INEX Efficiency Track meets XQuery Full Text in BaseX

Sebastian Gath, Christian Grün, Alexander Holupirek, and Marc H. Scholl

`<firstname>.<lastname>@uni-konstanz.de`
Department of Computer & Information Science
Box D 188, 78457 Konstanz, Germany
University of Konstanz

Abstract. BaseX is an early adopter of the upcoming XQuery Full Text Recommendation. This extended abstract describes the challenges of joining the INEX Efficiency Track using BaseX, an XML database system. We will describe some of the problems we encountered during the application of XQuery Full Text to the INEX topic set and discuss the remaining comparability problem.

1 Introduction: BaseX and XQuery Full Text

The existence of more than fifty XQuery processors clearly underlines the large interest in querying XML documents and collections. While many of the database-driven implementations offer their own extensions to support full-text requests, the upcoming XPath and XQuery Full Text 1.0 Recommendation [1] (XQFT) aims to satisfy the need for a unified language extension and might attract more developers and users from the Information Retrieval community. The recommendation offers a wide range of content-based query operations, classical retrieval tools such as Stemming and Thesaurus support, and an implementation-defined scoring model that allows developers to adapt their database to a large variety of use cases and scenarios. BaseX is, to the best of our knowledge, the first implementation to fully support all features of the specification. More implementations are expected to follow in the near future as soon as the recommendation has reached its final state.

In this extended abstract, we present some aspects encountered during our participation in the INEX Efficiency Track 2009 using our XQFT-based database system BaseX [3, 4, 6]. Due to the complexity of the language extension, we will focus on its NEXI related features. The first Ad-Hoc query of the INEX track is depicted as an example:

```
//article[.//text() ftcontains ("Nobel" ftand "prize")]
```

First of all, there is a path condition `//article` in front of the predicate, which demands that each node on the descendant-or-self-axis of the document root with an element tag `article` has to be taken into further considerations. Additionally, the predicate filters all descendant texts node of each `article` element for the tokens `"Nobel"` and `"prize"`. The existence of at least one text node containing both tokens yields a valid result of the query. Changing the full-text operand from conjunction to disjunction would lead to additional results containing at least one of the tokens. In general, any XQFT query has at least one input item and at least one full-text condition, and valid results always have to satisfy both conditions. Keeping this in mind, the following observations try to give an insight into the challenges we encountered during our participation in INEX 09 Efficiency Track using an XQFT based engine.

2 Proceedings

First of all, we have introduced a query rewriting step to take advantage of the internal query optimization in BaseX. In detail, we have modified the atomization of context nodes in the given queries:

```
(1) //article[. ftcontains ("Nobel" ftand "prize")]
(1') //article[.//text() contains ("Nobel" ftand "prize")]
```

The atomization of the the context node in the first query `(1)` purges element information from the subtree of each `article` node and concatenates the text nodes of its descendants. That means, in a valid result, the tokens `"Nobel"` and `"prize"` do not necessarily have to occur in the same text node. Our rewritten version `(1')` applies the search on the specified terms on each text node, which is contained in the subtree of each `article` node, and the terms are now expected to occur in the same text node. This rewriting allows us to use the full-text index in BaseX, which is optimized to run on single text nodes. This way, all kinds of XQuery location paths can be traversed, independent from the structure of the input document. The results of the rewritten queries represent a subset of the atomized query results, and the usage of `FTOr` instead of `FTAnd`, as proposed for the efficiency track, might lead to results similar to the original query, because the tokens do not have to ocurre in the same text node.

A second type of query rewritings keeps the original semantics:

```
(1) //article[.//(sec|p) ftcontains ("mean" ftand "average" ftand
"precision" ftand "reciprocal")]
(2) //article[.//sec ftcontains ("mean" ftand "average" ftand
"precision" ftand "reciprocal") or .//p ftcontains ("mean" ftand
"average" ftand "precision" ftand "reciprocal")]
```

Here, the replacement of `|` in `(1)` by a union operator `(2)` is helpful to trigger index-based query processing in BaseX, and results in less compact but equivalent queries. This optimization step is currently not applied automatically by the query compiler due to some side-effects, which are not relevant for the INEX context.

After this pre-processing step, the BaseX query optimizer performs further internal rewritings and applies the full-text index structure whenever possible and cheap enough. Disjunct or conjunct full-text tokens within an XQFT expression are evaluated by the index as well. A cost-based evaluation strategy is applied on queries with several possible index requests (which is the case, e.g., if queries have two or more full-text predicates). Details can be found in [5].

Next, we have extended our full-text index structure by including a normalized, TF/IDF-based scoring value for each indexed token. The score is pre-computed at database creation time and is accessible at query time without additional computations. Individual scores values for several terms are generally calculated by full-text operands. The XQFT operators `FTAnd`, `FTOr` and `FTNot` are taken into further considerations because of their relevance in the INEX context. We have introduced a minimum-based scoring for `FTAnd`, i.e., the minimum of several score values is adopted in an `FTAnd` expression. For `FTOr`, we are using a maximum-based scoring approach, which is similar to the `FTAnd` based score, and for `FTNot`, the inverted score value is returned $(1 - score)$ [7]. To get XQFT and NEXI queries semantics closer, NEXI-like scoring for `FTAnd` and `FTOr` operands would be interesting. Axis steps performed in a query can influence score values as well (score propagation). Due to the high heterogeneity of the INEX documents and a lack of well-known solutions, we have chosen a simple, but efficient approach, multiplying a score value with a constant for each location step.

All BaseX query results conform to the typical structure of XQuery results. To comply with the INEX submission format, the path for each topic result had to be extracted out of the query results, which was realized by a simple XQuery function:

```
declare namespace basex = "http://www.basex.com";
declare function basex:sum-path ( $n as node()? )
as xs:string {
 string-join( for $a in $n/ancestor-or-self::*
 let $ssn := $a/../*[name() = name($a)]
 return concat(name($a), '[',
  basex:index-of($ssn, $a), ']'), '/')};
declare function basex:index-of (
 $n as node()* , $ntf as node() ) as xs:integer* {
 for $s in (1 to count($n))
 return $s[$n[$s] is $ntf]};
```

The function `basex:sum-path($n as node() ?) as xs:string` returns the node path from the root node to `$n` by traversing the document tree and caching each element and its index.

3 Similarities and Differences between XQuery Full Text and NEXI

INEX, as an extension of XPath, transfers the semantics defined in the XPath queries to the evaluation system [8]. Generally the semantic of XPath queries is well defined by the query grammar and independent of the evaluation system. In contrast the semantics of INEX queries is more independent and depends mostly on the engine. Since the INEX Efficiency Track offers for each topic a NEXI and an XQFT query, the contest attracts new XML database systems with XQFT implementations, and leads to additional competition with established IR systems. However, to compare performance or measure the quality of results, semantically equivalent queries for database and information retrieval systems are crucial.

Basically, there is a main big difference between XML database systems using XQuery and IR systems using with NEXI. The XQuery 1.0 Recommendation [2] guarantees that the same query returns the same result on any system that is processing XQueries. Even if scoring is applied in the query, which is totally implementation defined in XQFT, combined with top-k conditions, the set and structure of the results will always be the same. But the visible results and their ordering may change because of a scoring based ordering and the top-k condition. In contrast, NEXI engines could have a major influence on the semantics of a query and do not provide a guarantee like that. This observation leads to an incomparability of simple XQFT- and NEXI-based queries.

In detail, there are two major differences between the query languages:

```
//article[about(.//p, "information retrieval")]
```

Considering the query above, the strict interpretation demands that article tags are returned as a result, and the path condition within the predicate (`.//p`) is handeld as a suggestion for the information retrieval search. Even the `//article` path condition can be dealt with as (only) a suggestion [8].

The example above illustrates the hint-like behavior in structural conditions in NEXI-based queries even in the strict interpretation compared to XQuery, where there is always a strict path condition in a query which has to be evaluated. To get some additional freedom, the combination of different path conditions with a logical OR could be useful, but in this case, the user has to know the structure of the document. Alternatively, the usage of a unspecified path condition is possible, but this leads to a completely different query and many irrelevant results.

Another point is the interpretation of `AND` and `OR` in the NEXI context:

```
(1) //article[about(., apple) and about(., computer)]
(2) //article[about(., apple) or about(., computer)]
```

The first query `(1)` will return article elements from documents about "apple" and about "computer", the second `(2)` about "apple" or about "computer". The predicate conditions, however, are only hints. Looking at the loose interpretation, the `AND` is interpreted as `ANDish`, `OR` as `ORish`. In that case the contained Boolean operators are rather hints on how to resolve the information need [8].

The strict interpretation is close to the `FTAnd` and `FTOr` operands, but is still semantically inequivalent, due to the hint behavior in NEXI. Next, looking at the loose interpretation, there is no similarity to the full-text operands. To get this hint behavior and `ANDish` or `ORish` interpretation, a scoring model is needed which, for example, scores results having more `OR` disjunct search tokens higher than results having less disjunct results. Scoring-oriented query processing is very much dependent on the input data. A general-purpose database systems, such as BaseX, aims to perform well in all kinds of applications. Therefore we decided not to add INEX-specific meta information to the database and indexing engine, such as a scoring priority for certain elements, etc. The scoring model does not have any additional information about the input data, therefore a proper solution could be the usage of individual scoring models depending on the use case. For example, in some use cases it might be reasonable to include structural knowledge of the document in the scoring model, in other use cases this does not create an additional value.

In a nutshell, we believe that the semantic equivalence between the NEXI and XQFT queries in the Efficiency Track is not given. As a consequence, NEXI- and XQFT-based submissions might not be directly comparable at the current stage. The semantic distance between the queries increase with their complexity, and, to get almost semantically similar queries, the fuzziness of the NEXI operands would have to be mapped to XQFT. This might lead to verbose XQFT queries with numerous sub-queries and, most likely, longer evaluations times. Alternatively, additional knowledge on the document structure would have to be utilized to get equivalent results.

References

1. Sihem Amer-Yahia et al. XQuery and XPath Full Text 1.0. W3C Candidate Recommendation. http://www.w3.org/TR/xpath-full-text-10, May 2008.
2. Scott Boag et al. XQuery 1.0: An XML Query Language. W3C Recommendation. http://www.w3.org/TR/xquery, January 2007.
3. Christian Grün et al. Pushing XPath Accelerator to its Limits. In *Proc. of the 1st ExpDB Workshop*, Chicago, Illinois, USA, 2006.
4. Christian Grün et al. Visually Exploring and Querying XML with BaseX. In *Proc. of the 12th BTW Conference, Demo Tracks*, pages 629–632, Aachen, Germany, 2007.
5. Christian Grün et al. XQuery Full Text Implementation in BaseX. In *Proc. of the 6th International XML Database Symposium (XSym 2009)*, pages 114–131, Lyon, France, 2009.
6. Alexander Holupirek et al. BaseX & DeepFS: Joint Storage for Filesystem and Database. In *Proc. of the 12th EDBT Conference*, pages 1108–1111, 2009.

7. Gerard Salton. Extended Boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.
8. Andrew Trotman. Narrowed Extended XPath I (NEXI). In *Lecture Notes in Computer Science*, pages 16–40, Berlin / Heidelberg, 2005.