# SeSQL extra features

## Contents



## 1 Text highlighting

SeSQL supports basic text highlighting. There is a highlighting function, which takes two parameters : a text and a list of words. It'll return a list of matching positions, as a triplet (begin, end, word number).

For example

```
>>> from sesql.highlight import highlight
>>> highlight('sesql : full text search, with text highlight',
              [ 'sesql', 'text' ])
 [(0, 5, 0), (13, 17, 1), (31, 35, 1)]
```

The text given to the highlight module must have been preprocessed first with the cleanup function.

If you're using different cleanup or dictionaries for different indexes, you can specify which index configuration to use with the `index` parameter of the highlight function.

# 2 Dependency tracking

## 2.1 The problem

SeSQL supports composite indexes, for example, indexing the name of the section of an article within the article, the section being a related model.

Now, if the name of the section is changed, you'll probably want to reindex all content within the section. That can be a lot of content.

## 2.2 How to specify dependencies

On each of your Django models, you can implement a method named

```
get_related_objects_for_indexation
```

This method doesn't take any parameter (except the `self`, of course), and should return a lit of related objects to reindex when the `self` changed.

The list can either be a list of `(classname, id)` tuples, or a list of Django objects.

## 2.3 The daemon

Because there could be a lot of related objects to reindex, they won't be reindexed instantly. They'll be inserted in a simple table, called `sesql_reindex_schedule`, just as `(classname, id, date)`.

A daemon, available in the `daemon/` directory of SeSQL, will then process the reindexation, chunk by chunk (100 by 100 by default), waiting some time (2 minutes by default) between two chunks. This will allow the content to be up-to-date in a mater of minutes (or hours, if you really have a lot of content), without being a too heavy burden for the live performance of the system.

You can change the size of a chunk and the delay to compromise between the acceptable delay in reindexing and the system load.

# 3 SeSQL admin

By setting `ENABLE_SESQL_ADMIN` to `True` in the configuration file, you'll be able to use `sesql:<index_name>` in any Django query, especially in the search field to use in the admin model. For example using :

```
search_fields = ['sesql:private_index']
```

In the `ModelAdmin` for a model will make Django admin look into SeSQL field called *private_index* whenever the users of the admin uses the search feature.

This feature is functional and stable, but not totally optimized and uses an heavy monkey-patch (susceptible to break with newer Django versions). It is therefore disabled by default.

# 4 Search history

## 4.1 General idea

The general idea is to collect information about searches done by users of the site, and to be able to later on suggest related searches to perform.

Three factors are taken in consideration on each search :

1. The number of the results the search gave (a small amount is probably a typo or spelling mistake, for example).

2. The number of times the query was performed.

3. The freshness of those queries (SeSQL was initially made for a newspaper site, http://www.liberation.fr , and in the world of newspapers, freshness is an important issue).

From those three factors, a weight is computed, allowing web developers to build additional features on top of that. n Please note that all the data are totally anonymized inside SeSQL search history tables.

## 4.2 The process

The process works in three steps :

1. When a query is performed, the code making the query can ask SeSQL to historize the search. It is inserted into a temporary table, in a raw form.

2. Regularly (typically, every day, but could be even more frequent) a cron task (as a `manage.py` command) is executed to process the information from the temporary table, and insert it in two other tables :

    1. A table of unprocessed searches, very similar to the initial one, which additional indexes, that can be used for datamining or any other purpose.

    2. A table of search statistics, where similar searches are aggregated and given a score.

## 4.3 What can be done from the data

The data can be used as you want to. You could make a portlet with the best scored searches on the homepage of your site, and just use them in the back office to understand how people use your site.

One feature that is currently under development is to suggest alternative searches (typically in case of mispelling) looking at searches with a high score and similar phonex. This code is partly implemented for the french language only.

# 5 Benchmarking tool

## 5.1 Summary

A small benchmarking tool is provided with SeSQL to allow yourself to test your specific configuration and request patterns.

## 5.2 Disclaimer

There are three types of lies : lies, damn lies and benchmarks. Don't give any absolute value to the results you obtain, and be aware that reality may be significantly distinct from benchmarks.

## 5.3 Concept of the benchmark tool

The SeSQL benchmark tool works by running threads performing operations like short queries, long queries and indexation. Each of those threads has a tight loop in which in a perform one operation, and then wait a bit, before doing another one.

You can configure how many threads of each kind are used, and how long they wait between each operation.

## 5.4 Query file

For the short queries and long queries threads, you must provide a query file. A query file is a file with a Q expression on each line, for example something like :

```
Q(classname='Article') & Q(fulltext__containswords="search engine sql")
Q(classname__in=['Article', 'Blog']) & \
 Q(fulltext__containswords="search engine sql")
```

## 5.5 Syntax examples

For a 10 minutes benchmark of a single thread doing continuous queries :

```
./manage.py sesqlbench --queryfile=queries.txt --duration=600
```

For a 10 minutes benchmark of 4 threads doing continuous queries :

```
./manage.py sesqlbench --queryfile=queries.txt --duration=600 \
  --short-threads=4
```

For a 10 minutes benchmark of 2 threads doing continuous short queries and one doing long queries :

```
./manage.py sesqlbench --queryfile=queries.txt --duration=600 \
  --short-threads=2 --long-threads=1
```

For a 10 minutes benchmark of 2 threads doing continuous short queries and 1 thread doing regular, but not continuous, reindexation :

```
./manage.py sesqlbench --queryfile=queries.txt --duration=600 \
  --short-threads=2 --index-threads=1 --index-delay=0.5 \
  --index-type=Article
```