TeXQuery: A Full-Text Search Extension to XQuery

Sihem Amer-Yahia
AT&T Labs–Research

In collaboration with:
Chavdar Botev (Cornell University)
Jayavel Shanmugasundaram (Cornell University)
Jonathan Robie (DataDirect Technologies)
Outline

1. Full-Text search: Definition and Use Cases.
2. Existing proposals.
3. TeXQuery: Data model and primitives.
4. Design principles: Composability and scoring.
**Full-Text Search**

- **Context expression**: defines nodes where search occurs.
- **Return expression**: defines document fragments that are returned to users.
- **Search expression**: defines FT search conditions.
- **Score expression**: defines an expression that might be used to score returned fragments.
Use Cases

- Use Case ”WORD”: Words and Phrases
- Use Case ”STOP-WORD”: Ignoring and Overriding Stop Words
- Use Case ”WILDCARD”: Word Wildcard
- Use Case ”STEMMING”: Word Stemming
- Use Case ”THESAURUS”: Thesauri, Dictionaries, and Taxonomies
- Use Case ”BOOLEAN”: Or, And, and Not
- Use Case ”DISTANCE”: Distance (Proximity, Window)
- Use Case ”SCORE”: Unique to Score
- Use Case ”IGNORE”: Ignoring Markup
- Use Case ”COMPOSABILITY”: Composability of Full-Text with XQuery
The usability of a Web site is...
Use Cases

- Find all books containing the word ”usability”.
- Find all book titles containing the word ”usability”.
- Find all books (title and authors) with the phrase ”web site” and the word ”usability” in the same subject.
- Find all books with the word ”way” with one prefix character.
- Find all books with information on software engineers.
- Find the best two books on usability.
- Find book chapters containing the phrase ”heuristic is Ten Usability” ignoring citation tags.
- **FTSelection** corresponds to Boolean, proximity, window, number of occurrences, order and scope.

- **FTContextModifier** defines the FTS environment, which can modify the operational semantics of FTSelection such as stemming, stop-words, diacritics and case.
Find the title and price of books on usability and sort books from the cheapest to the most expensive:

```xml
for $item in //books/book
let $pval := $item/metadata/price
where ft:contains($item//content,"usability")
order by $pval ascending
return <result>
  $item/title
  <price>
    $pval
  </price>
</result>
```
Existing Proposals

- Cerisent & Oracle

  \[\text{text-contains}(\text{node}\ast \ $\text{searchText}, \ \text{string}? \ $\text{language})\]
  \[\text{text-score}(\text{node}\ast \ $\text{searchText}, \ \text{string}? \ $\text{language})\]

- IBM & Microsoft

  \[\text{text-contains}(\text{source as node}\ast,\]
  \[\quad \text{match-options as string}?,\]
  \[\quad \text{terms as string}\ast) \ \text{as} \ \text{xs:boolean}\]
  \[\text{proximity-contains}(\text{source as node}\ast,\]
  \[\quad \text{proximity-options as xs:string},\]
  \[\quad \text{match-options as xs:string}\ast,\]
  \[\quad \text{terms1 as xs:string}\ast,\]
  \[\]
$terms2 as xs:string*)

ft:score(OrExprFT) as float*

OrExprFT ::= AndExprFT ("or" AndExprFT)*
Example: Stop-Words

4.2.4 Q4 Query on Phrase Ignoring Multiple Stop Words

*Find all books which have not been approved by a Web users organization.*

**Cerisent & Oracle**

```
text-contains($book//content,
   ' "not been approved" STOPWORDS IGNORE')
```

**IBM & Microsoft**

```
text-contains($book//content,
   "match-stopwords='false' lang='en'",
   "not been approved")
```
TeXQuery: Primitives

- **FTContainsExpr**:\( ::= \text{ContextExpr} \ "ftcontains" \ \\
  \text{FTSelection} \)
  returns true if at least one node in ContextExpr satisfies FTSelection.

- **FTScoreExpr**:\( ::= \text{ContextExpr} \ "ftscore" \ \\
  \text{FTSelectionWithScoreWeights} \)
  returns a sequence of scores. Provides access to fine-grained ranking (e.g., threshold and top-k.)

- **FTSearchExpr**:\( ::= \text{ContextExpr}? \ "ftsearch" \ \\
  \text{FTSelectionWithScoreWeights} \)
  returns most specific nodes.
TeXQuery: Primitives

FTSelection ::= FTStringSelection | FTAndConnective | FTOrConnective | FTNegation | FTMildNegation | FTOrderSelection | FTScopeSelection | FTDistanceSelection | FTWindowSelection | FTTimesSelection | FTSelection FTContextModifier
\textbf{TeXQuery: Primitives}

\texttt{FTSelectionWSW ::= FTStringSelection | FTAndConnective | FTOrConnective | FTNegation | FTMildNegation | FTOderSelection | FTScopeSelection | FTDistanceSelection | FTWindowSelection | FTTimesSelection | FTSelectionWSW FTContextModifier | FTSelectionWSW "weight" xs:float}
\textbf{TeXQuery: Primitives}

\begin{verbatim}
FTContextModifier ::= FTCaseCtxMod | FTDiacriticsCtxMod | 
FTSpecialCharCtxMod | FTStemCtxMod | FTThesaurusCtxMod | 
FTStopWordCtxSpec | FTLanguageCtxMod | 
FTIgnoreCtxMod
\end{verbatim}
**FTSelections: Grammar**

FTStringSelection ::= Expr ("any" | "all" | "phrase")?

FTDistanceSelection ::= FTSelection "with"? ("word" | "sentence" | "paragraph") "distance" FTRangeSpec

FTWindowSelection ::= FTSelection ("within")?
"window" FTRangeSpec

FTOrderSelection ::= FTSelection "ordered"

FTTimesSelection ::= FTSelection FTRangeSpec
"occurrences"

FTRangeSpec ::= ("exactly"? Expr) | ("at least" Expr)
("at most" Expr)
FTContextModifiers: Grammar

FTThesaurusCtxMod ::= "with"? "thesaurus" Expr
    | "without" "thesaurus"

FTStopWordsCtxMod ::= "with" "additional"? "stopwords" Expr ?
    | "without" "stopwords" Expr?

FTLanguageCtxMod ::= "language" Expr

FTIgnoreCtxMod ::= "without" ("tags" | "content") Expr
FTSelections and FTCtxModifiers Examples

("mustang" && "great") ordered

("mustang" && "great") window at most 30

("mustang" || "great") same paragraph

("mustang" && "great" || "rust") at least 2 occurrences

(("mustang" && "great" word distance 2)
&& "rust") word distance at most 5
for $item$ in $ContextExpr$ ftcontains FTSelection
return <hit>$item</hit>

for $item$ in $ContextExpr$
let $score := $item ftscore FTSelectionWSW
return <hit>$item$score$score</hit>

for $item$ in $ContextExpr$
let $score := $item ftscore FTSelectionWSW
order by $score$ descending
return <hit>$item$score$score</hit>
TeXQuery: Generic Examples

for $item$ in ContextExpr
let $score :=$ item ftscore FTSelectionWSW
where $score > 0$
order by $score$ descending
return <hit>$item$<score>$score</score></hit>

for $item$ in ContextExpr ftcontains FTSelection
let $score :=$ item ftscore FTSelectionWSW
order by $score$ descending
return <hit>$item$<score>$score</score></hit>
for $item$ in  
\texttt{ContextExpr ftcontains FTSelection}

let $score :=$item \texttt{ftscore FTSelectionWSW}

where $score >$threshold

order by $score$ descending

return \texttt{<hit>$item<$score>$score</score></hit>}

for $hit$ at $i$ in

for $item$ in  
\texttt{ContextExpr ftcontains FTSelection}

let $score :=$item \texttt{ftscore FTSelectionWSW}

order by $score$ descending

return \texttt{<hit>$item<$score>$score</score></hit>}

where $i < 20$

return $hit$
4.2.4 Q4 Query on Phrase Ignoring Multiple Stop Words

*Find all books which have not been approved by a Web users organization.*

**Cerisent & Oracle**

```
text-contains($book//content,
  ' "not been approved" STOPWORDS IGNORE''
)
```

**IBM & Microsoft**

```
text-contains($book//content,
  "match-stopwords='false' lang='en'",
  "not been approved")
```

**TeXQuery**

```
$book//content ftcontains "not been approved"
  without stopwords
```
9.2.1 Q1 Or Query

*Find all books with the words "web" or "software" in the text.*

**Cerisent & Oracle**

```
text-contains($content, ' "web" OR "software" ')```

**IBM & Microsoft**

```
text-contains($content, "web")
or text-contains($content, "software")```

**TeXQuery**

```
$content ftcontains "web" & "software"
```
3.2.9 Q9 Query on Attribute

*Find all books with "improve" "web" "usability" in the short title.*

**Cerisent & Oracle**

```
text-contains($title/@shortTitle,
    'NEAR ((STEMMED FORM OF "improve",
    STEMMED FORM OF "usability"),6)')
```

**IBM & Microsoft**

```
proximity-contains($title/@shortTitle,"stemming='true'",
    "max-distance='6' unit='tokens'",
    ("improve", usability"))
```

**TeXQuery**

```
$title/@shortTitle ftcontains "improve" && "usability"
    with stems word distance 6
```
Pros and Cons: IBM & Microsoft

- + functions and arguments
- + simple extension to XQuery (two functions)
- + XQuery-like and composable with XQuery
- - requires a second-order function for score
- - not easily readable
- - not clear how to compose expansion operators e.g. stem of (synonym of (dog) )
- - asymmetrical syntax for all terms and any term
Pros and Cons: Cerisent&Oracle

- + pure sub-language approach
- + simple extension to XQuery (two functions)
- + easy to read and write
- + based on proven approach (SQL/MM)
- + many existing implementations use this approach
- - string sub-language requires special parsing
- - variable binding requires concat, string-building
- - not extensible in a standard way
**Differences**

**FTSelections:** proximity-contains() (IBM&Microsoft), keywords (Cerisent&Oracle, TeXQuery).

**FTCtxModifiers:** positional arguments (IBM&Microsoft), explicit composition order in sublanguage (Cerisent&Oracle, TeXQuery).

**Scores and Ranking:** second-order function (IBM&Microsoft), text-score function (Cerisent&Oracle), ftscore extension (TeXQuery).

**XQuery Leverage:** XQuery boolean operators (IBM&Microsoft), XQuery order by clause, XQuery to evaluate context nodes.
Rationale for TeXQuery

- XQuery is defined on XML structures.
- FTS operates on linguistic units (e.g., words, sentences) not represented in the XQuery 1.0 and XPath 2.0 Data Model.
- Basic FTS operators and the way they are composed is different from XQuery operators (e.g., ’and’ and ’or’ in XQuery are Boolean, but their full-text equivalents are operators on linguistic units).
- FTS uses token positions (e.g., textual proximity).
- TeXQuery introduces new operators for the full-text sublanguage and integrates them to XQuery.
TeXQuery Data Model
Composability

- Calling \textit{ftcontains} from XQuery.
- Calling XQuery from \textit{ftcontains}.
- Composing \textit{FTContextModifiers}.
- Composing \textit{FTSelections}.

Goal: \textit{Understand how the choice of a FT syntax impacts composability.}
Composability Figure

TeXQuery Expression
Convert a FullMatch to
a sequence of items

Evaluate to a
sequence of items

XQuery Expression

FTSelection Expression
Convert a sequence of
items to a FullMatch

Evaluate to a FullMatch
Composability in TeXQuery

- **Conversion from FullMatch to XQuery data model:** To enable TeXQuery expressions to be nested and composed with regular XQuery expressions, we propose three new XQuery expressions for FTS which convert a FullMatch to a sequence of items.

- **Conversion from XQuery data model to a FullMatch:** We use the result of an XQuery expression as a search token in an FTSelection by converting the XQuery expression to the FullMatch associated with that search token.

- "`XML' && article/title, the string result of article/title is converted to a StringMatch."
Calling `ftcontains` from XQuery

All proposals achieve this because the assumption is to apply a function to a set of nodes produced by any XQuery expression.

```xquery
for $item in ContextExpr ftcontains FTSelection
return <hit>$item</hit>
```

**Issue:** how to represent and what to do with scores that are returned by the FT expression?
**Calling XQuery from ftcontains**

All proposals call XQuery from FT expression by binding a variable to the result of an XQuery expression and use that variable within the FT expression. Both proposals limit the XQuery expression that is called inside FT search to return ”a sequence of strings”. Not any XQuery can be called from FT search.

We extend this to:

- *Integers* in range specification in **FTDistanceSelection**
- *Element nodes* in **FTIgnoreCtxSpec**
- *Strings*: search tokens, stopword list in **FTStopWordsCtxSpec**, thesaurus name in **FTThesaurusCtxSpec**
**Composing FTContextModifiers**

- *FTContextModifiers* have to be specified as substrings because they are closely related to the search tokens. Easier to specify than function arguments.

- IBM/MS does not specify the order in which modifiers are applied.

- TeXQuery uses a sublanguage and can specify/compose modifiers in any order.
Composing FTSelections

( ("mustang" &&
   (("great" || "excellent") at least 2 occurrences)
) window at most 30
&& ! "rust"
)
same node

ft:window-atmost ( ft:contains ($item,"mustang")
AND ( ft:occurrence-atleast ( ft:contains($item,"great")
OR
ft:contains($item,"excellent"),2)
),30)
AND NOT ft:contains ($item,"rust")
Composing FTSelctions

- Each FTS primitive can be translated to an XQuery functions but XQuery data model would need to be extended to achieve full composability.

- FTS primitives can be called from a function (Cerisent&Oracle) but FTS has to remain within the sublanguage for composability.

- TeXQuery primitives can be called from a function but FT has to remain within the sublanguage for composability.
Scores and Ranking: Issues

- Use a ranking expression that is different from the search expression. The ranking expression can be as complex as the search expression. This is the case where we search for documents on AIDS and rank them by their relevance to "health benefits".

- Provide ability to express user weights in scoring expression.

- Rank on both scalar and FT predicates.
Scores and Ranking: Possible Solutions

How to express scores in the language’s syntax?

1. Extend the FLWOR expression with a score clause

   \[
   \text{score } s1 \text{ as } \text{ft:contains($book, "usability"),}
   \]
   \[
   s2 \text{ as } \text{ft:contains($book, "matrix"),}
   \]
   \[
   \text{top 50}
   \]

2. Add a score expression to XQuery.

   \[
   \text{order by } \text{ft:score(ft:text-contains($book,}
   \]
   \[
   "usability")})
   \]
   \[
   \text{top 50}
   \]

3. Return more than one item from a function call and parallel bind across multiple returned values.
Scores and Ranking: Possible Solutions

How to carry score information from FTS expression to XQuery?

1. Extend the data model.

2. Allow FT search to append new attributes to the elements that it manipulates. Those attributes would carry the ”extra information”.

3. The current Cerisent&Oracle proposal on multiple variable bindings.

4. IBM/MS: Idea of being able to score scalar and FT predicates together with second-order function creates situations where interpretation of Boolean connectors depends on operands.
Scores and Ranking in TeXQuery

- *ftscore* returns a sequence of floats.
- Scoring function is implementation-defined but must satisfy:
  1. The scoring function should produce score values in the range 0-1 (inclusive)
  2. If context node does not satisfy FTSelection used in *ftscore*, score is 0
  3. If context node satisfies FTSelection used in *ftscore*, score should be $> 0$ (use this condition in where clause to filter out answers).
  4. For context nodes that satisfy the FTSelection, a higher score implies a higher relevance to the FTSelection.
**Extras at zero cost**

*ftsearch* is easy to incorporate and provides great flexibility.

```plaintext
for $item in ContextExpr ftsearch FTSelection
let $score := $item ftscore FTSelectionWSW
order by $score
return <hit>$item$score</hit>
```

Answers are be returned in the order *ftsearch* decides to return them which is implementation-dependent (depends on ranking algorithms).
Lessons

1. A sublanguage approach is necessary to achieve full composability.

2. Simple and powerful scoring mechanism.

3. Clean Semantics and well-defined default behavior for easy extensibility.

4. No extension to the XQuery data model should be necessary.
TeXQuery Highlights

- Sublanguage is composed of atomic FTSelections that are fully composable (any number of times!).
- Support for thesauri, ability to share modifiers, specify/override stopwords, control number of occurrences, scope, fine granularity control over search token sequence.
- Scoring expression allows user weights.
- Evaluation model is represented in an XML Schema. Semantics of TeXQuery is specified in XQuery!
- Does not require any change to XQuery constructs and data model.