Lucene - Query parser syntax

Overview

Although Lucene provides the ability to create your own queries through its API, it also provides a rich query language through the Query Parser, a lexer which interprets a string into a Lucene Query using JavaCC.

Generally, the query parser syntax may change from release to release. This page describes the syntax as of the current release. If you are using a different version of Lucene, please consult the copy of docs/queryparsersyntax.html that was distributed with the version you are using.

Before choosing to use the provided Query Parser, please consider the following:

1. If you are programmatically generating a query string and then parsing it with the query parser then you should seriously consider building your queries directly with the query API. In other words, the query parser is designed for human-entered text, not for program-generated text.
2. Untokenized fields are best added directly to queries, and not through the query parser. If a field's values are generated programmatically by the application, then so should query clauses for this field. An analyzer, which the query parser uses, is designed to convert human-entered text to terms. Program-generated values, like dates, keywords, etc., should be consistently program-generated.
3. In a query form, fields which are general text should use the query parser. All others, such as date ranges, keywords, etc. are better added directly through the query API. A field with a limit set of values, that can be specified with a pull-down menu should not be added to a query string which is subsequently parsed, but rather added as a TermQuery clause.

https://lucene.apache.org/core/9_6_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#Overview

Terms

A query is broken up into terms and operators. There are two types of terms: Single Terms and Phrases.

A Single Term is a single word such as "test" or "hello".

A Phrase is a group of words surrounded by double quotes such as "hello dolly".

Multiple terms can be combined together with Boolean operators to form a more complex query (see below).

Note: The analyzer used to create the index will be used on the terms and phrases in the query string. So it is important to choose an analyzer that will not interfere with the terms used in the query string.

Fields

Lucene supports fielded data. When performing a search you can either specify a field, or use the default field. The field names and default field is implementation specific.

You can search any field by typing the field name followed by a colon ":" and then the term you are looking for.

As an example, let's assume a Lucene index contains two fields, title and text and text is the default field. If you want to find the document entitled "The Right Way" which contains the text "don't go this way", you can enter:

```
title:"The Right Way" AND text:go
```

or

```
title:"The Right Way" AND go
```

Since text is the default field, the field indicator is not required.

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

```
title:The Right Way
```

Will only find "The" in the title field. It will find "Right" and "Way" in the default field (in this case the text field).

Term Modifiers

Lucene supports modifying query terms to provide a wide range of searching options.

Wildcard Searches

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

To perform a single character wildcard search use the "?" symbol.

To perform a multiple character wildcard search use the "*" symbol.

The single character wildcard search looks for terms that match that with the single character replaced. For example, to search for "text" or "test" you can use the search:

```
te?t
```

Multiple character wildcard searches looks for 0 or more characters. For example, to search for test, tests or tester, you can use the search:

```
test*
```

You can also use the wildcard searches in the middle of a term.

```
te*t
```

Note: You cannot use a * or ? symbol as the first character of a search.

Regular Expression Searches

Lucene supports regular expression searches matching a pattern between forward slashes "/". The syntax may change across releases, but the current supported syntax is documented in the `RegExp` class. For example to find documents containing "moat" or "boat":

```
/[mb]oat/
```

Fuzzy Searches

Lucene supports fuzzy searches based on Damerau-Levenshtein Distance. 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.

An additional (optional) parameter can specify the maximum number of edits allowed. The value is between 0 and 2, For example:

```
roam~1
```

The default that is used if the parameter is not given is 2 edit distances.

Previously, a floating point value was allowed here. This syntax is considered deprecated and will be removed in Lucene 5.0.

Proximity Searches

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
```

Range Searches

Range Queries allow one to match documents whose field(s) values are between the lower and upper bound specified by the Range Query. Range Queries can be inclusive or exclusive of the upper and lower bounds. Sorting is done lexicographically.

```
mod_date:[20020101 TO 20030101]
```

This will find documents whose mod_date fields have values between 20020101 and 20030101, inclusive. Note that Range Queries are not reserved for date fields. You could also use range queries with non-date fields:

```
title:{Aida TO Carmen}
```

This will find all documents whose titles are between Aida and Carmen, but not including Aida and Carmen.

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

Boosting a Term

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
```

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)

Boolean Operators

Boolean operators allow terms to be combined through logic operators. Lucene supports AND, "+", OR, NOT and "-" as Boolean operators(Note: Boolean operators must be ALL CAPS).

OR

The OR operator is the default conjunction operator. This means that if there is no Boolean operator between two terms, the OR operator is used. The OR operator links two terms and finds a matching document if either of the terms exist in a document. This is equivalent to a union using sets. The symbol || can be used in place of the word OR.

To search for documents that contain either "jakarta apache" or just "jakarta" use the query:

```
"jakarta apache" jakarta
```

or

```
"jakarta apache" OR jakarta
```

AND

The AND operator matches documents where both terms exist anywhere in the text of a single document. This is equivalent to an intersection using sets. The symbol && can be used in place of the word AND.

To search for documents that contain "jakarta apache" and "Apache Lucene" use the query:

```
"jakarta apache" AND "Apache Lucene"
```

+

The "+" or required operator requires that the term after the "+" symbol exist somewhere in a the field of a single document.

To search for documents that must contain "jakarta" and may contain "lucene" use the query:

```
+jakarta lucene
```

NOT

The NOT operator excludes documents that contain the term after NOT. This is equivalent to a difference using sets. The symbol ! can be used in place of the word NOT.

To search for documents that contain "jakarta apache" but not "Apache Lucene" use the query:

```
"jakarta apache" NOT "Apache Lucene"
```

Note: The NOT operator cannot be used with just one term. For example, the following search will return no results:

```
NOT "jakarta apache"
```

-

The "-" or prohibit operator excludes documents that contain the term after the "-" symbol.

To search for documents that contain "jakarta apache" but not "Apache Lucene" use the query:

```
"jakarta apache" -"Apache Lucene"
```

Grouping

Lucene supports using parentheses to group clauses to form sub queries. This can be very useful if you want to control the boolean logic for a query.

To search for either "jakarta" or "apache" and "website" use the query:

```
(jakarta OR apache) AND website
```

This eliminates any confusion and makes sure you that website must exist and either term jakarta or apache may exist.

Field Grouping

Lucene supports using parentheses to group multiple clauses to a single field.

To search for a title that contains both the word "return" and the phrase "pink panther" use the query:

```
title:(+return +"pink panther")
```

Escaping 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
```

Java classes

Interface Summary

Interface	Description
QueryParserConstants	Token literal values and constants.

Class Summary

Class	Description
MultiFieldQueryParser	A QueryParser which constructs queries to search multiple fields.
QueryParser	This class is generated by JavaCC.
QueryParserBase	This class is overridden by QueryParser in QueryParser.jj and acts to separate the majority of the Java code from the .jj grammar file.
QueryParserTokenManager	Token Manager.
Token	Describes the input token stream.

Enum Summary

Enum	Description
QueryParser.Operator	The default operator for parsing queries.

Exception Summary

Exception	Description
ParseException	This exception is thrown when parse errors are encountered.

Error Summary

Error	Description
TokenMgrError	Token Manager Error.

Revision #2
Created 27 September 2025 09:27:46 by Sion Vives
Updated 23 October 2025 16:37:05 by Sion Vives