SOLR Search

**fl**

This parameter can be used to specify a set of fields to return, limiting the amount of information in the response. When returning the results to the client, only fields in this list will be included.

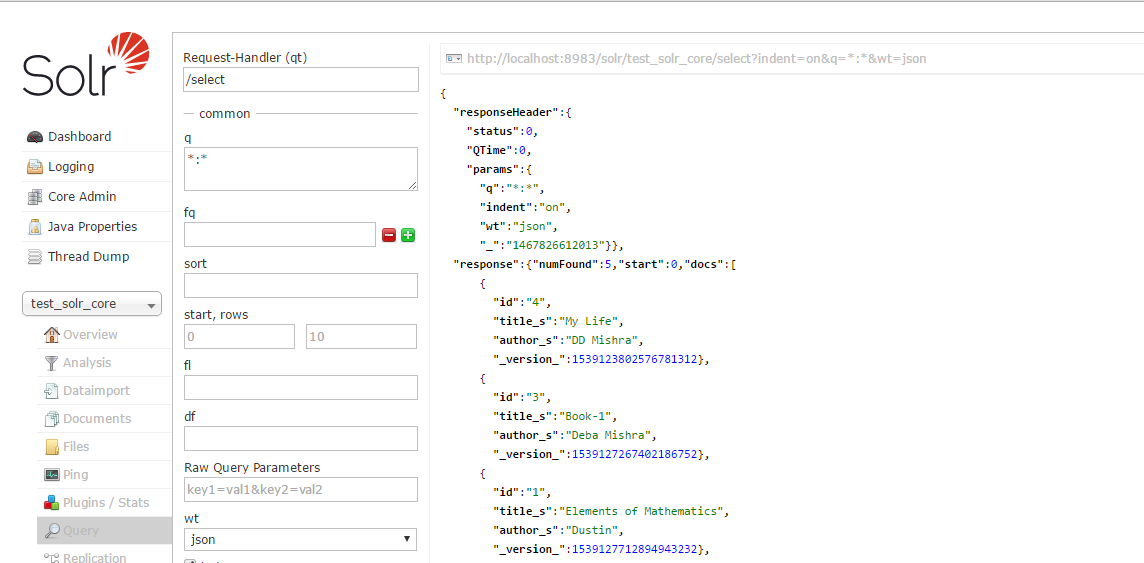
The set of fields to be returned can be specified as a space (or comma) separated list of field names. The string "score" can be used to indicate that the score of each document for the particular query should be returned as a field, and the string "\*" can be used to indicate all stored fields the document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

|  |  |
| --- | --- |
| **Examples** | |
| **Field List** | **Meaning** |
| id name price | Only return the "id", "name", and "price" fields |
| id,name,price | Same as above |
| id name, price | Same as above |
| id score | Return the "id" field and the score |
| \* | Return all fields the each document has |
| \* score | Return all fields each document has, along with the score |

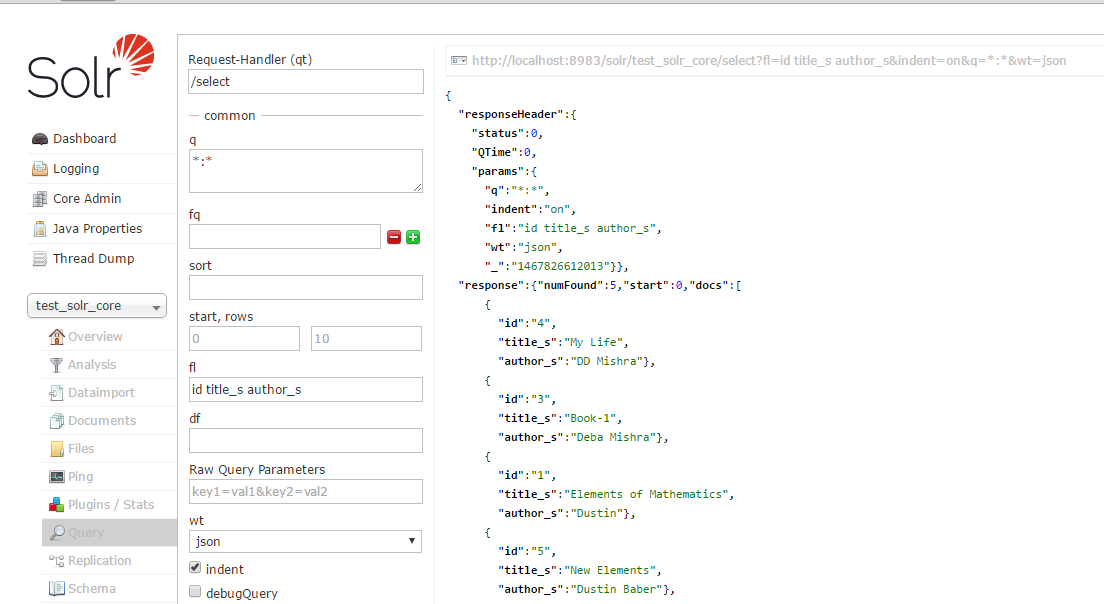
The default value is "\*"

Example is give below.

# **Without fl**



# **With fl field information, fi = id title\_s author\_s**



In stead of search field with spaces, you can also expect the return fields with comma(,). The example is given below.

Fl=id,title\_s,author\_s

## fq

"fq" stands for Filter Query.

This parameter can be used to specify a [query](https://wiki.apache.org/solr/SolrQuerySyntax) 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](https://wiki.apache.org/solr/SolrCaching) to learn about the caches Solr uses.

NOTE:

1. 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.
2. fq=popularity:[10 TO \*]

& fq=section:0

1. 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

1. 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.

**sort**

|  |  |
| --- | --- |
| **Examples** | |
| **sort** | **Meaning** |
|  | by default, it is the same as score desc |
| score desc | sort from highest score to lowest score |
| price asc | sort in ascending order of the "price" field |
| inStock desc, price asc | sort by "inStock" descending, then "price" ascending |
| sum(x\_f, y\_f) desc | sort by the sum of x\_f and y\_f in a descending order |

Sorting can be done on the "score" of the document, or on any multiValued="false" indexed="true" field provided that field is either 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 [Solr3.1](https://wiki.apache.org/solr/Solr3.1), sorting can also be done on arbitrary function queries (as in [FunctionQuery](https://wiki.apache.org/solr/FunctionQuery)) that produce a single value per document.

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.

NOTE:

1. A sort ordering must include a field name (or the pseudo-field score), followed by whitespace (escaped as + or %20 in URL strings), followed by a sort direction (asc or desc).
2. Multiple sort orderings can be separated by a comma, ie: sort=<field name>+<direction>[,<field name>+<direction>]...
   1. When more than one sort criteria is provided, the second entry will only be used if the first entry results in a tie. If there is a third entry, it will only be used if the first AND second entries are tied. This pattern continues with further entries.

An example screenshot is given below.

## start

This parameter is used to paginate results from a query. When specified, it indicates the **offset in the complete result set** for the queries where the set of returned documents should begin. (i.e. the first record appear in the result set is the offset)

The default value is "0".

## rows

This parameter is used to paginate results from a query. It specify the **maximum number of documents** from the complete result set to return to the client for every request. You can consider it as the maximum number of result appear in the page.

The default value is "10", which is used if the parameter is not specified. If you want to tell Solr to return all possible results from the query without an upper bound, specify rows to be 10000000 or some other ridiculously large value that is higher than the possible number of rows that are expected.

## df

This parameter overrides the default field defined in [SchemaXml](https://wiki.apache.org/solr/SchemaXml), if provided.

DisMax

**DisMax** is an abbreviation Disjunction Max, and is a popular query mode with Solr.

Simply put, it's your choice for all user generated queries.

Out of the box, Solr uses the standard Solr query parser which is pretty stupid, understanding only syntactically correct boolean queries like "title:foo OR body:foo", it can only search one field by default, and it may very well throw an exception in your face if you put in some characters it does not like.

Therefore a new, more robust query mode was needed and the DisMax and ExtendedDisMax Query Parsers were born. They are designed to process simple user entered phrases (without heavy syntax) and search for the individual words across several fields using different weighting (boosts) based on the significance of each field, and it should never throw an exception.

**Disjunction** refers to the fact that your search is executed across multiple fields, e.g. title, body and keywords, with different relevance weights

**Max** means that if your word "foo" matches both title and body, the max score of these two (probably title match) is added to the score, not the sum of the two as a simple OR query would do. This gives more control over your ranking.

DisMax and eDisMax are usually the short name for the actual query parser, so don't be confused.

# [**The DisMax Query Parser**](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser)

The DisMax query parser is designed to process simple phrases (without complex syntax) entered by users and to search for individual terms across several fields using different weighting (boosts) based on the significance of each field. Additional options enable users to influence the score based on rules specific to each use case (independent of user input).

In general, the DisMax query parser's interface is more like that of Google than the interface of the 'standard' Solr request handler. This similarity makes DisMax the appropriate query parser for many consumer applications. It accepts a simple syntax, and it rarely produces error messages.

The DisMax query parser supports an extremely simplified subset of the Lucene QueryParser syntax. As in Lucene, quotes can be used to group phrases, and +/- can be used to denote mandatory and optional clauses. All other Lucene query parser special characters (except AND and OR) are escaped to simplify the user experience. The DisMax query parser takes responsibility for building a good query from the user's input using Boolean clauses containing DisMax queries across fields and boosts specified by the user. It also lets the Solr administrator provide additional boosting queries, boosting functions, and filtering queries to artificially affect the outcome of all searches. These options can all be specified as default parameters for the handler in thesolrconfig.xml file or overridden in the Solr query URL.

Interested in the technical concept behind the DisMax name? DisMax stands for Maximum Disjunction. Here's a definition of a Maximum Disjunction or "DisMax" query:

A query that generates the union of documents produced by its subqueries, and that scores each document with the maximum score for that document as produced by any subquery, plus a tie breaking increment for any additional matching subqueries.

Whether or not you remember this explanation, do remember that the DisMax request handler was primarily designed to be easy to use and to accept almost any input without returning an error.

## DisMax Parameters

In addition to the common request parameter, highlighting parameters, and simple facet parameters, the DisMax query parser supports the parameters described below. Like the standard query parser, the DisMax query parser allows default parameter values to be specified in solrconfig.xml, or overridden by query-time values in the request.

| **Parameter** | **Description** |
| --- | --- |
| [q](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-TheqParameter) | Defines the raw input strings for the query. |
| [q.alt](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Theq.altParameter) | Calls the standard query parser and defines query input strings, when the q parameter is not used. |
| [qf](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Theqf(QueryFields)Parameter) | Query Fields: specifies the fields in the index on which to perform the query. If absent, defaults to df. |
| [mm](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Themm(MinimumShouldMatch)Parameter) | Minimum "Should" Match: specifies a minimum number of clauses that must match in a query. If no 'mm' parameter is specified in the query, or as a default in solrconfig.xml, the effective value of the q.op parameter (either in the query, as a default insolrconfig.xml, or from the defaultOperator option in the Schema) is used to influence the behavior. If q.op is effectively AND'ed, then mm=100%; if q.op is OR'ed, then mm=1. Users who want to force the legacy behavior should set a default value for the 'mm' parameter in their solrconfig.xml file. Users should add this as a configured default for their request handlers. This parameter tolerates miscellaneous white spaces in expressions (e.g., " 3 < -25% 10 < -3\n", " \n-25%\n ", " \n3\n "). |
| [pf](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Thepf(PhraseFields)Parameter) | Phrase Fields: boosts the score of documents in cases where all of the terms in the q parameter appear in close proximity. |
| [ps](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Theps(PhraseSlop)Parameter) | Phrase Slop: specifies the number of positions two terms can be apart in order to match the specified phrase. |
| [qs](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Theqs(QueryPhraseSlop)Parameter) | Query Phrase Slop: specifies the number of positions two terms can be apart in order to match the specified phrase. Used specifically with the qf parameter. |
| [tie](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Thetie(TieBreaker)Parameter) | Tie Breaker: specifies a float value (which should be something much less than 1) to use as tiebreaker in DisMax queries. Default: 0.0 |
| [bq](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Thebq(BoostQuery)Parameter) | Boost Query: specifies a factor by which a term or phrase should be "boosted" in importance when considering a match. |
| [bf](https://cwiki.apache.org/confluence/display/solr/The+DisMax+Query+Parser#TheDisMaxQueryParser-Thebf(BoostFunctions)Parameter) | Boost Functions: specifies functions to be applied to boosts. (See for details about function queries.) |

The sections below explain these parameters in detail.

### The q Parameter

The q parameter defines the main "query" constituting the essence of the search. The parameter supports raw input strings provided by users with no special escaping. The + and - characters are treated as "mandatory" and "prohibited" modifiers for terms. Text wrapped in balanced quote characters (for example, "San Jose") is treated as a phrase. Any query containing an odd number of quote characters is evaluated as if there were no quote characters at all.

The q parameter does not support wildcard characters such as \*.

### The q.alt Parameter

If specified, the q.alt parameter defines a query (which by default will be parsed using standard query parsing syntax) when the main q parameter is not specified or is blank. The q.alt parameter comes in handy when you need something like a query to match all documents (don't forget &rows=0 for that one!) in order to get collection-wide faceting counts.

### The qf (Query Fields) Parameter

The qf parameter introduces a list of fields, each of which is assigned a boost factor to increase or decrease that particular field's importance in the query. For example, the query below:

qf="fieldOne^2.3 fieldTwo fieldThree^0.4"

assigns fieldOne a boost of 2.3, leaves fieldTwo with the default boost (because no boost factor is specified), and fieldThree a boost of 0.4. These boost factors make matches in fieldOne much more significant than matches in fieldTwo, which in turn are much more significant than matches infieldThree.

### The mm (Minimum Should Match) Parameter

When processing queries, Lucene/Solr recognizes three types of clauses: mandatory, prohibited, and "optional" (also known as "should" clauses). By default, all words or phrases specified in the q parameter are treated as "optional" clauses unless they are preceded by a "+" or a "-". When dealing with these "optional" clauses, the mm parameter makes it possible to say that a certain minimum number of those clauses must match. The DisMax query parser offers great flexibility in how the minimum number can be specified.

The table below explains the various ways that mm values can be specified.

| **Syntax** | **Example** | **Description** |
| --- | --- | --- |
| Positive integer | 3 | Defines the minimum number of clauses that must match, regardless of how many clauses there are in total. |
| Negative integer | -2 | Sets the minimum number of matching clauses to the total number of optional clauses, minus this value. |
| Percentage | 75% | Sets the minimum number of matching clauses to this percentage of the total number of optional clauses. The number computed from the percentage is rounded down and used as the minimum. |
| Negative percentage | -25% | Indicates that this percent of the total number of optional clauses can be missing. The number computed from the percentage is rounded down, before being subtracted from the total to determine the minimum number. |
| An expression beginning with a positive integer followed by a > or < sign and another value | 3<90% | Defines a conditional expression indicating that if the number of optional clauses is equal to (or less than) the integer, they are all required, but if it's greater than the integer, the specification applies. In this example: if there are 1 to 3 clauses they are all required, but for 4 or more clauses only 90% are required. |
| Multiple conditional expressions involving > or < signs | 2<-25% 9<-3 | Defines multiple conditions, each one being valid only for numbers greater than the one before it. In the example at left, if there are 1 or 2 clauses, then both are required. If there are 3-9 clauses all but 25% are required. If there are more then 9 clauses, all but three are required. |

When specifying mm values, keep in mind the following:

* When dealing with percentages, negative values can be used to get different behavior in edge cases. 75% and -25% mean the same thing when dealing with 4 clauses, but when dealing with 5 clauses 75% means 3 are required, but -25% means 4 are required.
* If the calculations based on the parameter arguments determine that no optional clauses are needed, the usual rules about Boolean queries still apply at search time. (That is, a Boolean query containing no required clauses must still match at least one optional clause).
* No matter what number the calculation arrives at, Solr will never use a value greater than the number of optional clauses, or a value less than 1. In other words, no matter how low or how high the calculated result, the minimum number of required matches will never be less than 1 or greater than the number of clauses.
* When searching across multiple fields that are configured with different query analyzers, the number of optional clauses may differ between the fields. In such a case, the value specified by mm applies to the maximum number of optional clauses. For example, if a query clause is treated as stopword for one of the fields, the number of optional clauses for that field will be smaller than for the other fields. A query with such a stopword clause would not return a match in that field if mm is set to 100% because the removed clause does not count as matched.

The default value of mm is 100% (meaning that all clauses must match).

### The pf (Phrase Fields) Parameter

Once the list of matching documents has been identified using the fq and qf parameters, the pf parameter can be used to "boost" the score of documents in cases where all of the terms in the q parameter appear in close proximity.

The format is the same as that used by the qf parameter: a list of fields and "boosts" to associate with each of them when making phrase queries out of the entire q parameter.

### The ps (Phrase Slop) Parameter

The ps parameter specifies the amount of "phrase slop" to apply to queries specified with the pf parameter. Phrase slop is the number of positions one token needs to be moved in relation to another token in order to match a phrase specified in a query.

### The qs (Query Phrase Slop) Parameter

The qs parameter specifies the amount of slop permitted on phrase queries explicitly included in the user's query string with the qf parameter. As explained above, slop refers to the number of positions one token needs to be moved in relation to another token in order to match a phrase specified in a query.

### The tie (Tie Breaker) Parameter

The tie parameter specifies a float value (which should be something much less than 1) to use as tiebreaker in DisMax queries.

When a term from the user's input is tested against multiple fields, more than one field may match. If so, each field will generate a different score based on how common that word is in that field (for each document relative to all other documents). The tie parameter lets you control how much the final score of the query will be influenced by the scores of the lower scoring fields compared to the highest scoring field.

A value of "0.0" - the default - makes the query a pure "disjunction max query": that is, only the maximum scoring subquery contributes to the final score. A value of "1.0" makes the query a pure "disjunction sum query" where it doesn't matter what the maximum scoring sub query is, because the final score will be the sum of the subquery scores. Typically a low value, such as 0.1, is useful.

### The bq (Boost Query) Parameter

The bq parameter specifies an additional, optional, query clause that will be added to the user's main query to influence the score. For example, if you wanted to add a relevancy boost for recent documents:

|  |
| --- |
| q=cheese  bq=date:[NOW/DAY-1YEAR TO NOW/DAY] |

You can specify multiple bq parameters. If you want your query to be parsed as separate clauses with separate boosts, use multiple bq parameters.

### The bf (Boost Functions) Parameter

The bf parameter specifies functions (with optional boosts) that will be used to construct FunctionQueries which will be added to the user's main query as optional clauses that will influence the score. Any function supported natively by Solr can be used, along with a boost value. For example:

|  |
| --- |
| recip(rord(myfield),1,2,3)^1.5 |

Specifying functions with the bf parameter is essentially just shorthand for using the bq param combined with the {!func} parser.

For example, if you want to show the most recent documents first, you could use either of the following:

|  |
| --- |
| bf=recip(rord(creationDate),1,1000,1000)    ...or...  bq={!func}recip(rord(creationDate),1,1000,1000) |

## Examples of Queries Submitted to the DisMax Query Parser

All of the sample URLs in this section assume you are running Solr's "techproducts" example:

|  |
| --- |
| bin/solr -e techproducts |

Normal results for the word "video" using the StandardRequestHandler with the default search field:

http://localhost:8983/solr/techproducts/select?q=video&fl=name+score

The "dismax" handler is configured to search across the text, features, name, sku, id, manu, and cat fields all with varying boosts designed to ensure that "better" matches appear first, specifically: documents which match on the name and cat fields get higher scores.

http://localhost:8983/solr/techproducts/select?defType=dismax&q=video

Note that this instance is also configured with a default field list, which can be overridden in the URL.

http://localhost:8983/solr/techproducts/select?defType=dismax&q=video&fl=\*,score

You can also override which fields are searched on and how much boost each field gets.

http://localhost:8983/solr/techproducts/select?defType=dismax&q=video&qf=features^20.0+text^0.3

You can boost results that have a field that matches a specific value.

http://localhost:8983/solr/techproducts/select?defType=dismax&q=video&bq=cat:electronics^5.0

Another instance of the handler is registered using the qt "instock" and has slightly different configuration options, notably: a filter for (you guessed it)inStock:true).

http://localhost:8983/solr/techproducts/select?defType=dismax&q=video&fl=name,score,inStock

http://localhost:8983/solr/techproducts/select?defType=dismax&q=video&qt=instock&fl=name,score,inStock

One of the other really cool features in this handler is robust support for specifying the "BooleanQuery.minimumNumberShouldMatch" you want to be used based on how many terms are in your user's query. These allows flexibility for typos and partial matches. For the dismax handler, one and two word queries require that all of the optional clauses match, but for three to five word queries one missing word is allowed.

http://localhost:8983/solr/techproducts/select?defType=dismax&q=belkin+ipod

http://localhost:8983/solr/techproducts/select?defType=dismax&q=belkin+ipod+gibberish

http://localhost:8983/solr/techproducts/select?defType=dismax&q=belkin+ipod+apple

Just like the StandardRequestHandler, it supports the debugQuery option to viewing the parsed query, and the score explanations for each document.

http://localhost:8983/solr/techproducts/select?defType=dismax&q=belkin+ipod+gibberish&debugQuery=true

http://localhost:8983/solr/techproducts/select?defType=dismax&q=video+card&debugQuery=true