XQuery/IR: Integrating XML Document and Data Retrieval

Jan-Marco Bremer Michael Gertz
Department of Computer Science
University of California, Davis, CA
{bremer|gertz}@cs.ucdavis.edu

1 Introduction

One of the most important features of XML is to provide a unified view to all kinds of structured and semistructured data as well as loosely structured documents. Here, *document* means a coherent unit of usually textual information while *data* stands for uninterpreted, raw content without a fixed context. Most content XML is applied to today is very text-rich.

The structural aspects of the unified XML view are rigid enough to support *data retrieval* (DR) queries as known from database systems. Over the past few years, increasingly powerful query language, most notably the recent XQuery standard [2], have exploited this fact to provide expressive DR query capabilities for XML. On the other hand, XML's structural aspects are transparent enough to treat arbitrary parts of the XML-represented data as documents. Document or *information retrieval* (IR) providing one of the most important capabilities for querying text-rich documents, however, is not supported by XQuery or any of the earlier XML query languages so far.

DR provides means to formulate queries based on *exact* matches of data. IR is based on the notion of (relative) *relevance* of documents within a document collection. Thus, some major issues for the integration of IR into an XML query language arise: Today's (toy) collections of small XML documents are eventually to be replaced by large, hierarchically structured XML databases. In such databases, the notions of *static* document collection and document are lost and have to be replaced by some kind of view on documents and document collections that is established *dynamically* by a query. This poses challenges for new, embedded IR algorithms and their application *within* the host language rather than only as the final operation as in standard IR. However, besides these challenges, significant rewards in form of an increased expressiveness of the resulting integrated query language might arise.

IR in general is based on detecting fine differences in term distributions throughout a document collection leading to valuable information [12]. However, the value of information is highly context-dependent. Currently, the context for IR is always an entire, static document collection. This changes when term statistics for dynamic document views are available and exploited by an IR algorithm. A DR language is a natural means to specify a context for a relevance-based query. Moreover, when IR is truly embedded into the query language, derived information can be obtained after the IR-style ranking has identified the most relevant components. For example, the mean publication date for information related to a certain event reported in textual news naturally leads to an apex of that event.

In this paper, we present the results of our ongoing research to integrate IR capabilities into XQuery and through this, provide more expressive queries than DR or IR alone can answer. XQuery-related standards [2, 5] define document fragment sequences (DFSs) as in- and output for all intermediate and final query results. We identify document fragments (DFs) and DFSs as the equivalent of documents and document collections within XML queries. We introduce a single, new operator called *rank* into XQuery. The operator orders DFs within arbitrary intermediate DFSs. It is used through a *Rankby* expressions that is very similar to XQuery's *Sortby* expression. The operator is orthogonal to other XQuery operators and can be arbitrarily nested.

The rank operator itself imposes no requirements on the implementation of the relevance-based ordering of DFs. Thereby, we establish a general framework for research into query language-embedded IR algorithms. However, we define a *dynamic ranking principle* that captures IR algorithms which allow the richest type of queries based on a local context. These algorithms are based on term statistics only within the current DFS as returned by a (sub-)query. We briefly study consequences of the rank operator for index structures and query optimization.

Related work. Oracle's *contains* operator implements a standard, static IR approach based on table columns [1]. The operator is used within an SQL *Where* clause. In [6], only exact keyword search within certain XML elements is introduced into XML-QL. Earlier related approaches for structured documents can be found in [11]. The aim of [13] is an improved Web search by means of an IR-enabled XML query language. The similarity operator of [3] is mainly a vehicle to execute fuzzy similarity joins based on atomic values within two DFs. Again, XML-QL is used as host language. Based on XQL, [8] proposes an IR-style operator that computes similarity between DFs and queries by means of XML leaf elements' similarity weights that are propagated upwards. However, XQL is less expressive than XQuery and lacks important data retrieval capabilities.

Common to all of these approaches is the extension of a *Where* clause which implies a selection and thus, a partitioning of the input into relevant and non-relevant objects. The relevance-based sorting employed by our scheme more naturally reflects IR's ordering properties while still allowing the partitioning by means of a relevance threshold. Moreover, unlike our approach, none of the existing approaches is able to demonstrate meaningful and useful nested queries involving both DR and IR operations. We have not yet seen other work applying IR to a local context established by a DR (sub-) query or a previous IR-style query.

Passage retrieval approaches [9] have used more fine-grained document components to improve the ranking but without supporting IR on document parts as a local context. The feasibility and usefulness of IR index structures for semistructured data as required by our approach have been shown in [10].

Overview. In Section 2, we establish the background for this paper. In Section 3, we introduce the new operator and demonstrate its application. Section 4 discusses implementation issues of the operator including dynamic ranking, index structures and query optimization. In Section 5, we conclude this paper and briefly point out some future work.

2 Background

2.1 XML Data Model

We employ an XML model in which documents and fragments of documents are represented as ordered, node-labeled trees. We leave out details such as comments, processing instructions, references, and the distinction between elements and attributes.

Assume a set E of element names and a set T of text string values disjoint from E. Given a set X, let $L(X)$ denote the set of all lists that can be built over elements from X. Then an *XML document fragment* F is a 4-tuple $(V, r, \mathit{label}, \mathit{elem})$ where

- V is a set of vertices with a distinguished element r, called the *root node*,
- *label* is a mapping from vertices to element labels, i.e., $\mathit{label} : V \rightarrow E$, and
- *elem* is a mapping from vertices to their children, i.e., $\mathit{elem} : V \rightarrow L(V \cup T)$.

Let an *XML document* D be an XML document fragment (DF). Furthermore, let $\mathcal{F}$ denote the set of all document fragments over E and T. Then an XML document fragment sequence (DFS) S is a sequence of elements of $\mathcal{F}$, i.e., $S \in L(\mathcal{F})$. Let $First_k(S)$ $(k \in \mathbf{N} \cup \infty)$ denote the sub-sequence of S consisting of the first k or $|S|$ elements, whichever is smaller.

2.2 XQuery

XQuery [2] combines features from several earlier XML query languages, in particular XPath [4]. Through XPath, DFs can be extracted from an XML document. Nested loops iterate over these fragments to further extract DFs and construct sequences of output DFs. Variable assignment supports complex computations based on content and structure of the input.

Details of XQuery's formal semantics can be found in [7] and are not discussed here. For our approach, it is sufficient to note that in- and output to XQuery queries are always DFSs, i.e., a query q is a mapping $q : L(\mathcal{F}) \rightarrow L(\mathcal{F})$. We illustrate the syntax and semantics of XQuery by means of a few simple examples. We will extend these examples in Section 3 to illustrate our new operator.

Example 2.1 *Select paragraphs of articles dating back to Feb. 15th, 2002 from the* news *document database.*

```
document("news.xml")
      //article[./date="2002-02-15"]//paragraph
```

The example consists of a single XPath expression, which alone is a valid XQuery query.

Example 2.2 *List all articles that appeared before 1996 with their first author and title, in sorted order.*

```
FOR $a IN document("bib.xml")//article
WHERE $a/year < 1996
RETURN
 <early_paper>
```

```
  <fstAuth> {$a/authors/author[1]/text()} </fstAuth>
   {$a/title}
  </early_paper>
SORTBY (author[1], title DESCENDING)
```

Example 2.2 consists of a single loop that goes through articles and extracts the author and title. The complete title DF builds a part of the result fragments. A new tag name fstAuth is introduced for the author. The returned fragments are ordered by author and title.

Example 2.3 *Convert a list of news articles classified under a certain category to a list of categories with their related articles.*

```
<news_by_category>
{FOR $c IN document("newsmeta.xml")//category
 RETURN
  <category>
   <name> {$c/name/text()} </name>
   {FOR $a IN document("news.xml")
                      //article[@cid = $c/@id]
    RETURN
     <title aid={$a/@id}> $a/title/text() </title>
  </category> }
</news_by_category>
```

Example 2.3 represents a join between category and article DFs based on a news category identifier. The join is implemented as two nested loops.

2.3 Document Retrieval

Document retrieval or, more commonly used, *information retrieval* (IR) is concerned with ordering documents from a document collection by relevance to a query [12]. The query is usually simple and consists of just a few terms. Relevance is based on term distribution statistics. A numerical weight is assigned to each term occurrence in each document. The weight represents the term's significance within the document content. The relevance ordering is obtained by summing up query term weights for every document and determining the highest sum. A second flavor of IR is only concerned with partitioning the collection into relevant and non-relevant documents. It can be implemented by means of the ordering approach and the application of a threshold below which documents are regarded as not relevant.

A standard approach for weighting terms is the term frequency-inverse document frequency (tf-idf) approach [12]. Tf-idf assigns higher weights to term occurrences with high in-document and low overall document frequencies, i.e., few documents that contain the term.

Term distributions may vary widely throughout a document collection. Hence, the significance of a term occurrence and thus, its relevance, highly depends on the context. In most existing retrieval approaches, the context is always the complete document collection.

3 XQuery Rank Operator

In traditional IR, the result of a usually stand-alone query is a total order or partitioning of documents that are the single unit within a collection. The result is directly presented to the user. When integrating IR into an XML query language, this raises two major questions:

1. What is the appropriate equivalent for documents and document collections within a single database-like XML document source (or set of such)?

2. How and where can IR be of use within a query? In particular, can it make sense to consider IR not only as a final operation, but one that does something meaningful in an intermediate query step?

XML queries extract document fragments (DFs) from XML sources. DFs are the only data unit suitable to replace the notion of document from standard IR. In XQuery in particular, all results are *sequences* of DFs (DFSs). It is only natural to modify the order of DFs within a DFS by means of an ordering IR approach, very similar to the sort operator in XQuery. This can be accomplished by a single operator. DFSs replace the notion of document collections. The order is transparent to subsequent queries that do not rely on any order, but can be exploited by other sub-queries or shown as an end result. The relevance-based partitioning can then be implemented on top of the ordering IR as discussed in Section 2.3.

The above observations let us derive the following requirements for integrating an IR operator into an XML query language, in particular XQuery.

Total order. The operator should be able to order a sequence of DFs based on relevance.

Local context. The partitioning flavor of IR should be supported to establish a local context for subsequent queries.

Closure. The operator should be closed within XQuery and thus, be applicable within arbitrary XQuery expressions.

Transparency. The operator should not affect queries in ways other than changing the DFS-internal order or eliminating elements of the DFS.

Exchangeability. The IR weighting algorithm underlying the operator should be exchangeable. A user should have means to choose the weighting algorithm for a query.

There is an additional requirement we impose, because it appears to make the IR-style ranking within the host language even more useful:

Visibility. The operator should assign visible ranking weights to DFs, but without causing side-effects to the embedding query.

The exchangeability property of the operator allows for different IR algorithms to be plugged into the query engine. Although we have not introduced any limitations here, we identify a certain class of algorithms called *dynamic ranking* algorithms as required to establish a real local context for sub-queries. These algorithms will be subject of the next section.

In the following, we introduce the syntax and semantics of an operator that meets the above requirements. The elegant simplicity of the XQuery extension should allow to easily understand the operator's functionality, even though space limitations prevent us from going into all the details.

3.1 Syntax

We propose to add a single new operator called *rank* to the XQuery language. The operator is used within a Rankby expression that extends the set of base XQuery expressions. A Rankby expression is very similar to XQuery's Sortby expression [2, Section 2.4].

Definition 3.1 (Rankby expression)

```
RankBy::=Expr "rankby" "("QuerySpecList")"
                  ("ascending" | "descending")?
                  ["basedon" "(" TargetSpecList ")"]
                  ["limit" n ["%"]]
                  ["using" MethodFunctCall]
QuerySpecList ::=Expr ("," QuerySpecList)?
TargetSpecList::=PathExpr ("," TargetSpecList)?
```

QuerySpecList is a list of strings (constant DFs) or expressions that return DFs that can be interpreted as strings in XQuery as well. TargetSpecList is a list of context node-dependent path expressions. MethodFunctCall refers to an XQuery function. □

The syntax and semantics of XQuery functions (FunctionCall in the specification) are not yet fully specified by the W3C. MethodFuntCall is more of symbolic nature. It represents an IR algorithm that may or may not be implemented as some kind of stored procedure.

3.2 Semantics

Assume the set $L(\mathcal{F})$ of document fragments (Section 2.1). Let *weight* be a special element name in E.

Definition 3.2 (Weighting algorithm)
Assume a set of IR queries $\mathcal{Q} = L(\mathcal{F})$ which includes the DFs consisting of only a string from T, in particular single terms. Let $\mathcal{W}$ be the set of operators

$$W : L(\mathcal{F}) \times \mathcal{Q} \rightarrow L(\mathcal{F}),$$
$$W(S, Q) = S'$$

such that S' is equal to S except that each DF in S' has an additional element named *weight* at the root. The weight element has a single number string as child representing the relative relevance weight of the respective DF within S. Then a $W \in \mathcal{W}$ is called a *weighting algorithm.* □

Within the Rankby expression, the MethodFunctCall part represents a certain exchangeable weighting algorithm. Related to the QuerySpecList, queries are DFs that can be interpreted as query text. The actual query interpretation and limitations to the allowed query types depend on the weighting algorithm W. In particular, typical IR term queries like ("sun", "moon") can be regarded as DFs with only a single text string node. More advanced weighting algorithms may make use of the structure of query DFs. In the following, we define the actual ranking operator in terms of a weighting algorithm and XQuery's existing *sort* operator.

Definition 3.3 (Rank operator)
The *rank operator* is an operator:

$$rank : L(\mathcal{F}) \times \mathbf{N} \times \mathcal{Q} \times \mathcal{W} \longrightarrow L(\mathcal{F})$$
$$rank(S, k, Q, W) = First_k(sort_{weight}(W(S, Q))),$$

where $\mathbf{N}$ is the set of natural numbers and $\mathcal{Q}$ defined as above. $sort_{weight}$ refers to the sorting of DFs within S based on the weight element. $First_k$ eliminates all but the first k elements from S ($First_\infty(S) = S$). □

If there is a non-empty TargetSpecList in the Rankby expression, for each DF in S, the input to *rank* are the fragments selected by the TargetSpecList's path expression, otherwise the complete DFs. The elimination of all but the first k DFs relates to the limit clause of the Rankby expression. As an extension, the limit could also be based on a certain percentage of the DFS's total weight to be kept after the elimination.

The weighting algorithm W, exchangeable within *rank*, implements the total order as called for in the requirements. It adds a weight element to a DF making the ranking visible, a further requirement. Note that in XML, we envision an attribute to be added. An attribute is much more transparent to the rest of a query, but not an explicit part of our simplified data model. Full transparency of the ranking, however, can only be achieved through introducing a reserved name for *weight* in XQuery. XQuery's closure properties are kept as DFSs remain the only type of in- and output.

We call the XQuery query language extended by the *rank* operator *XQuery/IR*. Due to space limitations we have to illustrate the expressiveness of XQuery/IR by means of the following examples in favor of a more complete discussion.

3.3 Examples

Example 2.1 (simplified) can be extended to retrieve a maximum of 100 paragraphs with relevant information about New York as follows:

```
document(“news.xml”)//article//paragraph
rankby (“New York”) limit 100
```

The result might look like:

```
<paragraph weight=0.96>
   The New Yorker fire fighters...
</paragraph>
<paragraph weight=0.81>
   Weekend weather in New York promisses to...
</paragraph>
<paragraph weight=0.79>
  ...
```

In Example 2.2, instead of sorting the result by author, a relevance-based ranking only based on the articles' abstracts can be obtained through:

```
FOR $a IN document("bib.xml")//article
WHERE $a/year < 1996
RETURN
  <early_paper>
   <fstAuth> {$a/authors/author[1]/text()} </fstAuth>
   {$a/title}
  </early_paper>
RANKBY (“Albert”, “Einstein”) BASEDON (abstract)
```

Note that abstract as argument to BASEDON is an XPath expression relative to the context node $a. The terms “Albert” and “Einstein” could be replaced by a subquery that selects text to be used as query from another document like:

```
document(“authors.xml”)//author[./name="Einstein"]
                                   /accomplishments
```

A modification to the earlier Example 2.3 extracts only the first 10 articles in each category. Here, the ranking occurs in the inner loop based on the actual relevance for the category an article is classified under.

```
<news_by_category>
{FOR $c IN document(“newsmeta.xml”)//category
  RETURN
    <category>
      <name> {$c/name/text()} </name>
      {FOR $a IN document(“news.xml”)
                        //article[@cid = $c/@id]
        RETURN
          <title aid={$a/@id}> $a/title/text() </title>
        RANKBY ($c/keywords) LIMIT 10
    </category> }
</news_by_category>
```

4 Implementation Aspects

Weighting algorithms underlying the new operator have some novel properties. Furthermore, *rank* has some implications for a system implementation, in particular index structures and query optimization. In this section, we briefly address both issues.

4.1 Dynamic Ranking

In standard IR, queries are always evaluated in the context of the complete document collection, a notion that we have replaced by the concept of DFS. DFSs are not static anymore but dynamically established by a (sub-)query. Nothing prevents us from still using static, pre-computed term weights. However, most of the power of the integrated query approach with its local contexts is lost in that case.

For instance, it is unsatisfactory to always use the same, fixed inverse document frequency for terms when using an adapted tf-idf scheme, because the sub-query result to which the weighting is applied might not even once contain the respective term. This will lead to unexpected and even useless ranking results. Consequently, more general term counting statistics have to be kept. For each ranking operation, term statistics for the current intermediate result have to be derived. Then, a scheme like, e.g., tf-idf could be used as before, or an extension thereof that utilizes the additional information encoded in the structure of DFs.

We call this general principle underlying IR embedded into a data retrieval query language *dynamic ranking principle*. An approach following this principle allows us to detect the local significance of an otherwise rather common term, considered as irrelevant in a more global context.

4.2 Indexes and Query Optimization

For an IR index to be useful for hierarchically structured XML data, it needs to capture term occurrences within single document fragments. This is obviously more expensive than a static index on a collection of documents as single units. However, by definition only leaf nodes contain content. Thus, storing term statistics for these and relying on an index for the document structure as required for structure-based queries anyway can be sufficient to derive statistics for arbitrary DFs.

Furthermore, the query engine needs to be extended to keep track of where parts of DFs within an intermediate result originate from. This is made even more difficult as DFs from different contexts may be put together through XQuery's construction operations. In some cases, e.g., in case of aggregated data, no direct relationships between DFs in a DFS and their origin can be maintained. In this case, either ranking is not possible or term statistics have to be collected from scratch, which might be acceptable for small results.

The expenses associated with a full text index on top of XML data can be compensated by an IR index-aware query engine, an advantage that is ignored when deploying IR outside of a database. For semistructured data without a lot of schema information, optimization via the IR index is especially promising [10]. Not only point term indices directly to the data, but detailed data statistics, e.g., about the variance of certain element values are easily integrated into an IR index.

5 Conclusions and Future Work

We have introduced a new operator into XQuery that naturally extends XML queries by information retrieval capabilities. XML document fragment sequences are intermediate and end results in XQuery. We have identified arbitrary document fragments within such dynamically selected sequences as suitable replacement for the notions of document (collection) in today's information retrieval. The extended language, which we have dubbed XQuery/IR, is conceptually simple yet more expressive than the sum of its parts. We have identified dynamic ranking, the usage of term statistics in the local context established by a query, as the most important property of an underlying IR algorithm to achieve this expressiveness.

Currently, we are implementing a system able to demonstrate important properties of the presented approach. A hindrance is the non-existence of freely available query engines and large, deeply nested XML databases with diverse types of textual information.

Other aspects we are investigating include reformulation and thus, optimization rules for queries involving the rank operator. Questions are, for instance, to what extend ranking and data extraction are commutative.

References

[1] O. Alonso. Oracle Text White Paper. Technical report, Oracle Corp., Redwood Shores, U.S.A, May 2001.

[2] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, M. Stefanescu. XQuery 1.0: An XML Query Language. W3C Working Draft, W3C, June 2001.

[3] T.T. Chinenyanga and N. Kushmerick. An Expressive and Efficient Language for XML Inf. Retrieval. In *Proc. of the 2001 SIGIR Conf.*, pp. 163-171, 2001.

[4] J. Clark and S. DeRose. XML Path Language (XPath). W3C Recommendation, W3C, Nov. 1999.

[5] M. Fernandez and J. Marsh. XQuery 1.0 and XPath 2.0 Data Model. W3C Work. Draft, W3C, 2001.

[6] D. Florescu, D. Kossmann, and I. Manolescu. Integrating Keyword Search into XML Query Processing. In *Proceedings of the 9th Int'l Word Wide Web Conference/Computer Networks*, 33(1-6):119-135, 2000.

[7] P. Frankhauser, M. Fernandez, A. Malhotra, M. Rys, J. Siméon, and P. Wadler. XQuery 1.0 Formal Semantics. W3C Working Draft, W3C, June 2001.

[8] N. Fuhr and K. Grossjohann. XIRQL: A Query Language for Information Retrieval in XML Documents. In *Proc. of the SIGIR 2001 Conf.*, pp. 172-180, 2001.

[9] M. Kaszkiel, J. Zobel, and R. Sacks-Davis. Efficient Passage Ranking for Document Databases. *ACM Transactions on Inf. Systems*, 17(4):406–439, 1999.

[10] J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajaraman. Indexing Semistructured Data. Technical report, Stanford University, February 1998.

[11] G. Navarro and R. Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. *ACM Transactions on Information Systems*, 15(4):400-435, 1997.

[12] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[13] A. Theobald and G. Weikum. Adding Relevance to XML. In *Proc. of the 3rd Int'l Workshop on the Web and Datab.*, LNCS 1997, pp. 105-124, Springer, 2001.