Topic: Scope Search Investigation

Author: Darin McBeath

Date: July 10, 2014

As part of the Scopus Hothouse, there was a need to find a solution for scope searching. ScienceDirect has basic one-level scope searching requirements (such as searching for a given-name or surname within a specific author) for which the team was able to develop a solution. However, Scopus has more complex scope searching needs (such as within a given reference searching for a given-name or surname for a specific author). This additional level of sub scoping spawned the investigation into possible solutions. The initial approach centered on creating a custom query plug-in (leveraging the Lucene SpanNearQuery) but low-level bugs were uncovered with the Lucene SpanNearQuery (when using the SpanNotQuery) that ultimately prevented this as a viable option. After discussions with Search Technologies, it was requested they create a custom jar file (consisting of their implementation of SpanAnd and SpanBetween based on Lucene Spans) that was subsequently leveraged in the custom query plug-in (instead of the SpanNearQuery approach). The remainder of this memo highlights the general approach and provides results from some basic testing.

**Environment**

Solr/Lucene 4.7.2

between-operators-1.0-snapshot.jar (provided by Search Technologies)

**Content Preparation**

The content needs to be ‘massaged’ prior to indexing to enable scope searching. The ‘massaging’ essentially replaces the ‘begin’ and ‘end’ XML tags with a begin/end token that will be searchable. These begin/end tokens are upper-cased so that a user can’t inadvertently search for these tokens (and the tokens we want the user to search are lower-cased). This is currently accomplished with the following charFilters (specified in the schema.xml) for the ‘author’ scope field.

<fieldType name="authorScope" class="solr.TextField">

<analyzer>

<charFilter class="solr.MappingCharFilterFactory" mapping="mapping-uppertolowercase.txt"/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/? \*[a-zA-Z]\*:?(bold|sup|inf|hsp|vsp) \*>" replacement=""/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?authors \*>" replacement=" BAUTHORS "/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?authors \*>" replacement=" EAUTHORS "/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?author \*>" replacement=" BAUTHOR "/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?author \*>" replacement=" EAUTHOR "/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?surname \*>" replacement=" BLNAME "/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?surname \*>" replacement=" ELNAME "/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?given-name \*>" replacement=" BFNAME "/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?given-name \*>" replacement=" EFNAME "/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?email \*>" replacement=" BEMAIL "/>

<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?email \*>" replacement=" EEMAIL "/>

<charFilter class="solr.HTMLStripCharFilterFactory"/>

<tokenizer class="solr.ClassicTokenizerFactory"/>

<filter class="solr.ClassicFilterFactory"/>

<filter class="solr.ASCIIFoldingFilterFactory"/>

</analyzer>

</fieldType>

Assuming the original XML for the authors field in some sample documents was something like the following,

<ce:authors><ce:author><ce:surname>Jones</ce:surname><ce:given-name>Jeff</ce:given-name></ce:author></ce:authors>

<ce:authors><ce:author><ce:surname>Jeff</ce:surname><ce:given-name>Jones</ce:given-name></ce:author></ce:authors>

<ce:authors><ce:author><ce:surname>McBeath</ce:surname><ce:given-name>Darin William</ce:given-name></ce:author><ce:author><ce:surname>Fulford</ce:surname><ce:given-name>Darby</ce:given-name></ce:author><ce:author><ce:surname>McBeath</ce:surname><ce:given-name>Darby</ce:given-name></ce:author></ce:authors>

<ce:authors><ce:author><ce:surname>McBeath</ce:surname><ce:given-name>Darin</ce:given-name></ce:author><ce:author><ce:surname>Fulford</ce:surname><ce:given-name>Darin</ce:given-name></ce:author></ce:authors>

<ce:authors><ce:author><ce:surname>Johnson-Smith</ce:surname><ce:given-name>Joe</ce:given-name> </ce:author></ce:authors>

the tokens sent to Solr for indexing these sample documents would look something like the following.

BAUTHORS BAUTHOR BLNAME jones ELNAME BFNAME jeff EFNAME EAUTHOR EAUTHORS

BAUTHORS BAUTHOR BLNAME jeff ELNAME BFNAME jones EFNAME EAUTHOR EAUTHORS

BAUTHORS BAUTHOR BLNAME mcbeath ELNAME BFNAME darin william EFNAME EAUTHOR BAUTHOR BLNAME fulford ELNAME BFNAME darby EFNAME EAUTHOR BAUTHOR BLANME mcbeath ELNAME BFNAME darby ELNAME EAUTHOR EAUTHORS

BAUTHORS BAUTHOR BLNAME mcbeath ELNAME BFNAME darin EFNAME EAUTHOR BAUTHOR BLNAME fulford ELNAME BFNAME darin EFNAME EAUTHOR EAUTHORS

BAUTHORS BAUTHOR BLNAME johnson-smith ELNAME BFNAME joe EFNAME EAUTHOR EAUTHORS

Similar token replacement would occur for other scoped fields such as affiliations and references. There are certainly better (and more efficient) ways of performing this task. The goal of this exercise was simply to explore whether we could meet the feature needs of Scopus scope searching.

**Custom Plugin**

After Solr indexed the content, a custom query plugin was developed to resolve a ‘scope’ query. The nested query syntax for invoking a ‘scope’ query would look like the following:

{!scope field='authscope' query='fname:Darin AND lname:Fulford'}

* !scope identifies our ‘scope’ query plugin.
* Field identifies the scoped filed (in this case ‘authscope’)
* Query identifies the actual query (in Lucene syntax). The default connector is currently an AND. So, ‘fname:Darin lname:Fulford’ would be equivalent. Leading, embedded, and trailing wildcards are supported. Phrases are also supported (with the exception of wildcards with phrases).

The end-user would certainly not be expected to enter a search with the above syntax. Rather, the end-user syntax would be ‘transformed’ into something similar in the above syntax. This is really no different than what would be done for edismax type queries today. The ‘scope’ query could be combined with other query types in the overall query (much like is done today with edismax).

The above query does not restrict fname:Darin AND lname:Fulford to a specific author. So, when this query is executed against the 5 sample documents mentioned above, 2 documents would be returned. However, if the desire is to restrict this to a specific author, the following query syntax would be used (and only 1 result would be returned). Note the use of ‘scope:author’.

{!scope field='authscope' query='scope:author(fname:Darin AND lname:Fulford)'}

It’s important to note that there can only be one ‘analyzer’ for the scoped field (such as authscope) and all of its subfields (such as fname, lname). This is because the ‘authscope’ field is indexed as one field (including the logical subscope fields (such as fname, lname). So, to be clear, you can’t have different tokenizers, etc. for the scope field and it’s subscope fields.

Here are some other types of queries that would be supported (with the number of results returned based on the above sample data).

This query searches within an author element where the given-name contains ‘Darin’ or ‘Darby’ and the surname contains ‘McBeath’. Two results will be returned for this query.

{!scope field='authscope' query='scope:author(fname:(Darin OR Darby) AND lname:McBeath)'}

This query searches within an author element where the given-name contains the prefix ‘dar’ and the surname contains ‘mcbeath’. Two results will be returned for this query.

{!scope field='authscope' query='scope:author(fname:dar\* AND lname:mcbeath)'}

This query searches within an author element where the given-name contains the wildcard expression ‘da?in’ and the surname contains ‘Fulford. One result will be returned for this query.

{!scope field='authscope' query='scope:author(fname:da?in AND lname:fulford)'}

This query searches within an author element where the given-name contains the phrase “darin william” and the surname contains ‘mcbeath’. One result will be returned for this query.

{!scope field='authscope' query='scope:author(fname:"darin william" AND lname:mcbeath) '}

But, the above queries are basic single-level scope queries. To prove that additional sub-scoping can be supported, the references field was investigated. The following analyzer (with charFilters) was specified in the schema.xml for the ‘reference’ scope field.

<fieldType name="referenceScope" class="solr.TextField">  
<analyzer>  
<charFilter class="solr.MappingCharFilterFactory" mapping="mapping-uppertolowercase.txt"/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/? \*[a-zA-Z]\*:?(bold|sup|inf|hsp|vsp) \*>" replacement=""/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?references \*>" replacement=" BREFERENCES "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?references \*>" replacement=" EREFERENCES "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?reference \*>" replacement=" BREFERENCE "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?reference \*>" replacement=" EREFERENCE "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?eid \*>" replacement=" BEID "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?eid \*>" replacement=" EEID "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?ref-authors \*>" replacement=" BAUTHORS "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?ref-authors \*>" replacement=" EAUTHORS "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?author \*>" replacement=" BAUTHOR "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?author \*>" replacement=" EAUTHOR "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?surname \*>" replacement=" BLNAME "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?surname \*>" replacement=" ELNAME "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?given-name \*>" replacement=" BFNAME "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?given-name \*>" replacement=" EFNAME "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?titletext \*>" replacement=" BTITLE "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?titletext \*>" replacement=" ETITLE "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?lang \*>" replacement=" BLANG "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?lang \*>" replacement=" ELANG "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?srctitle \*>" replacement=" BSRCTITLE "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?srctitle \*>" replacement=" ESRCTITLE "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?pubyr \*>" replacement=" BPUBYR "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?pubyr \*>" replacement=" EPUBYR "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?pubyrtxt \*>" replacement=" BPUBYRTXT "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?pubyrtxt \*>" replacement=" EPUBYRTXT "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?pg \*>" replacement=" BPG "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?pg \*>" replacement=" EPG "/>   
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?pgfirst \*>" replacement=" BPGFIRST "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?pgfirst \*>" replacement=" EPGFIRST "/>   
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?artnum \*>" replacement=" BARTNUM "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?artnum \*>" replacement=" EARTNUM "/>   
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?scp \*>" replacement=" BSCP "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?scp \*>" replacement=" ESCP "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt; \*[a-zA-Z]\*:?authid \*>" replacement=" BAUTHID "/>  
<charFilter class="solr.PatternReplaceCharFilterFactory" pattern="&lt;/ \*[a-zA-Z]\*:?authid \*>" replacement=" EAUTHID "/>   
<charFilter class="solr.HTMLStripCharFilterFactory"/>  
<tokenizer class="solr.ClassicTokenizerFactory"/>  
<filter class="solr.ClassicFilterFactory"/>  
<filter class="solr.ASCIIFoldingFilterFactory"/>  
</analyzer>  
</fieldType>

The following sample document (references field) was then indexed by Solr.

<ce:references>  
 <ce:reference>  
 <ce:titletext>Investigation into Dementia</ce:titletext>  
 <ce:srctitle>Brain Research</ce:srctitle>  
 <ce:pubyr>2012</ce:pubyr>  
 <ce:ref-authors>  
 <ce:author>  
 <ce:given-name>Darin</ce:given-name>  
 <ce:surname>McBeath</ce:surname>  
 </ce:author>  
 <ce:author>  
 <ce:given-name>Darin</ce:given-name>  
 <ce:surname>Fulford</ce:surname>  
 </ce:author>  
 <ce:author>  
 <ce:given-name>Darby</ce:given-name>  
 <ce:surname>Fulford</ce:surname>  
 </ce:author>  
 </ce:ref-authors>  
 </ce:reference>  
 <ce:reference>  
 <ce:titletext>Living with Dementia</ce:titletext>  
 <ce:srctitle>Brain Review</ce:srctitle>  
 <ce:pubyr>2011</ce:pubyr>  
 <ce:ref-authors>  
 <ce:author>  
 <ce:given-name>Darin</ce:given-name>  
 <ce:surname>McBeath</ce:surname>  
 </ce:author>  
 <ce:author>  
 <ce:given-name>William</ce:given-name>  
 <ce:surname>McBeath</ce:surname>  
 </ce:author>  
 <ce:author>  
 <ce:given-name>Darby</ce:given-name>  
 <ce:surname>McBeath</ce:surname>  
 </ce:author>  
 </ce:ref-authors>  
 </ce:reference>  
 <ce:reference>  
 <ce:titletext>Case Study</ce:titletext>  
 <ce:srctitle>Lancet</ce:srctitle>  
 <ce:pubyr>2010</ce:pubyr>  
 <ce:ref-authors>  
 <ce:author>  
 <ce:given-name>Darin William</ce:given-name>  
 <ce:surname>McBeath</ce:surname>  
 </ce:author>  
 <ce:author>  
 <ce:given-name>Annie</ce:given-name>  
 <ce:surname>McBeath</ce:surname>  
 </ce:author>  
 <ce:author>  
 <ce:given-name>Kinsey</ce:given-name>  
 <ce:surname>McBeath</ce:surname>  
 </ce:author>  
 </ce:ref-authors>  
 </ce:reference>  
</ce:references>

For brevity, the tokens sent to Solr for reference scope indexing are not included (although it would be very similar to the author scope example above). Here are some queries that would be supported and an indication for whether the above sample document would be returned for the query.

This query searches within a reference element where the given-name contains ‘darin’. One result will be returned for this query.

{!scope field='refscope' query='scope:reference(fname:darin)'}

This query searches within a reference element and within an author element where the given-name contains ‘darin’. One result will be returned for this query.

{!scope field='refscope' query='scope:reference((scope:author(fname:darin)))'}

This query searches within a reference element and within an author element where the given-name contains ‘darin’ and the surname contains ‘fulford’. One result will be returned for this query.

{!scope field='refscope' query='scope:reference((scope:author(fname:darin lname:fulford)))'}

This query searches within a reference element and within an author element where the given-name contains ‘annie’ and the surname contains ‘fulford’. No results will be returned for this query.

{!scope field='refscope' query='scope:reference((scope:author(fname:annie lname:fulford)))'}

This query searches within a reference element and within an author element where the given-name contains ‘darin’ and the surname contains ‘fulford’ and within a pubyr element that contains 2012. One result will be returned for this query.

{!scope field='refscope' query='scope:reference((scope:author(fname:darin lname:fulford)) AND pubyr:2012)'}

This query searches within a reference element and within an author element where the given-name contains ‘darin’ and the surname contains ‘fulford’ and within a pubyr element that contains 2011. No results will be returned for this query.

{!scope field='refscope' query='scope:reference((scope:author(fname:darin lname:fulford)) AND pubyr:2011)'}

This query searches within a reference element and within an author element where the given-name contains ‘darin’ and the surname contains ‘mcbeath’ and within an author element where the given-name contains ‘darby’ and the surname contains ‘mcbeath’. One result will be returned for this query.

{!scope field='refscope' query='scope:reference((scope:author(fname:darin lname:mcbeath)) AND (scope:author(fname:darby lname:mcbeath)))'}

This query searches within a reference element and within an author element where the given-name contains ‘darin’ and the surname contains ‘mcbeath’ or within an author element where the given-name contains ‘darby’ and the surname contains ‘mcbeath’. One result will be returned for this query.

{!scope field='refscope' query='scope:reference((scope:author(fname:darin lname:mcbeath)) OR (scope:author(fname:darby lname:mcbeath)))'}

This query searches within a reference element and within an author element where the given-name contains ‘darin’ and the surname contains ‘mcbeath’ and within an author element where the given-name contains ‘darby’ and the surname contains ‘mcbeath’ and within a srctitle element that contains the phrase ‘brain review’ and within a pubyr element that contains 2011. One result will be returned for this query.

{!scope field='refscope' query='scope:reference((scope:author(fname:darin lname:mcbeath)) AND (scope:author(fname:darby lname:mcbeath)) AND srctitle:"brain review" AND pubyr :2011)'}

**Conclusions**

I’m confident that the SearchTechnologies SpanBetween and SpanAnd features could be leveraged to meet the scope searching needs for Scopus. While the SpanNot is currently not implemented (it could be), consideration should be given as to whether this is even necessary for scope searches in Scopus.

The current implementation of the search plugin ‘works’ but it is certainly not optimal. Once again, the goal was to prove that we could meet the scope feature requirements for Scopus. There could be advantages to leveraging the Search Technologies query parser specifically for the scope query. Currently, the Lucene query parser is leveraged and the lower level SpanBetween/SpanAnd classes are incorporated. This leads to some sub-optimal approaches that were taken to make things work. There are also some assumptions/coupling between the schema.xml and the search plugin. Some other potential limitations could potentially also be eliminated.

The approach/solution is dependent on licensing the jar file from Search Technologies. To be clear, this is not QPL. This is only a subset of classes that have been developed by Search Technologies. However, it could be advantageous to leverage a few more capabilities that could be provided by Search Technologies.

The code, etc. will be made available on github should the implementation team wish to further investigate/pursue this approach.