XQuery Full Text Implementation in BaseX

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

Database and Information Systems Group
University of Konstanz, Germany
Motivation

XQuery/XPath Full Text 1.0

- upcoming W3C Recommendation for content-based XML queries
- brings DB and IR world together
- first implementations available (Qizx, MXQuery, xDB, BaseX)

Challenges

- large text corpora/XML instances
- complete embedding in XQuery language
- classical retrieval features: stemming, thesaurus, stop words

→ all features need to be supported, yet performance is essential
Queries

Document-based location path with predicate

```
//book[title ftcontains 'Crime and Punishment']
```

Optional filters and options

```
//book[section ftcontains ('heritage' ftand 'claim'
   window 10 words) language 'en' with stemming]/title
```

Queries without document reference

```
'a b c' ftcontains 'b'
```

Dynamic item values

```
func:merge($a) ftcontains { func:stem($a), $b, $c }
```
BaseX

Native XML Database and XQuery Processor

- first complete XQuery Full Text implementation
- high XQuery conformance (99.9%)
- various index structures: names, paths, values, full text
- tight backend/frontend coupling, real-time querying
- open source (BSD) since 03/07
Storage

Document Storage

- inspired by XQuery Accelerator\(^1\) and MonetDB/XQuery
- flat, compressed table storage, using `pre/dist/size` encoding:

\[\text{pre dist size data} \quad \begin{array}{ccc}
1 & 1 & 11 \quad A \\
2 & 1 & 1 \quad B \\
3 & 1 & 0 \quad \text{yw xy} \\
4 & 3 & 6 \quad C \\
5 & 1 & 1 \quad D \\
6 & 1 & 0 \quad \text{x} \\
7 & 3 & 1 \quad D \\
8 & 1 & 0 \quad \text{xy xw} \\
9 & 5 & 1 \quad D \\
10 & 1 & 0 \quad \text{x y} \\
11 & 10 & 1 \quad E \\
12 & 1 & 0 \quad \text{y x}
\end{array}\]

\(^1\) Torsten Grust, Accelerating XPath Location Steps. SIGMOD 2002
Storage

Indexes

• names (tags, attribute names)
• paths (unique location paths)
• values (texts, attribute values)

Full Text Index

• Compressed Trie
• node: characters and pre, pos value pairs
• value pairs are sorted
→ essential for pipelined evaluation
Evaluation

Sequential Scan
- performs the predicate test for each location path
- touches all addressed nodes at least once

Index-based processing
- performs the predicate test first
- traverses the inverted path for all index items

Hybrid Approach
- combination of sequential and index-based processing

\[ /A/C[D/text() ftcontains "x"] \]
Evaluation: Index-Based

Indexing on document level

- popular approach in relational databases
  - no performance boost for large documents

Indexing of location paths

- simple queries with fixed path can be easily sped up
  - does not work for nested/more complex queries

XQuery Index Functions

- allows for explicit index calls
  - no benefit for internal query optimization
Evaluation: Index-Based

Dynamic Approach

• all text nodes are indexed
• predicates with \texttt{ftcontains} are analyzed for index access
• costs are estimated for each index access
• cheapest predicates are rewritten to index operators
• remaining location paths are inverted (utilizing the XPath Symmetries\textsuperscript{2})

Advantages

+ many queries with nested/complex location paths can be optimized
+ query writing and query optimization are uncoupled

\textsuperscript{2} Dan Olteanu et al., XPath: Looking Forward. XMLDM Workshop 2002
Evaluation: Index-Based

\[ /A/C[D/\text{text()} \text{ftcontains} \ "x"\] \]
Evaluation: Index-Based

/\text{descendant::D}[\text{text()} \text{ftcontains} \ "x"]

\text{FTIndex("x")}/\text{parent::D}
Evaluation: Index-Based

let $auction := doc("XMark.xml")
return
  for $p in $auction/site/people/person
  where $p/address/country = "United States"
  and $p/name ftcontains "Nikil" ftand "Stolovitch"
return $p/emailaddress
Evaluation: Hybrid

Hybrid Approach

- the \texttt{ftnot} operator cannot be processed by only using the index
- yet, index can be applied to avoid tokenization of all text nodes
- optimized plan combines seq. scan and index access
- sortedness of nodes and index results leads to linear costs
Evaluation: Pipelining

Iterative/pipelined Evaluation

- items are processed one-by-one
- constant memory consumption
- most efficient if large results are reduced to small, final result sets

Index Access

- all XQFT operators can be processed in an iterative manner
- a pipelined index operator returns single items
  - this way, the same full-text operators can be applied on both sequential and index-based processing
- again, the sortedness of index results avoids pipeline blocking
Evaluation: Pipelining

Index-based evaluation of `ftand`

- FTIntersection operator merges index results:
  
  ```xml
  //*[text() ftcontains "x" ftand "y"]
  ```

  ```
  QueryPlan
  LocationPath
  FTIntersection
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIntersection
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
  ```

  ```
  FTIndex
Evaluation: Pipelining

Index-based evaluation of wildcards

- Wildcards results are merged by FTIndex (ftor works similar):

```
1
  A
    2
    B
      3
      xw
      xy
    4
      5
      D
        6
        x
        y
        xw
    7
      9
      D
    11
      12
      E
      y
      x

Trie

1
  x
    6,0
    10,0
    12,1
  y
    10,1
    12,0

w
  3,0
  8,1

y
  3,1
  8,0
```

- [3,0] and [3,1] are merged and returned
- next results are: [6,0], [8,0 | 8,1], [10,0], [12,1]
Q1: \( \text{doc("xmark")//keyword[text()} \text{ftcontains "barrel"}] } \)

Q2: for $mail in doc("xmark")/site/regions/*/item/mailbox/mail
where $mail//text/text() \text{ftcontains "seeking.*" with wildcards}
return $mail/from

Q3: for $item in doc("xmark")/site/regions/*/item
where $item//listitem/text/text() \text{ftcontains ftnot "preventions"}
return <result>{ $item/location/text() }</result>
Frontend XQUERY FULL TEXT
Frontend

XQUERY FULL TEXT

XQuery Full Text Implementation in BaseX
Conclusion

XQuery Full Text is getting popular!

- many of our users are already working with XQFT
- more and more implementations arise

Open Challenges

- suitable scoring algorithms for XML data (see INEX, SIGIR, …)
- runtime optimizations to allow for index access of variable ftcontains strings

...thanks for listening!