TeXQuery: A Full-Text Search Extension to XQuery

Part III – Use Cases Solutions

Sihem Amer-Yahia [1]
sihem@research.att.com

Chavdar Botev [2]
cbotev@cs.cornell.edu

Jonathan Robie [3]
jonathan.robie@datadirect-technologies.com

Jayavel Shanmugasundaram [2]
jai@cs.cornell.edu

[1] AT&T Labs

[2] Computer Science Department, Cornell University

[3] DataDirect Technologies

14 July 2003

Contents

1. XQuery and XPath Full-Text Use Cases

In this appendix we present sample queries that implement the XQuery full-text use cases as given in [3].

Use Case 2: "WORD" – Word and Phrase Queries

Q2.1 – Single Word Query

Find all books containing the word "usability". This query finds a single word within an element.

- Operands: "usability"
- Functionality: word query
- Context: books/book/metadata/title
- Return: books/book/metadata/title
- Comments: This is the simplest query possible, a query on one word within one element. This query does not employ wildcards, stemming, or thesaurus support. While this query finds useful results in the sample data, most queries such as one on the word "test" would not. A query on the word "test" would return no results, missing the word variants which exist in the sample data: "pretest" "tested" "testers" "testimony" "testing" and "tests".

```
//book[. ftcontains "usability"]
```

Q2.2 – Single Phrase Query

Find all book subjects containing the phrase "usability testing". This query finds a single phrase within an element.

- Operands: "usability testing"
- Functionality: phrase query
- Context: books//subject
- Return: books//subject
- Content: This is a simple query on one phrase within one element. Unlike an unordered proximity query, the words in this phrase query must be adjacent to each other and must appear in the order specified. While this query finds useful results in the sample data, most queries such as one on "software developer" would not. A query on the phrase "software developer" would return no results, missing "developer of software" which exists in the sample data.

```
//book//subject[. ftcontains "usability testing"]
```

Q2.3 – Single Phrase Query on Long Text Excerpts

Find all book text containing the phrase "would a user know by looking at the screen how to complete the first step of the task". This query finds a single phrase in an element.

- Operands: "would a user know by looking at the screen how to complete the first step of the task"
- Functionality: phrase query
- Context: books//content
- Return: books//content
- Comments: This query shows phrases can be of any length.

```
//book//content[. ftcontains "would a user know by looking at the
screen how to complete the first step of the task"]
```

Q2.4 – Single Phrase Query on Chinese Characters

Not different from Q2.2 or Q2.4 uses the "Language" context modifier.

Use Case 3: "ELEMENT" – Queries on XML Elements and Attributes

Q3.1 – Query on Element

Find all book paragraphs with the word "step". This query finds a word in an element.

– Operands: "step"

– Functionality: word query

– Context: books//p

– Return: books/book/@number, books//p

– Comments: This query finds the word "step" in the p element, not in any other element.

```
for $book in //book
let $para=$book//p[. ftcontains "step"]
where fn:count($para)>0
return
   <book number="{$book/@number}"> {$para} </book>
```

Q3.2 – Query on Multiple Operands in Same Instance of an Element

Find all books with the phrase "web site" and the word "usability" in the same subject. This query finds a word and a phrase within one instance of an element.

– Operands: "web site" "usability"

– Functionality: phrase query, word query, `and` query

– Context: books//subject

– Return: books/book/@number, books//subject

– Comments: This query find words and phrases within one instance of an element, not allowing one of the operands to be found in one sibling and the other operand in a different sibling of the same name. This query does not find book 2 which has "usability" and "web site" in different instances of an element. It uses an `and` query introduced in Section 9 (BOOLEAN).

```
for $book in //book
let $subj=$book//subject[. ftcontains ("web site" &&
                                        "usability")]
where fn:count($subj)>0
return <book number="{$book/@number}"> {$subj} </book>
```

Q3.3 – Query on Multiple Operands in Any Instance of an Element

Find all books with the phrase "web site" and the word "usability" in any subject. This query finds a word and a phrase within any instance of an element, across the siblings of the same name. The word and the phrase may be in different elements.

- Operands: "web site" "usability"
- Functionality: phrase query, word query, `and` query
- Context: books//subject
- Return: books/book/@number, books//subject
- Comments: This query returns words and phrases within any instance of an element, allowing one of the operands to be found in one sibling and the other operand in a different sibling of the same name. It uses an `and` query introduced in Section 9 (BOOLEAN).

```
for $book in //book
let $subj1 := $book//subject[. ftcontains “web site”]
let $subj2 := $book//subject[. ftcontains “usability”]
return <book number=”{$book/@number}”> {$subj1, $subj2} </book>
```

Q3.4 – Query on Multiple Operands in Every Instance of an Element

Find all books with the words "ersatz" and "publications" in every publisher name. This query finds two words within every instance of an element.

- Operands: "ersatz" "publications"
- Functionality: word query, `and` query
- Context: books//publisher
- Return: books/book/metadata/title, books//publisher
- Comments: This query does not return a book if at least one of its publishers does not contain the words "ersatz" and "publications". Book 1 is returned because it has a single publisher which satisfies the full-text conditions (contains both "Ersatz" and "Publications"). Book 2 is not returned because even though it has a publisher that satisfies the full-text conditions, it also has a publisher that does not satisfy it (universal quantification). Book 3 is not returned because it has a single publisher which does not satisfy the query (contains "Ersatz" but not "Publications"). Finally, universal quantification will allow books without publishers to qualify. It uses an `and` query introduced in Section 9 (BOOLEAN).

```
for $book in //book
where every $pub in $book//publisher
      satisfies ($pub ftcontains (“ersatz” && “publications”))
return <result> {$book//title,$book//publisher} </result>
```

Q3.5 – Query on Element Returning Different Elements

Find all books with the phrase "usability testing" in some subject. This query finds a single phrase within an element and returns different elements from the same document.

- Operands: "usability testing"
- Functionality: phrase query
- Context: books//subject
- Return: books/book/metadata/title, books//author, books//publicationInfo/dateRevised
- Comments: This query queries the subject element, but does not return it. It returns three different elements which provide useful information to the user.

```
for $book in //book
where $book//subject ftcontains “usability testing”
return <result> {$book/metadata/title,
                 $book//author,
                 $book//dateRevised}
       </result>
```

Q3.6 – Query on Multiple Elements

Find all books with "usability tests" in book or chapter titles. This query finds a phrase in multiple elements.

– Operands: "usability tests"

– Functionality: phrase query

– Context: books/book/metadata/title, books/book/content/part//chapter/title

– Return: books/book/metadata/title, books/book/content/part//chapter/title

– Comments: This query is an example of a word query in two elements.

– Version: For consideration in v.1

```
for $book in //book
where $book/metadata/title ftcontains “usability tests”
      or
      $book/content/part//chapter/title ftcontains “usability
tests”
return <book> {$book} </book>
```

Q3.7 – Query across Element Boundaries

Find all books with the phrase "usability testing once the problems". This query finds a phrase which begins in one element and ends in a second.

– Operands: "usability testing once the problems"

– Functionality: phrase query, ignoring element tags

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query crosses sibling element boundaries.

```
for $book in //book
let $cont := $book//content[. ftcontains
                           "usability testing once the problems"]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q3.8 – Query on Element and Its Descendants

Find all books with the word "tests". This query finds a word in an element or its descendants.

– Operands: "tests"

– Functionality: word query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query crosses parent-child element boundaries.

```
for $book in //book
let $cont := $book//content[. ftcontains "tests"]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q3.9 – Query on Attribute

Find all books with "improve" "web" "usability" in the short title. This query finds multiple words within an attribute allowing word variants and allowing the words in any order with up to a specified number of intervening words.

– Operands: "improve" "web" "usability"

– Functionality: word queries, stemming, unordered proximity (0 to 2 intervening words)

– Context: books/book//title/@shortTitle

– Return: books/book/@number, books/book//title/@shortTitle

– Comments: This query illustrates full-text querying within an attribute. It uses stemming introduced in Section 7 (STEMMING) and an ordered proximity query introduced in Section 10 (DISTANCE).

```
for $book in //book
let $stitle := $book/title/@shortTitle[. ftcontains ("improve" &&
"web" && "usability") with stems with word distance at most 2]
where fn:count($stitle)>0
return <book number="{$book/@number}"> {$stitle} </book>
```

Q3.10 – Query on Element and Attribute

Find all books with the phrase "manuscript guides" in the short title and the phrase "user profiling" in the text. This query finds a phrase within an attribute and a phrase within an element and its descendants.

– Operands: "manuscript guides" "user profiling"

– Functionality: phrase queries, stemming, `and` query

– Context: books/book//title/@shortTitle, books//content

– Return: books/book/@number, books/book//title/@shortTitle, books//content

– Comments: This query combines querying within an element with querying within an attribute. It uses stemming introduced in Section 7 (STEMMING) and an `and` query introduced in Section 9 (BOOLEAN).

```
for $book in //book
let $stitle := $book/title/@shortTitle[. ftcontains
                                       "manuscript guides"
                                       with stems]
```

```
let $cont := $book//content[. ftcontains
                           "user profiling" with stems]
where fn:count($stitle)>0 and fn:count($cont)>0
return <book number="{$book/@number}"> {$stitle, $cont} </book>
```

Use Case 4: "STOP-WORD" – Queries Ignoring and Overriding Stop Words

Q4.1 – Query on Word Ignoring Single Stop Word

Find all books with the word "the" in the text. This query ignores a word which has been identified as a stop word.

- Operands: "the"
- Functionality: identify and ignore stop words (the), word query
- Context: books//content
- Return: books/book/metadata/title
- Comments: This query on a stop word will either return no results or an error depending on the implementation.

```
//book/metadata/title[. ftcontains "the" with stopwords "the"]
```

Q4.2 – Query on Phrase Ignoring Single Stop Word

Find all books with the phrase "planning then conducting" in the text. This query finds a phrase ignoring a word which has been identified as a stop word.

- Operands: "planning then conducting"
- Functionality: identify and ignore stop words (then), phrase query
- Context: books//content
- Return: books/book/metadata/title, books//content
- Comments: Once the stop word "then" is ignored, this query is reduced to a query on the words "planning" and" conducting", allowing one intervening word. It finds both "planning and conducting" and "planning then conducting" in the sample data.

```
for $book in //book
let $cont := $book//content[. ftcontains
                            "planning then conducting"
                            with stopwords "then"]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q4.3 – Query on Phrase Overriding Single Stop Word

Find books with the phrase "Guides and finding aids" in the subject. This query finds a phrase overriding the stop word identification of a word.

- Operands: "guides and finding aids"
- Functionality: identify and override stop words (and), phrase query

– Context: books//subject

– Return: books/book/metadata/title, books//subject

– Comments: This query finds the phrase "guides and finding aids", which might be in a subject element built from a controlled vocabulary where querying the exact phrase may avoid undesired results.

```
for $book in //book
let $subj := $book//subject[. ftcontains
                           "Guides and finding aids"
                           without stopwords "and"]
where fn:count($subj)>0
return <result> {$book/metadata/title, $subj} </result>
```

Q4.4 – Query on Phrase Ignoring Multiple Stop Words

Find all books which have not been approved by a Web users organization. This query finds a phrase ignoring multiple words which has been identified as stop words.

– Operands: "not been approved"

– Functionality: identify and ignore stop words (not, been), phrase query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: Once the stop words are ignored, this query is reduced to a query on the word "approved". It returns books with the word "approved" in the phrases "been approved" "not been approved" and "approved travel orders" in the sample data.

```
for $book in //book
let $cont := $book//content[. ftcontains
                           "not been approved"
                           with stopwords ("been" "not")]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q4.5 – Query on Phrase Overriding Multiple Stop Words

Find all the books which have not been approved by a Web users organization. This query finds a phrase overriding the stop word identification of a multiple words.

– Operands: "not been approved"

– Functionality: identify and override stop words (not, been), phrase query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query finds the phrase "not been approved". It does not return the phrases "been approved" and "approved travel orders" in the sample data.

```
for $book in //book
let $cont := $book//content[. ftcontains
```

```
                            “not been approved”
                            without stopwords]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Use Case 5: “CHARACTER-MANIPULATION” – Queries Manipulating Normalized Characters and Tokenized Words, Spaces, and Punctuation

Q5.1 – Query Returning Characters with Diacritics Only

Verify the existence of a résumé in the papers of John Wesley Usabilityguy. This query finds a word only when diacritics are present.

– Operands: "résumé"

– Functionality: collation requiring presence of diacritics, word query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: The desired return is only the word "résumé" with diacritics, not the more often used word "resume" which is present in the sample data.

```
for $book in //book[./metadata/title ftcontains
                    “John Wesley Usabilityguy”]
let $cont := $book//content[. ftcontains “résumé”
                            with diacritics]
return <result> {$book/metadata/title, $cont} </result>
```

Q5.2 – Query Returning Characters with and without Diacritics

Verify the existence of a résumé in the papers of John Wesley Usabilityguy. This query finds a word whether its diacritics are present or not.

– Operands: "resume"

– Functionality: collation ignoring the presence of diacritics, word query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: The desired return is either the word "résumé" with diacritics or the word "resume" without diacritics. The user wants to find résumé, but has entered resume possibly because the system does not allow the entry of diacritics, the user does not know how to enter diacritics, or the user did not want to take the time to enter them.

```
for $book in //book[./metadata/title ftcontains
                    “John Wesley Usabilityguy”]
let $cont := $book//content[. ftcontains “resume”
                            diacritics insensitive]
return <result> {$book/metadata/title, $cont} </result>
```

Q5.3 – Query Returning Upper Case Characters Only

Find out whether John Wesley Usabilityguy included research on AIDS among the charities he supported. This query finds a word in upper case letters only.

– Operands: "AIDS"

– Functionality: collation which is case-sensitive or limited to upper case, word query

– Context: books//book

– Return: books/book/metadata/title, books//book

– Comments: This query does not return the word "aids" in lower case which exists in the sample data.

```
for $book in //book[. ftcontains “AIDS” uppercase]
return <result> {$book/metadata/title, $book} </result>
```

Q5.4 – Query Returning Upper Case and Lower Case Characters

Find out whether John Wesley Usabilityguy included research on AIDS among the charities he supported. This query finds a word in upper or lower case letters.

– Operands: "AIDS"

– Functionality: collation which is case-insensitive, word query

– Context: books//book

– Return: books/book/metadata/title, books//book

– Comments: This query returns the word "aids" in upper and lower case.

```
for $book in //book[. ftcontains “AIDS” case insensitive]
return <result> {$book/metadata/title, $book} </result>
```

Q5.5 – Query Returning Word with Special Characters

– Find all books with "walk-through". Operands: "walk-through"

– Functionality: word query, collation returning words containing special characters

– Context: books//book

– Return: books/book/metadata/title, books//book

– Comments: The desired return is only the hyphenated word "walk-through", not the phrase "walk through" which is also present in the sample data.

```
for $book in //book[. ftcontains “walk-through”
                        with special characters]
return <result> {$book/metadata/title, $book} </result>
```

Q5.6 – Query Returning Word with Special Character or Phrase with Space

Find all books with "walk-through". This query finds a word with a special character or a phrase where the special character is replaced by a space.

– Operands: "walk-through"

– Functionality: word query, collation returning words containing special characters or phrases containing a space

– Context: books//book

– Return: books/book/metadata/title, books//book

– Comments: The desired return is the hyphenated word "walk-through" or the phrase "walk through".

```
for $book in //book[. ftcontains “walk-through”
                    without special characters]
return <result> {$book/metadata/title, $book} </result>
```

Use Case 6: “WILDCARD”: Character Wildcard (Prefix, Infix, Suffix) and Word Wildcard Queries

Q6.1 – Single Prefix Character Wildcard Query

Find all books with the word "way" with any one character prefix. This query finds a word allowing any one character prefix (any one character before the first character).

– Operands: "way"

– Functionality: word query, character wildcard (prefix)

– Context: books//content

– Return: books/book/@number, books//content

– Comments: This query specifies that one and only one character be added to the word.

```
for $book in //book
let $cont := $book//content[. ftcontains “.way”]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q6.2 – Single Suffix Character Wildcard Query

Find all books with the word "test" with any one character suffix. This query finds a word allowing any one character suffix (any one character after the last character).

– Operands: "test"

– Functionality: word query, character wildcard (suffix)

– Context: books//content

– Return: books/book/@number, books//content

– Comments: This query finds "tests", but not "pretest" "test" "tested" "testers" "testimony" and "testing" which also appear in the sample data.

```
for $book in //book
let $cont := $b//content[. ftcontains “test.”]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q6.3 – Single Infix Character Wildcard Query

Find all books with the words "step or stop". This query finds words allowing any one infix character (any one character in the middle of a word).

– Operands: "st" "p"

– Functionality: word query, character wildcard (infix)

– Context: books//content

– Return: books/book/@number, books//content

– Comments: This query allows one and only one character to be added to the word.

```
for $book in //book
let $cont := $book//content[. ftcontains "st.p"]
where fn:count($cont)>0
return <book number="{$book/@number}"> {$cont} </book>
```

Q6.4 – 0 or More Prefix Character Wildcard Queries

Find all books with the word "test" with any prefix. This query finds a word allowing any prefix (0 or more characters before the first character).

– Operands: "test"

– Functionality: word query, character wildcard (prefix)

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query finds "pretest", but not "test" "testers" "testimony" "testing" or "tests" which also appear in the sample data.

```
for $book in //book
let $cont := $book//content[. ftcontains ".*step"]
where fn:count($cont)>0
return <book number="{$book/@number}"> {$cont} </book>
```

Q6.5 – 0 or More Suffix Character Wildcard Queries

Find all books with the word "test" with any suffix. This query finds a word allowing any suffix (0 or more characters after the last character).

– Operands: "test"

– Functionality: word query, character wildcard (suffix)

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query finds "test" "testers" "testimony" "testing" and "tests", but not "pretest" which also appear in the sample data.

```
for $book in //book
let $cont := $book//content[. ftcontains "step.*"]
where fn:count($cont)>0
return <book number="{$book/@number}"> {$cont} </book>
```

Q6.6 – 0 or More Suffix Character Wildcard Queries on a Part of a Word

Find all books with the phrases "usability testing" or "user testing". This query finds a phrase allowing any suffix (0 or more characters after the last character) on a part of one of the words.

– Operands: "us testing"

– Functionality: phrase query, character wildcard (suffix)

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This is an example of a suffix query on a part of a word "us" which is not one of the words or one of the roots of the words desired in the results. The query on "us" will find "usability" and "user". Where stemmed queries (presented below in Section 7 (STEMMING) attempt to return linguistic variants on a word or the root of a word, wildcards may be applied to any part of a word and will return all character combinations found.

```
for $book in //book
let $cont := $book//content[. ftcontains “us.* testing”]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q6.7 – 0 or More Infix Character Wildcard Queries

Find all books with the words "serve" or "service". This query finds words allowing any infix characters (0 or more characters inserted in the middle of a word).

– Operands: "serv", "e"

– Functionality: word query, character wildcard (infix)

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query returns the word "service" and would return the word "serve" if it existed in the sample data. It does not return the word "served" which exists in the sample data.

```
for $book in //book
let $cont := $book//content[. ftcontains “serv.*e”]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q6.8 – Specified Range Suffix Characters Wildcard Query

Find all books with the word "test" with any three to four character suffix. This query finds a word allowing a number of characters within a specified range in a suffix (specified range of characters after the last character).

– Operands: "test"

– Functionality: word query, character wildcard (suffix)

– Context: books//content

– Return: books/book/@number, books//content

– Comments: This query allows any three or four character suffix. It returns "testers" and "testing", but not "pretest" "tests" "tested" and "test" which also appear in the sample data.

```
for $book in //book
let $cont := $book//content[. ftcontains “test.{3, 4}”]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q6.9 – Word Wildcard Query

Find all books with a phrase which begins with "propagating", has any word in the middle, and ends with "errors". This query finds a phrase where one of the words (represented by a wildcard) is unspecified.

– Operands: "propagating errors"

– Functionality: phrase query, word wildcard

– Context: books//content

– Return: books/book/@number, books//content

– Comment: This query is a word wildcard query. A one word wildcard query returns the same result as an ordered proximity query allowing one intervening word.

```
for $book in //book
let $cont := $book//content[. ftcontains
                            “propagating errors”
                            with word distance 2 ordered]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q6.10 – Specified Range Word Wildcard Query

Find all books with a phrase which begins with "propagating", has up to ten words in the middle, and ends with "errors". This query finds a phrase where a number of words within a specified range (represented by wildcards) are inserted after the first word and before the last.

– Operands: "propagating errors"

– Functionality: phrase query, word wildcard

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query illustrates a multiple word wildcard representing a specified range. In this query the range is set as 0 to 10.

```
for $book in //book
let $cont := $book//content[. ftcontains
                           “propagating errors”
                           with word distance at most 11
                           ordered]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Use Case 7: “STEMMING” – Word Stemming Queries

Q7.1 – Single Word Stemming Query

Find all books with the word "test". This query finds words in an element and its descendants applying a stemming algorithm.

- Operands: "test"
- Functionality: word query, stemming
- Context: books//content
- Return: books/book/metadata/title, books//content
- Comments: Unlike the wildcard queries in Section 6 (WILDCARD) which allow any suffix, this query will probably not return the word "testimony" which occurs in the sample data.

```
for $book in //book
let $cont := $book//content[. ftcontains “test” with stems]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q7.2 – Multiple Word Stemming Query

Find all books with the phrases "usability testing" or "user testing". This query finds phrases in an element and its descendants applying a stemming algorithm to multiple words.

- Operands: "usable testing" "use testing"
- Functionality: phrase query, stemming
- Context: books//content
- Return: books/book/metadata/title, books//content
- Comments: Unlike the wildcard queries in Section 6 (WILDCARD) which allow any suffix, a stemmed query on "us" will not return the desired results because "user" and "usability" do not share the share root.

```
for $book in //book
let $cont := $book//content[. ftcontains
                          (“usable testing” | “use testing”)
                          with stems]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Use Case 8: “THESAURUS’ – Queries Which Use Thesauri, Dictionaries, and Taxonomies

Q8.1 – Query on Synonyms Identified by a Thesaurus

Find all books which contain quotes. This query finds a word using a thesaurus to return synonyms.

- Operands: "quote"
- Functionality: word query, thesaurus support

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query uses thesaurus support to identify synonyms for the word "quote" via preferred and used for terms. It returns the following words: said, says, stated, states, spoke, speaks, replied, replies, reply, remarks, remarked, responded, response, reports, reported, quotes, quoted, according to, commented, discussed, expressed, told--which become additional operands

```
for $book in //book
let $cont := $book//content[. ftcontains “quote”
                            with thesaurus “Synonyms”]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q8.2 – Query on Narrower Terms Identified by a Thesaurus

Find all books with text on improving web site components. This query finds a word in one element or its descendants using a thesaurus to identify narrower terms.

– Operands: "web site components"

– Functionality: phrase query, thesaurus support

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query employs a thesaurus to identify web site components via narrower terms, including: layout, terminology, graphics, menus, navigation--which become additional operands.

```
for $book in //book
let $cont := $book//content[. ftcontains “web site components”
                            with thesaurus “NarrowerTerms”]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q8.3 – Query on Broader Terms Identified by a Thesaurus

Are there any letters or holiday cards in John Wesley Usabilityguy's papers? This query finds a word in one element or its descendants using a thesaurus to identify broader terms.

– Operands: "letters" "holiday cards"

– Functionality: word query, phrase query, thesaurus support

– Context: books/book[3]//content

– Return: books/book[3]/metadata/title, books/book[3]//content

– Comments: This query employs a thesaurus to identify the broader term "correspondence"--which becomes an additional operand.

```
for $book in //book[./metadata/title ftcontains
                    “John Wesley Usabilityguy”]
let $cont := $book//content[. ftcontains
```

```
                           (“letters” | “holiday cards”)
                           with thesaurus “BroaderTerms”]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q8.4 – Query on Word Which Sounds Like Other Words

Find all books with words which sound like "Merrygould". This query finds words in an element and its descendants using a dictionary of words which sound like the word queried.

– Operands: "Merrygould"

– Functionality: word query, sounds-like dictionary support

– Context: books//book

– Return: books/book/metadata/title, books//book

– Comments: This query uses sounds-like support to identify words which sound like the word "Merrygould". It returns the word "Marigold"-- which becomes an additional operand.

```
for $book in //book[. ftcontains “Merrygould”
                    with thesaurus “soundex”]
return <result> {$book/metadata/title, $book} </result>
```

Q8.5 – Query on Word Spelled Similarly to Other Words

Find all books which contain words that are close in spelling to "sucessfull". This query finds words in an element and its descendants using a dictionary of words that are spelled similarly.

– Operands: "sucessfull"

– Functionality: word query, similarly spelled dictionary support

– Context: books//book

– Return: books/book/metadata/title, books//book

– Comments: This query uses support for similarly spelled words to identify words close in spelling to "sucessfull". It returns the word "successful"--which becomes an additional operand.

```
for $book in //book[. ftcontains “Merrygould”
                    with thesaurus “spellcheck”]
return <result> {$book/metadata/title, $book} </result>
```

Q8.6 – Query on Subordinate Terms Identified by a Taxonomy

Find out whether John Wesley Usabilityguy included research on AIDS and other infectious diseases among the charities he supported. This query finds a word in an element and its descendants in upper case letters using a taxonomy to identify subordinate terms.

– Operands: "AIDS"

– Functionality: word query, collation which is case-sensitive or limited to upper case, taxonomy support

– Context: books/book[3]//component

– Return: books/book/@number, books/book[3]//component

– Comments: This query uses a taxonomy to identify other infectious diseases (e.g., Hepatitis, Tuberculosis) which are then added as operands.

```
for $book in /books/book[3]
let $comp := $book//component[. ftcontains “AIDS”
                              with thesaurus “wordnet”
                              uppercase]
where fn:count($comp)>0
return <book number=”{$book/@number}”> {$comp} </book>
```

Use Case 9: “BOOLEAN” – Or, And, and Not Queries

Q9.1 – Or Query

Find all books with the words "web" or "software" in the text. This query finds one or both of the words in an element and its descendants.

- Operands: "web" "software"
- Functionality: word query, `or` query
- Context: books//content
- Return: books/book/metadata/title, books//content
- Comment: This `or` query returns either or both of words queried.

```
for $book in //book
let $cont := $book//content[. ftcontains “web” || “software”]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q9.2 – Or Query on More than Two Words

Find all books with the words "web" or "software" or "internet" in the text. This query finds any or all of the words in an element and its descendants.

– Operands: "web" "software" "internet"

– Functionality: word queries, `or` query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: An `or` query can have any number of words as operands. Since the word "internet" does not appear in the sample data, books with both or either of the remaining two words are returned.

```
for $book in //book
let $cont := $book//content[. ftcontains
                             “web” || “software” || “internet”]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q9.3 – Or Query on Phrases

Find all books with the phrases "heuristic evaluation" or "cognitive walk-through". This query finds any or all of the phrases in an element or its descendants.

– Operands:"heuristic evaluation" "cognitive walk-through"

– Functionality: word query, `or` query

– Context: books//content

– Return: books/book/@number, books//content

– Comments: An `or` query can have any number of words and phrases as operands.

```
for $book in //book
let $cont := $book//content[. ftcontains
                           ("heuristic evaluation" ||
                            "cognitive walk-through")
                           with special characters]
where fn:count($cont)>0
return <book number="{$book/@number}"> {$cont} </book>
```

Q9.4 – And Query

Find all books with the words "web" "software" in the text. This query finds all of the words in one element and its descendants.

– Operands: "web" "software"

– Functionality: word queries, `and` query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: The `and` query finds both words.

```
for $book in //book
let $cont := $book//content[. ftcontains "web" && "software"]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q9.5 – And Query on More than Two Words

Find all books with the words "web" "software" "internet" in the text. This query finds all of the words queried in an element or its descendants.

– Operands: "web" "software" "internet"

– Functionality: word queries, `and` query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: An `and` query can have any number of words as operands. The word "internet" does not appear in the sample data, so no results are returned.

```
for $book in //book
```

```
let $cont := $book//content[. ftcontains
                            “web” && “software” && “internet”]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q9.6 – And Query on Phrases

Find all books with the phrases "heuristic evaluation" and "cognitive walk-through" in the text. This query finds all of the phrases queried in an element or its descendants.

– Operands: "heuristic evaluation" "cognitive walk-through"

– Functionality: phrase queries, `and` query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: An `and` query can have any number of words or phrases as operands.

```
for $book in //book
let $cont := $book//content[. ftcontains
                            (“heuristic evaluation” &&
                             “cognitive walk-through”)
                            with special characters]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q9.7 – Unary Not Query

Find all books which do not belong in a collection on usability testing. This query finds books which do not contain a phrase in any element.

– Operands: "usability testing"

– Functionality: phrase query, character wildcard (suffix), `unary not` query

– Context: books//book

– Return: books/book/metadata/title

– Comments: Unlike the `and not` query below, the `unary not` query requires only one operand. This query has value for information architects and data managers who will use it for checks such as this one, to find nonconforming data and documents in a collection.

```
//book[. ftcontains ! (“usablity testing” with
stems)]/metadata/title
```

Q9.8 – And Not Query

Find all books with the word "usability" and not the word "plan" in the metadata. This query finds a word only when another is not found.

– Operands: "usability" "plan"

– Functionality: word query, `and not` query

– Context: books/book//metadata

– Return: books/book//metadata

- Comments: The `and not` query is also called a `but`, `but not`, and `without` query. Unlike the `unary not` query above,this query requires two operands. Book 2 which contains the words "usability" and "plan" in the metadata is not returned.

```
//book/metadata[. ftcontains "usability" && ! "plan"]
```

Q9.9 – And Not Query Where Second Operand is a Subset of the First Operand

Find all books with listings for résumés, drafts, and correspondence, and not book drafts in the metadata or text. This query finds books with multiple words and not a phrase containing one of those words in an element.

- Operands: "résumés" "drafts" "correspondence" "book drafts"
- Functionality: word queries, `or` query, phrase query, `and not` query, collation requiring presence of diacritics
- Context: books//content
- Return: books/book/metadata/title, books//content
- Comments: This query will not return a result which contains everything the user wants which also includes what the user does not want, "book drafts". The user will lose results which contain everything he wants when that book also contains what he does not want.

```
for $book in //book
let $cont := $book//content[. ftcontains
                            ("résumés" | "drafts" |
                            "correspondence") &&
                            ! "book drafts"]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q9.10 – Mild Not Query Where Second Operand is a Subset of the First Operand

Find all books with listings for résumés, drafts, and correspondence, and not book drafts, in the metadata or text. This query finds books with multiple words, not considering a phrase containing one of those words, in an element.

- Operands: "résumés" "drafts" "correspondence" "book drafts"
- Functionality: word queries, `or` query, phrase query, `mild not` query, collation requiring presence of diacritics
- Context: books//book
- Return: books/book/metadata/title, books//book
- Comments: This query will return a result which contains all the terms user wants which may also include some of the terms user deprecated via the mild not, "book drafts". The user will not lose results which contain everything he wants when that book also contains what he does not want. Books containing instances of "book drafts" (a subset of "drafts") are not excluded, merely not considered.

```
for $book in //book[. ftcontains
                    ("résumés" | "drafts" | "correspondence")
                     mildnot "book drafts"]
```

```
return <result> {$book/metadata/title, $book} </result>
```

UseCase 10: “DISTANCE” – Queries on Distance Relanstionships Including Proximity, Window, Sentence, and Paragraph Queries

Q10.1 – Unordered Proximity Query

Find all books with information on software developers. This query finds multiple words in an element in any order allowing up to a specified number of intervening words.

- Operands: "software" "developer"
- Functionality: word queries, stemming, unordered proximity (0 to 3 intervening words)
- Context: books//content
- Return: books/book/metadata/title, books//content
- Comments: This query returns "developer of software", which also occurs in the sample data, which a phrase query in Section 2 (WORD) could not.

```
for $book in //book
let $cont := $book//content[. ftcontains
                           (“software” && “developer”)
                            with word distance at most 3
                           with stems]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q10.2 – Ordered Proximity Query

Find all books with information on efficient task completion. This query finds multiple words in an element and its descendants in the order queried allowing up to a specified number of intervening words.

- Operands: "efficient" "task" "completion"
- Functionality: word queries, ordered proximity (0 to 10 intervening words)
- Context: books//content
- Return: books/book/metadata/title, books//content
- Comments: This query is more permissive than an phrase query on "efficient task completion" which would return no results.

```
for $book in //book
let $cont := $book//content[. ftcontains
                           (“efficient” && “task completion”)
                           ordered
                           with word distance at most 10
                           with stems]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q10.3 – Unordered Window Query

Find all books on how expert reviews identify and correct problems. This query finds books with multiple words within an unordered window of up to a specified number of words.

- Operands: "problems" "correct" "identify" "expert reviews"
- Functionality: word queries, phrase query, stemming, unordered window of 0 to 30 words
- Context: books//content
- Return: books/book/@number, books//content
- Comments: This query opens a window on the first found word or phrase (which can be any of the words or phrases queried, and counts a specified number of words from that first word within which it may find the remaining word or words, finding them in any order.

```
for $book in //book
let $cont := $book//content[. ftcontains
                             (“problems” && “correct” &&
                              “identify” && “expert reviews”)
                              within window at most 30 with stems]
where fn:count($cont)>0
return <book number=”{$book/@number}”> $cont </book>
```

Q10.4 – Ordered Window Query

Find all books about users feeling well-served. This query finds books with multiple words within an ordered window of up to a specified number of words.

- Operands: "users" "feeling" "well" "served"
- Functionality: word queries, stemming, collation which treats the hyphen as a space, unordered window of 0 to 15 words
- Context: books//content
- Return: books/book/metadata/title, books//content
- Comments: This query opens a window on the first found word (which must be the first word queried) and counts a specified number of words from that first word within which it may find the remaining word or words, finding them in the order queried.

```
for $book in //book
let $cont := $book//content[. ftcontains
                             (“users” && “feeling” &&
                              “well” && “served”)
                              ordered within window at most 15
                              with stems]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q10.5 – Unordered Window Within a Sentence Query

Find books which discuss questions asked during cognitive walk-throughs. This query finds books with multiple words in any order within a sentence.

- Operands: "users" "would" "know" "step"
- Functionality: word queries, stemming, unordered sentence query
- Context: books//content
- Return: books/book/@number, books//content
- Comments: This query expects an implementation-defined tokenized sentence or a sentence element.

```
for $book in //book
let $cont := $book//content[. ftcontains
                            (“users” && “would” &&
                             “know” && “step”)
                             same sentence with stems]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q10.6 – Ordered Window Within a Sentence Query

Find books which discuss questions asked during cognitive walk-throughs. This query finds books with multiple words in the order queried within a sentence.

- Operands: "users" "would" "know" "step"
- Functionality: word queries, stemming, ordered sentence query
- Context: books//content
- Return: books/book/@number, books//content
- Comments: This query expects an implementation-defined tokenized sentence or a sentence element.

```
for $book in //book
let $cont := $book//content[. ftcontains
                             (“users” && “would” &&
                              “know” && “step”)
                              ordered same sentence with stems]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q10.7 – Unordered Within a Paragraph Query

Find all paragraphs which define what Web site usability is. This query finds books with multiple words and phrases in any order within a paragraph.

- Operands: "usability" "Web site" "efficiency" "satisfaction"
- Functionality: word queries, phrase query, stemming, unordered paragraph query

– Context: books//content

– Return: books/book/@number, books//content

– Comments: This query expects an implementation-defined tokenized paragraph or a paragraph element.

```
for $book in //book
let $cont := $book//content[. ftcontains
                             (“usability” && “Web site” &&
                              “efficiency” && “satisfaction”)
                              same paragraph with stems]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q10.8 – Ordered Within a Paragraph Query

Find all paragraphs which define what Web site usability is. This query finds books with multiple words and phrases in any order within a paragraph.

– Operands: "usability" "Web site" "is"

– Functionality: word queries, phrase query, stemming, ordered paragraph query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query expects an implementation-defined tokenized paragraph or a paragraph element.

```
for $book in //book
let $cont := $book//content[. ftcontains
                            (“usability” && “Web site” && “is”)
                             ordered same paragraph with stems]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Use Case 11: “ADVANCED-WORD” – Advanced Word and Phrase Queries

Q11.1 – Ordered Word Query

Find all book text containing the words "goal" "obstacles" "task" in this order. This query finds multiple words within an element and its descendants in the order queried.

– Operands: "goal" "obstacles" "task"

– Functionality: ordered word query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query requires multiple words be found in a specified order. It is more permissive than a phrase query. It is comparable to an ordered proximity query where the number of intervening words is 0 or more.

```
for $book in //book
let $cont := $book//content[. ftcontains
                            (“goal” && “obstacles” && “task”)
                            ordered]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q11.2 – AtLeast Query

Find all books which repeat the phrase "expert review methods" at least 2 times. This query finds a phrase that is repeated a specified number of times in an element and its descendants.

– Operands: "expert review methods"

– Functionality: phrase query

– Context: books//content

– Return: books/book/metadata/title, books//content

– Comments: This query employs the `atleast` query. Here it excludes books with only minor references to "expert review methods".

```
for $book in //book
let $cont := $book//content[. ftcontains
                             “expert review methods”
                             at least 2 occurrences]
where fn:count($cont)>0
return <book number=”{$book/@number}”> {$cont} </book>
```

Q11.3 – Starts-With Query

Find book titles which start with "improving the usability". This query finds an element which starts with a phrase.

– Operands: "improving the usability"

– Functionality: phrase query, starts-with functionality

– Context: books/book/metadata/title

– Return: books/book/metadata/title

– Comments: The starts-with functionality restricts the query to the first phrase in an element. It is especially useful in querying journal titles (e.g., *Journal of Psychology*) in large library collections. This query does not find book 2 which contains the phrase "improving the usability" in the title element, because "improving the usability" is not the first phrase in the title element.

```
//book/metadata/title[. ftcontains “^improving the usability”]
```

Q11.4 – Exact Phrase Only

Find all books with the exact title "Improving the Usability of a Web Site Through Expert Reviews and Usability Testing". This query finds the exact phrase allowing no other words or phrases in the element.

– Operands: "improving the usability of a web site through expert reviews and usability testing"

– Functionality: phrase query

– Context: books/book/metadata/title

– Return: books/book/metadata/title

– Comments: This query insists that the content of the element is exactly the same, no more, no less than what is queried.

```
//book/metadata/title[text() = "Improving the Usability of a Web
Site Through Expert Reviews and Usability Testing"]
```

Use Case 12: "SCORE" – Queries Unique to Score

Q12.1 – Multiple Word Query

Find all book text containing the words "task" "completion" "goals". This query finds multiple words within an element, returning books containing all the words first, then those with fewer, then those with one.

– Operands: "task" "completion" "goals"

– Functionality: word query, stemming, implementation-defined scoring

– Context: books//content

– Return: books/book/@number, books//content

– Comments: This query on multiple words cannot be written as a query with pure Boolean full-text predicate without additional functionality (e.g., `and`, `or`, proximity). This is however a common scored query. The scoring methodology in this use case is for illustrative purposes only. Scoring methodologies will be implementation-defined.

```
for $book in //book
let $score := $book//content ftscore ((“task” || “completion” ||
                                        “goals”) with stems)
where $score > 0
order by $score descending
return <book number=”{$book/@number}”> {$book//content} </book>
```

Q12.2 – Multiple Phrase Query

Find all books containing the phrases "heuristic evaluation" and "cognitive walk-through". This query finds multiple phrases within an element, returning books containing all phrases queried first, books with one phrase or none after.

– Operands: "heuristic evaluation" and "cognitive walk-through".

– Functionality: phrase query, implementation-defined scoring

– Context: books//content

– Return: books/book/@number, books//content

– Comments: This query on multiple phrases cannot be written as a query with pure Boolean full-text predicate without additional functionality (e.g., `and`, `or`, proximity). This is however a common scored query. The scoring methodology in this use case is for illustrative purposes only. Scoring methodologies will be implementation-defined.

```
for $book in //book
let $score := $book//content ftscore (“heuristic evaluation” ||
                                      (“walk-through”
                                       with special characters)
                                      with stems)
order by $score descending
return <book number=”{$book/@number}”> {$book//content} </book>
```

Q12.3 – Query Which Returns Scores

Find all books which mention usability. Return the book numbers and scores. This query performs a word query and returns scores, highest first.

– Operands: "usability"

– Functionality: word query, implementation-defined scoring, returns score

– Context: books//book

– Return: books/book/metadata/title, score (constructed element)

– Comments: This query is only possible as a scored query. Scores are included between 0 and 1. The scoring methodology in this use case is for illustrative purposes only. Scoring methodologies will be implementation-defined.

```
for $book in //book
let $score := $book ftscore (“usability”)
where $score > 0
order by $score descending
return <result> <title> {$book//title} </title>
                <score> {$score} </score>
       </result>
```

Q12.4 – Query Returning Results with Top Scores

Find the best two books on usability. This query performs a word query and returns only the results with the top scores as specified.

– Operands: "usability"

– Functionality: word query, implementation-defined scoring, returns top scores (returns books with the top 2 scores)

– Context: books//book

– Return: books/book/metadata/title

– Comments: This query returns results only for the 2 books with the highest scores. This query is only possible as a scored query. The scoring methodology in this use case is for illustrative purposes only. Scoring methodologies will be implementation-defined.

```
for $hit at $i in
  for $book in //book
  let $score := $book score by ("usability")
  where $score > 0
  order by $score descending
  return <result> <title> {$book//title} </title> </result>
where $i < 2
return $hit
```

Q12.5 – Query Which Filters on Scores

Find all books that focus on usability. This query performs a word query and filters on scores.

– Operands: "usability"

– Functionality: word query, implementation-defined scoring, filters on scores (accepts only those with score over .10)

– Context: books//book

– Return: books/book/metadata/title

– Comments: This query is only possible as a scored query. The scoring methodology in this use case case is for illustrative purposes only. Scoring methodologies will be implementation-defined.

```
for $book in //book
let $score := $book score by ("usability")
where $score > 0.1
order by $score descending
return <result> <title> {$book//title} </title> </result>
```

Q12.6 – Query Which Returns All Documents Ordered

Find all books on software. This query performs a word query, returns all the documents in the database, and orders them returning those with found word first, those without last.

– Operands: "software"

– Functionality: word query, implementation-defined scoring

– Context: books//content

– Return: books/book/@number books//content

– Comments: This query is only meaningful as a scored query. This query is probably only desirable in a small collection or database. The scoring methodology in this use case case is for illustrative purposes only. Scoring methodologies will be implementation-defined.

```
for $book in //book
let $score := $book//content score by ("heuristic evaluation" ||
                                       (“walk-through”
                                        with special characters))
order by $score descending
return <book number=”{$book/@number}”> {$book//content} </book>
```

Use Case 13: “STRUCTURE” – Queries using XPath Axes

Q13.1 – Query on Element and Its Children

Find all books with chapter titles containing the word "usability" and chapter paragraphs containing the phrase "computer workstation". This query finds a word in a child of an element and a phrase in another child of the same element.

- Operands: "usability", "computer workstation"
- Functionality: word query, phrase query
- Context: books//chapter/title, books//chapter/p
- Return: books/book/metadata/title, books//chapter/title, books//chapter/p
- Comments: This query looks for a word in the immediate title child of a chapter and for a phrase in that chapter's immediate paragraph child, and returns the chapter only if that word and that phrase appear in those children.

```
for $book in //books
let $cont := $book//chapter[./title ftcontains “usability”
                           and
                           ./p ftcontains “computer workstation”]
where fn:count($cont)>0
return <result> {$book/metadata/title, $cont} </result>
```

Q13.2 – Query on Element Returning Its First Two Children

Find the first two steps in chapters on conducting usability tests. This query finds words in an element and returns the first two children of the element.

- Operands: "usability", "test"
- Functionality: word queries, stemming
- Context: books/book//chapter/p
- Return: books/book/@number, books/book/content/part/chapter/p/step[1], books/book/content/part/chapter/p/step[2].
- Comments: This query finds words in a chapter paragraph element and uses XPath to return the two first children of the element.

```
for $book in //books[.//chapter/p ftcontains
                    (“usability” && “test”) with stems]
return <book number=”{$book/@number}”>
         {$book/content/part/chapter/p/step[1],
          $book/content/part/chapter/p/step[2]}
```

```
       </book>
```

Q13.3 – Query on Element and Its Descendants

Find all books with paragraphs containing the phrase "computer workstation" and footnotes within the paragraph containing the word "comfortable". This query finds a phrase in an element, then finds a descendant of that element that contains a word.

- Operands: "computer workstation" "comfortable"
- Functionality: phrase query, word query
- Context: books//p, books//footnote
- Return: books/book/@number, books//p, books//footnote
- Comments: This query combines phrase and word search in different elements which have an ancestor-descendant relationship.

```
for $book in //books
let $para := $b//p[. ftcontains "computer workstation"]
let $fn := $para//footnote[. ftcontains "comfortable"]
where fn:count($fn)>0
return <book number="{$book/@number}"> {$para} </book>
```

Q13.4 – Query on Element and Its Parent

Are there any flow diagrams of human computer interaction scenarios in John Wesley Usabilityguy's papers? This query finds a phrase in one element and a phrase in its parent element.

- Operands: "flow diagrams" "human computer interaction"
- Functionality: phrase queries, character wildcard (suffix)
- Context: books//subcomponent/componentTitle, books//subsubcomponent/ componentTitle
- Return: books/book/@number, books//componentTitle
- Comments: This query looks for two different phrases in two different elements and imposes a parent/child relationship between the two elements.

```
for $book in //books[./metadata/title ftcontains
                     "John Wesley Usabilityguy"]
let $comp := ($book//subcomponent/componentTitle
              $book//subsubcomponent/componentTitle)
               [. ftcontains ("flow diagram.*"
                              "human computer interaction.*")]
where fn:count($comp)>0
return <book number="{$book/@number}"> {$comp} </book>
```

Q13.5 – Query on Element and Its Ancestors

Are there any flow diagrams of human computer interaction scenarios in John Wesley Usabilityguy's papers? This query finds a phrase in one element and a phrase in its ancestor element.

- Operands: "flow diagrams" "human computer interaction"
- Functionality: phrase queries, character wildcard (suffix)
- Context: books//subcomponent/componentTitle, books//subsubcomponent/componentTitle
- Return: books/book/@number, books//componentTitle
- Comments: This query looks for a phrase in an element and a different phrase in all its ancestor elements.

No ancestor axis in XQuery.

Q13.6 – Query on Element and Its Right Siblings

Find all book chapters with paragraphs on usability testing followed by paragraphs on information architecture. This query finds a phrase in an element and another phrase in one of its right siblings.

- Operands: "usability testing" "information architecture"
- Functionality: phrase queries
- Context: books/book/content/part//introduction|chapter
- Return: books/book/@number books/book/content/part//introduction|chapter
- Comments: This query is a Boolean query that returns book chapters if they contain paragraphs following each other as specified in the query.

No sibling axes in XQuery.

Q13.7 – Query on Element and Its Siblings

Find all books with information on identifying problems in a chapter or in an introduction to part. This query finds words in an element and one of its siblings.

- Operands: "identifying" "problems"
- Functionality: word queries, character wildcard (suffix), unordered proximity query (0 to 3 intervening words)
- Context: books/book/content/part//introduction|chapter
- Return: books/book/@number, books/book/content/part//introduction|chapter
- Comments: The query could return the two siblings which satisfy the full-text predicates in any order.

No sibling axes in XQuery.

Q13.8 – Query on Same Element in Different Sub-Trees

Find all books with word "identify" in book introductions and part introductions. This query finds a word in instances of the same element in different sub-trees.

- Operands: "identify"
- Functionality: word query, character wildcard (suffix)
- Context: books/book/content/introduction/p, books/book/content/part/introduction/p

– Return: books/book/@number, books/book/content/introduction, books/book/content/part/ introduction

– Comments: This query looks for a word in multiple instances of the introduction element which appear as a child content or part elements.

```
for $book in //books
let $bi := $book/content/introduction[./p ftcontains "identify"]
let $pi := $book/content/part/introduction[./p ftcontains
                                           "identify"]
where fn:count($bi)>0 and fn:count($pi)>0
return <book number="{$book/@number}"> {$bi, $pi} </book>
```

Q13.9 – Query on Different Elements in Different Sub-Trees

Find all books with a title containing the word "usability", with a book introduction containing the word "satisfaction", and with a part introduction containing the phrase "identify problems". This query finds a word in one element, then finds words and phrases in different descendants of that element.

– Operands: "usability" "satisfaction" "identify problems"

– Functionality: word queries, phrase query

– Context: books/book/metadata/title, books/book/content//introduction, books/book/content/ part//introduction

– Return: books/book/@number, books/book/metadata/title, books/book/content/introduction, books/book/content/part/introduction

– Comments: This query looks for phrases and words in specific descendants of an element that might appear anywhere under the element.

```
for $book in //books[./metadata/title ftcontains "usability"]
let $bi := $book/content//introduction[./p ftcontains
                                       "satisfaction"]
let $pi := $book/content/part//introduction[. ftcontains
                                            "identify problems"]
where fn:count($bi)>0 and fn:count($pi)>0
return <book number="{$book/@number}">
         {$book/metadata/title, $bi, $pi}
       </book>
```

Q13.10 –Conditional Query on Different Elements in Different Subtrees

Find paragraphs in books on "usability testing" with remarks by Millicent Marigold. This query finds a phrase in one element. If found then finds words in another element in a different sub-tree, using a thesaurus to return synonyms.

– Operands: "usability testing" "remarks" "Marigold"

– Functionality: phrase query, conditional logic, word queries, stemming, unordered proximity query (0 to 3 intervening words between "remarks" and "Marigold")

– Context: books//subject, books//content

– Return: books/book/metadata/title, books//content

– Comments: This query looks for phrases and words in any descendant of an element that must contain a specific phrase.

```
for $book in //books[.//subject ftcontains “usability testing”]
let $cont := $book//content[. ftcontains(“remarks” && “Marigold”)
                            with word distance at most 3
                            with stems]
where fn:count($cont)>0
return <result> {$b/metadata/title, $cont} </result>
```

Q13.11 – Query on Element and Its Descendants, Returning Ancestors and Descendants

Find component, subcomponent, and subsubcomponent elements and their descendants which contain information about Usabilityguy's research on human computer interactions. Return the corresponding physical container numbers and componentTitles of each component, subcomponent, or subsubcomponent. This query finds a phrase in an element, then moves up the sub-tree to locate another element, returns that element and one of its descendants.

– Operands: "human computer interaction research"

– Functionality: phrase query

– Context: books//component

– Return: books/book/@number, books//container, books//componentTitle

– Comments: This query exercises the possibility of returning any ancestor and descendant of qualified elements (i.e., those containing the phrase).

No ancestor axis in XQuery

Use Case 14: “IGNORE” – Queries Through Tags and Content

Q14.1 – Query Ignoring Tags within Word

Find part introductions containing the word "prototypes" ignoring bold markup. This query crosses element boundaries, ignoring the tags of the element b (bold highlighting).

– Operands: "prototypes"

– Functionality: word query, ignore tags of the element b

– Context: books/book/content/part/introduction

– Return: books/book/@number, books/book/content/part/introduction

– Comments: This query should be able to cross any number of bold highlightings in the word.

```
for $book in //books
let $intro := $book/content/part/introduction[. ftcontains
                                               “prototypes”
                                               without tags “b”]
```

```
where fn:count($intro)>0
return <result> {$book/metadata/title, $intro} </result>
```

Q14.2 – Query Ignoring Tags within Multiple Words

Find part introductions containing the word "prototypes" and the word "wireframes" ignoring bold markup. This query crosses element boundaries, ignoring the tags of the element b (bold highlighting).

– Operands: "prototypes" "wireframes"

– Functionality: word query, `or` query, ignore tags of the element b

– Context: books/book/content/part/introduction

– Return: books/book/@number, books/book/content/part/introduction

– Comments: Similar queries could ignore different tags to match different words.

```
for $book in //books
let $intro := $book/content/part/introduction[. ftcontains
                                              (“prototypes” ||
                                               “wireframes”)
                                              without tags .//b]
where fn:count($intro)>0
return <result> {$book/metadata/title, $intro} </result>
```

Q14.3 – Phrase Query Ignoring All Tags of Descendant Elements

Find book chapters containing the phrase "lists of heuristics is Ten Usability" ignoring any intervening tags. This query crosses element boundaries, ignoring tags.

– Find book chapters containing the phrase "lists of heuristics is Ten Usability" ignoring any intervening tags. This query crosses element boundaries, ignoring tags. Operands: "lists of heuristics is Ten Usability"

– Functionality: phrase query, ignore tags of any element

– Context: books//chapter

– Return: books/book/metadata/title, books//chapter

– Comments: This query is equivalent to using the "string" function in XQuery that converts the whole sub-tree under an element into a string by removing all markup.

```
for $book in //book
let $chap := $book//chapter[. ftcontains “lists of heuristics is
Ten Usability” without tags .//element()]
where fn:count($chap) > 0
return <result> {$book/metadata/title, $chap} </result>
```

Q14.4 – Phrase Query Ignoring Explicit List of Tags of Descendant Elements

Find book chapters containing the phrase "lists of heuristics is Ten Usability" ignoring the tags of the element citation. This query must cross element boundaries, ignoring the tags of an element.

- Operands: "lists of heuristics is Ten Usability"
- Functionality: phrase query, ignore tags of the citation element
- Context: books//chapter
- Return: books/book/metadata/title, books//chapter
- Comments: The query could ignore any number of tags.

```
for $book in //book
let $chap := $b//chapter[. ftcontains “lists of heuristics is Ten
Usability” without tags .//citation]
where fn:count($chap) > 0
return <result> {$book/metadata/title, $chap} </result>
```

Q14.5 – Phrase Query Ignoring All Tags and Content of Descendant Elements

Find book chapters containing "users can be tested at any computer workstation or in a lab" ignoring any intervening tags and content. This query ignores intervening tags and content of all descendant elements.

- Operands: "users can be tested at any computer workstation or in a lab"
- Functionality: phrase query, ignore content of an element
- Context: books//chapter
- Return: books/book/metadata/title, books//chapter
- Comments: This query goes beyond the character string functions in XQuery by ignoring not only the tags but also the content of all descendant elements.

```
for $book in //book
let $chap := $book//chapter[. ftcontains "users can be tested at
any computer workstation or in a lab" without content
.//element()]
where fn:count($c) > 0
return <result> {$book/metadata/title, $chap} </result>
```

Q14.6 – Phrase Query Ignoring Explicit List of Tags and Content of Descendant Elements

Find book chapters containing "own workstation" ignoring element footnote. This query ignores intervening tags and content of an element and queries a phrase inside the ignored tags and content of the descendant.

- Operands: "own workstation"
- Functionality: phrase query, ignore content of element footnote, query inside ignored element footnote
- Context: books//chapter
- Return: books/book/metadata/title, books//chapter
- Comments: The query could specify any number of tags and content to ignore.

```
for $book in //book
let $chap := $book//chapter[. ftcontains “own workstation”
                            without content .//footnote]
where fn:count($chap) > 0
return <result> {$book/metadata/title, $chap} </result>
```

Q14.7 – Phrase Query Ignoring Tags and Content of Descendant Elements Identified by a Full-Text Query

Find book chapters containing the phrase "at any computer workstation or in a lab" ignoring footnotes on workstations. This query crosses element boundaries, ignoring the tags and content of a descendant element only if that element contains a word.

- Operands: "at any computer workstation or in a lab"
- Functionality: phrase query, character wildcard (suffix), ignore tag and content specified with a word query
- Context: books//chapter/p, books//chapter/p/footnote
- Return: books/book/metadata/title, books//chapter
- Comments: This query returns chapters only if they contain the specified phrase. In order to match the phrase, footnote identified using a word query on workstations must be ignored. If the footnote that must be ignored does not contain the word "workstation", it cannot be ignored to match the phrase and the query would return an empty result.

```
for $book in //book
let $chap := $b//chapter[.//p ftcontains “at any computer
workstation or in a lab” without content (.//footnote[. contains
“workstation”])]
where fn:count($chap) > 0
return <result> {$book/metadata/title, $chap} </result>
```

Q14.8 – Phrase Query Ignoring Tags and Content of Descendant Elements Identified by an XPath Query

Find book chapters containing the phrase "two testers, one to ask questions, another to take notes" ignoring footnotes on testing procedures. This query must cross element boundaries, ignoring the tags of a descendant element and the content of this element only if this element refers to testing procedures.

- Operands: "two testers one to ask questions another to take notes"
- Functionality: phrase queries, ignore tag and content specified with an XPath query
- Context: books//chapter/p, books//chapter/p/footnote/testingProcedure
- Return: books/book/metadata/title, books//chapter
- Comments: This query returns chapters only if they contain the phrase. In order to match the phrase, footnote elements identified using an XPath query must be ignored.

```
for $book in //book
let $chap := $book//chapter[.//p ftcontains “two testers one to
ask questions another to take notes” without content
.//footnote/testingProcedure]
```

```
where fn:count($chap) > 0
return <result> {$book/metadata/title, $chap} </result>
```

Q14.9 – Proximity Query Ignoring All Tags of Descendant Elements

Find book chapters on usability tests and contracting with computer professionals where all tags of descendant elements are ignored. This query crosses element boundaries, ignoring the tags, but not the content of descendants.

- Operands: "usability tests" "computer" "professionals"
- Functionality: phrase query, word queries, ordered proximity query (0 to 20 intervening words), ignore tags of footnote element
- Context: books//chapter, ignore books/book//chapter/title, books/book//chapter/p, books/book//chapter//footnote
- Return: books/book//title, books//chapter
- Comments: If this query had ignored the content of footnote elements, the words "computer" and "professionals" would not have matched.

```
for $book in //book
let $chap := $book//chapter[.//p ftcontains ((("usability tests"
&& "computer" && "professionals") in this order) within window
atmost 24) without tags .//footnote]
where fn:count($c) > 0
return <result> {$b/metadata/title,$c} </result>
```

Q14.10 Proximity Query Ignoring All Tags and Content of Descendant Elements

Find advice on whether an observer in a usability test should correct or help the user. Return the chapter containing the found words and titles for the book and the chapter. This query ignores intervening tags and content of a descendant element.

- Operands: "usability testing" "user" "help"
- Functionality: phrase query, character wildcard (suffix), word queries, unordered proximity query (0 to 3 intervening words between "user" and "help"), `and` query, stemming
- Context: books//chapter, ignore books/book/content/part/chapter/p/footnote
- Return: books/book/metadata/title, books//chapter
- Comments: This query can ignore any number of footnotes. In particular, if footnotes are nested in each other.

```
for $book in //book
let $chap := $book//chapter[.//p ftcontains
                             ("usability testing" &&
                               (("user" && "help")
                                 word distance atmost 3)
                              with stems
                              with thesaurus "Synonyms"
                              without tags .//footnote]
```

```
where fn:count($chap) > 0
return <result> {$book/metadata/title, $chap} </result>
```

Use Case 15: "COMPOSABILITY": Queries Illustrating Composability of Full-Text and other XQuery Functionality

Q15.1 – Query Combining Full-Text with Creation of New Elements

Create a flat list of all the title-author pairs, for books on usability, with each pair enclosed in a result element. This query finds a word in an element and returns it and a different element wrapper in a new element.

- Operands: "usability"
- Functionality: word query, construction of new elements
- Context: books/book/metadata/title
- Return: books/book/metadata/title, books/book//author, results (constructed element)
- Comments: This query requires looking for the word "usability" in the title of a book and building title-author pairs for those books.

```
for $book in //books[./metadata/title ftcontains "usability"]
return <result> {$book/metadata/title, $book//author} </result>
```

Q15.2 – Query Combining Full-Text with Aggregate on Number of Elements

Find all books with a chapter title on usability tests. Return it and the number of steps in chapters. This query finds a set of words in an element using stemming and returns the number of specified elements.

- Operands: "usability" "test"
- Functionality: word queries, stemming, fn:count()
- Context: books/book/content/part/chapter/title
- Return: books/book/metadata/title, books/book/content/part/chapter/title, books/book/content/part/chapter//step, number-of-steps (constructed element) which satisfy the query
- Comments: This query requires looking for the word "usability" and stemmed forms of the word "test" in the title of book chapters and returning chapter titles along with their number of steps. The query returns a number of steps equal to 0 if the chapter is on usability testing but does not specify any steps.

```
for $book in //book
let $ct := $book/content/part/chapter/title[. ftcontains
                                          ("usability" && "text")
                                           with stems]
where fn:count($ct)>0
return <result> {$book/metadata/title}
                {for $title in $ct
                 return {$title}
                        <number-of-steps>
```

```
                              {count($title/..//step)}
                        </number-of-steps>}
       </result>
```

Q15.3 – Query Combining Full-Text with Conditional Return

For each book with "usability" in the book title, return the book title and authors, if it has authors, or return the book title and publishers, if it has no authors. This query finds a word in an element and contains a conditional return.

- Operands: "usability"
- Functionality: word query, conditional query
- Context: books/book/metadata/title
- Return: books/book/metadata/title, books/book/metadata//author, books/book/metadata//publisher
- Comments: This query requires looking for the word "usability" in the title of a book and returning title-author pairs for those books when there are authors. If there are no authors, it returns publishers.

```
for $book in //book[./metadata/title ftcontains "usability"]
return <result>
         {$book/metadata/title
          if fn:count($book/metadata//author)>0
          then $book/metadata//author
          else $book/metadata//publisher}
       </result>
```

Q15.4 – Query Combining Full-Text with Functions on Numerics

For each book with "usability" in the book title, return its book title and the round number of its suggested price if the price exceeds $25. This query finds a word in an element and contains a conditional return based on a function on a numeric value.

- Operands: "usability"
- Functionality: word query, fn:round(), numeric value comparison
- Context: books/book/metadata/title, books/book/metadata/price
- Return: books/book/metadata/title, books/book/metadata/price
- Comments: This query finds the word "usability" in the title of a book and returns the round number of its price if the price exceeds a specified value.

```
for $book in //book[./metadata/title ftcontains "usability"]
return <result>
         {$book/metadata/title
         if $book/metadata/price > 25
         then <price>fn:round($book/metadata/price}</price>
       </result>
```

Q15.5 – Query Combining Full-Text with Query on Character String

Find introductions in books, which were published in New York, which include listings for résumés, drafts, and correspondence. This query finds words in an element and combines the result with a character string query on a different element.

- Operands: "résumés" "drafts" "correspondence" character string "New York"
- Functionality: word query, `and` query, fn:contains
- Context: books//introduction, books//publicationInfo/place
- Return: books/book/metadata/title, books//introduction
- Comments: This query combines full-text and character string queries. It uses "fn:contains", but other character string functions could have been used.

```
for $book in //book[.//publicationInfo/place/text() = ″New York″]
let $intro := $book//introduction[. ftcontains
                                   (“résumés″ && “drafts″ &&
                                    “correspondence″)]
where fn:count($intro)>0
return <result> {$book/metadata/title, $intro} </result>
```

Q15.6 – Query Combining Full-Text with Operators on Booleans

For each book on "usability", return its book title and a new element called has-publisher with value `true` if the book has publishers, return its title and a new element called has-publisher with value `false` if the book does not have publishers. This query finds a word in an element and return Boolean values.

- Operands: "usability"
- Functionality: word query, construction of new Boolean values, construction of new element, op:boolean-equal
- Context: books/book//title
- Return: books/book//title, has-publisher (constructed element)
- Comments: This query requires looking for the word "usability" in the title of a book and returning Boolean values showing whether the book has a publisher or not.

```
for $book in //book[./metadata/title ftcontains “usability″]
return <result>
         {$book/metadata/title}
         <has-publisher>
          {if fn:count($book//publisher)>0 then “true″ else
“false″}
         </has-publisher>
       </result>
```

Q15.7 – Query Combining Full-Text with Queries on Nodes and Dates

Find books about conducting usability tests which have more than one author and are published after 2000. This query finds a phrase in multiple elements, counts the number of instances of another element, runs a greater than comparison on a date element, and combines the results.

- Operands: "usability testing", "2000"
- Functionality: phrase query, character wildcard (suffix), fn:count, op:numeric-greater-than, op:date-greater-than
- Context: books//subject, books//author, books//publicationInfo/dateIssued|dateRevised
- Return: books/book/metadata/title, books//author, books//subject which satisfy query
- Comments: This is a full-text query on the phrase "usability testing" using a wildcard on the word "test", restricted to the subjects element. The query also counts the number of author elements, runs a greater than comparison on dates, and combines the results.

```
for $book in //book[
         (op:date-greater-than($book//publicationInfo/dateIssued,
                               “2000-12-31”) or
         op:date-greater-than($book//publicationInfo/dateRevised,
                               “2000-12-31”)
         ) and fn:count($book//author)>1]
let $subj := $book//subject[. ftcontains “usability test.*”]
where fn:count($subj)>0
return <result> {$book/metadata/title, $book//author, $subj}
       </result>
```

Use Case 16: “COMPLEX” – Complex Queries

Q16.1 – Query Entered in More than One Language

Find all books with the subject "□□□□□", with an introduction written by Elina Rose, which mention the name of the usability expert, Millicent Marigold. This query finds a phrase and words entered in more than one language.

- Operands: "□□□□□" "Elina" "Rose" "Millicent" "Marigold"
- Functionality: phrase query, word queries, unordered proximity queries (0 to 3 intervening words), capacity to enter, query, and return multiple languages
- Context: books//subject, books/book/content/introduction/author, books//content
- Return: books/book/metadata/title, books//subject, books/book/content/introduction/author, books//content
- Comments: This query accepts multiple languages as input and returns multiple languages as output.

```
for $book in //book
let $subj := $book//subject[. ftcontains "□□□□□"
```

```
                              language “Chinese”]
let $iauthor := $book/content/introduction/author[. ftcontains
                                                  “Elina Rose”]
let $content := $book//content[. ftcontains
                               “Millicent” && “Marigold”]
return <result>
         {$book/metdata/title, $subj, $iauthor, $content}
       </result>
```

Q16.2 – Query Combining Proximity, Phrase, and Stemmed Queries within an Instance of an Element

Find all paragraphs in books which discuss the role of task performance in expert review methods. This query finds multiple words close to each other together with a phrase in one instance of a paragraph.

– Operands: "task" "performance" "expert review"

– Functionality: word queries, stemming, phrase query, unordered proximity (0 to 3 intervening words between "task" and "performance"), all within one instance of an p element.

– Context: books/book/content//p

– Return: books/book/metadata/title, books/book/content//p

– Comments: This query combines functionalities introduced in previous sections.

```
for $book in //book
let $para := $book/content//p[. ftcontains
                              ((“task” && “performance”)
                                 with word distance at most 3) &&
                               “expert review”)
                              with stems
where fn:count($para)>0
return <result> {$book/metadata/title, $para} </result>
```

Q16.3 – Nested Proximity Query with Wildcards, Thesaurus Support, and Stemming

Find all books on usability testing which quote Millicent Marigold on the value of multiple iterations. This query finds multiple words, allowing a specified number of intervening words in any order, then finds those words and a phrase, allowing a specified number of words between the words and the phrase in any order. It uses wildcards, thesaurus support, and stemming.

– Operands: "usability testing" "quote" "millicent" "marigold" "iterations"

– Functionality: phrase query, character wildcard (suffix), word queries, thesaurus support, stemming, ordered proximity (0 to 3 intervening words between "millicent" and "marigold"), unordered proximity (0 to 3 intervening words between ("millicent ... marigold" and ("quote" or said or says or stated or states or statement or spoke or speaks or replied or replies or reply or remarks or remarked or responded or response or reports or reported or quote or quoted or according or commented or discussed or expressed or told), unordered proximity (0 to 30 intervening words between ("millicent ... marigold" ... ("quote" or said or says or stated or states or statement or spoke or speaks or replied or replies or reply or remarks or remarked or responded or response or reports or reported or quote or quoted or according or commented or discussed or expressed or told) and "usability testing" and "iterations")

- Context: books//content
- Return: books/book/metadata/title, books//content
- Comments: This query combines functionalities introduced in previous sections.

```
for $book in //book
let $chap := $book//content[
      . ftcontains ((((“millicent" && “marigold”) ordered
                      with word distance at most 3)
                     && (“quote” with stems
                                 with thesaurus “Synonyms”)
                    window 6)
                    && “usability testing”
                    && “iterations”) window 36]
where fn:count($chap)>0
return <result> {$book/metadata/title, $chap} </result>
```

Q16.4 – Query Combining Proximity, Boolean, and Stemming Queries which Ignores Tags and Content of a Descendant Element

Find advice on whether an observer in a usability test should correct or help the user in a book co-authored by Montana Marigold. This query finds words in close proximity to each other, those words in close proximity to one of four other words, finds both of those near a phrase, ignoring the tags and content of a descendant element.

- Operands: "usability testing" "correct" "guidance" "assistance" "help" "montana" "marigold"
- Functionality: phrase query, word queries, character wildcard (suffix), `or` query ("correct" "guidance" "assistance" "help"), unordered proximity query (0 to 10 intervening words between "usability testing" and ("correct" or "guidance" or "assistance" or "help"), proximity query (0 to 3 intervening words between "montana" and "marigold"), proximity query (0 to 50 intervening words between the results of those proximity queries. Ignores interveing tags and content.
- Context: books//subject, books//content, books//author
- Return: books/book/metadata/title, books//author, books//content
- Comments: This query combines functionalities introduced in previous sections.

```
for $book in //book
let $res := ($book//author $book//content $book//subject)
   [. ftcontains (((“montana” && “marigold”)window at most 5) &&
                 ((“usability testing” &&
                   (“correct” || “guidance” || “assistance” ||
                     “help”))
                   window 13)
                  ) window at most 55
                  without content.//element()]
where fn:count($res) > 0
return <result> {$book/metadata/title
                 $res//author
                 $res//content}
       </result>
```

Q16.5 – Query on Different Elements in Different Sub-Trees with a Conditional Return

For each book with a title containing the word "usability", with a book introduction containing the word "satisfaction", and a part introduction containing the phrase "identify problems", return the book title and the authors if it has authors, or return the book title and publisher, if it has no authors, then return the content surrounding the found words. This query finds words and a phrase in different elements, then finds a descendant of the first element and it contains a conditional return.

– Operands: "usability" "satisfaction" "identify problems"

– Functionality: word queries, phrase query, conditional query

– Context: books/book/metadata/title, books/book/content//introduction, books/book/content/part//introduction

– Return: books/book/metadata/title, books/book/metadata//author, books/book/metadata//publisher, books/book/content//introduction, books/book/content/part//introduction

– Comments: This query combines functionalities introduced in previous sections.

```
for $book in //book
let $title := $book/metadata/title[. ftcontains “usability”]
let $i := $book/content/introduction[. ftcontains “satisfaction”]
let $pin := $book/content/part//introduction[. ftcontains
                                            “identify problem”]
where fn:count($title)>0 and fn:count($in)>0 and fn:count($pin)>0
return <result> {$title
        (if fn:count($book/metadata//author)>0 then
          $book/metadata//author
        else $book/metadata//publisher)
        $i, $pin
      }
      </result>
```

Q16.6 – Query Combining Full-Text with Character String, Node, and Date Queries

Find introductions in books, which were published in New York after 2000 and have more than one author, which include listings for résumés, drafts, and correspondence. This query finds words in an element and combines the result with a character string query on a different element, counts the number of another element, runs a greater than comparison on a date element, and combines the results.

- Operands: "résumés", "drafts", "correspondence", character string "New York", "2000"
- Functionality: word query, `and` query, fn:contains, fn:count, op:numeric-greater-than, op:date-greater-than
- Context: books//introduction, books//publicationInfo/place, books/book//author, books//publicationInfo/dateIssued|dateRevised
- Return: books/book/metadata/title, books/book//author, books//introduction
- Comments: This query combines full-text and character string queries, node and date queries.

```
for $book in //book[
               (op:date-greater-than(publicationInfo/dateIssued,
                                     "2000-12-31") or
               (op:date-greater-than(publicationInfo/dateRevised,
                                     "2000-12-31"))
               and (publicationInfo/place eq "New York")
            and (fn:count(.//author)>1)]
let $i := $book//introduction[. ftcontains
                              ("résumés" && "drafts" &&
                               "correspondence")]
where fn:count($i)>0
return <result> {$book/metadata/title,$book//author, $i}
       </result>
```

2. References

[1] XQuery Language. The W3 Consortium. http://www.w3.org/TR/xquery/

[2] XQuery 1.0 and XPath 2.0 Data Model. The W3 Consortium. http://www.w3.org/TR/xpath-datamodel/

[3] XQuery and XPath Full-Text Use Cases. The W3 Consortium. http://www.w3.org/TR/xmlquery-full-text-use-cases/