# SolrRelevancyFAQ

## Solr Relevancy FAQ

Relevancy is the quality of results returned from a query, encompassing both what documents are found, and their relative ranking (the order that they are returned to the user.)



### Should I use the standard or dismax Query Parser

The standard Query Parser 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 Query Parser 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.

⚠️ Extended Dismax parser] From Solr 3.1 we recommend starting with the new [ExtendedDisMax enabled by **defType=edismax**

### How can I search for "superman" in both the title and subject fields

The standard request handler uses SolrQuerySyntax for **q**:

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

Using the dismax request handler, specify the query fields using the **qf** param.

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

### How can I make "superman" 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
```

### Why are search results returned in the order they are?

If no other sort order is specified, the default is by relevancy score.

### How can I see the relevancy scores for search results

Request that the pseudo-field named "score" be returned by adding it to the **fl** (field list) parameter. The "score" will then appear along with the stored fields in returned documents. `q=Justice League&fl=*,score`

## Why doesn't my query of "flash" match a field containing "Flash" (with a capital "F")

The fieldType for the field containing "Flash" must have an analyzer that lowercases terms. This will cause all searches on that field to be case insensitive.

See AnalyzersTokenizersTokenFilters for more.

## How can I make exact-case matches score higher

Example: 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.

## I'm getting query parse exceptions when making queries

For the standard request handler, the **q** parameter must be correctly formatted SolrQuerySyntax, with any special characters escaped. If this is a user-entered query, consider using the dismax handler.

Many other parameters such as **fq** and **facet.query** must also conform to SolrQuerySyntax regardless of which handler is used.

## 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 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

### Query Elevation Component

To raise certain documents to the top of the result list based on a certain query, one can use the QueryElevationComponent.

### 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.

Under the covers, solr uses this numeric boost value as a factor that contributes to the "norm" for the field (along with teh length of the field in terms) so it is only valid if 'omitNorms="false"' for the fields you use it on. It will also be encoded according to the rules of the Similarity class used, which means it may loose precision. (ie: in the DefaultSimilarity, the numeric norm value is encoded as a single byte using a 3-bit mantissa, so differences in boost values of less than 25% may end up being rounded out)

Index-time boosts may also be assigned with the optional attribute "boost" in the <doc> section of the XML updating messages - in which case it is equivilent to specifying the boost param on all of the individual fiels. See UpdateXmlMessages for more information.

Using Field and/or Document boosts has been supported since the very early days of Lucene, but is some what limiting and antiquated at this point, and instead people should strongly consider indexing their boost values as numeric fields instead (see next section)

### Field Based Boosting

You can structure your queries to include "boosts" based on specific attributes of documents. This might be a simple matter of adding an optional query clause that "boosts" documents for matching an "important:true" query, or by using a function on the value of a numeric field (see the next section)

## How can I change the score of a document based on the *value* of a field (say, "popularity")

Use a FunctionQuery as part of your query.

Solr can parse function queries in the following syntax.

Some examples...

```
# simple boosts by popularity
defType=lucene&df=text&q=%2Bsupervillians+_val_:"popularity"
defType=dismax&qf=text&q=supervillians&bf=popularity
q={!boost b=popularity}text:supervillians

# boosts based on complex functions of the popularity field
defType=lucene&q=%2Bsupervillians+_val_:"sqrt(popularity)"
defType=dismax&qf=text&q=supervillians&bf=sqrt(popularity)
q={!boost b=sqrt(popularity)}text:supervillians
```

These functions can also operate on an ExternalFileField

## How are documents scored

By default, a "TF-IDF" based Scoring Model is used. The 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.

## Why does id:archangel come before id:hawkgirl when querying for "wings"

Add **debugQuery=on** to your request, and you will get (fairly dense) detailed scoring information for each document returned.

```
q=wings&indent=on&debugQuery=on
```

This extra information will appear in the "explain" section of the "debug" section in the response.

```
<response>
<result>[...]</result>
<lst name="debug">
 <str name="rawquerystring">wings</str>
 <str name="querystring">wings</str>
 <str name="parsedquery">text:wings</str>
 <str name="parsedquery_toString">text:wings</str>
 <lst name="explain">
  <str name="id=archangel,internal_docid=4">
0.46632254 = (MATCH) fieldWeight(text:wings in 4), product of:
  1.7320508 = tf(termFreq(text:wings)=3)
  2.871802 = idf(docFreq=2)
  0.09375 = fieldNorm(field=text, doc=4)
</str>
  <str name="id=hawkgirl,internal_docid=24">
0.35897526 = (MATCH) fieldWeight(text:wings in 24), product of:
  1.0 = tf(termFreq(text:wings)=1)
  2.871802 = idf(docFreq=2)
  0.125 = fieldNorm(field=text, doc=24)
</str>
[...]
```

In this specific example, we see that the main scoring difference between the two documents is the **tf** or (term frequency) factor. The text field for the **id: archangel** document contains the term **wings** 3 times (`termFreq(text:wings)=3`) while the **id:hawkgirl** document only contains it once.

Debug info is expensive to generate, and should only be used for debugging problems with specific queries.

Debug info can also be selected from the admin query page, http://localhost:8983/solr/admin/form.jsp

## 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 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)
    http://lucene.apache.org/core/api/queries/org/apache/lucene/queries/function/valuesource/ReciprocalFloatFunction.html http://lucene.apache.org/solr/api/org/apache/solr/search/BoostQParserPlugin.html

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

```
http://localhost:8983/solr/select?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:

```
http://localhost:8983/solr/select?q={!boost b=$dateboost v=$qq}&dateboost=recip(ms(NOW,manufacturedate_dt),3.16e-11,1,1)&qq=ipod
```

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 parser such as a dismax query, the value of the boost query is a full sub-query and hence can use the {!queryParser} 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.

```
http://localhost:8983/solr/select?q={!boost b=$dateboost v=$qq defType=dismax}&dateboost=recip(ms(NOW,
manufacturedate_dt),3.16e-11,1,1)&qf=text&pf=text&qq=ipod
```

Consider using reduced precision to prevent excessive memory consumption. You would instead use **recip(ms(NOW/HOUR,mydatefield),3.16e-11,1,1)**. See this thread for more information.

## How do I give a negative (or very low) boost to documents that match a query?

True negative boosts are not supported, but you can use a very "low" numeric boost value on query clauses. In general the problem that confuses people is that a "low" boost is still a boost, it can only improve the score of documents that match. For example, if you want to find all docs matching "foo" or "bar" but penalize the scores of documents matching "xxx" you might be tempted to try...

```
    q = foo^100 bar^100 xxx^0.00001    # NOT WHAT YOU WANT
```

...but this will still help a document matching all three clauses score higher then a document matching only the first two. One way to fake a "negative boost" is to give a large boost to everything that does *not* match. For example...

```
    q =  foo^100 bar^100 (*:* -xxx)^999
```

**NOTE:** When using (e)dismax, people sometimes expect that specifying a pure negative query with a large boost in the "bq" param will work (since Solr automatically makes top level purely negative positive queries by adding an implicit "*:*" – but this doesn't work with "bq", because of how queries specified via "bq" are added directly to the main query. You need to be explicit...

```
    ? defType = dismax
    & q = foo bar
    & bq = (*:* -xxx)^999
```

## TODO

⚠️ :TODO: ⚠️

- shorter fields score higher
- filter vs query clause
- when should index-time boosts be used
- searching one aggregate field vs many
- using relevancy as a secondary sort
- admin page for analyzer debugging