Writing SeSQL queries

Contents

1 Short queries and full queries

SeSQL can be queried in two modes : *short queries* and *full queries*.

1.1 Short queries

Short queries are designed to retrieve a small amount (like 50) recent (according to any date field indexed by SeSQL) content matching the given criteria in an extra-fast manner.

It's typical used for a portlet, infobox or similar widget.

It has several limitations :

- it doesn't count the total amount of content matching the query ;
- it must be used on a query sorted by a date field ;

- several optimizations will be disabled if the content types spawn on more than one SeSQL table (see the typemap).

1.2 Full queries

Full queries (sometimes called *long queries* in the code or documentation) are a more common search. They are a bit slower than short queries, especially if there is a lot of content matching, but propose additional features :

- exact count of matching items ;
- stable pagination support (even if content is inserted or modified between the query and when the page is displayed, the pagination will remain stable) ;
- support of sorting by relevance, if the query includes a full-text search.

2 General rules

Making queries with SeSQL is made by calling the `shortquery` or `longquery` function with a Django Q object.

All the general rules of Q objects still work : using ~ to make a negation, | to make OR queries, & to make AND queries.

The kind of supported queries depend of the index types.

3 Queries by type

3.1 IntField

- default is = (`id = 12` for `id == 12`);
- `__lt`, `__gt`, `__gte`, `__lte` for <, >, <=, >= (`id__lt = 12` for `id < 12`) ;
- `__in` to test if it's inside a list (`id__in = (12, 13)`);
- `__range=(min,max)` to test if it's inside the range (`id__range = (12, 42)`)

3.2 StrField

- default is = (`workflow = "published"`);
- `__in` to test if it's inside a list (`workflow__in == ("published", "pending")`) ;

3.3 ClassField

- default is = (`objclass = Article`);
- `__in` to test if it's inside a list (`objclass__in == (Article, Photo)`) ;

3.4 DateField

- default is = (`date = now()` for today);
- `__lt`, `__gt`, `__gte`, `__lte` for <, >, <=, >= (`id__lt = now()` for before today) ;
- `__range=(min,max)` to test if it's inside the range.

3.5 DateTimeField

- like DateField, only different for ordering.

3.6 IntArrayField

- default is contains (`authors = 12` for contains the author 12) ;
- `__all` for contains all of a list (`authors__all = (12, 13)` for contains both authors) ;
- `__any` for contains any of a list (`authors__any = (12, 13)` for contains at leats one of the two authors) ;

3.7 FullTextField

- `__containswords` to test for inclusing of all the given words (`fulltext__containswords = "presidential elections in france"`);
- `__containsexact` to test for a specific sentence (`fulltext_containsexact = "france 2"`);
- `__matches` to test with a !PostgreSQL full-text query string (which can contain & for and, | for or, ...) (`fulltext_matches = "presidential | legislative & elections"`);
- `__like` to match with a SQL like (`fulltext_like = "fra%"`).

4 Sorting with SeSQL

4.1 Problematics

SeSQL needs the sort order to perform various optimization. It must be known early in the process of deciding which heuristics to use.

4.2 General syntax

The sort order must be given as a second, optional, 'order' argument of the methods. For example

```
shortquery(Q(classname__in = ('Article', 'PaperPage')) &
           Q(fulltext__containswords = 'python postgresql'),
           order = ( '-publication_date', 'page' ))
```

The `order` argument must be a list of field names, on which sorting makes sense (you can't sort of full text indexes for example). If a field is prefixed by "-" it'll use a descending order (defaut is ascending order).

4.3 Special ordering

SeSQL support special ordering modes, prefixed by `sesql_`. Only one is implemented in this version :

sesql_relevance will sort by relevance of the full text query on the primary full text index. Will only work if the query includes a filter on the primary full text index. Will disable most heuristics, so be careful to not overuse it.

4.4 Default order

The default search order is taken from the `DEFAULT_ORDER` varibale in `config.py`.

5 Brains

Since version 0.15, SeSQL can, instead of returning full objects constructed through the ORM, expose "brains" (the term comes from Zope catalog). Brains are taken directly from SeSQL table, and are much faster to retrieve, and usually take much less memory.

To get brains, you need :

1. To specify the name of fields (sesql indexes, postfixed with `_text` for the full-text fields) you want to fetch, in the constructor of the `SeSQLQuery` object, or as a keyword parameter of the `shortquery` and `longquery` functions.

2. On the resulting `SeSQLResultSet`, call the `brains()` methods, which will return an iterator of dictionaries.