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 Toptal

• 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
Symbols
32gb Heap boundary, 642

A
ACID transactions, 545, 556
action, in bulk requests, 57, 69
ad hoc searches, 15
aggregations, 20, 417
  aggs parameter, 424
  and analysis, 483
  approximate, 457
    cardinality, 458
    percentiles, 462
  basic example
    adding a metric, 426
    adding extra metrics, 429
    buckets nested in other buckets, 427
  buckets, 419
  combining buckets and metrics, 420
  metrics, 420
  limiting memory usage, 487
  fielddata circuit breaker, 490
  fielddata size, 488
  monitoring fielddata, 489
  managing efficient memory usage, 507
  nested, 567
    reverse_nested aggregation, 568
  operating alongside search requests, 418
  preventing combinatorial explosions, 500
    depth-first versus breadth-first, 502
  returning empty buckets, 439
  scoping, 445
    global bucket, 447
  Significant Terms, 471
  significant terms
Simple sentences

1. The quick brown fox jumped over the lazy dog

2. Quick brown foxes leap over lazy dogs in summer
### Inverted Index

<table>
<thead>
<tr>
<th></th>
<th>Term</th>
<th>Doc_1</th>
<th>Doc_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>3</td>
<td>Quick</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>The</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>brown</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>dog</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>dogs</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td>fox</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>foxes</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>in</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>11</td>
<td>jumped</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>lazy</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>13</td>
<td>leap</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>14</td>
<td>over</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>15</td>
<td>quick</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>summer</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>17</td>
<td>the</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Inverted index

<table>
<thead>
<tr>
<th></th>
<th>Term</th>
<th>Doc_1</th>
<th>Doc_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>brown</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>quick</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Total</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Inverted index: normalization

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc_1</th>
<th>Doc_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>The</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>brown</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>dog</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>dogs</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>fox</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>foxes</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>in</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>jumped</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>lazy</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>leap</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>over</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>quick</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>summer</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>the</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc_1</th>
<th>Doc_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>brown</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>dog</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>fox</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>foxes</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>in</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>jump</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>lazy</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>over</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>quick</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>summer</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>the</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Search Engines
PostgreSQL Full Text Search support from version 8.3
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

\texttt{ts\_rank()} -> float4 - based on the frequency of their matching lexemes

\texttt{ts\_rank\_cd()} -> float4 - cover density ranking for the given document vector and query
SELECT ts_headline('english',
   'python conference 2016',
   to_tsquery('python & 2016'));

<\texttt{ts\_headline}>

\texttt{ts\_headline}

\begin{verbatim}
\texttt{python}\texttt{conference}\texttt{2016}
\end{verbatim}
PostgreSQL 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

```python
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

```python
>>> Page.objects.search("documentation & about")

[<Page: Page: Home page>]

>>> Page.objects.search("about l documentation l django l home", raw=True)


https://github.com/linuxlewis/djorm-ext-pgfulltext
>>> 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?
ElasticSearch: Quick Intro

Relational DB → Databases → Rows → Columns → Tables

ElasticSearch → Indices → Types → Documents → Fields
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"

    } 

  } 

}'
$ 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

```python
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/
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
<table>
<thead>
<tr>
<th>Python clients</th>
<th>Python 3</th>
<th>Django support</th>
</tr>
</thead>
<tbody>
<tr>
<td>elastic search-py</td>
<td>YES</td>
<td>haystack + elasticstack</td>
</tr>
<tr>
<td>elastic search-dsl-py</td>
<td></td>
<td></td>
</tr>
<tr>
<td>elastic search-py-async</td>
<td></td>
<td></td>
</tr>
<tr>
<td>psycopg2</td>
<td>YES</td>
<td>djorm-ext-pgfulltext</td>
</tr>
<tr>
<td>aiopg</td>
<td></td>
<td>django.contrib.postgres</td>
</tr>
<tr>
<td>asyncpg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sphinxapi</td>
<td>NOT YET</td>
<td>django-sphinx</td>
</tr>
<tr>
<td></td>
<td>(Open PR)</td>
<td>django-sphinxql</td>
</tr>
<tr>
<td>Whoosh</td>
<td>YES</td>
<td>support using haystack</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>Indexes</th>
<th>Without indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>elastic</td>
<td>Apache Lucene</td>
</tr>
<tr>
<td></td>
<td>No support</td>
</tr>
<tr>
<td></td>
<td>GIN / GIST</td>
</tr>
<tr>
<td></td>
<td>to_tsvector()</td>
</tr>
<tr>
<td>Sphinx</td>
<td>Disk / RT / Distributed</td>
</tr>
<tr>
<td></td>
<td>No support</td>
</tr>
<tr>
<td>WHOOSH</td>
<td>index folder</td>
</tr>
<tr>
<td></td>
<td>No support</td>
</tr>
<tr>
<td></td>
<td>ranking / relevance</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------</td>
</tr>
<tr>
<td><strong>elastic</strong></td>
<td>TF/IDF</td>
</tr>
<tr>
<td><strong>Sphinx</strong></td>
<td>cd_rank</td>
</tr>
<tr>
<td><strong>WHOOSH</strong></td>
<td>max_lcs+BM25</td>
</tr>
<tr>
<td><strong>Python search library</strong></td>
<td>Okapi BM25</td>
</tr>
<tr>
<td>Synonyms</td>
<td>Scale</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>elastic</td>
<td>YES</td>
</tr>
<tr>
<td>Sphinx</td>
<td>YES</td>
</tr>
<tr>
<td>WHOOSH</td>
<td>NO SUPPORT</td>
</tr>
</tbody>
</table>
1 million music Artists

Evie Tamala
Jean-Pierre Martin
Deejay One
wecamewithbrokekteeth
The Blackbelt Band
Giant Tomo
Decoding Jesus
Elvin Jones & Jimmy Garrison Sextet
Infester
...
David Silverman
Aili Teigmo
<table>
<thead>
<tr>
<th>Performance</th>
<th>Database size</th>
</tr>
</thead>
<tbody>
<tr>
<td>elastic</td>
<td>~ 1 million records</td>
</tr>
<tr>
<td>9 ms</td>
<td>~ 1 million records</td>
</tr>
<tr>
<td>4 ms</td>
<td>~ 1 million records</td>
</tr>
<tr>
<td>Sphinx</td>
<td>~ 1 million records</td>
</tr>
<tr>
<td>6 ms</td>
<td>~ 1 million records</td>
</tr>
<tr>
<td>WHOOSH</td>
<td>~ 1 million records</td>
</tr>
<tr>
<td>~2 s</td>
<td>~ 1 million records</td>
</tr>
</tbody>
</table>
Books
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://lucene.apache.org/core/3_6_0/scoring.html
Slides

Thank You

@a_soldatenko

Follow me! @a_soldatenko

andrii.soldatenko@gmail.com
Hire the top 3% of freelance developers

(n) -> [:Questions]