

Query Language

Introduction

The Daisy Query Language can be used to search for documents (more precisely, document variants). Queries can be used in various places:

- explicitly via the "Query Search" page
- embedded inside documents
- embedded inside navigation trees

The implementation of various Daisy features is also based on queries, such as the recent changes page or the referrers page. And of course it is possible to execute queries from your own applications, using the HTTP interface or Java API.

The query language is a somewhat SQL-like language that allows to search on various document properties (including the fields), fulltext on the part content, or a combination of those. The sort order of the results can also be defined. The resulting document list is filtered to only include documents to which the user has at least read access.

An example query, searching all documents in a collection call "mycollection":

```
select id, name where InCollection('mycollection') order by name
```

Internally, non-fulltext queries are translated to SQL and executed on the database and fulltext queries are executed by [Jakarta Lucene](#)¹.

Although the query language is somewhat SQL-like, it hides the complexity of the actual SQL-queries that are performed by the repository server on the relational database, which can quickly grow quite complex.

WARNING

Note: everytime in this document when we talk about "searching documents", this is equivalent to "searching document variants". The result of query is a set of document variants, i.e. each member of the result set is identified by a tripple (document ID, branch, language).

Query Language

General structure of a query

```
select
...
where
...
order by
...
limit x
option
...
```

The `select` and `where` parts are required, the rest is optional. Whitespace is of no importance.

1. <http://jakarta.apache.org/lucene/>

The select part

The `select` part should list one or more identifiers, separated by commas. Available identifiers are listed further on.

The where part

The `where` part should contain a conditional expression, thus an expression which tests the value of identifiers using operators, or uses some built-in functions.

Besides the identifiers listed in the table below, the operations AND and OR are supported, and parentheses can be used for grouping.

Operators & datatypes

	string	long	double	decimal	date	datetime	boolean
=	X	X	X	X	X	X	X
!=	X	X	X	X	X	X	X
<	X	X	X	X	X	X	
>	X	X	X	X	X	X	
<=	X	X	X	X	X	X	
>=	X	X	X	X	X	X	
[NOT] LIKE	X						
[NOT] BETWEEN	X	X	X	X	X	X	
[NOT] IN	X	X	X	X	X	X	
IS [NOT] NULL	X	X	X	X	X	X	X

Wildcards for `LIKE` are `_` and `%`, escape using `_` and `\%`.

All keywords such as `AND`, `LIKE`, `BETWEEN`, ... can be written in either uppercase or lowercase (but not mixed case).

If these operators are used on multi-value fields, they return true if at least one of the values of the multi-value field satisfies. See further on for a set of conditions specifically for multi-value fields.

Identifiers

The table below lists the available identifiers.

Some notes:

- identifier names are case sensitive
- non-searchable identifiers are identifiers which can only be used in the select clause of the query, not in the where clause
- the datatype symbolic means it should be a string, but the string is internally translated into another code. For example, when searching on `ownerLogin`, the given string is internally translated to a user

id, which is then used when performing the database search. This means that certain operators will not work on it (such as like, less than, greater than, ...)

- version dependent means that the searched or retrieved data is version dependent data. By default this will search in, or retrieve data from, the live version of the document, but by specifying the query option `search_last_version` (see further on) the last version can also be searched.
- the names in italic, i.e. *partTypeName*, *fieldName* and *customFieldName* must be replaced by an actual name.

name	searchable	datatype	version dependent	remarks
id	yes	long	no	
name	yes	string	yes	
branch	yes	symbolic	no	
branchId	yes	long	no	
language	yes	symbolic	no	
languageId	yes	long	no	
documentType	yes	symbolic	no	
versionId	yes	long	yes	ID of the live version, or if the query option <code>search_last_version</code> is specified, of the last version
creationTime	yes	datetime	no	
ownerId	yes	long	no	
ownerLogin	yes	symbolic	no	
ownerName	no	string	no	
summary	no	string	no	always of last published version

retired	yes	boolean	no	
private	yes	boolean	no	
lastModified	yes	datetime	no	
lastModifierId	yes	long	no	
lastModifierLogin	yes	symbolic	no	
lastModifierName	no	string	no	
variantLastModified	yes	datetime	no	
variantLastModifierId	yes	long	no	
variantLastModifierLogin	yes	symbolic	no	
variantLastModifierName	no	string	no	
%partTypeName.mimeType	yes	string	yes	
%partTypeName.size	yes	long	yes	
%partTypeName.contentType	yes	xml	yes	only works for part types for which the flag 'daisy html' is set to true, and additionally the actual part must have the mime type 'text/xml'
versionCreationTime	yes	datetime	yes	
versionCreatorId	yes	long	yes	
versionCreatorLogin	yes	symbolic	yes	
versionCreatorName	yes	string	yes	
versionState	yes	symbolic	yes	'draft' or 'publish'
totalSizeOfParts	yes	long	yes	sum of the size of all parts in document
versionStateLastModified	yes	datetime	yes	
lockType	yes	symbolic	no	'pessimistic' or 'warn'
lockTimeAcquired	yes	datetime	no	
lockDuration	yes	long	no	(in milliseconds)
lockOwnerId	yes	long	no	
lockOwnerLogin	yes	symbolic	no	
lockOwnerName	no	string	no	
collections	yes	symbolic	no	The collections (the names of the

				collections) the document belongs too. Behaves the same as a multi-value field with respect to applicable search conditions.
<code>collections.valueCount</code>	yes	symbolic	no	The number of collections a document belongs too.
<code>\$fieldName</code>	yes		yes	datatype depends on field type
<code>\$fieldName.valueCount</code>	yes	long	yes	Useful for multi-value fields. Searching for a value count of 0 does not work, use the "is null" condition instead.
<code>#customFieldName</code>	yes	string	no	

Literals

String literals

Strings (text) should be put between single quotes, the single quote is escaped by doubling it, for example:

```
' 't is mooi weer vandaag'
```

Numeric literals

These consists of digits (0-9), the deicmal separator is a dot (.).

Numeric literals can be put between single quotes like strings, but it is not required to do so.

Date & datetime literals

Date format: 'YYYY-MM-DD'

Datetime format: 'YYYY-MM-DD HH:MM:SS'

Special conditions for multi-value fields

```
$fieldName has all (value1, value2, value3, ...)
```

Tests that the multi-value field has all the specified values (and possibly more).

```
$fieldName has exactly (value1, value2, value3, ...)
```

Tests that the multi-value field has all the specified values, and none more. The order is not important.

```
$fieldName has some (value1, value2, value3, ...)  
or  
$fieldName has any (value1, value2, value3, ...)
```

`has some` and `has any` are synonyms. They test that the multi-value field has at least one of the specified values.

```
$fieldName has none (value1, value2, value3, ...)
```

Tests that the multi-value field has none of the specified values.

In addition to these conditions, you can use `is null` and `is not null` to check if a document has a certain (multi-value) field. The special sub-identifier `$fieldName.valueCount` can be used to check the number of values a multi-value field has.

Other special conditions

InCollection

```
InCollection('collectionname' [, collectionname, collectionname])
```

Searches documents contained in at least one of the specified collections. To search documents that occur in multiple collections (thus in the intersection of those collections), use the function `InCollection` multiple times with AND in between: `InCollection('collection1')` and `InCollection('collection2')`. This also works for OR but in that case it is more efficient to give the collections as arguments to one `InCollection` call.

Instead of the `InCollection` condition, you can use the `collections` identifier in combination with the multi-value field search conditions such as `has some`, `has all` or `has none` for more powerful search possibilities. The `InCollection` condition predates the existence of multi-value fields, but remains supported.

LinksTo, LinksFrom, LinksToVariant, LinksFromVariant

```
LinksTo(documentId, inLastVersion, inLiveVersion)  
LinksFrom(documentId, inLastVersion, inLiveVersion)  
LinksToVariant(documentId, branch, language, inLastVersion, inLiveVersion)  
LinksFromVariant(documentId, branch, language, inLastVersion, inLiveVersion)
```

Searches documents which link to or from the specified document (or document variant). The other two parameters, `inLastVersion` and `inLiveVersion`, are interpreted as booleans: 0 is false, any other (numeric) value is true.

If `inLastVersion` is true, only documents whose last version link to the specified document are included.

If `inLiveVersion` is true, only documents whose live version link to the specified document are included.

If both parameters are true or both are false, all documents are returned for which either the last or live version link to the specified document.

IsLinked, IsNotLinked

```
IsLinked()  
IsNotLinked()
```

`IsLinked()` evaluates to true for any document which is linked by other documents, `IsNotLinked()` evaluates to true for any document that is not linked from any other document (thus not reachable by following links in documents or the navigation tree).

HasPart

```
HasPart('partTypeName')
```

Searches documents which have a part of the specified part type. This search is version-dependent.

HasPartWithMimeType

```
HasPartWithMimeType('some mimetype')
```

Searches documents having a part with the given mime type. This search is version-dependent. This uses a 'like' condition, thus the % wildcard can be used in the parameter. For example, to search all images:
`HasPartWithMimeType('image/%')`

DoesNotHaveVariant

```
DoesNotHaveVariant(branch, language)
```

Searches documents that do not have the specified variant. See also the page on [variants](#)² for more information.

Full text queries

For full text queries, the `where` part takes a special form. There are two possibilities: either only a full text search is performed, or the `fulltext` query is further restricted using 'normal' conditions. The two possible forms are:

```
... where FullText('word')  
or  
... where FullText('word') AND <other conditions>  
for example:  
... where FullText('word') AND $myfield = 'abc' AND InCollection('mycollection')
```

Note that the combining operator between the `FullText` condition and other conditions is always `AND`, thus the result of the full text query is further refined. The further conditions can of course be of any complexity, and can thus again contain `OR`.

If no `order by` clause is included when doing a full text query, the results are ordered according to the score assigned by the `fulltext` search engine.

2. [daisy:155-cd \(Variants\)](#)

The parameter of the `FullText (. . .)` function is a query which is passed on to the full text engine, in our case Lucene. See [here](#)³.

The `FullText ()` function can have 3 additional parameters which indicate if the search should be performed on the document name, document content or field content. By default, all three are searched. These parameters should be numeric: 0 indicates false, and any other value true.

For example:

```
FullText('word', 1, 0, 0)
```

Searches for 'word', but only in the document name.

Additionally, you can specify a branch and language as parameters to the `FullText` function, to specify that only documents of that branch/language should be searched. Thus the full syntax of the `FullText` function is:

```
FullText(lucene query, searchInName, searchInContent, searchInFields, branch, language)
```

Specifying the branch and language as part of the `FullText` function is more more efficient then using:

```
FullText(lucene query) and branch = 'my_branch' and language = 'my_language'
```

The order by part

The `order by` part is optional.

The `order by` part contains a comma separated listing of identifiers, each of these optionally followed by `ASC` or `DESC` to indicate ascending (the default) or descending order. The identifiers listed here have no connection with those in the `select`-part, i.e. it does not have to be subset of those.

"null" values are put at the end (when using `ASC` order).

The limit part

This can be used to limit the number of results returned from a query. This part is optional.

The option part

The `option` part allows to specify options that influence the execution of the query. The options are defined as:

```
option_name = 'option_value' (, option_name = 'option_value')*
```

Supported options:

name	value	default
include_retired	true/false	false
search_last_version	true/false	false
style_hint	(anything)	(empty)

3. <http://jakarta.apache.org/lucene/docs/queryparsersyntax.html>

include_retired is used to indicate that retired documents should be included in the result (by default they are not).

search_last_version is used to indicate that the last version of metadata should be searched and retrieved, instead of the live version. When using this, documents that do not have a live version will also be included in the query result (otherwise they are not included). Full text searches are always performed on the live data, regardless of whether this option is specified.

style_hint is used to supply a hint to the publishing layer for how the result of the query should be styled. The repository server does not do anything more than add the value of this option as an attribute on the generated XML query results (<searchResult styleHint="my hint" ...). It is then up to the publishing layer to pick this up and do something useful with it. For how this is handled in the DaisyWiki, see the page on [Query Styling](#)⁴.

Example queries

List of all documents

```
select id, name where true
```

Search on document name

```
select id, name where name like 'p%' order by creationTime desc limit 10
```

Show the 10 largest documents

```
select id, name, totalSizeOfParts where true order by totalSizeOfParts desc limit 10
```

Show documents of which the last version has not yet been published

```
select id, name, versionState, versionCreationTime  
where versionState = 'draft' option search_last_version = 'true'
```

Overview of all locks

```
select id, name, lockType, lockOwnerName, lockTimeAcquired, lockDuration  
where lockType is not null
```

All documents having a part containing an image

```
select id, name where HasPartWithMimeType('image/%')
```

4. daisy:54-cd (Query Styling)