# Writing SeSQL queries

## Short queries and full queries

FIXME

## 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.

## Queries by type

### 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)`)

### StrField

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

### ClassField

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

### 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.

### DateTimeField

- like DateField, only different for ordering.

### 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) ;

### 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%"`).

## Sorting with SeSQL

### Problematics

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

### 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).

### Special ordering

SeSQL support special ordering modes, prefixed by `sesql_`. Only one is implemented in the first 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.

### Default order

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