Apache Solr

http://heliosearch.org/
http://splainer.io/
http://explain.solr.pl/
http://opensourceconnections.com/blog/2014/08/18/introducing-splainer-the-open-source-search-sandbox-that-tells-you-why/
https://quepid.com/secure/#/
http://www.sitepoint.com/geospatial-search-solr-solarium/
https://www.sitepoint.com/search-engine-node-elasticsearch

https://github.com/leonardofoderaro/
https://github.com/evolvingweb/ajax-solr/wiki/Solr-proxies
https://github.com/adsabs/solr-service
https://github.com/adsabs/adsws
http://ui.adslabs.org/#index/
https://github.com/adsabs/bumblebee

Consul
SolrJ
CloudSolrServer

TIKA

Google Mini Search appliance

https://github.com/agazzarini/SolRDF
http://docs.lucidworks.com/download/attachments/11927696/ApacheSolrRefGuide-v3.4.pdf
https://www.apache.org/dyn/closer.cgi/lucene/solr/ref-guide/
http://www.packtpub.com/apache-solr-beginners-guide/book
http://www.sitepoint.com/using-solarium-solr-search-solarium-gui/
http://curator.apache.org/index.html
http://www.foragejs.net/
https://github.com/cominvent/exactmatch
https://www.apache.org/dyn/closer.cgi/lucene/solr/ref-guide/

http://vimeo.com/43913870
http://heliosearch.org/
http://www.youtube.com/watch?v=eVK0wLkLw9w
http://wiki.apache.org/solr/HierarchicalFaceting

http://www.cubrid.org/?mid=wiki_tutorials&entry=full-index-of-cubrid-database-using-solr-dataimporthandler&l=ko
http://www.scribd.com/doc/210143379/Apache-Solr-Ref-Guide-4-4

Apache UIMA
http://opennlp.apache.org/
https://annomarket.com/
http://www.flax.co.uk/blog/2012/06/12/clade-a-freely-available-open-source-taxonomy-and-autoclassification-tool/
http://gate.ac.uk/download/

SolrCloud
ZooKeeper

ClusterJ
LeaderLatch, Barrier and NodeCache
Overseer and OverseerCollectionProcessor
Nutch
http://fergiemcdowall.github.com/solrstrap/
http://manifoldcf.apache.org/en_US/index.html

http://www.quizmeup.com/quiz/apache-solr-configuration
http://www.datastax.com/dev/blog/dse-solr-backup-restore-and-re-index
http://rockyj.in/2012/05/08/setting_up_solr.html
http://charlesleifer.com/blog/solr-ubuntu-revisited/
http://wiki.apache.org/solr/SolrCloud/

We at Lucidworks are pleased to announce the release of Lucidworks Fusion 1.0. Fusion is built to overlay on top of Solr (in fact, you can manage multiple Solr clusters — think QA, staging and production — all from our Admin). In other words, if you already have Solr, simply point Fusion at your instance and get all kinds of goodies like Banana (https://github.com/LucidWorks/Banana — our port of Kibana to Solr + a number of extensions that Kibana doesn't have), collaborative filtering style recommendations (without the need for Hadoop or Mahout!), a modern signal capture framework, analytics, NLP integration, Boosting/Blocking and other relevance tools, flexible index and query time pipelines as well as a myriad of connectors ranging from Twitter to web crawling to Sharepoint. The best part of all this? It all leverages the infrastructure that you know and love: Solr. Want recommendations? Deploy more Solr. Want log analytics? Deploy more Solr. Want to track important system metrics? Deploy more Solr.

Fusion represents our commitment as a company to continue to contribute a large quantity of enhancements to the core of Solr while complementing and extending those capabilities with value adds that integrate a number of 3rd party (e.g connectors) and home grown capabilities like an all new, responsive UI built in AngularJS. Fusion is not a fork of Solr. We do not hide Solr in any way. In fact, our goal is that your existing applications will work out of the box with Fusion, allowing you to take advantage of new capabilities w/o overhauling your existing application.

We do not want to have to write a separate init script for Solr. If we are already running Tomcat, and if Tomcat already have an init script, we should deploy Solr using Tomcat.

My unanswered questions on Solr
Unread articles
Miscellaneous
Running Apache Solr in a cloud

I was having a problem with using wildcard. It seems that the wildcard does not work when it is at the end of a word. If I search for patient (* is not appended to the search term), it returns results. If I search for patient (* is appended to the search term), it does not return result. To support wild card search:

- Try using EdgeNGrams. Just add the edgytext field type to schema.xml, and change the field type of the field you want to search
- Use EDisMax (ExtendedDismaxQParser). It handle both trailing and leading wildcards.

I was also having a problem with phrase synonym. Try to use the \ to escape spaces: hold\ up, delay

How to escape special characters?

Lucene supports escaping special characters that are part of the query syntax. The current list special characters are + - && || ! ( ) { } [ ] ^ " ~ * ? :
To escape these character use the \ before the character. For example to search for (1+1):2 use the query:

```
\(1\+1\)\:2
```

Why is "Medical" highlighted when user search for "medication"?

This is because of SnowballPorterFilter. If we don't want this to happen, we should remove the SnowballPorterFilter, or see if we can use protwords.txt

What are the differences between string and text?

String is not analyzed. Text is analyzed. (String is not tokenized. Text is tokenized). An indexed string might be useful for faceting, highlighting.

What is the purpose of the dataDir directive in solrconfig.xml?

Used to specify an alternate directory to hold all index data other than the default ./data under the Solr home.

What is fuzzy search?

Fuzzy search is a search for words that are similar in spelling.

Lucene supports fuzzy searches based on the Levenshtein Distance, or Edit Distance algorithm. To do a fuzzy search use the tilde, "~", symbol at the end of a Single word Term. For example to search for a term similar in spelling to "roam" use the fuzzy search:

```
roam~
```

This search will find terms like foam and roams. tarting with Lucene 1.9 an additional (optional) parameter can specify the required similarity. The value is between 0 and 1, with a value closer to 1 only terms with a higher similarity will be matched. For example:

```
roam~0.8
```

The default that is used if the parameter is not given is 0.5.

What is Searcher?

"Searcher" tends to refer to an instance of the SolrIndexSearcher class. This class is responsible for executing all searches done against the index, and manages several caches. There is typically one Searcher per SolrCore at any given time, and that searcher is used to execute all queries against that SolrCore, but there may be additional Searchers open at a time during cache warming (in which and "old Searcher" is still serving live requests while a "new Searcher" is being warmed up).

How to implement partial word?

Lucene fundamentally search on words. Using advance ngram analysis, it can do partial words too.

Does Lucene support wild card search in a phrase?

No. Lucene supports single and multiple character wild card searches within single terms (not within a phrase queries).

How to specify the search query?

```
q=solr+rocks
```

How to query multiple fields?

```
q=myField:Java AND otherField:developerWorks
q=title:"The Right Way" AND text:go
q=title:"The Right Way" AND go (we did not specify the field, the default searc
```

How to search for documents containing Lucene in the title field, and Java in the content field? (the name of the fields are title and content)

```
q=title:Lucene AND content:Java
```

When querying Solr, can the field name be omitted?

Yes. In the following examples:

```
q=solr+rocks
q=title:"The Right Way" AND go
```

the field names are omitted. In the second example, the field name was specify for the first field, but was omitted for the second field. When the field name is omitted, the defaultSearchField (defined in schema.xml) is used.

How to search for a range?

```
q=age:[18 TO 35]
title:{Aida TO Carmen} // This will find all documents whose titles are between
```

Inclusive range queries are denoted by square brackets. Exclusive range queries are denoted by curly brackets.

How to do an open-ended range query?

```
field:[* TO *]
```

What is fq, and how is it different from q?

fq specifies an optional filtering query. The results of the query are restricted to searching only those results returned by the filter query. Filtered queries are cached by Solr. They are very useful for improving speed of complex queries. The value for fq is any valid query that could be passed to the q parameter, not including sort information.

I was working on a application that takes a search term from the user, and the application adds other conditions before sending it to Solr. These other conditions are finite (does not change based on user input). We use fq to handle these conditions. So to take advantage of fq, we need to analyze our query mix, see what parts are used most frequent. For those that are used most frequent, use fq.

"fq" stands for Filter Query. This parameter can be used to specify a query that can be used to restrict the super set of documents that can be returned, without influencing score. It can be very useful for speeding up complex queries since the queries specified with fq are cached independently from the main query. Caching means the same filter is used again for a later query (i.e. there's a cache hit). See SolrCaching to learn about the caches Solr uses. See FilterQueryGuidance for an explanation of how filter queries may be used for increased efficiency.

The fq param can be specified multiple times. Documents will only be included in the result if they are in the intersection of the document sets resulting from each fq. In the example below, only documents which have a popularity greater then 10 and have a section of 0 will match.

```
  fq=popularity:[10 TO *]
& fq=section:0
```

Filter Queries can be complicated boolean queries, so the above example could also be written as a single fq with two mandatory clauses:

```
fq=+popularity:[10 TO *] +section:0
```

The document sets from each filter query are cached independently. Thus, concerning the previous examples: use a single fq containing two mandatory clauses if those clauses appear together often, and use two separate fq params if they are relatively independent.

"fq" is a filter, therefore **does not influence the scores. You must always have a "q" parameter.**

How to specify which fields you want in the result?

```
fl=name,id,score
fl=*,score
```

The set of fields to be returned can be specified as a space (or comma) separated list of field names. The string "score" can be used to indicate that the score of each document for the particular query should be returned as a field, and the string "*" can be used to indicate all stored fields the document has.

How to specify the starting offset into the result set?

```
start=15
```

Useful for paging through results. This returns results starting with fifteenth ranked result.

How to specify the format of the result?

```
wt=json
```

How to specify the number of rows / records / documents to return?

```
rows=10
```

How to specify that you want highlighting enabled?

```
hl=true
```

q.op: specifies the default operator for query expressions. Possible values are "AND" or "OR"

How to specify the default search field?

Use df parameter, which overrides the default search field defined in schema.xml

How to boost a term? How to give a term more weight?

Lucene provides the relevance level of matching documents based on the terms found. To boost a term use the caret, "^", symbol with a boost factor (a number) at the end of the term you are searching. The higher the boost factor, the more relevant the term will be.

Boosting allows you to control the relevance of a document by boosting its term. For example, if you are searching for

```
jakarta apache
jakarta apache
```

and you want the term "jakarta" to be more relevant boost it using the ^ symbol along with the boost factor next to the term. You would type:

```
jakarta^4 apache
```

This will make documents with the term jakarta appear more relevant. You can also boost Phrase Terms as in the example:

```
"jakarta apache"^4 "Apache Lucene"
```

By default, the boost factor is 1. Although the boost factor must be positive, it can be less than 1 (e.g. 0.2)

How to sort the result?

```
sort=price+desc
sort=inStock+asc,price+desc
```

Can I do relevance and sorting together?

- &sort=score asc,date desc,title asc
- Boost functions, or function queries, may also be what you're looking for.
  See FunctionQuery and Boost function (bf) to increase score of documents whose date is closest to NOW

slop is related to phrase searching.

Sorting can be done on the "score" of the document, or on any multiValued="false" indexed="true" field provided that field is non-tokenized (ie: has no Analyzer) or uses an Analyzer that only produces a single Term (ie: uses the KeywordTokenizer)

You can sort by index id using sort=_docid_ asc or sort=_docid_ desc

As of Solr1.5, sorting can also be done by any single-valued function (as in FunctionQuery)

The common situation for sorting on a field that you do want to be tokenized for searching is to use a <copyField> to clone your field. Sort on one, search on the other.

Multiple sort orderings can be separated by a comma:

```
sort=inStock desc, price asc
```

Sorting can also be done on the result of a function:

```
sort=sum(x_td, y_td) desc
```

What is LocalParam?

LocalParam is a way to provide additional information about each query parameter / argument. Assume that we have the existing query parameter:

```
q=solr+rocks
```

We can prefix this query string with {!} to provide more information to the query parser:

```
q={!}solr+rocks
```

The above code does not provide more information to the query parser. The {!} is the syntax for LocalParams.

To provide additional information to the query parser:

```
q={!q.op=AND df=title}solr+rocks
```

The above change the default operator to "AND" and the default search field to "title".

To indicate a LocalParam, the argument is prefixed with curly braces whose contents begin with an exclamation point and include any number of key=value pairs separated by whitespace.

Values in the key-value pairs may be quoted via single or double quotes, and backslash escaping works within quoted strings.

There may only be one LocalParams prefix per argument.

How to specify that the search term should be search in multiple columns?

Use LocalParam qf:

```
q={!type=dismax qf='myfield yourfield'}solr+rocks
```

How to filter for 'not equal'?

Use the - sign:

```
fq=-iMemberId:351
```

Of course, there should be other ways to express 'not equal'.

How to search for words that is spell similarly to a given word?

This is known as Fuzzy search. Fuzzy search is a search for words that are similar in spelling. Lucene supports fuzzy searches based on the Levenshtein Distance, or Edit Distance algorithm. To do a fuzzy search use the tilde, "~", symbol at the end of a Single word Term. For example to search for a term similar in spelling to "roam" use the fuzzy search:

```
roam~
```

This search will find terms like foam and roams.

Starting with Lucene 1.9 an additional (optional) parameter can specify the required similarity. The value is between 0 and 1, with a value closer to 1 only terms with a higher similarity will be matched. For example:

```
roam~0.8
```

The default that is used if the parameter is not given is 0.5.

How to search for documents that contains 'apache' and 'jakarta' within 10 words apart from each other?

Lucene supports finding words are a within a specific distance away. To do a proximity search use the tilde, "~", symbol at the end of a Phrase. For example to search for a "apache" and "jakarta" within 10 words of each other in a document use the search:

```
"jakarta apache"~10
```

How can I search for one term near another term (say, "batman" and "movie")?

A proximity search can be done with a sloppy phrase query. The closer together the two terms appear in the document, the higher the score will be. A sloppy phrase query specifies a maximum "slop", or the number of positions tokens need to be moved to get a match.

This example for the standard request handler will find all documents where "batman" occurs within 100 words of "movie":

```
q=text:"batman movie"~100
```

The dismax handler can easily create sloppy phrase queries with the pf (phrase fields) and ps (phrase slop) parameters:

```
q=batman movie&pf=text&ps=100
```

The dismax handler also allows users to explicitly specify a phrase query with double quotes, and the qs(query slop) parameter can be used to add slop to any explicit phrase queries:

```
q="batman movie"&qs=100
```

How can I increase the score for specific documents?

index-time boosts: To increase the scores for certain documents that match a query, regardless of what that query may be, one can use index-time boosts.

Index-time boosts can be specified per-field also, so only queries matching on that specific field will get the extra boost. An Index-time boost on a value of a multiValued field applies to all values for that field.

Index-time boosts are assigned with the optional attribute "boost" in the <doc> section of the XML updating messages.

Query Elevation Component: To raise certain documents to the top of the result list based on a certain queries, one can use the QueryElevationComponent.

How can I change the score of a document based on the *value* of a field?

Use a FunctionQuery as part of your query.

Solr can parse function queries in the following syntax.

Some examples:

```
  # simple boosts by popularity
  q=%2Bsupervillians+_val_:"popularity"
  defType=dismax&qf=text&q=supervillians&bf=popularity

  # boosts based on complex functions of the popularity field
  q=%2Bsupervillians+_val_:"scale(popularity,0,100)"
  defType=dismax&qf=text&q=supervillians&bf=sqrt(popularity)
```

How are documents scored?

Basic scoring factors:

- tf stands for term frequency - the more times a search term appears in a document, the higher the score
- idf stands for inverse document frequency - matches on rarer terms count more than matches on common terms
- coord is the coordination factor - if there are multiple terms in a query, the more terms that match, the higher the score
- lengthNorm - matches on a smaller field score higher than matches on a larger field
- index-time boost - if a boost was specified for a document at index time, scores for searches that match that document will be boosted.
- query clause boost - a user may explicitly boost the contribution of one part of a query over another.

See the Lucene scoring documentation for more info.

How can I boost the score of newer documents?

- Do an explicit sort by date (relevancy scores are ignored)
- Use an index-time boost that is larger for newer documents
- Use a FunctionQuery to influence the score based on a date field. In Solr 1.3, use something of the form recip(rord(myfield),1,1000,1000). In Solr 1.4, use something of the form recip(ms(NOW,mydatefield),3.16e-11,1,1)

See ReciprocalFloatFunction and BoostQParserPlugin.

A full example of a query for "ipod" with the score boosted higher the newer the product is:

```
q={!boost b=recip(ms(NOW,manufacturedate_dt),3.16e-11,1,1)}ipod
```

One can simplify the implementation by decomposing the query into multiple arguments:

```
q={!boost b=$dateboost v=$qq}&dateboost=recip(ms(NOW,manufacturedate_dt),3.16e-
```

Now the main "q" argument as well as the "dateboost" argument may be specified as defaults in a search handler in solrconfig.xml, and clients would only need to pass "qq", the user query.

To boost another query type such as a dismax query, the value of the boost query is a full sub-query and hence can use the {!querytype} syntax. Alternately, the defType param can be used in the boost local params to set the default type to dismax. The other dismax parameters may be set as top level parameters.

```
q={!boost b=$dateboost v=$qq defType=dismax}&dateboost=recip(ms(NOW,manufacture
&qf=text&pf=text&qq=ipod
```

How do I give a very low boost to documents that match my query?

In general the problem is that a "low" boost is still a boost, it can only improve the score of documents that match. One way to fake a "negative boost" is to give a high boost to everything that does *not* match. For example:

```
bq=(*:* -field_a:54)^10000
```

If "bq" supports pure negative queries then you can simplify that to bq=-field_a:54^10000

How to tell Solr to output debug information?

Use debugQuery parameter. If this parameter is present (regardless of its value) then additional debugging information will be included in the response, including "explain" info for each of the documents returned. This debugging info is meant for human consumption… its XML format could change in the future.

We can also use the 'debug' parameter:

```
&debug=true
```

How to include even more debug information?

Use explainOther parameter.

Why doesn't document id:juggernaut appear in the top 10 results for my query?

Since debugQuery=on only gives you scoring "explain" info for the documents returned, the explainOther parameter can be used to specify other documents you want detailed scoring info for:

```
q=supervillians&debugQuery=on&explainOther=id:juggernaut
```

Now you should be able to examine the scoring explain info of the top matching documents, compare it to the explain info for documents matching id:juggernaut, and determine why the rankings are not as you expect.

How to specify the query parser?

Use the defType parameter.

What is LocalParam type?

Specify the query parser. If a LocalParams value appears without a name, it is given the implicit name of "type". This allows short-form representation for the type of query parser to use when parsing a query string. Thus:

```
q={!dismax qf=myfield}solr+rocks
```

is equivalent to:

```
q={!type=dismax qf=myfield}solr+rocks
```

What is LocalParam v?

A "v" within local parameters is an alternate way to specify the value of that parameter. For example,

```
q=solr+rocks
```

is equivalent to:

```
q={!v='solr+rocks'}
```

What is LocalParam parameter dereferencing / indirection?

Parameter dereferencing or indirection allows one to use the value of another argument rather than specifying it directly. This can be used to simplify queries, decouple user input from query parameters, or decouple front-end GUI parameters from defaults set in solrconfig.xml. For example,

```
q=solr+rocks
```

is equivalent to

```
q={!v=$qq}&qq=solr+rocks
```

How to specify the query parser?

Users can specify the type of a query in most places that accept a query string using LocalParams syntax. For example, the following query string specifies a lucene/solr query with a default operator of "AND" and a default field of "text":

```
q={!lucene q.op=AND df=text}myfield:foo +bar -baz
```

In standard Solr search handlers, the defType param can be used to specify the default type of the main query (ie: the q param) but it only affects the main query — The default type of all other query parameters will remain "lucene".

q={!func}popularity is thus equivalent to defType=func&q=popularity in the standard Solr search handler.

What are the differences between default Solr query parser and Lucene query parser?

The standard Solr Query Parser syntax is a superset of the Lucene Query Parser syntax.

Differences in the Solr Query Parser include:

- Range queries [a TO z], prefix queries a*, and wildcard queries a*b are constant-scoring (all matching documents get an equal score). The scoring factors tf, idf, index boost, and coord are not used. There is no limitation on the number of terms that match (as there was in past versions of Lucene). Lucene 2.1 has also switched to use ConstantScoreRangeQuery for its range queries.
- A * may be used for either or both endpoints to specify an open-ended range query. field:[* TO 100] finds all field values less than or equal to 100. field:[100 TO *] finds all field values greater than or equal to 100. field:[* TO *] matches all documents with the field.
- Pure negative queries (all clauses prohibited) are allowed. -inStock:false finds all field values where inStock is not false. -field:[* TO *] finds all documents without a value for field.
- A hook into FunctionQuery syntax. Quotes will be necessary to encapsulate the function when it includes parentheses. Example: _val_:"recip(rord(myfield),1,2,3)"
- Nested query support for any type of query parser (via QParserPlugin). Quotes will often be necessary to encapsulate the nested query if it contains reserved characters. Example: _query_:"{!dismax qf=myfield}how now brown cow"

How to do an open-ended range query?

A * may be used for either or both endpoints to specify an open-ended range query. field:[* TO 100] finds all field values less than or equal to 100. field:[100 TO *] finds all field values greater than or equal to 100. field:[* TO *] matches all documents with the field.

How to select documents that has NULL value?

Pure negative queries (all clauses prohibited) are allowed. -inStock:false finds all field values where inStock is not false. -field:[* TO *] finds all documents without a value for field.

If negative queries does not work, prepend it with *:*

```
*:* -fieldName:value
```

```
Field1:Val1 AND (*:* NOT Field2:Val2)
```

That should be equivalent to Field1:Val1 -Field2:Val2. You only need the *:* trick if all of the clauses of a boolean query are negative.

How to specify the type of query for the main query parameter?

In standard Solr search handlers, the defType param can be used to specify the default type of the main query (ie: the q param) but it only affects the main query — The default type of all other query parameters will remain "lucene". For example,

```
q={!func}popularity
```

is thus equivalent to:

```
defType=func&q=popularity
```

in the standard Solr search handler.

How to specify the type of query for other query parameter?

Users can specify the type of a query in most places that accept a query string using LocalParams syntax. The following query string specifies a lucene/solr query with a default operator of "AND" and a default field of "text":

```
q={!lucene q.op=AND df=text}myfield:foo +bar -baz
```

The above changes the type of query parser for the main query parameter (q). But this syntax can be applied to other query parameter (using this syntax, we can change the query parser using for other query parameter)

Does Lucene support wild card search?

Lucene supports single and multiple character wildcard searches within single terms (not within phrase queries).

What is the wild card character that match single character?

To match a single character, use '?'. Lucene supports single and multiple character wildcard searches within single terms (not within phrase queries)

What is the wild card character that match multiple characters?

To match multiple characters, use '*'. Lucene supports single and multiple character wildcard searches within single terms (not within phrase queries)

Can we have a wild card character at the beginning of a search term?

No. You cannot use a * or ? symbol as the first character of a search. There may be some way to work around this.

How to make other data type searchable / query-able?

For the most part, Lucene only deals with strings, so integers, floats, dates, and doubles require special handling to be searchable.

If we have a integer field, and you were not able to query on it with fieldName:some_value, you need to index that field.

When should we use standard request handler? When should we use the dismax request handler?

The standard request handler uses SolrQuerySyntax to specify the query via the q parameter, and it must be well formed or an error will be returned. It's good for specifying exact, arbitrarily complex queries.

The dismax request handler has a more forgiving query parser for the q parameter, useful for directly passing in a user-supplied query string. The other parameters make it easy to search across multiple fields using disjunctions and sloppy phrase queries to return highly relevant results.

For servicing user-entered queries, start by using dismax.

How can I search for a term in more than one fields?

If we are using the standard request handler, use q:

```
q=title:superman subject:superman
```

If we are using the dismax request handler, specify the query fields using the qf param:

```
q=superman&qf=title subject
```

How can I make a search term in the title field score higher than in the subject field?

For the standard request handler, "boost" the clause on the title field:

```
q=title:superman^2 subject:superman
```

Using the dismax request handler, one can specify boosts on fields in parameters such as qf:

```
q=superman&qf=title^2 subject
```

How can I make exact-case matches score higher?

A query of "Penguin" should score documents containing "Penguin" higher than docs containing "penguin".

The general strategy is to index the content twice, using different fields with different fieldTypes (and different analyzers associated with those fieldTypes). One analyzer will contain a lowercase filter for case-insensitive matches, and one will preserve case for exact-case matches.

Use copyField commands in the schema to index a single input field multiple times.

Once the content is indexed into multiple fields that are analyzed differently, query across both fields.

How can I make queries of "spiderman" and "spider man" match "Spider-Man"?

WordDelimiterFilter can be used in the analyzer for the field being queried to match words with intra-word delimiters such as dashes or case changes.

How to work with date?

See http://wiki.apache.org/solr/SolrQuerySyntax

For a specific search term / phrase / query, can we put specific documents at the top of the result?

Yes. Take a look at the elevate.xml file:

```
<elevate>
 <query text="foo bar">
  <doc id="1" />
  <doc id="2" />
  <doc id="3" />
 </query>

 <query text="ipod">
   <doc id="MA147LL/A" />  <!-- put the actual ipod at the top -->
   <doc id="IW-02" exclude="true" /> <!-- exclude this cable -->
 </query>

</elevate>
```

If this file is found in the config directory, it will only be loaded once at startup. If it is found in Solr's data directory, it will be re-loaded every commit.

What is the purpose of copyField?

The <copyField> mechanism lets you create the all field without manually adding all the content of your document to a separate field. Copy fields are convenient ways to index the same content in multiple ways. For instance, if you wanted to provide both exact matching respecting case and matching ignoring case, you could use a copy field to automatically analyze the incoming content. Content would then be indexed exactly as received, with all letters in lowercase.

What is dynamicField?

One of the powerful features of Lucene is that you don't have to pre-define every field when you first create your index. Even though Solr provides strong datatyping for fields, it still preserves that flexibility using "Dynamic Fields". Using <dynamicField> declarations, you can create field rules that Solr will use to understand what datatype should be used whenever it is given a field name that is not explicitly defined, but matches a prefix or suffix used in a dynamicField.

For example the following dynamic field declaration tells Solr that whenever it sees a field name ending in "_i" which is not an explicitly defined field, then it should dynamically create an integer field with that name…

```
<dynamicField name="*_i"  type="integer"  indexed="true"  stored="true"/>
```

Dynamic fields are special kinds of fields that can be added to any document at any time with the attributes defined by the field declaration. The key difference between a dynamic field and a regular field is that dynamic fields do not need to have a name declared ahead of time in the schema.xml. Solr applies the glob-like pattern in the name declaration to any incoming field name not already declared and processes the field according to the semantics defined by its <dynamicField> declaration. For instance, <dynamicField name="*_i" type="sint" indexed="true" stored="true"/> means that a field named myRating_i would be treated as an sint by Solr, even though it wasn't declared as a field. This is convenient, for example, when letting users define the content to be searched.

```
<dynamicField name="*_dt" type="date" indexed="true" stored="true"/>
```

If at index-time a document contains a field that isn't matched by an explicit field definition, but does have a name matching this pattern (that is, ends with _dt such as updated_dt), then it gets processed according to that definition. This also applies to searching the index.

A dynamic field is declared just like a regular field in the same section. However, the element is named dynamicField, and it has a name attribute that must start or end with an asterisk (the wildcard). If the name is just *, then it is the final fallback.

Using dynamic fields is most useful for the * fallback if you decide that all fields attempted to be stored in the index should succeed, even if you didn't know about the field when you designed the schema. It's also useful if you decide that instead of it being an error, such unknown fields should simply be ignored (that is, not indexed and not stored).

Why might I want to index single data in multiple field?

It often make sense to take what's logically speaking a single field (e.g. product name) and index it into several different Solr fields, each with different field options and/or analyzers.

If I had a field with a list of authors, such as: 'Schildt, Herbert; Wolpert, Lewis; Davies, P.', I might want to index the same data differently in three different fields (perhaps using the Solr copyField directive):

- For searching: Tokenized, case-folded, punctuation-stripped: schildt / herbert / wolpert / lewis / davies / p
- For sorting: Untokenized, case-folded, punctuation-stripped: schildt herbert wolpert lewis davies p
- For faceting: Primary author only, using a solr.StringField: Schildt, Herbert

What is 'term vectors'?

The storage of Lucene term vectors can be triggered using the following field options (in schema.xml):

- termVectors=true|false
- termPositions=true|false
- termOffsets=true|false

These options can be used to accelerate highlighting and other anciliary functionality, but impose a substantial cost in terms of index size. They are not necessary for typical uses of Solr (phrase queries, etc., do not require these settings to be present).

How to copy all source field to a single field?

A common requirement is to copy or merge all input fields into a single solr field. This can be done as follows:

```
<copyField source="*" dest="text"/>
```

Should we do synonym expansion both at index-time and at query-time?

No. Synonym expansion should be done either index-time or query-time, but not both, as that would be redundant.

Why is index-time synonym expansion considered better than query-time synonym expansion?

For a variety of reasons, it is usually better to do this at index-time:

- A synonym containing multiple words (example: i pod) isn't recognized correctly at query-time because the query parser tokenizes on whitespace.
- The IDF component of Lucene's scoring algorithm will be much higher for documents matching a synonym appearing rarely, as compared to its equivalents that are common. This reduces the scoring effectiveness.
- Prefix, wildcard, and fuzzy queries aren't analyzed, and thus won't match synonyms.

Why is index-time synonym expansion considered less flexible than query-time synonym expansion?

Index-time synonym expansion is less flexible than query-time synonym expansion because with index-time synonym expansion, changes to the synonyms will require a complete re-index to take effect. Moreover, the index will get larger if you do index-time expansion.

It's plausible to imagine the issues above being rectified at some point. However, until then, index-time is usually best.

Is there any alternative for synonym expansion?

Alternatively, you could choose not to do synonym expansion. This means that for a given synonym term, there is just one term that should replace it. This requires processing at both index-time and query-time to effectively normalize the synonymous terms. However, since there is query-time processing, it suffers from the problems mentioned above with the exception of poor scores, which isn't applicable (see why index-time synonym expansion is considered better than query-time synonym expansion). The benefit to this approach is that the index size would be smaller, because the number of indexed terms is reduced.

You might also choose a blended approach to meet different goals. For example, if you have a huge index that you don't want to re-index often but you need to respond rapidly to new synonyms, then you can put new synonyms into both a query-time synonym file and an index-time one. When a re-index finishes, you empty the query-time synonym file. You might also be fond of the query-time benefits, but due to the multiple word term issue, you decide to handle those particular synonyms at index-time.

What is the purpose of StopFilterFactory?

Filter out "stop words", words that are very common, such as "a", "the", "is", "are" …

For indexes with lots of text, common uninteresting words like "the", "a", and so on, make the index large and slow down phrase queries. To deal with this problem, it is best to remove them from fields where they show up often. Fields likely to contain more than a sentence are ideal candidates.

The trade-off when omitting stop words from the index is that those words are no longer query-able. This is usually fine, but in some circumstances like searching for **To be or not to be**, it is obviously a problem. See "shingling".

How to determine which words appear commonly in your index?

We should determine which words that appear frequently in our index can be considered as stop words. This help reduce the size of the index.

In order to determine which words appear commonly in your index, access the SCHEMA BROWSER menu option in Solr's admin interface. A list of your fields will appear on the left. In case the list does not appear at once, be patient. For large indexes, there is a considerable delay before the field list appears because Solr is analyzing the data in your index. Now, choose a field that you know contains a lot of text. In the main viewing area, you'll see a variety of statistics about the field including the top-10 terms appearing most frequently.

What is the purpose of phonetic sound-alike analysis?

Correct misspelled words.

A filter is used at both index and query-time that phonetically encodes each word into a phoneme.

How many phonetic sound-alike algorithms does Solr offer?

1. DoubleMetaphone
2. Metaphone
3. RefinedSoundex
4. Soundex.

Among the phonetic sound-alike algorithms that Solr offers, which one seems to be the best?

Anecdotally, DoubleMetaphone appears to be the best, even for non-English text. However, you might want to experiment in order to make your own choice.

RefinedSoundex declares itself to be most suitable for spellcheck applications. However, Solr can't presently use phonetic analysis in its spellcheck component.

How many tools does Solr have for aggressive in-exact searching?

Solr has three tools at its disposal for more aggressive in-exact searching: phonetic sounds-like, query spellchecking, and fuzzy searching.

With regard to phonetic sound-alike analysis, how to use the analysis page?

Using Solr's analysis admin page, it can be shown that this field type encodes Smashing Pumpkins as SMXNK|XMXNK PMPKNS. The use of a vertical bar | here indicates both sides are alternatives for the same position. This is not supposed to be meaningful, but it is useful for comparing similar spellings to detect its effectiveness.

What are the two options for DoubleMetaphoneFilterFactory?

1. inject: A boolean defaulting to true that will cause the original words to pass through the filter. It might interfere with other filter options, querying, and potentially scoring. Therefore, it is preferred to disable this, and use a separate field dedicated to phonetic indexing.
2. maxCodeLength: The maximum phoneme code (that is Phonetic character, or syllable) length. It defaults to 4. Longer code are truncated.

How to use the DoubleMetaphone filter factory?

```
<filter class="solr.DoubleMetaphoneFilterFactory" inject="false" maxCodeLength=
```

How to use the other phonetic sound-alike filter factory?

To use the DoubleMetaphone filter factory:

```
<filter class="solr.DoubleMetaphoneFilterFactory" inject="false" maxCodeLength=
```

To use the other 3 phonetic sound-alike filter factories:

```
<filter class="solr.PhoneticFilterFactory" encoder="..." inject="false"/>
```

where the encoder attribute is one of Metaphone, RefinedSoundex, Soundex.

Does Solr support leading wild card search?

No.

Usually, text indexing technology is employed to search entire words. Occasionally however, there arises a need for a search to match an arbitrary substring of a word or across them. Lucene supports leading and trailing wildcards (example: *) on queries. However, only the latter is supported by Solr without internal modification.

Moreover, this approach only scales for very small indices before it gets very slow and/or results in an error. The right way to solve this is to venture into the black art of n-grams.

Before employing this approach, consider if what you really need is better tokenization for special code. For example, if you have a long string code that internally has different parts that users might search on separately, then you can use a PatternReplaceFilterFactory with some other analyzers to split them up.

How does N-gram analysis works in general?

N-gram analysis slices text into many smaller substrings ranging between a minimum and maximum configured size. For example, consider the word Tonight. An NGramFilterFactory configured with minGramSize of 2 and maxGramSize of 5 would yield all of the following indexed terms: (2-grams:) To, on, ni, ig, gh, ht, (3-grams:) Ton, oni, nig, igh, ght, (4-grams:) Toni, onig, nigh, ight, (5-grams:) Tonig, onigh, night. Note that Tonight fully does not pass through because it has more characters than the maxGramSize. N-gram analysis can be used as a filter for processing on a term-by-term basis, and it can also be used as a tokenizer with NGramTokenizerFactory, which will emit n-grams spanning across the words of the entire source text.

What is the suggested analyzer configuration for using n-grams?

```
<fieldType name="nGram" class="solr.TextField" positionIncrementGap="100" store
    <analyzer type="index">
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <!-- potentially word delimiter, synonym filter, stop words, NOT stemmi
        <filter class="solr.LowerCaseFilterFactory"/>
        <filter class="solr.NGramFilterFactory" minGramSize="2" maxGramSize="15
    </analyzer>
    <analyzer type="query">
        <tokenizer class="solr.StandardTokenizerFactory"/>
        <!-- potentially word delimiter, synonym filter, stop words, NOT stemmi
        <filter class="solr.LowerCaseFilterFactory"/>
    </analyzer>
</fieldType>
```

Notice that the n-gramming only happens at index-time.

How should n-gram analysis be used?

This analysis would be applied to a field created solely for the purpose of matching substrings. Another field would exist for typical searches, and a dismax handler should be configured for searches to use both fields using a smaller boost for this field.

How do EdgeNGramTokenizerFactory and EdgeNGramFilterFactory work?

Another variation is EdgeNGramTokenizerFactory and EdgeNGramFilterFactory, which emit n-grams that are adjacent to either the start or end of the input text. For the filter-factory, this input-text is a term, and the tokenizer is the entire input. In addition to minGramSize and maxGramSize, these analyzers take a side argument that is either front or back. If only prefix or suffix matching is needed instead of both, then an EdgeNGram analyzer is for you.

What are the costs of n-gram analysis?

There is a high price to be paid for n-gramming. Recall that in the earlier example, Tonight was split into 15 substring terms, whereas typical analysis would probably leave only one. This translates to greater index sizes, and thus a longer time to index.

Note the ten-fold increase in indexing time for the artist name, and a five-fold increase in disk space. Remember that this is just one field!

Given these costs, n-gramming, if used at all, is generally only done on a field or two of small size where there is a clear requirement for substring matches.

The costs of n-gramming are lower if minGramSize is raised and to a lesser extent if maxGramSize is lowered. Edge n-gramming costs less too. This is because it is only based on one side. It definitely costs more to use the tokenizer-based n-grammers instead of the term-based filters used in the example before, because terms are generated that include and span whitespace. However, with such indexing, it is possible to match a substring spanning words.

What is the purpose for them n-gram analysis?

N-gram analysis is potentially useful for

- leading wild card search

Caution: N-gram analysis comes with a huge cost.

How to match substring spanning across words?

One possible way is to use n-gram analysis that span across words.

The costs of n-gramming are lower if minGramSize is raised and to a lesser extent if maxGramSize is lowered. Edge n-gramming costs less too. This is because it is only based on one side. It definitely costs more to use the tokenizer-based n-grammers instead of the term-based filters used in the example before, because terms are generated that include and span whitespace. However, with such indexing, it is possible to match a substring spanning words.

What is the purpose of StandardFilterFactory?

Works in conjunction with StandardTokenizer. It will remove periods inbetween acronyms and s at the end of terms:

```
"I.B.M. cat's" => "IBM", "cat"
```

What is the purpose of LowerCaseFilterFactory?

Simply lowercases all text. Don't put this before WordDelimeterFilterFactory if you want to split on case transitions.

What is the purpose of KeepWordFilterFactory?

Omits all of the words, except those in the specified file:

```
<filter class="solr.KeepWordFilterFactory" words="keepwords.txt" ignoreCase="tr
```

If you want to ensure a certain vocabulary of words in a special field, then you might enforce it with this.

What is the purpose of LengthFilterFactory?

Filters out the terms that do not have a length within an inclusive range.

```
<filter class="solr.LengthFilterFactory" min="2" max="5" />
```

What is the purpose of RemoveDuplicatesTokenFilterFactory?

Ensures that no duplicate terms appear at the same position. This can happen, for example, when synonyms stem to a common root. It's a good idea to add this to your last analysis step, if you are doing a fair amount of other analysis.

What is the purpose of ISOLatin1AccentFilterFactory?

This will normalize accented characters such as é to the unaccented equivalent e. An alternative and more customizable mechanism introduced in Solr 1.4 is a CharFilterFactory, which is something that actually comes before the lead tokenizer in the analysis chain. For more information about this approach, search Solr's Wiki for MappingCharFilterFactory.

What is the purpose of CapitalizationFilterFactory?

This capitalizes each word according to the rules that you specify. For more information, see the Javadocs at http://lucene.apache.org/solr/api/org/apache/solr/analysis/CapitalizationFilterFactory.html.

What is the purpose of PatternReplaceFilterFactory?

Takes a regular expression and replaces the matches. Example:

```
<filter class="solr.PatternReplaceFilterFactory" pattern=".*@(.*)" replacement=
```

This replacement happens to be a reference to a regexp group, but it might be any old string. The replace attribute is either first to only apply to the first occurrence, or all. This example is for processing an email address field to get only the domain of the address.

How to use explainOther?

If you want to determine why a particular document wasn't matched by the query, or the query matched many documents and you want to ensure that you see scoring diagnostics for a certain document, then you can put a query for this value, such as id:"Release:12345", and debugQuery's output will be sure to include documents matching this query in its output.

```
&q=patient&debugQuery=on&explainOther=id:juggernaut
```

How to import data into Solr?

See chapter 3 in the book "Solr 1.4 Enterprise Search Server", published by Packtpub.

Why is index-time boosting considered less flexible than query-time boosting?

index-time boosting, which is rarely done as compared to the more flexible query-time boosting. Index-time boosting is less flexible because such boosting decisions must be decided at index-time and will apply to all of the queries.

What is similarity?

A <similarity> declaration (in schema.xml) can be used to specify the subclass of Similarity that you want Solr to use when dealing with your index. If no Similarity class is specified, the Lucene DefaultSimilarity is used. Please see SolrPlugins for information on how to ensure that your own custom Similarity can be loaded into Solr.

How do I use copyField with wildcards?

The <copyField> directive allows wildcards in the source, so that several fields can be copied into one destination field without having to specify them all individually. The dest field may by a full field name, or a wildcard expression. A common use case is something like:

```
<copyField source="*_t"  dest="text" />
```

This tells Solr to copy the contents of any field that ends in "_t" to the "text" field. This is particularly useful when you have a large, and possibly changing, set of fields you want to index into a single field. In this example, **it's important that the "text" field be defined in schema.xml as multiValued since you intend to copy multiple sources into the single destination.**

What is the effect of having too many copyField directives?

Copying data to additional fields is necessary to supporting different indexing purposes (one field may have a different analyzer). However, copying data to additional fields increase indexing time, and will consume more disk space. So make sure that you remove copyField directives that you don't really need.

How to specify the amount of time for search to complete?

Use the timeAllowed parameter. It specifies the time allowed for a search to finish. This value only applies to the search and not to requests in general. Time is in milliseconds. Values <= 0 mean no time restriction. Partial results may be returned (if there are any).

How to tell Solr to not return the header?

Use omitHeader parameter (true | false). Exclude the header from the returned results. The header contains information about the request, such as the time it took to complete. The default is false.

What is the definition of "stop words":

Words that are frequently used that don't help distinguish one document from the other, such as a, an, the, in, and on, etc

Why is Solr better than database fulltext?

Fuzzy search in Solr is based on Levenshtein distance algorithm which looks at the number of changes to a string required to arrive at another string. Levenshtein is great, but we can improve the quality of these search results. One reason for inaccurate search queries is misspellings. Levenshtein attempts to find words that are close, but this ignores how people actually work with words. A better solution would be to look at what kinds of substitutions are required and give certain ones higher scores. This would be based on their use in language; their effective pronunciation.

There are a number of algorithms that support comparing two words based on how similar they are in pronunciation. One of the more common ones is called Soundex. This algorithm produces a hash from a given string. Other strings that have a similar pronunciation are supposed to hash to the same value. There are a few limitations with the method. One major one is that it can’t handle wrong first letters. So ‘psychology’ (P242), ‘sychology’ (S240), and ‘cychology’ (C420) will not match at all. There are a number of variations on Soundex as well as a few alternatives.

Solr supports a number of these phonetic filters.

Bottom line: Solr is more specialized for full-text search, and it is modular, and extend-able.

What is parametric search?

It is another name for faceted search. Faceted search is also known as faceted browsing, faceted navigation, guided navigation.

What is guided navigation?

It is another name for faceted search. Faceted search is also known as faceted browsing, faceted navigation, guided navigation, or parametric search.

What is faceted search?

Faceted search is feature of Solr, which **tells Solr to return extra information related to the current query**.

Faceted search is also known as guided navigation, or parametric search. Basically it allows user to drill down the the category structure. Imagine that you are in an online store (Amazon, or eBay) which sell all kinds of products (electronics, clothing, books, school supplies, etc). So a digital camera can be categorized as "digital cameras" (a sub-category of electronics).

Faceted search is not much different from a regular search. Imagine a user search for "camera", and you want your server to return extra information, break down by some conditions (manufacturers, price range, etc), so in your server logic, you added extra parameters to the URL.

This example assumes a field named "manufacturer" exists in the schema and that it has the manufacturer indexed as a single token. The Solr query to retrieve matches for "camera" would be:

```
http://localhost:8983/solr/select?q=camera
```

To retrieve facet counts for the "manufacturer" field, we would simply add the following parameters to that query request:

```
&facet=true&facet.field=manufacturer
```

Any number of facet commands can be added to a single query request. To facet on both the "manufacturer" field and the "camera_type" field, we would add the following parameters:

```
&facet=true&facet.field=manufacturer&facet.field=camera_type
```

So, faceted search is feature of Solr, which **tells Solr to return extra information (group by some condition)** related to the current query. The condition can be any field (for example, manufacturer), or a range (for example, price range). We can have multiple conditions / constraints.

What type of faceting does Solr offer by default?

- Field faceting: retrieve the counts for all terms, or just the top terms in any given field. The field must be indexed.
- Query faceting: return the number of documents in the current search results that also match the given query.
- Date faceting: return the number of documents that fall within certain date range.

Can we facet on multiple fields?

Yes. Any number of facet commands can be added to a single query request. To facet on both the "manufacturer" field and the "camera_type" field, we would add the following parameters:

```
&facet=true&facet.field=manufacturer&facet.field=camera_type
```

What is a facet?

Facet is like category. Price can be broken down into price ranges, and each of these ranges can be considered as a category. Date can also be broken down into date ranges, and each of these date ranges can be considered as a category also.

How to implement facet on price?

If we request field faceting on the "price" field, we get back counts for individual prices. However, we want price ranges, not individual prices. To implement facet on price we utilize "query faceting" that provide the ability to retrieve facet counts for arbitrary queries. Let's assume that we have an indexed "price" field and we want to get the facet counts for the following ranges of prices: $100 or less, $100-$200, $200-$300, $300-$400, $400-$500, and $500 or over. We simply add a facet.query command to our query request for each desired range:

```
&facet=true&facet.query=price:[* TO 100]
    &facet.query=price:[100 TO 200];&facet.query=[price:200 TO 300]
    &facet.query=price:[300 TO 400];&facet.query=[price:400 TO 500]
    &facet.query=price:[500 TO *]
```

What are the requirements for using faceting?

- We can facet on multiple field
- Each field must be indexed as single token (field type is string)

- We must add facet parameters to the URL

How to apply conditions / constraints on faceting?

Now that we've learned how to retrieve facet counts, how do we allow the user to drill down and narrow the search results with any of those constraints? The answer is standard Solr filter queries, where search results are filtered by any number of arbitrary queries.

Let's assume again that the user typed "camera" into the search box and we have queried solr for the top matching documents, and also requested some facet counts to be displayed to the user.

```
http://localhost:8983/solr/select?q=camera
  &facet=on&facet.field=manu&facet.field=camera_type
  &facet.query=price:[* TO 100]
  &facet.query=price:[100 TO 200];&facet.query=[price:200 TO 300]
  &facet.query=price:[300 TO 400];&facet.query=[price:400 TO 500]
  &facet.query=price:[500 TO *]
```

Now let's assume that the user wants to drill down on the constraint $400-$500 from the Price facet to get a new set of results that include only cameras in that price range. For this we use the fq (filter query) parameter, which allows one to filter by a query. We'll also send the relevant faceting commands again since we also want to update the facet counts.

```
http://localhost:8983/solr/select?q=camera
  &facet=on&facet.field=manu&facet.field=camera_type
  &fq=price:[400 to 500]
```

Notice that we no longer request the facet counts for the prices since we are constraining results to one price range and thus already know that the counts for other ranges would be zero.

To filter our search results further, we'll simply add an additional fq parameter.

```
http://localhost:8983/solr/select?q=camera
  &facet=on&facet.field=manu

  &fq=price:[400 to 500]
  &fq=camera_type:SLR
```

How can I rebuild my index from scratch if I change my schema?

1. Stop your application server
2. Change your schema.xml file
3. Start your application server
4. Delete the index directory in your data directory (or alternately delete all documents using <delete><query>*:*</query></delete> before shutting down Solr)
5. Send an <optimize/> command
6. Re-Index your data

How can I update a specific field of an existing document?

In Lucene to update a document the operation is really a delete followed by an add. You will need to add the complete document as there is no such "update only a field" semantics in Lucene.

What is the equivalent for Documents in term of data?

An index consist of a set of Documents, and each Document consist of one or more Fields. Each Field has a name and a value. You can think of a Document as a row in an RDBMS, and Fields as column in that row.

How to use separate analyzers for querying and indexing?

You can specify two analyzers in the same fieldType:

```
<fieldType name="..." class="...">
<analyzer type="index">

...
</analyzer>
<analyzer type="query">
...
</analyzer>
</fieldType>
```

How to handle acronym?

I am migrating from a pure Lucene application to using solr. For legacy reasons I must support a somewhat obscure query feature: lowercase words in the query should match lowercase or uppercase in the index, while uppercase words in the query should only match uppercase words in the index.

To do this with Lucene we created a custom Analyzer and custom TokenFilter. During indexing, the custom TokenFilter duplicates uppercase tokens as lowercase ones and sets their offsets to make them appear in same position as the upper case token, i.e., you get two tokens for every uppercase token. Then at query time a normal (case sensitive) analyzer is used so that lowercase tokens will match either upper or lower, while the uppercase will only match uppercase.

You can specify two analyzers in the same fieldType:

```
<fieldType name="..." class="...">
<analyzer type="index">
...
</analyzer>
<analyzer type="query">
...
</analyzer>
</fieldType>
```

How to use wild card search with highlighting?

To simulate wildcard searching I used EdgeNGrams following some of the instructions here: http://www.lucidimagination.com/blog/2009/09/08/auto-suggest-from-popular-queries-using-edgengrams/. Actually I really only added the edgytext fieldtype to schema.xml and changed the fieldtype of the field I wanted to search.

Or you can grab the latest nightly build and use edismax (ExtendedDismaxQParser). It handles both trailing and leading wildcards.

Is it possible to retrieve a multi-value field?

Yes, you'll get what is stored and asked for.

Is there any way to provide autocomplete while filtering results?

Yes, you can use facets to achieve that.

Introduction:

Lucene is a Java-based, high performance full-text search engine library originally written by Doug Cutting and donated to the Apache Software Foundation. **Lucene is a library.**

Apache Solr is a server (or a service) based on Lucene. Apache Solr is built on top of Lucene, and extends Lucene (**Lucene is a library, Solr is a server**). Solr was first developed at CNET Networks and donated to the Apache Software Foundation in early 2006 under the Lucene top-level project umbrella.

Features:

- Boosting
- Phrases
- Range filters
- Faceted search
- Hit highlighting
- Result highlighting
- Query spellcheck
- Auto-suggest queries
- More like this
- Multiple output formats (ncluding XML/XSLT and JSON)
- Can be extended it to meet the needs of your enterprise

Limitations:

- Solr does not support relational joins.
- Solr does not support wild card at the beginning of a search term

Comparison with other search engines:

- Sphinx
- MarkLogic
- Autonomy
- ElasticSearch

Notes:

Sorting only works properly on non-tokenized single-value fields.

Because Solr wraps and extends Lucene, it uses much of the same terminology. More importantly, indexes created by Solr are completely compatible with the Lucene search engine library. With proper configuration, and in some cases a bit of coding, Solr can read and use indexes built into other Lucene applications. Furthermore, **many Lucene tools like Luke work just as well on indexes created by Solr.**

In Solr and Lucene, an index is built of one or more documents. A document consist of one or more fields. A field consists of a name, content, and meta-data to tell Solr how to handle the content. For example, a field can contain string, number, boolean, date, or as well as any types you wish to add. A field can be described using a number of options that tell Solr how to treat the content during indexing and searching.

Lucene / Solr work primarily on words. So, to support partial word search, we may need to do something special.

Some fields do not need to be stored. Only store a field if you need to return it as part of your search result. You can search on or analyze a field without having to store it.

The field is only valid for the term that it directly precedes, so the query

```
title:Do it right
```

will find "Do" in the title field, and will find "it" and "right" in the default search field.

How to install, configure, start and stop Apache Solr?

```
Download, unzip and put it somewhere. On my machine, I had it at /usr/local/solr-1.4.1/
```

To start Apache Solr:

```
/usr/java/jre1.6.0_23/bin/java -Dsolr.solr.home=solr -jar start.jar
```

Which file contain the SQL statement that is used by data-import?

conf\data-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dataConfig>
    <dataSource type="JdbcDataSource"
                  driver="oracle.jdbc.driver.OracleDriver"
                  url="jdbc:oracle:thin:@IP:port:SID"
                  user="..."
                  password="..."/>
    <document name="doc">
        <entity
            name="ot_search"
            query="SELECT 'Y' AS IS_NEARMISS_APPLICABLE, ROWNUM, PROJECTSID, OT
            <field column="PROJECTSID" name="PROJECTSID" />
            <field column="OT_ID" name="OT_ID" />
            <field column="QNATURES_ID" name="QNATURES_ID" />
            <field column="QSUB_NATURES_ID" name="QSUB_NATURES_ID" />
            <field column="ENTITYTYPEID" name="ENTITYTYPEID" />
            <field column="LABEL" name="LABEL" />
            <field column="QNATURESLABEL" name="QNATURESLABEL" />
            <field column="QSUBNATURESLABEL" name="QSUBNATURESLABEL" />
            <field column="OTLABEL" name="OTLABEL" />
            <field column="ADDDATE" name="ADDDATE" />
            <field column="MODDATE" name="MODDATE" />
            <field column="OT_CATEGORY_ID" name="OT_CATEGORY_ID" />
            <field column="ENTITIESID" name="ENTITIESID" />
            <field column="AP_CATEGORY_ID" name="AP_CATEGORY_ID" />
            <field column="IS_ACTIVE" name="IS_ACTIVE" />
            <field column="ROWNUM" name="ROWNUM" />
            <field column="NATUREDETAIL" name="NATUREDETAIL" />
            <field column="IS_NEARMISS_APPLICABLE" name="IS_NEARMISS_APPLICABLE
        </entity>
    </document>
</dataConfig>
```

[Field Options By Use Case]
[SolrRelevancyFAQ]
[CommonQueryParameters]
[How to Get the Most Out of the Users' Mailing List]
[SolrTerminology]
[SolrQuerySyntax]
[Apache Lucene - Query Parser Syntax]

[schema.xml]
[Field types (data types) and field attributes]
[Analyzer]
[Stemming]
[Components]

Differences between Solr and Lucene parser
Combined versus single index
The statistic page
Facet Search
Performance
Deployment
Boolean operators
http://wiki.apache.org/solr/FilterQueryGuidance

page 100 (116 of 335)

Solr tutorial
Introduction to The Solr Enterprise Search Server
Search smarter with Apache Solr, Part 1
What's powering the Content API? - The Guardian speaks at Lucene Eurocon 2010

page revision: 139, last edited: 29 Sep 2016, 18:47 (225 days ago)