SchemaDesign

General tips & tricks in designing schemas.

Mapping databases to Solr

Solr provides one table. Storing a set database tables in an index generally requires denormalizing some of the tables. Attempts to avoid denormalizing usually fail.

Field contents

The more heterogeneous (different kinds of data) you have in one field or in one index, the less useful it is. For example, if you have text in different languages, it is more useful to store them in different fields: text_en, text_fr, etc. than all in one field. This way you can search for only English, only French etc.

Sorting

There are two ways of sorting available in Solr 1.4: Lucene's sorting feature and function queries.

Lucene Sorting

The Solr sort parameter uses the Lucene sorting tool. This creates an array containing an entry for every document in the index using the FieldCache. Sorting is then done against this array. This array is cached across requests using the IndexReader and so repeated sorts are fast. If the field type is 'integer' the array contains only that value and thus is 4 bytes * the number of documents. If the field type is anything else, this integer array is created and then a separate array is also created with that field's data per entry.

A Note on "sortable" FieldTypes

Sortable FieldTypes like sint, sdouble are a bit of a misnomer. They are not needed for Sorting in the sense described above, but are needed when doing RangeQuery queries. Sortables, in fact, refer to the notion of making the number sort correctly lexicographically as Strings. That is, if this is not done, the numbers 1..10 sort lexicographically as 1,10, 2, 3... Using an sint, however remedies this. If, however, you don't need to do RangeQuery queries and only need to sort on the field, then just use an int or double or the equivalent appropriate class. You will save yourself time and memory.

Function Query Sorting

Add this clause to your query string to sort the results using 'myIndexedField'. Do not use the 'sort=field+asc' parameter. See [FunctionQuery] for more.

```
_val_:"ord(myIndexedField)"
```

There may be performance differences with this technique v.s. the Lucene sorting algorithm.

Multiple Text Search Field types

The "text" field type in the example schema.xml provides basic text search for English text. But, it has a surprise: the actual text given to this field is not indexed as-is, and therefore searching for the raw text may not work. If you store "To Be Or Not To Be" in a "text" field, none of these words will found this document, nor will the phrase in quotes. The above words are all *stopwords* and are stripped from the input text. Another transform is *stemming*, which stores both 'change' and 'changing' as the word 'chang'. Stemming is done at both index and query time, so a query of 'changing' will match a document containing 'change'.

Raw text search

If you search this text field for "changing" you will also find "changes". It is also useful to be able to search for individual words, especially when searching phrases. This requires a separate text field without the stemming and stopword filters. This field will store "changing" in one document and "changes" in another document.

Phonemes

Programmers are perfect spellers and expect the same of their users. A *phoneme* represents (roughly) the sound of one syllable. Phoneme-based searching can give users a better search experience. To support misspelled search words phoneme filters cause the index to store phoneme-base representations of the text instead of the input. This only finds misspellings which sound like the original word.

To create a phoneme-based field, you need a text filter stack that does not include stemming or stopwords. You can then add the solr. PhoneticFilterFactory (see AnalyzersTokenizersTokenFilters) with one of the available encoders. This must be in both the indexing and query stack. Of the several available the "Double Metaphone" filter is the most popular and does well with non-English text.

Newly added in Solr 3.6 was the solr.BeiderMorseFilterFactory (see AnalyzersTokenizersTokenFilters) which is optimized for finding surnames that sound alike but may be spelled differently, especially Central European and Eastern European surnames. For example, a search for the surname "Maddow" can also turn up the name "Madoff" and vice versa.

For another take on assisting spelling, see SpellCheckComponent.

Unicode processing

Searching text in different languages is very difficult. The Latin1Accent filters downgrade all European "special characters" down to their US Ascii equivalents: the French spelling *protégé* becomes the English spelling *protege*. In Solr-1.3, use this in the filter stack of your "text" field type:

```
<tokenizer class="solr.WhitespaceTokenizerFactory" />
<filter class="solr.ISOLatin1AccentFilterFactory" />
```

In Solr-1.4, use this:

```
<charFilter class="solr.MappingCharFilterFactory" mapping="mapping-ISOLatin1Accent.txt"/>
<tokenizer class="solr.WhitespaceTokenizerFactory" />
```