctuation, and spaces. Searches in full-text find words and phrases instead of character strings. A full-text search for the word *man* returns the word *man* and possibly some synonyms, but it will not return the word *semantics*. The tokenized text-search capabilities of full-text search engines would be useful in relational database querying, but are not currently available in most relational databases. Full-text searching comes in many varieties, including open Web searching, Web site searching, enterprise searching (of varied network and intranet resources), federated searching (i.e., searching by transforming a query and broadcasting it to a group of disparate databases with the appropriate syntax), full-text database searching, and intra-document searching. They are similar in some ways and very different in others.

The dominant market for full-text search systems has become enterprise search, that is, searching the networks and intranets within a business or other organization for information such as phone directory entries, e-mails, shared documents, guidelines, databases, and other corporate resources. The focus of enterprise search is on searching heterogeneous documents in different formats (i.e., HTML, WPD, DOC, PDF) with heterogeneous content.

Full-text search engine vendors have tended to ignore standardization of the fundamental components of full-text searching. This has resulted in an amorphous state of the art, which is compounded by the tendency to include innovations in full-text search such as categorization, result clustering, and visualization in the enterprise search product category. For example, Gartner, Inc.'s *enterprise search* product category was renamed *information access technology* in 2005[9] in order to include the aforementioned innovations. While these innovative components are important, they cannot replace the functionalities addressed here, and building them into search engines (rather than making them available as plug-ins or add-ons) has led to a chaotic product category. This paper addresses only the fundamental components of full-text search systems and so continues to use the term *enterprise search*.

The distinction between relational databases and full-text search systems makes sense when looking at the dominant markets. The concentration of full-text search products on enterprise search is also understandable. The distinction, however, has had negative consequences for full-text database search systems. Full-text databases are often more homogenous in format and content than enterprise search systems. A simple word search in an enterprise search system may be more successful than a search in a full-text database search system. In a full-text database search system, an end user is more likely to get lost in too many results and to have difficulty distinguishing among them. Historically, only librarians and a few academics used multiple full-text databases. It is to be hoped that the growing number of end users conducting full-text database searches by means of Web interfaces, whether without charge or for a fee, will lead to the creation of products that better support full-text database searching and these new users, both novice and expert.

■ The use of XML is blurring the distinction between structured data and unstructured documents, making the disjunction between relational databases and full-text databases less and less defensible ■

The use of XML is blurring the distinction between structured data and unstructured documents. Previously unstructured documents, when they are converted to XML, gain at least minimal structure through the addition of semantically meaningful XML information such as title, author, and text tags. As this trend continues, it will become harder to tell data from documents, and the disjunction between relational databases and full-text databases will become less and less defensible.

The W3C standards for XQuery and XQ/XPFT also contribute to this trend. XQ/XPFT uses the XQuery data model (or *infoset*), extending it to retain word, sentence, and paragraph position numbers. The XQuery and XQ/XPFT standards are fully composable. XQuery relational data queries may be combined with and nested within XQ/XPFT searches and vice versa.

AN EXAMPLE OF A FULL-TEXT DATABASE SEARCH SYSTEM

The Library of Congress stores a wealth of relational data and full-text documents. Currently, in order to search data and documents together, the data must be moved from relational databases into full-text databases, where it is combined with full-text documents and then searched by a proprietary full-text search system. In this process, all of the functionality and standardization of relational databases is lost.

The example of full-text database search provided here was developed at the Library of Congress. Our team designs the search system architecture and interface for the Legislative Information System (LIS), a closed system available only to members of Congress and their staffs. LIS provides access to U.S. legislative information that is accurate, timely, and complete. It serves as a portal to House of Representatives, Senate, and Congressional support agency resources. It provides search interfaces for bills, committee reports, and the *Congressional Record*. LIS is not unique. It is typical of full-text database search systems at the Library of Congress and elsewhere. It is used here as a prototype for a search system that contains homogenous data and documents. The THOMAS system,[10] providing legislative information on the Internet, is the public version of LIS. Its search interface is different from the LIS interface, with fewer expert user options.

The transition to XML represents an opportunity to improve searching in LIS. Adding elements and attributes to structure full text is of course a boon, but the XQuery and XQ/XPFT standards will do much more for LIS. The first draft of the standards, *XQuery and XPath Full-Text Use Cases*, a W3C working draft as of this writing,[3] was prepared by Library of Congress end users and search system developers. It was then edited and enhanced by W3C XML Query and XSL (XML Stylesheet) Full-Text Task Force members. The functionalities in the *Use Cases* are those required by end users of LIS and other full-text database search systems at LC and elsewhere, and they will solve the search problems described in the following section. These search problems are typical of full-text database search systems.

THE BENEFITS OF THE XQUERY AND XQ/XPFT STANDARDS

In this section, we use LIS as an example to illustrate the benefits of the XQ/XPFT standards. Currently, like many systems, LIS is forced to choose between relational database query and full-text database search functionalities. LIS needs to port its data from a relational database to a database that supports full-text searching and thus loses the relational-database querying capabilities. XQ/XPFT-compliant search engines can offer full-text search functionalities and the relational database querying functionalities of XQuery in one package and in one system. In addition, these search engines should allow end users to combine and nest relational database queries with full-text searches, and vice versa.

The XQ/XPFT standards facilitate search engines that offer the following functions in a single system: exploiting the order and hierarchy of XML documents, ad hoc searching of any element and attribute, exact and scored searching, and provision of ordered and unordered distance operators. They also enable searching within a single instance of element, offer a full array of full-text search functionalities and of relational database queries on character strings, numbers, dates, and nodes, and allow the user to control the display of search results. These contributions and their application to LIS are described in the following subsections.

Exploiting the order and hierarchy of XML documents

The XQ/XPFT standards enable search engines to offer the user many advanced search options. Because XML documents are ordered and hierarchical, intra-document searches may be performed to locate documents that are related to each other in time and subject matter. For example, in the LIS system enhanced by these standards, a user can search for a document by a given author, which preceded or followed a given event, such as a speech by a member of Congress that preceded the introduction of a bill. Similarly, the user may search for the remarks of a given member or members of Congress that are tagged as related to (i.e., in response to) the remarks of another member of Congress in a particular debate. Inter-document and intra-document order, essential for fulfilling these requirements, are currently ignored by most full-text database search engines except where data is ordered by date.

Performing ad hoc searching of any element and attribute

Currently, in order to make a particular item of document-related information available to be searched, the related fields must be identified as searchable, field numbers and names must be assigned, and search programs and indexes must be modified. An example of such information would be "related bill information" (i.e., which bills are identical or similar to other bills) and the party designations of members of Congress. This is due to the format, used by most existing full-text database search engines, of predesignated fields. "Shoe-horning" XML elements and attributes into the predesignated field format does not provide unfettered access to the entire contents of a full-text database. Modifying predesignated fields is an unwelcome but common task, such as when a Congressional staffer calls with a complicated but urgent search request which necessitates searching on bill-related information elements that are not commonly searched, are not optimized for searching, or are not available through the graphical user interface.

With XQ/XPFT, these laborious field modifications will not be necessary (unless search optimization is desired). Search engines conforming to XQ/XPFT will be able to search and return the content of any element and the value of any attribute in any data or document (or document fragments) in the search system, treating XML tags as such and not as fields.

Performing exact and scored searching

Today, relational database systems support exact queries only on character strings, dates, numbers, and nodes. Scoring is supported only for full-text word and phrase searches. However, end users searching full text may want the exact phrase they search for (allowing for word variants and other full-text substitutions, such as for case and diacritics), and end users querying numbers may want to query, for example, for dollar amounts of roughly $10 million with results in ranked order, which requires scoring. XQ/XPFT-compliant search engines will enable both exact and scored searching on full text, eventually on all data types.

Providing ordered and unordered distance operators

Many search engines offer the option of advanced searching, but few people use these advanced search pages because they offer only marginal and unpredictable operators. On many full-text database search systems, the advanced search pages offer the operators shown in *Table 1*.

Table 1 Disadvantages of operators on advanced search pages

or	Useful only when searching for one of a group of synonyms; otherwise, finding either of two words anywhere in a document delivers too little precision to be useful.
and	Seldom useful; finding two words anywhere in a document still delivers too little precision to be useful.
not	Sometimes fails to return wanted results when the second operand is a subset of the first operand.
phrase	Often too precise and likely to miss wanted results.

The "or" operator is seldom successful as a full-text database default operator. The "and" operator may be useful in open Web searching or enterprise searching where the content is varied and non-repetitious, but in full-text database searching there is almost always a need to reduce the number of search results returned. As "and" is often the default operator for basic searching, it often adds nothing when offered from the advanced search pages. The "phrase" operator assumes more precision than usually exists in language. Allowing the user to specify an appropriate number of intervening words leads to more useful results; this is accomplished by the use of distance operators. If the distance operator is ordered, the words must appear in the document in the order that they appear in the search statement; if the distance operator is unordered, the words may appear in any order.

For example, a user may search for information on *elementary education* in the LIS database. The ordered distance search

(elementary) ***ordered distance of 10 words*** *(education)*

returns among other results *elementary and secondary education*, a very common phrase in the bills in LIS, and one that would be missed by the phrase search *elementary education.*

This search is also better than the search

elementary ***and*** *education,*

which returns results such as: *denial of the most elementary forms of personal freedom and human dignity . . . the need for continuing education*, which is unrelated to elementary education.

Few end users realize that the “not” operator sometimes fails to return wanted results. For example, in most full-text search engines, the query

mexico **not** “*new mexico*”

discards an entire document about Mexico if it mentions at any point that New Mexico was named after Mexico. Search engines that implement XQ/XPFT support a “mild not” operator, which is written as **not in**. The “mild not” operator does not discard a document result that contains both the first and second operands (Mexico and New Mexico in our example). For example,

estate tax **not in** *real estate tax*

does not discard a document that contains “real estate tax” if it also contains “estate tax” not preceded by the word “real.” Currently, pains are taken to rewrite LIS searches to remove “not” operators because “mild not” operators are not yet available. Search engines supporting XQ/XPFT use ordered and unordered distance operators and the mild not operators as defaults and offer them on advanced search pages.

Searching within a single instance of an element

A document, such as a bill in LIS, may have hundreds of instances of a field (currently) or element. For example, the bill status and subject fields of a document may be instantiated multiple times, as many documents have multiple authors and multiple subjects. Current full-text search engines search all instances of a field, which may lead to unwanted results. An unordered proximity search for the subject term *Housing for the Disabled*, will return the two subject terms *Housing* and *Disabled* when they follow each other among the assigned subject terms. Because a bill could easily be about housing and the disabled and not be about housing for the disabled, the bill returned would be an unwanted result. XQ/XPFT-compliant search engines allow the search to be limited to a single instance of an element, thus avoiding this problem, known as “subject term bleed.”

LIS end users also want to find bills with a specified legislative status step realized on a specified date. For example, they may want to find only bills with the status “passed by the Senate” on a given date. Currently end users can find bills passed by the Senate where the text of that legislative status step is close to the date specified, but LIS cannot ensure that the date does not apply to a preceding or subsequent legislative status step. Searching within a single instance of an element, which is easily done in relational database querying but impossible in most existing full-text database search systems, makes these searches return fewer unwanted results.

Offering a full array of full-text search functionalities

To deliver quality search results from a single-line search box, search engines or search engine plug-ins or add-ons need to perform real semantic searching. They need to know the difference, for example, between Congressional bills, bills that are paid, and ducks’ bills. They need to consider those supposedly insignificant words, which many search engines label “stop words” and drop from the search, words such as *du* and *in* in the phrase *Hotel du France in Cannes, France*, to determine meaning. The punctuation, such as the comma in this phrase, is also routinely dropped from searches. They need to linguistically analyze text and consistently return appropriate results. There may be some cross-pollination for full-text database search systems from the work being done on ontologies and Web services for the Semantic Web. Until semantic searching becomes available, full-text database search engines must return some control over searching to end users, and search interfaces for databases will often require more than a single-line search box.

XQ/XPFT search engines can be the vehicle for this restoration of user control, providing users with serious full-text search functionality. This functionality includes prefix, infix, and suffix wildcards, ordered and unordered distance operators, thesaurus integration, *starts-with* functionality, a safe “not” operator as described previously, and end user control over diacritics, case, and stop words.

Offering a full array of relational database queries on character strings, numbers, dates, and nodes

Like relational database systems, XQ/XPFT-compliant search engines need to enable searches on character strings, dates, numbers, and nodes. For example, LIS end users need to find and repurpose the dollar amounts in appropriation conference reports. LIS end users often need to search on ranges of contiguous bill numbers. The current full-text database search systems often require end users to enter numbers in a range one by one, not allowing a simple search such as "*hr1–10*" on the first 10 bills of a Congress from the House of Representatives. Most relational database querying functionalities are nonexistent or dramatically less robust in existing full-text search systems.

Offering end-user control of search results displays

Although not the first to do so, XQ/XPFT-compliant search engines can return any element, attribute, document, or document fragment in almost any combination and order, not only the data elements searched. End users usually only need a small part of what appears in standard displays of search results. The unwanted parts are distracting and often user-unfriendly. Giving end users control over the number of results returned or the sort is not sufficient; end users should be allowed to specify exactly which elements and attributes and which documents and document fragments are displayed and how they are displayed. Some end users repeat the same search regularly; some end users present the results to others. These end users and others would invest the time to customize the display of their results. XQ/XPFT-compliant search engines can allow the user to do so.

Although the XQuery and XQ/XPFT standards do not address highlighting, end users need intelligent highlighting on words, numbers, and any data within a result that meets the search criteria. Highlighting becomes intelligent when it highlights numbers and other data types as well as words and character strings, when it highlights words only in the designated relationships to other words in the search (such as, in a phrase only if the words appear within that phrase), when it does not highlight (or highlights in a different way) words that are operands of a "not" or "mild not" operator, and when it allows an end user to turn highlighting off and, if desired, highlight additional or different words.

CONCLUSION

The XQ/XPFT standards will not solve all of the problems of full-text database searching. Some capabilities that full-text database searching currently lacks are not affected by the XQ/XPFT standards and are not related to XML. These include returning fewer search results, returning no results when appropriate, computing result relevance, and employing a standard end user syntax.

If widely implemented, the XQ/XPFT standards will enable full-text database end users to profit from searches by using the tagging, hierarchy, and order of XML data and documents, from having access to functionalities previously only available in relational databases, from having full-text functionalities not offered by current enterprise search systems, and from standardization. These benefits will significantly improve searching for both advanced users and basic users (the latter through graphical user interfaces).

**Trademark, service mark, or registered trademark of Massachusetts Institute of Technology in the United States, other countries, or both.

CITED REFERENCES

1. S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon, *XQuery 1.0: An XML Query Language*, World Wide Web Consortium (November 2005), http://www.w3.org/TR/xquery.
2. S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, D. McBeath, M. Rys, and J. Shanmugasundaram, *XQuery 1.0 and XPath 2.0 Full-Text*, World Wide Web Consortium (November 2005), http://www.w3.org/TR/xquery-full-text/.
3. S. Amer-Yahia and P. Case, *XQuery 1.0 and XPath 2.0 Full-Text Use Cases*, World Wide Web Consortium (November 2005), http://www.w3.org/TR/xmlquery-full-text-use-cases/.
4. S. Amer-Yahia, P. Brown, E. Curtmola, and M. Fernández, *GalaTex: An XML Full Text Search Engine*, AT&T Labs Research, http://www.galaxquery.com/galatex/.
5. *Reports: Online Activities and Pursuits*, Pew Internet & American Life Project (2005), http://www.pewinternet.org/PPF/r/146/report_display.asp.
6. LexisNexis, Reed Elsevier Inc., http://www.lexisnexis.com/.
7. Dialog, The Thomson Corporation, http://www.dialog.com/.

8. FirstGov.gov, The U.S. Government's Official Web Portal, United States General Services Administration, http://firstgov.gov/.

9. W. Andrews and R. E. Knox, *Magic Quadrant for Information Access Technology*, Gartner, Inc. (October 2005).

10. THOMAS, The Library of Congress, http://thomas.loc.gov.

Accepted for publication December 13, 2005.
Published online April 27, 2006.

Pat Case
Congressional Research Service, Library of Congress, 101 Independence Avenue, SE, LM-223, Washington, DC 20540 (pcase@crs.loc.gov). Ms. Case holds a B.A. degree in philosophy and a Master's degree in library science. She has worked in academic, public, and special libraries. She currently works as a search-system and interface designer for the Legislative Information System (LIS), a closed system available only to members of Congress and their staffs. She is a member of the W3C XML Query Working Group and the Full-Text Task Force. ■