

LuceneCaveats

A list of classic Lucene mistakes. While these solutions are all documented in one place or another, they won't suffer from additional repetition. See also [HowTo](#) for more targeted tips.

Indexing

Luke is your friend

Luke is an invaluable tool to learn what actually went in your index. If it's not in the index, you can't query it. [Luke](#)

Use the same analyzer for indexing and querying

Make sure you use the same [Analyzer](#) class when building your index and later querying against that index. Analysis dictates what goes into your index and how. The [QueryParser](#) needs this information to generate the proper queries.

Dissimilar or incompatible analyzers lead to mysterious search behavior. See: [LuceneFAQ](#), "Why am I getting no hits / incorrect hits?".

Documents are truncated by default

The indexer by default truncates documents to `IndexWriter.DEFAULT_MAX_FIELD_LENGTH` or 10,000 terms in Lucene 2.0.

Rule of thumb: an average page of English text contains about 250 words. (Source: [Google Answers](#).) This means only about 40 pages are indexed by default. If any of your documents are longer than this (and you want them indexed in full), you should raise the limit with [IndexWriter.setMaxFieldLength\(\)](#).

Stopwords are removed

[StandardAnalyzer](#) (the most commonly recommended analyzer) does not index "stopwords". Stopwords are common English words such "the", "a", etc. – the default list is

`StopAnalyzer.ENGLISH_STOP_WORDS`. These words are completely ignored and cannot be searched for, at all. This means that even phrase queries aren't exact. E.g. the phrase query "to be or not to be" finds nothing at all.

Performance

Keep the index open

Especially in a Web application, where keeping state between requests requires additional work, it is tempting to open and close the index on every request. Unfortunately, this leads to very poor performance. At first this might work with small indexes or beefy hardware, but performance problems soon crop up – e.g. large garbage collections.

You should keep the index open as long as possible. Both [IndexReader](#) and [IndexSearcher](#) are thread-safe and don't require additional synchronization. One could cache the index searcher e.g. in the application context.

See also: [LuceneFAQ](#), "Is the [IndexSearcher](#) thread-safe?".

No need to cache search results

Lucene is amazingly fast at searching. Rather than caching hits and paging through them, merely re-executing the query is often fast enough.

See: [LuceneFAQ](#), "How do I implement paging, i.e. showing result from 1-10, 11-20 etc?".

When doing sub-searches (searching the results of a previous search), it's easier (and fast enough) to add the first query as a conditional to the second.

But if you really need to do this, see: [LuceneFAQ](#), "Can I cache search results with Lucene?". See also: [LuceneFAQ](#), "Can Lucene do a "search within search", so that the second search is constrained by the results of the first query?".

Use [RangeFilter](#) instead of [RangeQuery](#)

[RangeQuery](#) expands every term in the range to a boolean expression, and easily blows past the built-in `BooleanQuery.maxClauseCount` limit (Lucene 2.0 defaults to about 1000).

[RangeFilter](#) doesn't suffer from this limitation.

See: [LuceneFAQ](#), "Why am I getting a [TooManyClauses](#) exception?".

Querying

Edit the query rather than the string

Parsing free text is a surprisingly hard problem at which [QueryParser](#) does a pretty good job. Rather than editing the query string, change the [Query](#) objects returned by the [QueryParser](#).

E.g. when offering additional search options in a Web form, it's easier and safer to combine them with the parsed [Query](#) object rather doing text manipulations on the query string.

Lucene is not a true boolean system

Or: `apple AND banana OR orange` doesn't work. Surprising at first. [QueryParser](#) does its best to translate from a boolean syntax to Lucene's own set-oriented queries, but it falls short. Either use parentheses everywhere or try to design your user interface accordingly.

See [BooleanQuerySyntax](#).

Iterating over all hits takes a long time

This is by design. Try using a [HitCollector](#) instead if you need access to all the hits for a search.

Highlighting search results

The Lucene-contributed highlighter does the best it can, but it doesn't have the information it really needs to correctly highlight the search results.

See: [LuceneFAQ](#), "Is there a way to get a text summary of an indexed document with Lucene (a.k.a. a "snippet" or "fragment") to display along with the search result?".