EUROPYTHON

2016

Bilbao, 17-24 July

What is the best full text search engine for Python?

Andrii Soldatenko
@a_soldatenko

Agenda:

- Who am I?
- What is full text search?
- PostgreSQL FTS / Elastic / Whoosh / Sphinx
- Search accuracy
- Search speed
- What’s next?

Andrii Soldatenko

- Backend Python Developer at 
- CTO in Persollo.com
- Speaker at many PyCons and Python meetups
- blogger at https://asoldatenko.com

Preface

Text Search

➜ cpython time grep -r -i 'OrderedDict' .

grep -r -i 'OrderedDict' **2.35s user 0.10s system 97% cpu 2.510 total**

➜ cpython time ack OrderedDict

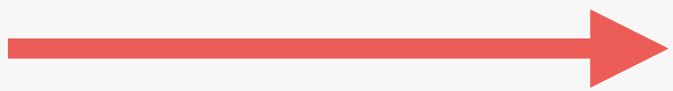
ack OrderedDict **1.74s user 0.14s system 96% cpu 1.946 total**

➜ cpython time pss OrderedDict

pss OrderedDict **0.85s user 0.09s system 96% cpu 0.983 total**

➜ **cpython time pt OrderedDict**

pt OrderedDict 0.14s user 0.10s system 462% cpu 0.051 total



Processor 2.5 GHz Intel Core i7

Memory 16 GB 1600 MHz DDR3

Full text search

Search index

Simple sentences

1. The quick brown fox jumped over the lazy dog

2. Quick brown foxes leap over lazy dogs in summer

Inverted index

```
Term       Doc_1  Doc_2
-------------------------
Quick     |       |  X
The       |   X   |
brown     |   X   |  X
dog       |   X   |
dogs      |       |  X
fox       |   X   |
foxes     |       |  X
in        |       |  X
jumped    |   X   |
lazy      |   X   |  X
leap      |       |  X
over      |   X   |  X
quick     |   X   |
summer    |       |  X
the       |   X   |
-------------------------
```

Inverted index

```
1   Term        Doc_1  Doc_2
2   ---------------------------
3   brown    |    X    |  X
4   quick    |    X    |
5   ---------------------------
6   Total    |    2    |  1
```

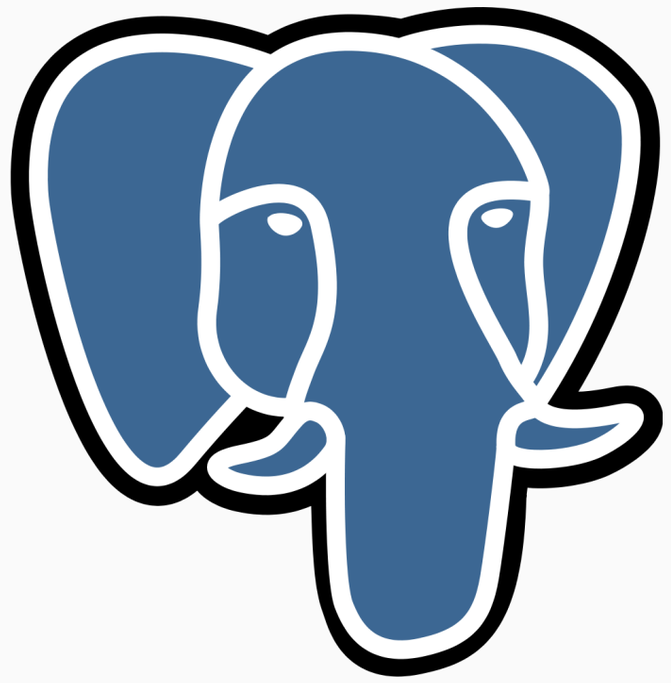
Inverted index: normalization

Term	Doc_1	Doc_2
Quick		X
The	X	
brown	X	X
dog	X	
dogs		X
fox	X	
foxes		X
in		X
jumped	X	
lazy	X	X
leap		X
over	X	X
quick	X	
summer		X
the	X	

Term	Doc_1	Doc_2
brown	X	X
dog	X	X
fox	X	X
in		X
jump	X	X
lazy	X	X
over	X	X
quick	X	X
summer		X
the	X	X

Search Engines

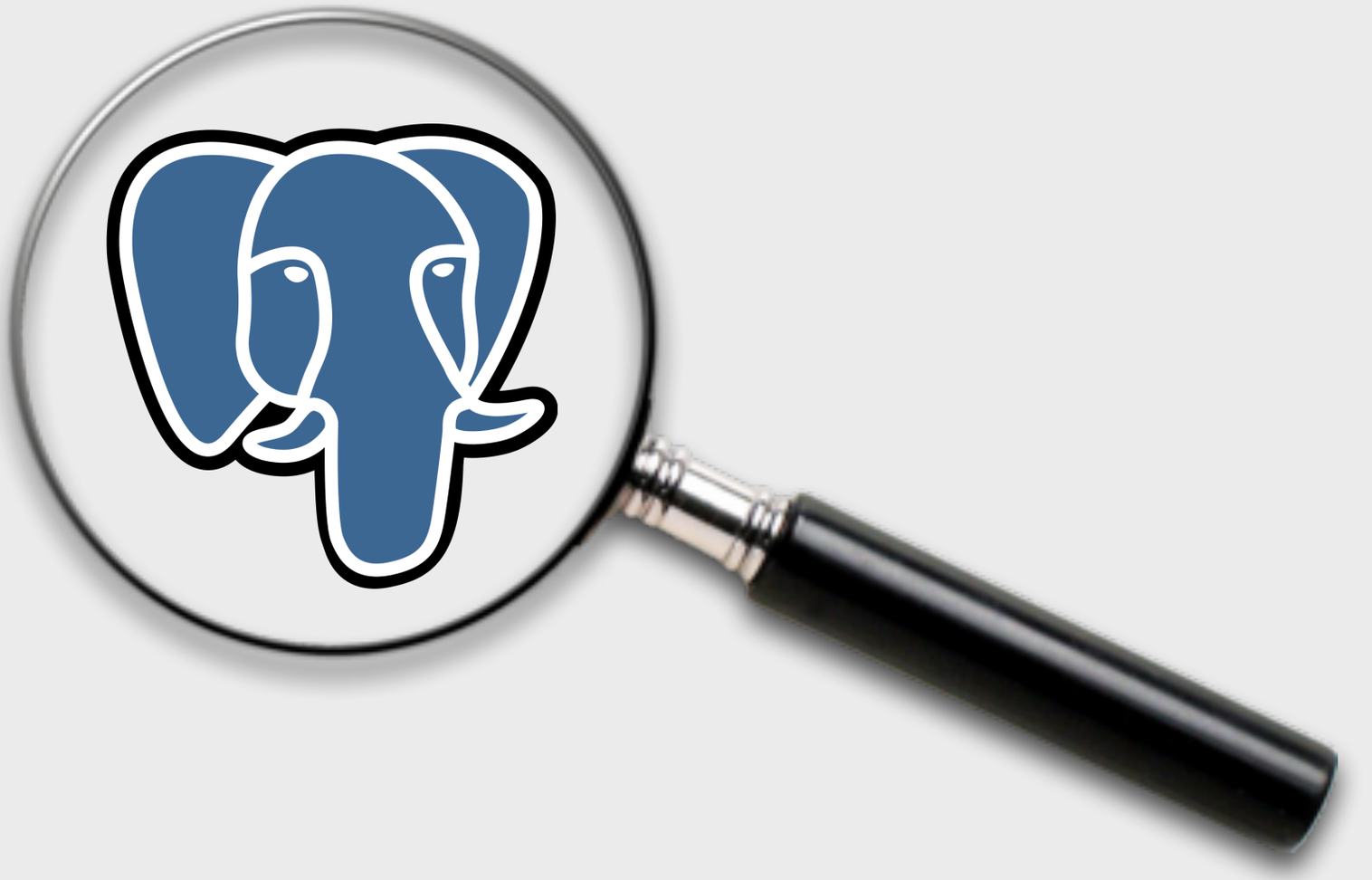
elastic



WHOOSH
Python search library

Sphinx

PostgreSQL
Full Text Search



support from version 8.3

PostgreSQL
Full Text Search

```
SELECT to_tsvector('text') @@
       to_tsquery('query');
```

Simple is better than complex. - by import this

Do PostgreSQL FTS
without index

```
SELECT ‘python bilbao 2016'::tsvector @@ 'python &
bilbao'::tsquery;
```

```
?column?
----------
 t
(1 row)
```

Do PostgreSQL FTS with index

```
CREATE INDEX name ON table USING GIN
(column);
```

```
CREATE INDEX name ON table USING
GIST (column);
```

PostgreSQL FTS:
Ranking Search Results

ts_rank() -> float4 - based on the frequency of their matching lexemes

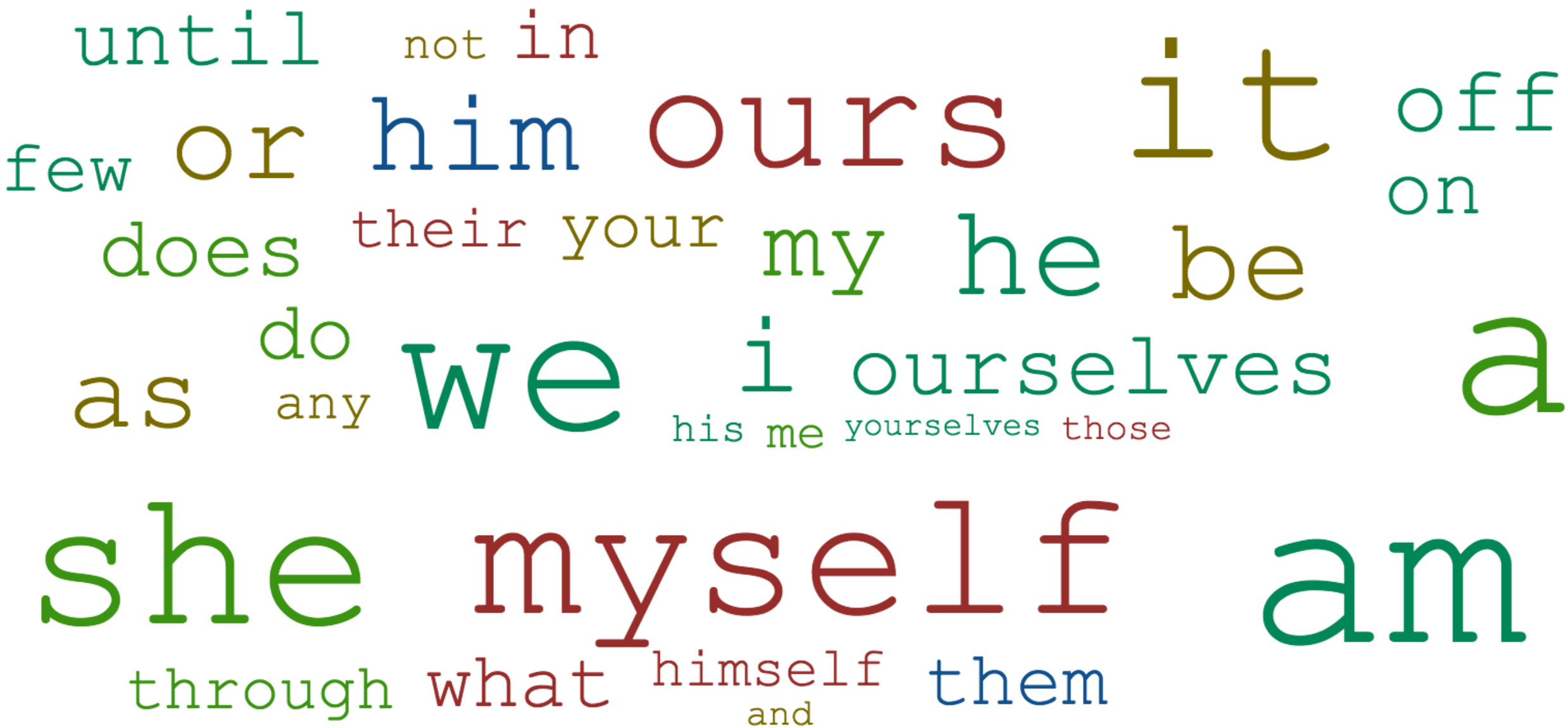
ts_rank_cd() -> float4 - cover density ranking for the given document vector and query

PostgresSQL FTS
Highlighting Results

```
SELECT ts_headline('english',
                   'python conference 2016',
                   to_tsquery('python & 2016'));
```

```
          ts_headline
-------------------------------
 <b>python</b> conference <b>2016</b>
```

Stop Words



postgresql/9.5.2/share/postgresql/tsearch_data/english.stop

PostgresSQL FTS
Stop Words

```
SELECT to_tsvector('in the list of stop words');
```

```
          to_tsvector
-------------------------------
 'list':3 'stop':5 'word':6
```

PG FTS
and Python

- Django 1.10 django.contrib.postgres.search
- djorm-ext-pgfulltext
- sqlalchemy

PostgreSQL FTS integration with django orm

```
from djorm_pgfulltext.models import SearchManager
from djorm_pgfulltext.fields import VectorField
from django.db import models

class Page(models.Model):
    name = models.CharField(max_length=200)
    description = models.TextField()

    search_index = VectorField()

    objects = SearchManager(
        fields = ('name', 'description'),
        config = 'pg_catalog.english', # this is default
        search_field = 'search_index', # this is default
        auto_update_search_field = True
    )
```

https://github.com/linuxlewis/djorm-ext-pgfulltext

For search just use search method of the manager

```
>>> Page.objects.search("documentation & about")
[<Page: Page: Home page>]
```

```
>>> Page.objects.search("about I documentation I django I home", raw=True)
[<Page: Page: Home page>, <Page: Page: About>, <Page: Page: Navigation>]
```

https://github.com/linuxlewis/djorm-ext-pgfulltext

Django 1.10

```
>>> Entry.objects.filter(body_text__search='recipe')
    [<Entry: Cheese on Toast recipes>, <Entry: Pizza
recipes>]

>>> Entry.objects.annotate(
...         search=SearchVector('blog__tagline',
'body_text'),
... ).filter(search='cheese')
    [
        <Entry: Cheese on Toast recipes>,
        <Entry: Pizza Recipes>,
        <Entry: Dairy farming in Argentina>,
    ]
```

https://github.com/django/django/commit/2d877da

PostgreSQL FTS

Pros:

- Quick implementation
- No dependency

Cons:

- Need manually manage indexes
- depend on PostgreSQL
- no analytics data
- no DSL only `&` and `|` queries

ElasticSearch

Who uses ElasticSearch?

GitHub

stackoverflow

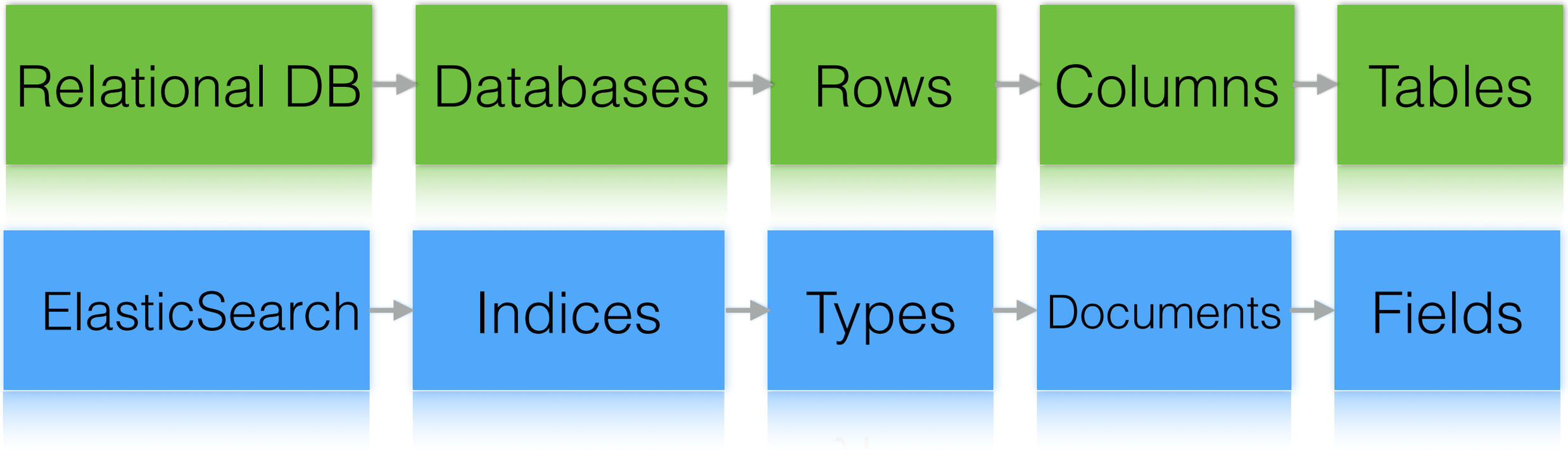
DATADOG

KLOUT

theguardian

WIKIPEDIA
The Free Encyclopedia

ElasticSearch: Quick Intro



ElasticSearch: Locks

- Pessimistic concurrency control

- Optimistic concurrency control

ElasticSearch and Python

- elasticsearch-py
- elasticsearch-py-async by Honza Kral
- elasticsearch-dsl-py by Honza Kral

ElasticSearch: FTS

```
$ curl -XGET 'http://localhost:9200/pyconua/talk/_search' -d '
{
    "query": {
        "match": {
            "user": "Andrii"
        }
    }
}'
```

ES: Create Index

```
$ curl -XPUT 'http://localhost:9200/
twitter/' -d '{
    "settings" : {
        "index" : {
            "number_of_shards" : 3,
            "number_of_replicas" : 2
        }
    }
}'
```

ES: Add json to Index

```
$ curl -XPUT 'http://localhost:9200/
pyconua/talk/1' -d '{
    "user" : "andrii",
    "description" : "Full text search"
}'
```

ES: Stopwords

```
$ curl -XPUT 'http://localhost:9200/europython' -d
'{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_english": {
          "type":                 "english",
          "stopwords_path": "stopwords/english.txt"
        }
      }
    }
  }
}'
```

ES: Highlight

```
$ curl -XGET 'http://localhost:9200/europython/
talk/_search' -d '{
    "query" : {...},
    "highlight" : {
        "pre_tags" : ["<tag1>"],
        "post_tags" : ["</tag1>"],
        "fields" : {
            "_all" : {}
        }
    }
}'
```

ES: Relevance

```
$ curl -XGET 'http://localhost:9200/_search?explain -d
'

{
    "query"    : { "match" : { "user" : "andrii" }}
}'

"_explanation": {
  "description": "weight(tweet:honeymoon in 0)
                  [PerFieldSimilarity], result of:",
  "value":        0.076713204,
  "details": [...]
}
```

- written in C+
- uses MySQL as data source (or other database)

Sphinx
search server

DB table ≈ Sphinx index

DB rows ≈ Sphinx documents

DB columns ≈ Sphinx fields and attributes

Sphinx
simple query

```
SELECT * FROM test1
WHERE MATCH('europython');
```

Whoosh

- Pure-Python
- **Whoosh** was created by *Matt Chaput*.
- Pluggable scoring algorithm (including BM25F)
- more info at video from PyCon US 2013

Whoosh: Stop words

```
import os.path
import textwrap


names = os.listdir("stopwords")
for name in names:
    f = open("stopwords/" + name)
    wordls = [line.strip() for line in f]
    words = " ".join(wordls)
    print '"%s": frozenset(u"""' % name
    print textwrap.fill(words, 72)
    print '""".split())'
```

http://anoncvs.postgresql.org/cvsweb.cgi/pgsql/src/backend/snowball/stopwords/

Whoosh: Highlight

```
results = pycon.search(myquery)
for hit in results:
    print(hit["title"])
    # Assume "content" field is stored
    print(hit.highlights("content"))
```

Whoosh:
Ranking search results

- Pluggable scoring algorithm
- including BM25F

	Python clients	Python 3	Django support
elastic	elasticsearch-py elasticsearch-dsl-py elasticsearch-py-async	YES	haystack + elasticstack
	psycopg2 aiopg asyncpg	YES	djorm-ext-pgfulltext django.contrib.postgres
Sphinx	sphinxapi	NOT YET (Open PR)	django-sphinx django-sphinxql
WHOOSH Python search library	Whoosh	YES	support using haystack

Haystack

Haystack

Haystack:
Pros and Cons

Pros:

- easy to setup
- looks like Django ORM but for searches
- search engine independent
- support 4 engines (Elastic, Solr, Xapian, Whoosh)

Cons:

- poor SearchQuerySet API
- difficult to manage stop words
- loose performance, because extra layer
- Model - based

	Indexes	Without indexes
elastic	Apache Lucene	No support
	GIN / GIST	to_tsvector()
Sphinx	Disk / RT / Distributed	No support
WHOOSH Python search library	index folder	No support

	ranking / relevance	Configure Stopwords	highlight search results
elastic	TF/IDF	YES	YES
	cd_rank	YES	YES
Sphinx	max_lcs+BM25	YES	YES
WHOOSH Python search library	Okapi BM25	YES	YES

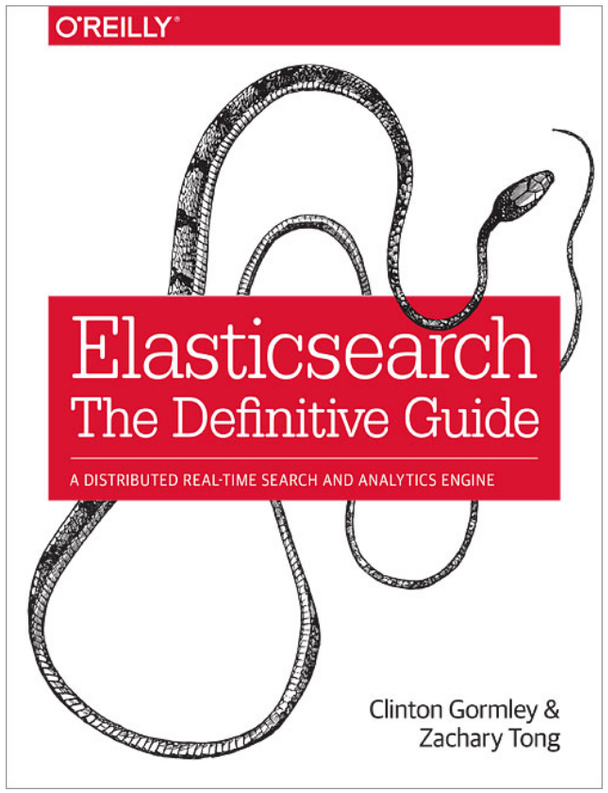
	Synonyms	Scale
elastic	YES	YES
	YES	Partitioning
Sphinx	YES	Distributed searching
WHOOSH Python search library	NO SUPPORT	NO

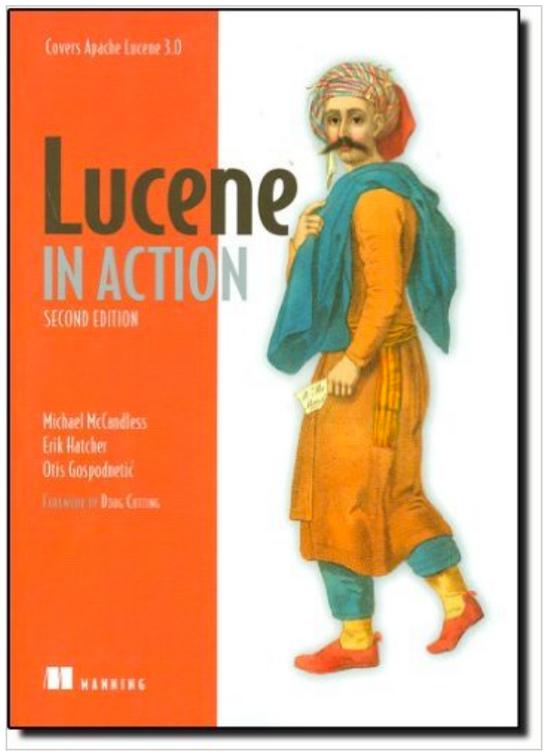
1 million music Artists

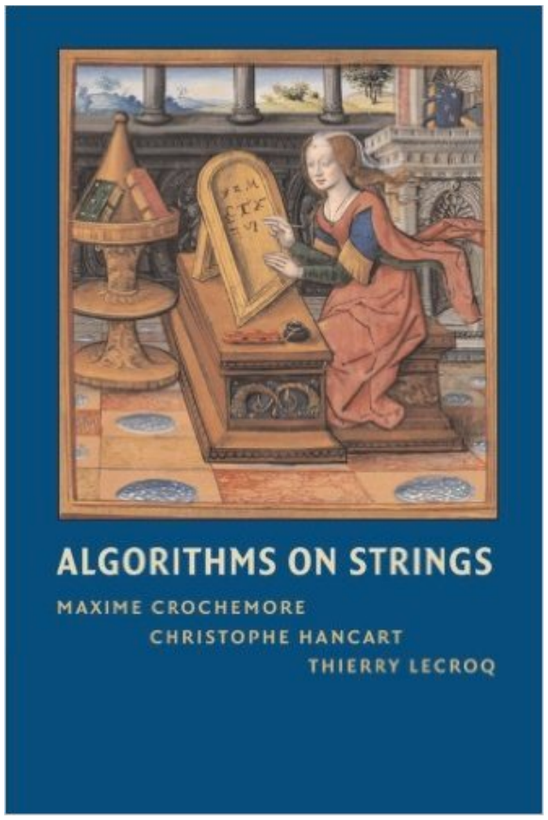
Evie Tamala
Jean-Pierre Martin
Deejay One
wecamewithbrokenteeth
The Blackbelt Band
Giant Tomo
Decoding Jesus
Elvin Jones & Jimmy Garrison Sextet
Infester
...
David Silverman
Aili Teigmo

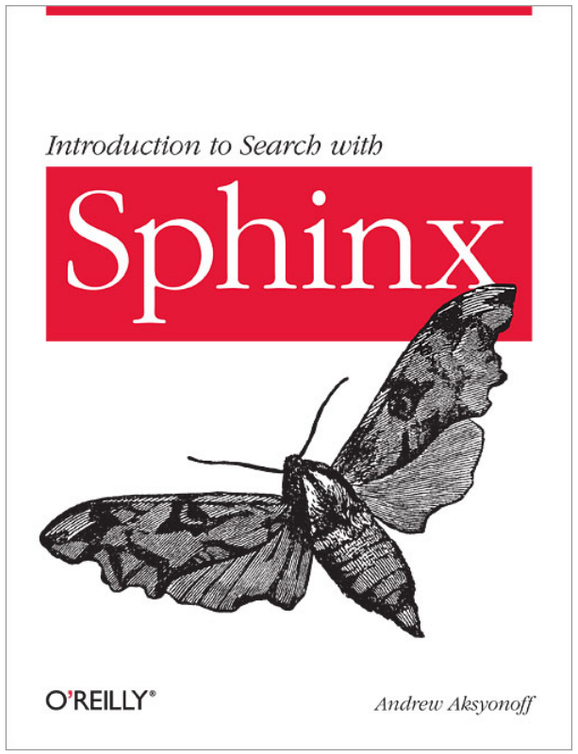
	Performance	Database size
elastic	9 ms	~ 1 million records
	4 ms	~ 1 million records
Sphinx	6 ms	~ 1 million records
WHOOSH Python search library	~2 s	~ 1 million records

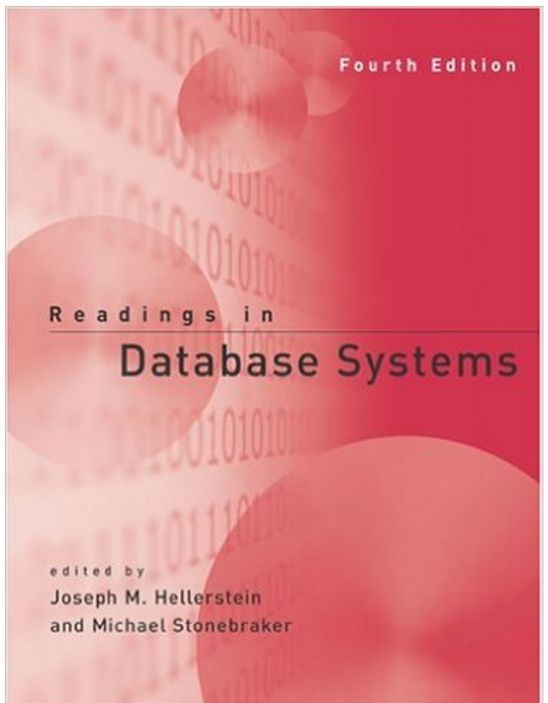
Books


O'REILLY
Elasticsearch
The Definitive Guide
A DISTRIBUTED REAL-TIME SEARCH AND ANALYTICS ENGINE
Clinton Gormley &
Zachary Tong


Covers Apache Lucene 3.0
Lucene
IN ACTION
SECOND EDITION
Michael McCandless
Erik Hatcher
Otis Gospodnetić
MANNING


ALGORITHMS ON STRINGS
MAXIME CROCHEMORE
CHRISTOPHE HANCART
THIERRY LECROQ


Introduction to Search with
Sphinx
O'REILLY
Andrew Aksyonoff


Fourth Edition
Readings in
Database Systems
edited by
Joseph M. Hellerstein
and Michael Stonebraker

Indexing references:

http://gist.cs.berkeley.edu/

http://www.sai.msu.su/~megera/postgres/gist/

http://www.sai.msu.su/~megera/wiki/Gin

https://www.postgresql.org/docs/9.5/static/gist.html

https://www.postgresql.org/docs/9.5/static/gin.html

Ranking references:

http://sphinxsearch.com/docs/current.html#weighting

https://www.postgresql.org/docs/9.5/static/textsearch-controls.html#TEXTSEARCH-RANKING

https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html

https://en.wikipedia.org/wiki/Okapi_BM25

https://lucene.apache.org/core/3_6_0/scoring.html

Slides

https://asoldatenko.com/EuroPython16.pdf

Thank You

@a_soldatenko

andrii.soldatenko@gmail.com

(n)->[:Questions]