

Query Language

Introduction

The Daisy Query Language can be used to search for documents (more precisely, document variants). In the Daisy Wiki, 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 Wiki 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 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 relational database while 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: every time 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 triple (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.

The select part

The `select` part should list one or more value expressions, separated by commas. A value expression can be an identifier, a literal or a function call. This is described in more detail further on.

The where part

The `where` part should contain a predicate expression, thus an expression which tests the value of *value expressions* using operators, or uses some built-in conditions.

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

Operators & data types

	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.

Normally the comparison operators work on values of the same type, though there is some relaxation for compatible types, e.g. it is possible to compare between all numeric types, and between the date and datetime types.

Value expressions

A value expression is:

- an identifier (= some property of a document, see list further on)
- a literal (= a fixed value such as a string, a number or a date)
- a function call (whose arguments can be identifiers, literals or function calls)

A function call usually has the following form:

```
functionName(arg1, arg2, ...)
```

However, for the basic mathematical functions (addition, subtraction, multiplication and division) "infix" notation is used instead, using the symbols `+`, `-`, `*` and `/`. Parentheses can be used to influence the order of the operations.

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 or will be of little meaning (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*, *fieldTypeName* and *customFieldName* must be replaced by an actual name.

name	searchable	datatype	version dependent	remarks
id	yes	string	no	
namespace	yes	string	no	The namespace part of the document ID
name	yes	string	yes	
branch	yes	symbolic	no	
branchId	yes	long	no	
language	yes	symbolic	no	
languageId	yes	long	no	
link	yes	link	no	The current document variant as a link. This is useful for comparison with link type fields. For example <code>\$someLinkField = link</code> to find documents which link to themselves in a certain field, or <code>\$someLinkField = ContextDoc(link)</code> to find documents which link to the context document.
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	yes	string	no	
%partTypeName.mimeType	yes	string	yes	
%partTypeName.size	yes	long	yes	
%partTypeName.content	no	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.
collections.valueCount	yes	symbolic	no	The number of collections a document belongs too.

<i>\$fieldName</i>	yes		yes	datatype depends on field type
<i>\$fieldName.valueCount</i>	yes	long	yes	Useful for multi-value fields. Searching for a value count of 0 does not work, use the "is null" condition instead.
<i>\$fieldName.documentId</i>	yes	string	yes	These special field sub-identifiers are only supported on fields of the type "link". For link field types, the <i>\$fieldName</i> identifier checks on the document ID, while these identifiers can be used to check on the branch and language.
<i>\$fieldName.branch</i>	yes	symbolic	yes	
<i>\$fieldName.branchId</i>	yes	long	yes	
<i>\$fieldName.language</i>	yes	symbolic	yes	
<i>\$fieldName.languageId</i>	yes	long	yes	
<i>\$fieldName.namespace</i>	yes	string	yes	
<i>#customFieldName</i>	yes	string	no	
score	no	double	no	The score of a document after doing a full text search. This score ranges from 0-1. When this identifier is used without the FullText() function it will just return 0.
variants	yes	link	no	All available variants of the document. Behaves the same as a multi-value field with respect to applicable search conditions. When searching, specify 'branch:lang' as literal, e.g. has all ('main:en')
variants.valueCount	yes	long	no	The number of variants of a document (= all existing branch-language combinations).
liveMajorChangeVersionId lastMajorChangeVersionId	yes	long	no	Last, or last-up-to-live, version that has major changes. Can be null.
versionComment	yes	string	no	Version comment, can be null.
versionChangeType	yes	string	yes	major or minor
referenceLanguage	yes	string	yes	Reference language name, can be null.
referenceLanguageId	yes	long	no	Reference language ID, can be null.
syncedWith	yes	link	no	The 'synced with' pointer of the current version (live or last version, according to the query option

				search_last_version). This is a dereferenceable link field.
syncedWith.language	yes	string	yes	
syncedWith.languageId	yes	long	no	
syncedWith.versionId	yes	long	no	

Addressing components of multivalue and hierarchical field identifiers

For multivalue and hierarchical field identifiers, an index-notation is supported using square brackets.

For multivalue fields:

```
$SomeField[index]
```

For multivalue hierarchical fields:

```
$SomeField[index][index]
```

For non-multivalue hierarchical fields, or if you only want to specify an index for the hierarchical value, you can use:

```
$SomeField[*][index]
```

The index is 1-based. You can address elements starting from the end by using negative indexes, e.g. -1 for the last element in a multivalue or hierarchy path.

In case you are using a sub-field identifier, it should be put after the square brackets:

```
$SomeField[index].documentId
```

Specifying an out-of-range index doesn't give an error, but simply finds/returns nothing.

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 decimal 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'

Link literals

When searching on fields of type "link", the link should be specified as:

```
'daisy:docid'           (assumes branch main and language default)
'daisy:docid@branch'    (assumes language default)
```

```
'daisy:docid@branch:lang' (branch can be left blank which defaults to main branch)
```

Branch and language can be specified either by name or ID.

So a search condition could be for example:

```
$someLinkField = 'daisy:35'
```

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.

Searching on hierarchical fields

matchesPath

For searching on hierarchical fields, a special `matchesPath` condition is available. It takes as argument an expression in which the elements of the hierarchical path are separated by a slash. For example, a basic usage is:

```
$fieldName matchesPath('/A/B/C')
$fieldName matchesPath('A/B/C') -> the initial slash is optional
```

This would return all documents for which the hierarchical field has as value the path `A/B/C`.

The values should be entered using the correct literal syntax corresponding to the type of the field. For example, for link type fields, you would use:

```
matchesPath('daisy:10-FOO/daisy:11-FOO')
```

It is possible to use wildcards (placeholders) in the expression, namely `*` and `**`. One stars (`*`) matches one path part. Two stars (`**`) matches multiple path parts. Two stars can only be used at the very start or at the very end of the expression (not at both ends at the same time). Some examples to give an idea of what's possible:

```
$fieldName matchesPath('/A/*')
$fieldName matchesPath('/A/**')
```

```
$fieldName matchesPath('/A/*/B')
$fieldName matchesPath('/*/*/*') -> finds all hierarchical paths of length 3
$fieldName matchesPath('/*/*/**') -> finds all hierarchical paths of at least length 3
$fieldName matchesPath('/A/**') -> finds all paths of any length starting on A
    thus e.g. A/B, A/B/C or A/B/C/D.
$fieldName matchesPath('**/A') -> finds all paths ending on A
```

The argument of `matchesPath` should be a string, but doesn't have to be a literal. Some examples:

```
$fieldName matchesPath($anotherField, '/*/*')

$fieldName matchesPath(Concat(ContextDoc(link), '/*/*'))

An example taken from Daisy's own knowledge base:
$fieldName matchesPath(String(ReversePath(
    GetLinkPath('KnowledgeBaseCategoryParent', 'true', ContextDoc(link))))))
```

Multi-value hierarchical fields

The `matchesPath` condition can also be used to search on multi-value hierarchical fields, in which case it will evaluate to true if at least one of the values of the multi-value field matches the path expression.

The special multi-value conditions such as 'has all', 'has some', etc. can also be used. There is no special syntax to specify hierarchical path literals in the query language, but they can be entered by using the `Path` function. For example:

```
$fieldName has all ( Path('/A/B/C'), Path('/X/Y/Z') )
```

The hierarchical paths specified using the `Path` function do not support wildcards.

Equals operator

When using the equals operator (`=`) or other binary operators with hierarchical fields, it will evaluate to true as long as there is one element in the hierarchy path which has the given value. For example, `$MyField = 'b'` will match a field whose value is `"/a/b/c"`. This is similar to the behaviour of this operator for multivalued fields.

Link dereferencing

When an expression returns a link as value (most often this is in the form a link field identifier, e.g. `$SomeLinkField`), then it is possible to 'walk through' this link to access properties of the linked-to document. This is known as link dereferencing.

The link dereferencing operator is written as `"=>"`. Notations for dereferencing in other languages are sometimes dot (`.`) or `"->"`, however since dash is a valid character in identifiers in Daisy, and dot is already used to access 'sub-field identifiers' (like `#SomePart.mimeType`), these could not be used.

```
[link expression]=>[identifier]
```

A practical example:

```
select name, $SomeLinkField=>name where $SomeLinkField=>name like 'A%' order by
    $SomeLinkField=>name
```

As shown in this example, the link dereferencing operator works in the select, where and order by parts of the query.

Link dereferencing can work multiple levels deep, e.g.

```
$SomeLink=>$SomeOtherLink=>name
```

If documents are linked together with the same type of field, this could of course be something like:

```
$SomeLink=>$SomeLink=>$SomeLink=>name
```

When dereferencing a link in the where-clause of the query, but one does not have access to the dereferenced document, then the evaluation of the where clause will be considered as 'false', e.g. the row will be excluded from the result set, since without access to the document it is not possible to know if it would evaluate to 'true'. Accessing non-accessible values in the select or order-by clauses will return a 'null' value.

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 [, linktypes])
LinksFrom(documentId, inLastVersion, inLiveVersion [, linktypes])
LinksToVariant(documentId, branch, language, inLastVersion, inLiveVersion [, linktypes])
LinksFromVariant(documentId, branch, language, inLastVersion, inLiveVersion [, linktypes])
```

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.

The optional parameter `linktypes` is a string containing a comma or whitespace separated list of the types of links to include, which is one or more of: `inline`, `out_of_line`, `image`, `include`, `field` or `other`.

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, the navigation tree, or linked by the content of other parts on which link extraction is performed).

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.

A "HasVariant" condition does not exist, however the same can be achieved using the newer `variants` identifier on which you can apply all multi-value conditions, for example:

```
variants has all ("main:en", "main:nl")
```

Similarly, the `DoesNotHaveVariant` can be written as:

```
variants has none ("main:en")
```

LangInSync, LangNotInSync

These conditions are useful for translation management.

Syntax:

```
LangInSync()           same as LangInSync('last')
LangInSync('live')

LangNotInSync()       same as LangNotInSync('last')
LangNotInSync('live')
```

These conditions test the relationship between the 'synced with' pointer of current version (normally the live version, unless the `search_last_version` option has been set) and the [last or live major change version ID](#)³ of the variant they point to.

`LangInSync` will evaluate to true for a document when:

- the synced with pointer is not null
- and the last or live major change version id of the synced-with variant is not null (otherwise there's nothing we can be in sync with)
- and the synced with version is greater than or equal to the last/live major change version ID

`LangNotInSync` will evaluate to true for a document when:

- the synced with pointer is null
- or the synced with pointer is not null, and:
 - the last or live major change version id of the synced-with variant is not null

- the synced with version is less than the last/live major change version ID

The following table clarifies some more behavioral details you might wonder about.

Evaluation result in case of	LangInSync	LangNotInSync
synced-with is null	false	true
synced with is not null but the last / live major change version of the synced-with variant is null	both false, can't be in-sync nor not-in-sync with something which has never had a major change (or never had a live version)	
synced-with is not null and synced-with variant does not exist	cannot occur, the repository enforces that the synced-with pointer points to an existing {lang-variant, version}	

In pseudo-query-language code the behaviour could be described as follows:

```
LangInSync() is similar to:

    syncedWith is not null
    and syncedWith=>lastMajorChangeVersionId is not null
    and syncedWith.versionId >= syncedWith=>lastMajorChangeVersionId

LangNotInSync() is similar to:

    syncedWith is null
    or syncedWith=>lastMajorChangeVersionId is null
    or syncedWith.versionId < syncedWith=>lastMajorChangeVersionId
```

One smallish difference between these expressions and Lang(Not)InSync() is that dereferencing the syncedWith identifier will only work when you have access to the document variant that it points to, while Lang(Not)InSync() bypasses these checks.

Often you will want to check if the last version is synced with the last major changes, therefore add the query option search_last_version.

Suppose you have documents with 'en' as reference language and 'fr' as one of the translated variants, than you can search for not-synced 'fr' translation like this:

```
select
  id, name
where
  language = 'fr'
  and referenceLanguage = 'en'
  and branch = 'main'
  and LangNotInSync()
option
  search_last_version = 'true'
```

These results will not include 'fr' translations which do not exist at all, for this you could use:

```
select
  id, name
where
  variants has none ('main:fr')
  and language = 'en'
  and referenceLanguage = 'en'
  and branch = 'main'
```

ReverseLangInSync, ReverseLangNotInSync

These conditions test if some other language variant is synced-with the current variant. This is similar to LangInSync and LangNotInSync, but in the reverse direction. Instead of searching for translated variants of which synced-with pointer is or is not in sync with the last major change version of the variant the synced-

with points to, here we search for variants for which there should be another variant which has a synced-with pointer pointing to their last major change version.

An important use case for these conditions is the [translation export](#)⁴, where you want to be able to export (and thus select) the reference language variants for which some other translation is not yet in sync.

Syntax:

```
ReverseLangInSync(language, 'live/last')
ReverseLangNotInSync(language, 'live/last')
```

The language argument specifies the language variant for which you want to check if it is synced-with the current language variant. The 'live/last' is optional (default: last) and specifies whether it should look at the synced-with of the last or live version. This is different from the 'live/last' argument of Lang(Not)InSync, where it applies to the major change version. Here, the major change version is taken from the version in which the search is performed: default live, unless the option search_last_version is specified.

The following table clarifies some more behavioral details you might wonder about.

Evaluation result in case of	ReverseLangInSync	ReverseLangNotInSync
variant in specified language does not exist	false	false
other language variant synced-with is null	false	true
no live / last major change version in current variant	both false, can't be in-sync nor not-in-sync with something which has never had a major change (or never had a live version)	
variant in specified language does not have a live version (when specifying 'live' as argument)	false	true

As an example, suppose you have French language variants which are synced with English language variants. Now you want to search for English variants for which the French variants are not up to date, or for which the French translation does not exist. You could do so with a query like:

```
select
  id, branch, language, name
where
  branch = 'main'
  and language = 'en'
  and ( ReverseLangNotInSync('fr', 'last') or variants has none('main:fr') )
option
  search_last_version = 'true'
```

Usually you will also add a condition `referenceLanguage = 'en'` to include only documents which are under translation management.

Functions

The following functions can be used in value expressions.

String functions

Concat

Syntax:

```
Concat(value1, value2, ...) : string
```

Concatenates multiple strings. If the arguments are not strings, they are converted to a string using the same logic as the String function.

Length

Syntax:

```
Length(string) : long
```

Returns the length of its string argument.

Left

Syntax:

```
Left(string, length) : string
```

Returns 'length' leftmost characters from the string. If 'length' is larger than the string, the whole string is returned. If length is 0, an empty string is returned.

Right

Syntax:

```
Right(string, length) : string
```

Returns 'length' rightmost characters from the string. If 'length' is larger than the string, the whole string is returned. If length is 0, an empty string is returned.

Substring

Syntax:

```
Substring(string, position, length) : string
```

Returns a string formed by taking 'length' characters from the string at the specified position. The 'length' argument is optional, if not specified, it will go till the end of the input string. The 'position' argument starts at 1 for the first character.

UpperCase

Syntax:

```
UpperCase(string) : string
```

LowerCase

Syntax:

```
LowerCase(string) : string
```

String

Syntax:

```
String(value) : string
```

Converts its argument to a string.

Some of the behaviours:

- date and datetime values are formatted using the syntax for literals
- link values are formatted as "daisy:" links

- hierarchical values are formatted with slashes between the elements of the hierarchical path (e.g. "/A/B/C")
- multivalued values are formatted like this: [A,B,C]

Date and datetime functions

CurrentDate

Syntax:

```
CurrentDate(spec?) : date
```

Returns the current date.

The optional spec argument allows to specify an offset to the current date. It is a string with the following syntax:

```
+/- <num> (days|weeks|months|years)
```

For example:

```
CurrentDate('- 7 days')
```

CurrentDateTime

Syntax:

```
CurrentDateTime(spec?) : date
```

Returns the current datetime.

The optional spec argument allows to specify an offset to the current datetime. It is a string with the following syntax:

```
+/- <num> (seconds|minutes|hours|days|weeks|months|years)
```

For example:

```
CurrentDateTime('- 3 hours')
```

Year, Month, Week, DayOfWeek, DayOfMonth, DayOfYear

These functions all take a date or datetime as argument, and return a long value.

DayOfWeek returns a value in the range 1-7, where 1 is Sunday.

For the Week function, the first week of the year is the first week containing a Sunday.

RelativeDate, RelativeDateTime

These functions take one string argument consisting of 3 words, each one taken from the following groups:

```
start    this    week
end      last   month
         next   year
```

So for example:

```
RelativeDate('start this month')
```

returns a date set to the first day of the current month.

Numeric functions

+, -, * and /

The basic mathematical operations.

Random

Returns a pseudo-random double value greater than or equal to 0 and less than or equal to 1.

Mod

Syntax:

```
Mod(number1, number2)
```

Abs, Floor, Ceiling

These functions all take one number as argument.

Round

Syntax:

```
Round(number, scale)
```

Rounds the given number to have at most *scale* digits to the right of the decimal point.

Special

ContextDoc

Syntax:

```
ContextDoc(expression [, position])
```

In some cases a *context document* is available when performing a query. For example, when a query is embedded inside a document, that document serves as the context document. It is possible to evaluate expressions on this context document by use of this ContextDoc function. The optional position argument allows to climb up in the stack of context documents (which is available in publisher requests).

Examples:

```
ContextDoc(id) -- the id of the context document
ContextDoc($someField) -- the value of a field of the context document
ContextDoc(Concat(name, ' ', $someField))
```

UserId

Returns the ID of the current user (= the user executing the query).

```
UserId() -> function takes no arguments
```

Path

Converts its argument to a hierarchical path literal. This function is useful because there is no special query language syntax for entering hierarchical path literals. The argument should be a slash-separated hierarchical path, e.g.:

```
Path('/A/B/C')
Path('A/B/C') -> the initial slash is optional
```

GetLinkPath

Syntax:

```
GetLinkPath(linkFieldName, includeCurrent, linkExpr)
```

Returns a hierarchical path formed by following a chain of documents linked through the specified link field.

The optional boolean argument `includeCurrent` indicates whether the current document should be part of the hierarchical path.

The optional `linkExpr` argument can be used to specify the start document, if it is not the current document.

Note that this expression can only be used in the `select` part of queries, or in the `where` clause if it is evaluated before performing the search. For example, as argument of `matchesPath`.

ReversePath

Reverses the order of the elements in a hierarchical path. For example useful in combination with `GetLinkPath`.

Syntax:

```
ReversePath(hierarchical-path)
```

Link

Links to a specific document. Not to be confused with the link identifier.

Syntax:

```
link(documentId, branch, language)
```

Full text queries

FullText() function

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`.

NOTE

The `FullText` clause needs to be the first after the word "where", it cannot appear at arbitrary positions in the `where`-clause.

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. See also the 'score' identifier and the order by part.

The parameter of the `FullText(. . .)` function is a query which is passed on to the full text engine, in our case Lucene. See [here](#)⁵.

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'
```

FullTextFragment() function

If you wish to have contextualized text fragments of the sought after terms. This function should be used in the `select` part of the query. By default this function will only return the first text fragment found. The fragments are returned as xml which has the following structure :

```
<html>
  <body>
    <div class="fulltext-fragment">
      ... full text fragment ... <span class="fulltext-hit">the term</span> ... more
    text ...
    </div>
    ...
  </body>
</html>
```

Usage of the function when you only wish to receive one fragment (default) :

```
select FullTextFragment() where FullText('word')
```

If you wish to have more text fragments you can specify the amount of fragments as a function parameter.

```
select FullTextFragment(5) where FullText('word')
```

NOTE

This function will only return fragments from the content of the document. This means that context from document name or fields will not appear in the result.

The order by part

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

When sorting ascending, "null" values are put at the end.

The `order by` clause is optional. If no `order by` is specified, the results are not sorted in a particular way, except when there is a `FullText` condition, in that case the results are sorted on the full-text score. Explicitly sorting on full-text score is also possible, using the score identifier.

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)
annotate_link_fields	true/false	true
chunk_offset	an integer (offset of the first row is 1)	1
chunk_length	an integer	N/A

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](#)⁶.

annotate_link_fields indicates whether selected fields of type "link" should be annotated with the document name of the document pointed to by the link. If you don't need this, you can disable this to gain some performance.

chunk_offset and **chunk_length** allow to retrieve a subset of the query results. This is useful for paged display of the query results.

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/%')
```

Order documents randomly

```
select name where true order by Random()
```

Documents ordered by length of their name

```
select name, Length(name) where true order by Length(name) DESC
```

1. <http://jakarta.apache.org/lucene/>
2. /daisy-docs-2_3/373-daisy/155-daisy.html
3. /daisy-docs-2_3/373-daisy/17-daisy.html#dsy17-daisy_major_change_version
4. /daisy-docs-2_3/impexp/538-daisy.html
5. <http://jakarta.apache.org/lucene/docs/queryparsersyntax.html>
6. /daisy-docs-2_3/374-daisy/54-daisy.html