Search Options in Django

Finding what you mean, not only what you type

Stefan Baerisch, stefan@stbaer.com, 2020-09-19

image: freepik.com premium, by Copperpipe

About

Stefan Baerisch

stefan@stbaer.com

Software since 2005

Python since 2006

Project Management / Test Management since 2010

Freelance Software Engineer since 2020

Some Background on Fulltext Search

(Fulltext-) Search

Search in some Text

Documents, Tweets, Emails, Patents, Websites, Product Descriptions, Product Reviews, Transcripts....

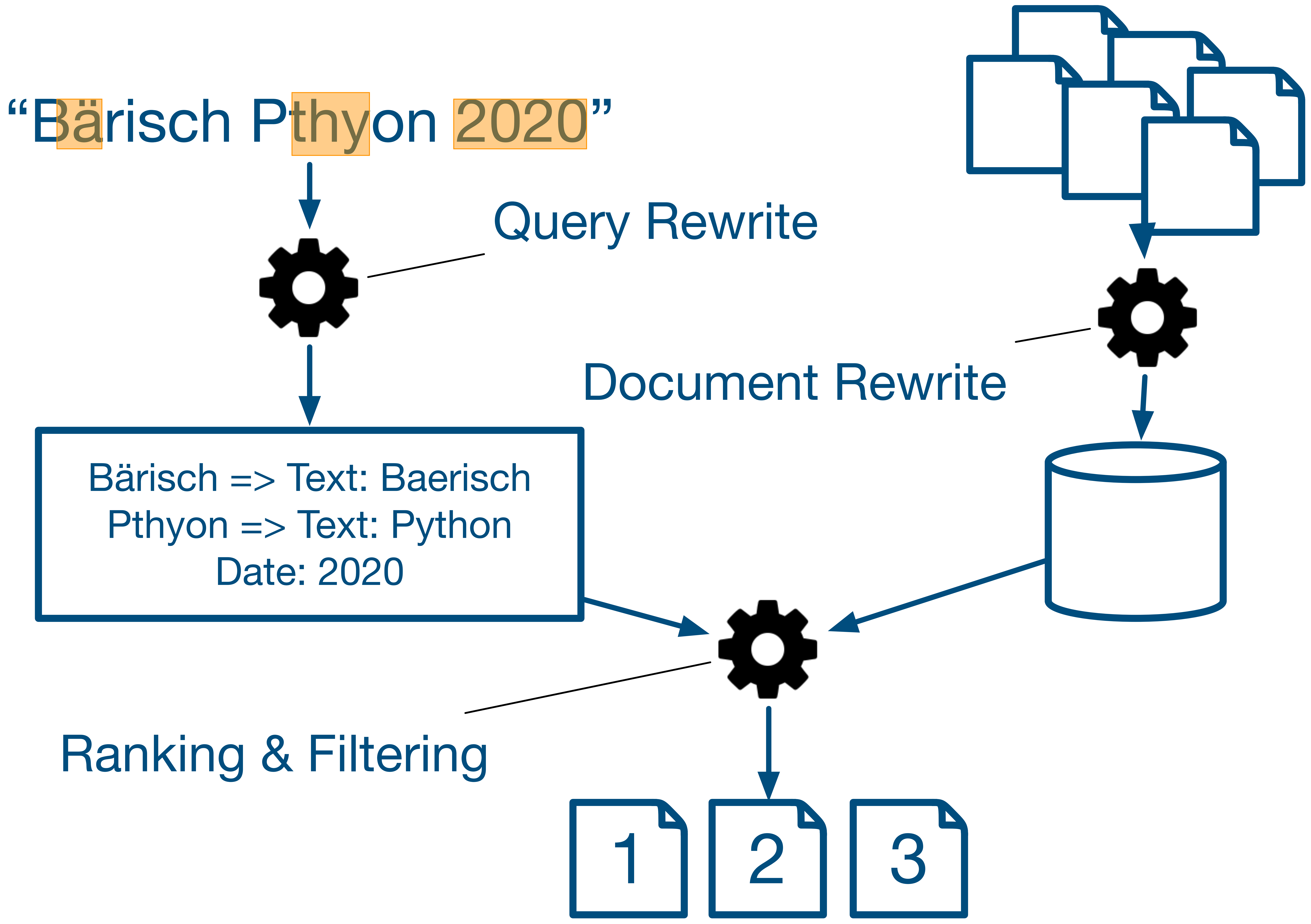
Language, Document lengths,

Image from freepics premium / @tutatama

Fulltext Seach - Why?

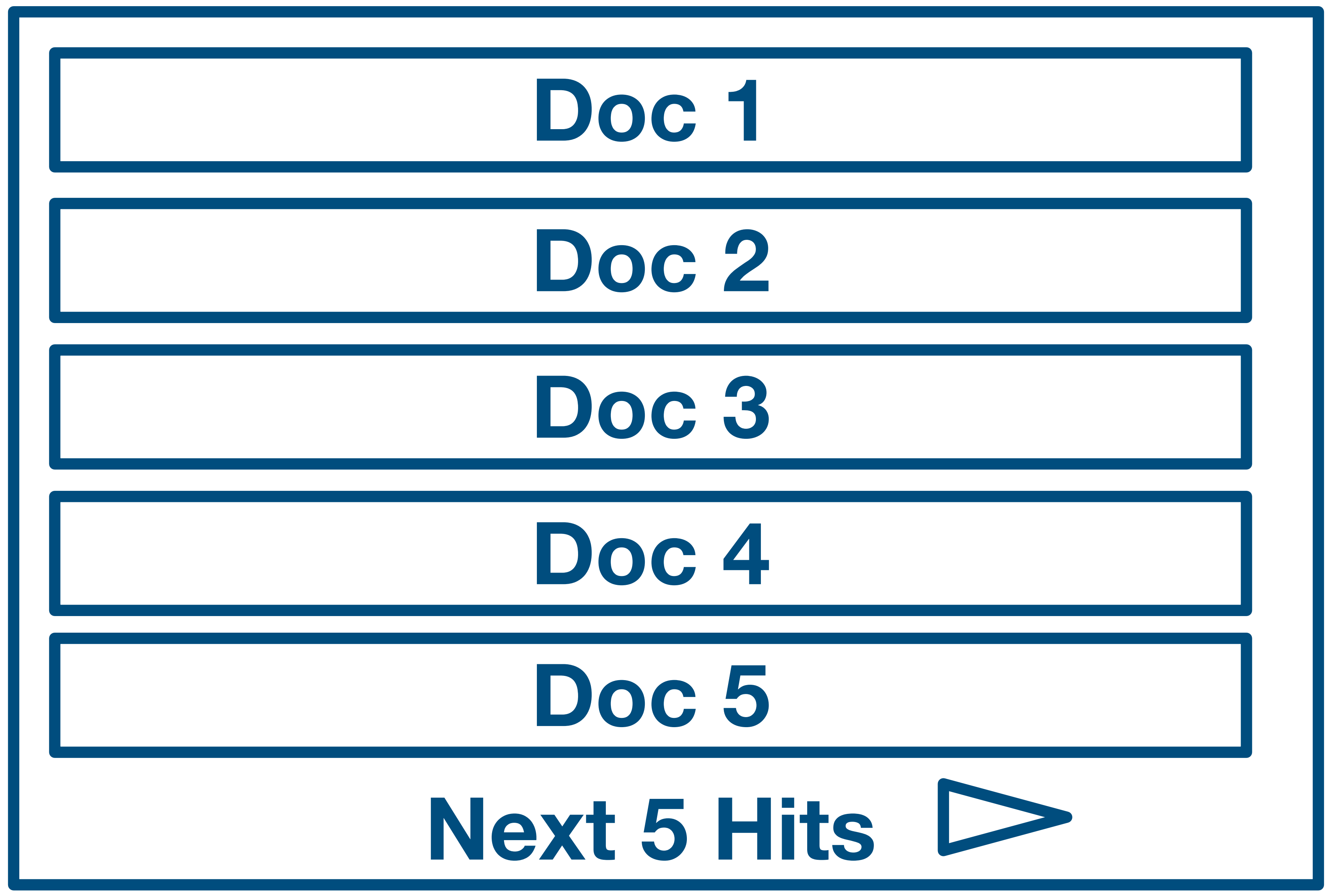
(SQL-) Query	Search
Exact Match	Fuzzy Match (Query / Document Rewriting)
Returns Set of Documents	Returns Relevance-Sorted List of Documents
Give me what I say	**Give me what I mean**

Document and Term Rewriting



Relevance

How manage pages of results do you look at ?



We want everything
on the first page

We want Ranking

What makes a document relevant?

# Terms present in document? In all documents?
Term position(s) in document?
Document specific factors (new, frequently seen)
Users specific factors (similar to others / recommendation /)
Not manipulated (think black SEO)

Other Aspects of Good Search

Product / User View

Fast ⟺	Latency / throughput of queries
Current ⟺	Quick indexing / updates
Correct ⟺	Precision / Recall
Relevant ⟺	Subjective, what do users thing
UX fits Users ⟺	Can query language express what users want?

Implementation / Operations View

Scalable ⟺	#docs / # queries
Well documented & Well known ⟺	documentation, books, experience reports
Maintainable ⟺	monitoring / deploy / operate / integrate
Easy things easy, hard things possible ⟺	minutes to change indexing, flexible processing and ranking

What to Search for: Documents

A Search Scenario

Movies!

Amazon Movie Review Dataset[1]

Nice dataset, contains a combination of structured data and text. ~8 million review in total

Field	Example
productID	B00006HAXW
userID	A1RSDE90N6RSZF
userName	Joe E. Xample
helpfulness	9/9 (nine of nine users….)
reviewscore	5.0
time	1042502400 (Epoch)
summary	**Pittsburgh - Home of the OLDIES**
text	**I have all of the doo wop DVD's and this one is as good….**

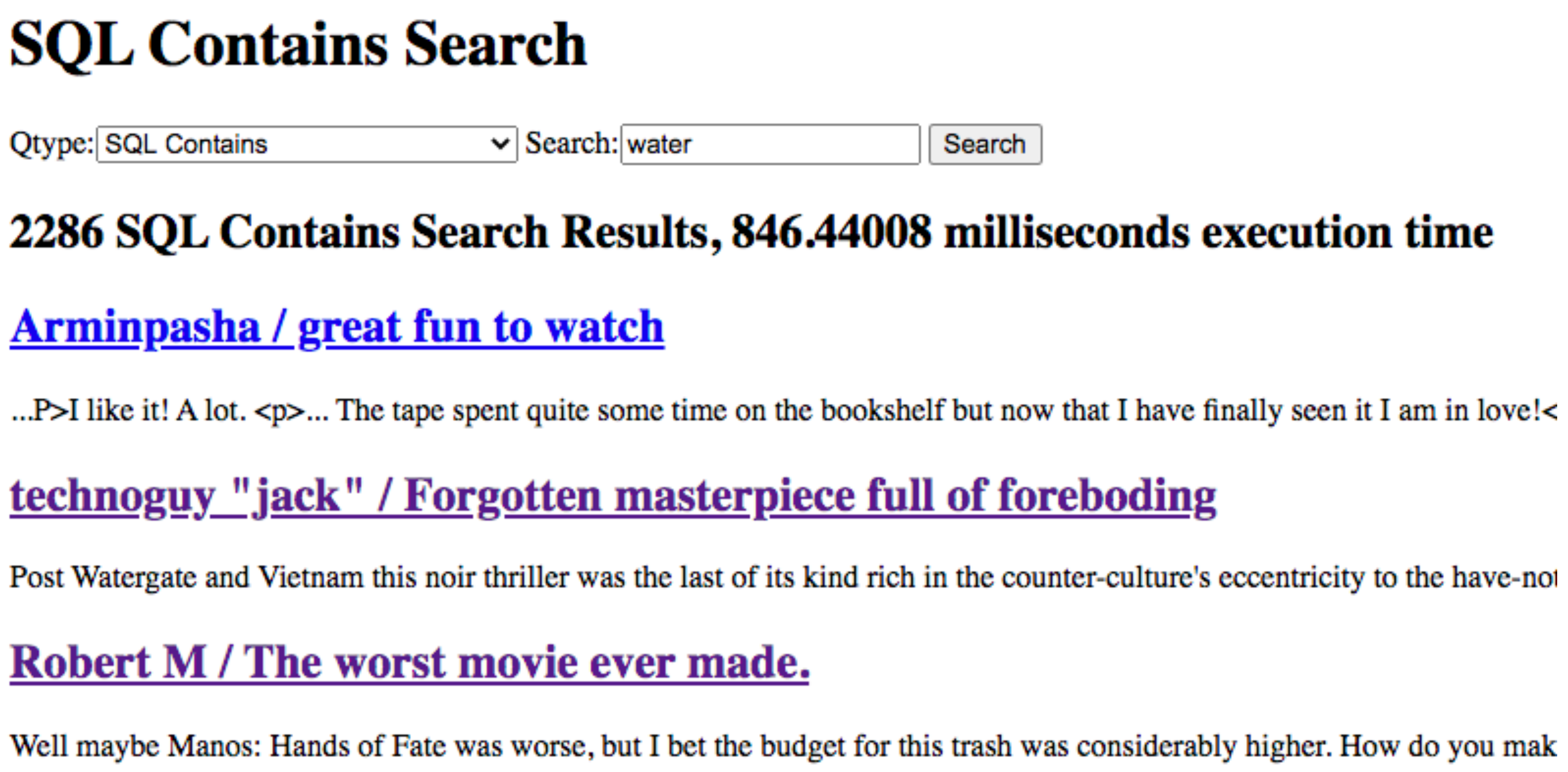
```
class FTSReview(models.Model):
    productId = models.CharField(max_length=200, db_index=True)
    userId = models.CharField(max_length=200, db_index=True)
    name = models.CharField(max_length=200)
    review_help_total = models.PositiveIntegerField()
    review_help_help = models.PositiveIntegerField()
    review_score = models.FloatField()
    review_time = models.DateTimeField()
    review_summary = models.TextField()
    review_text = models.TextField()
```

[1] J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. WWW, 2013.

Using PostgreSQL Fulltext Search

Regular Search

```
def sql_contains(qstring):
    q_summary = Q(review_summary__icontains=qstring)
    q_text = Q(review_text__icontains=qstring)
    search_query = q_summary | q_text
    reviews = FTSReview.objects.filter(
        search_query
    )
    return reviews, {}
```



SQL Contains Search

2286 SQL Contains Search Results, 846.44008 milliseconds execution time

Arminpasha / great fun to watch

...P>I like it! A lot. <p>... The tape spent quite some time on the bookshelf but now that I have finally seen it I am in love!<

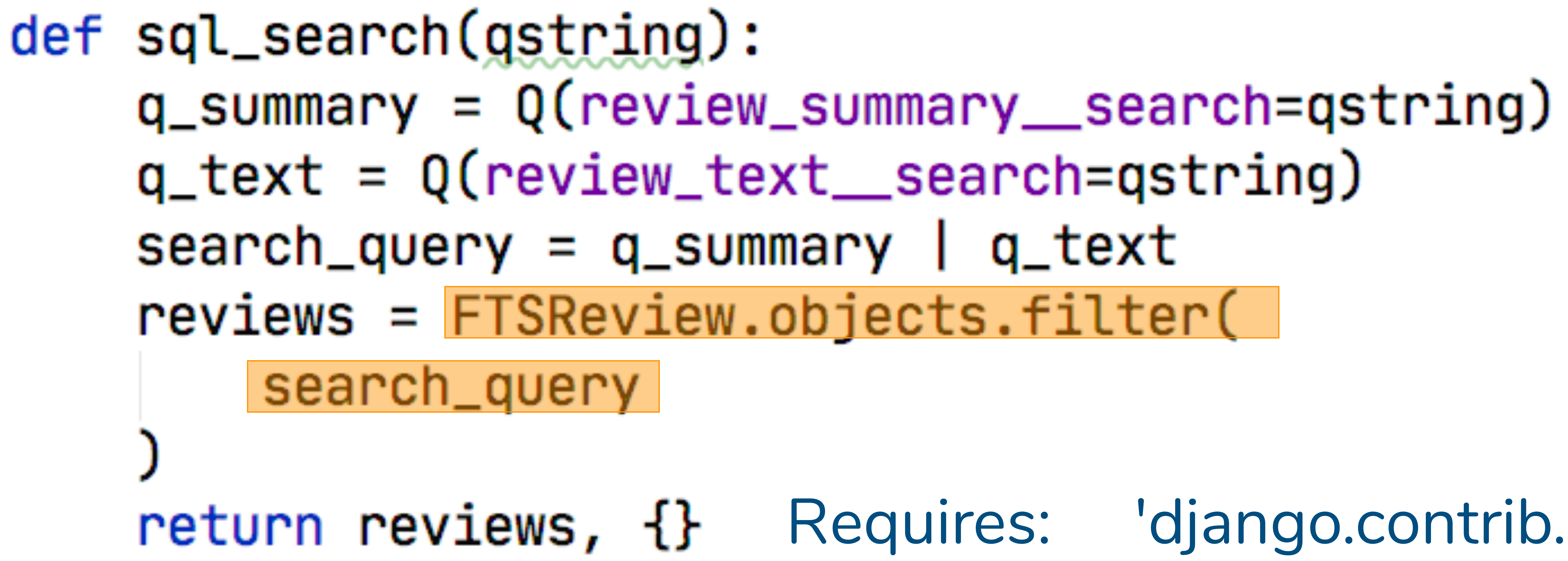
technoguy "jack" / Forgotten masterpiece full of foreboding

Post Watergate and Vietnam this noir thriller was the last of its kind rich in the counter-culture's eccentricity to the have-not

Robert M / The worst movie ever made.

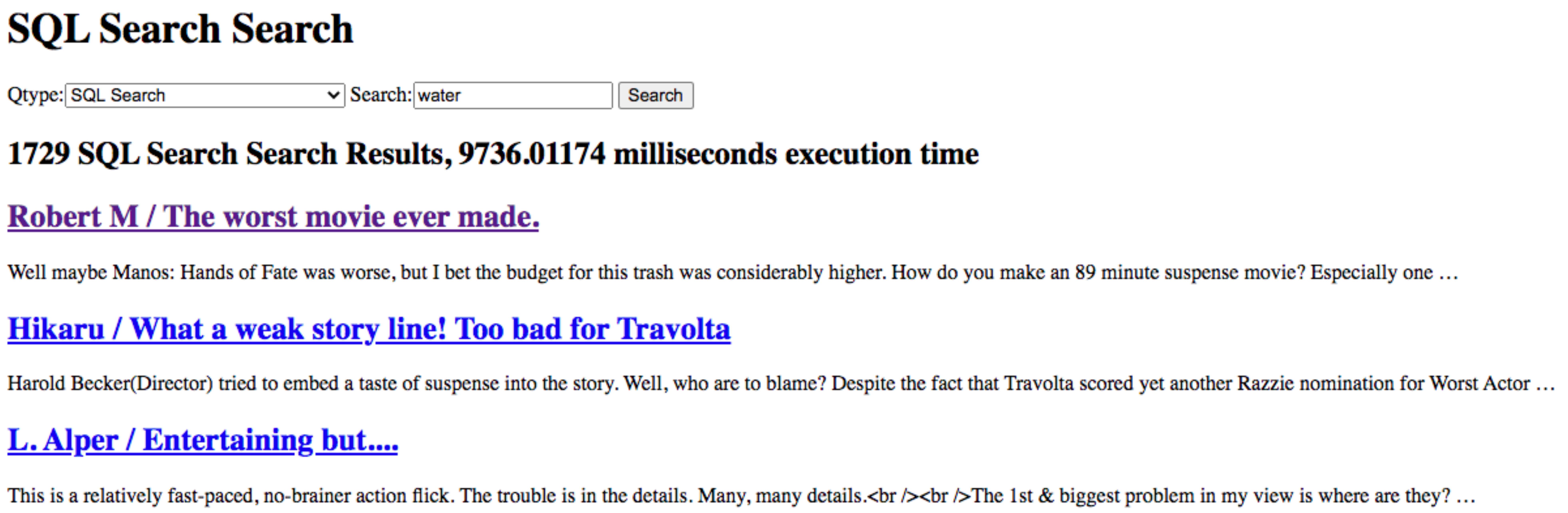
Well maybe Manos: Hands of Fate was worse, but I bet the budget for this trash was considerably higher. How do you mak

Fulltext Search in Postgresql



```
def sql_search(qstring):
    q_summary = Q(review_summary__search=qstring)
    q_text = Q(review_text__search=qstring)
    search_query = q_summary | q_text
    reviews = FTSReview.objects.filter(
        search_query
    )
    return reviews, {}
```

Requires: 'django.contrib.postgres',



SQL Search Search

Qtype: SQL Search Search: water Search

1729 SQL Search Search Results, 9736.01174 milliseconds execution time

Robert M / The worst movie ever made.

Well maybe Manos: Hands of Fate was worse, but I bet the budget for this trash was considerably higher. How do you make an 89 minute suspense movie? Especially one …

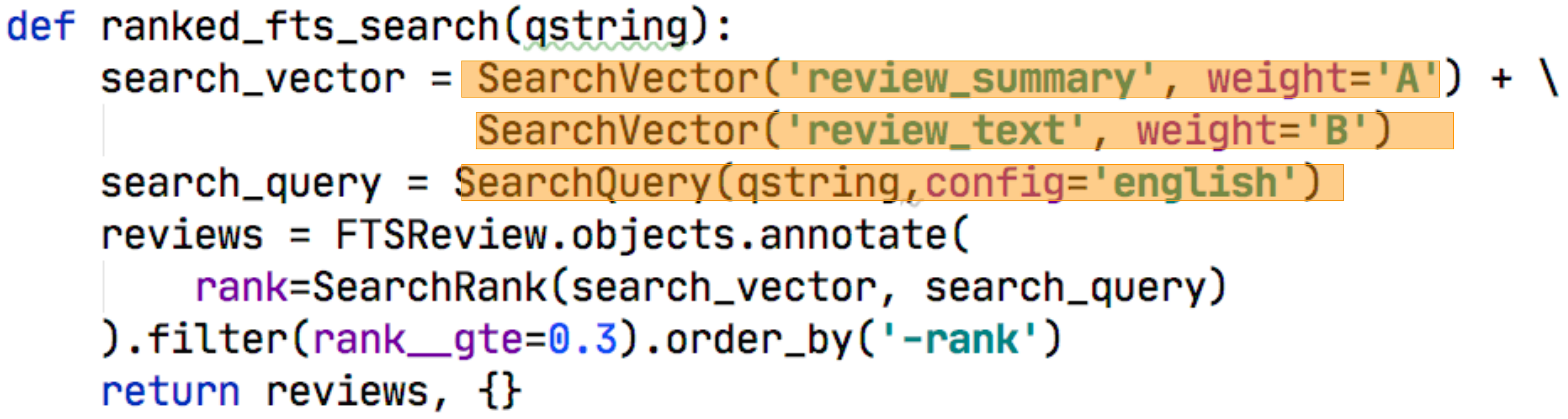
Hikaru / What a weak story line! Too bad for Travolta

Harold Becker(Director) tried to embed a taste of suspense into the story. Well, who are to blame? Despite the fact that Travolta scored yet another Razzie nomination for Worst Actor …

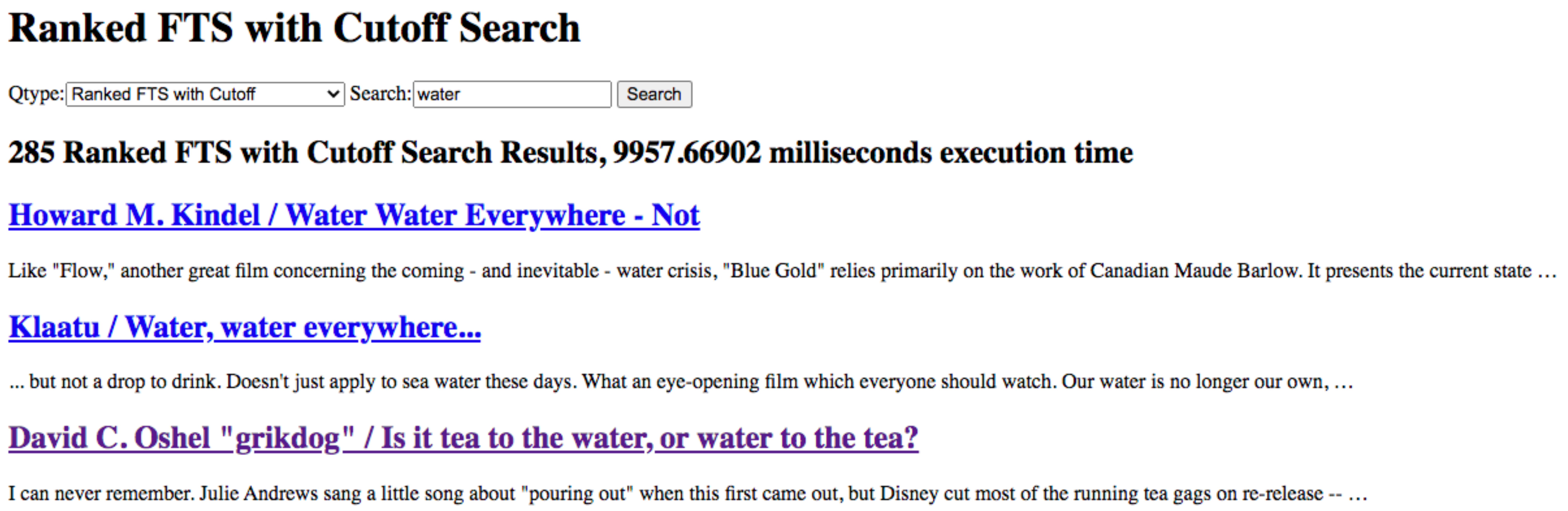
L. Alper / Entertaining but....

This is a relatively fast-paced, no-brainer action flick. The trouble is in the details. Many, many details.<br /><br />The 1st & biggest problem in my view is where are they? …

Ranked Search in Postgresql



```python
def ranked_fts_search(qstring):
    search_vector = SearchVector('review_summary', weight='A') + \
                    SearchVector('review_text', weight='B')
    search_query = SearchQuery(qstring,config='english')
    reviews = FTSReview.objects.annotate(
        rank=SearchRank(search_vector, search_query)
    ).filter(rank__gte=0.3).order_by('-rank')
    return reviews, {}
```



Ranked FTS with Cutoff Search

Qtype: Ranked FTS with Cutoff Search: water Search

285 Ranked FTS with Cutoff Search Results, 9957.66902 milliseconds execution time

Howard M. Kindel / Water Water Everywhere - Not

Like "Flow," another great film concerning the coming - and inevitable - water crisis, "Blue Gold" relies primarily on the work of Canadian Maude Barlow. It presents the current state …

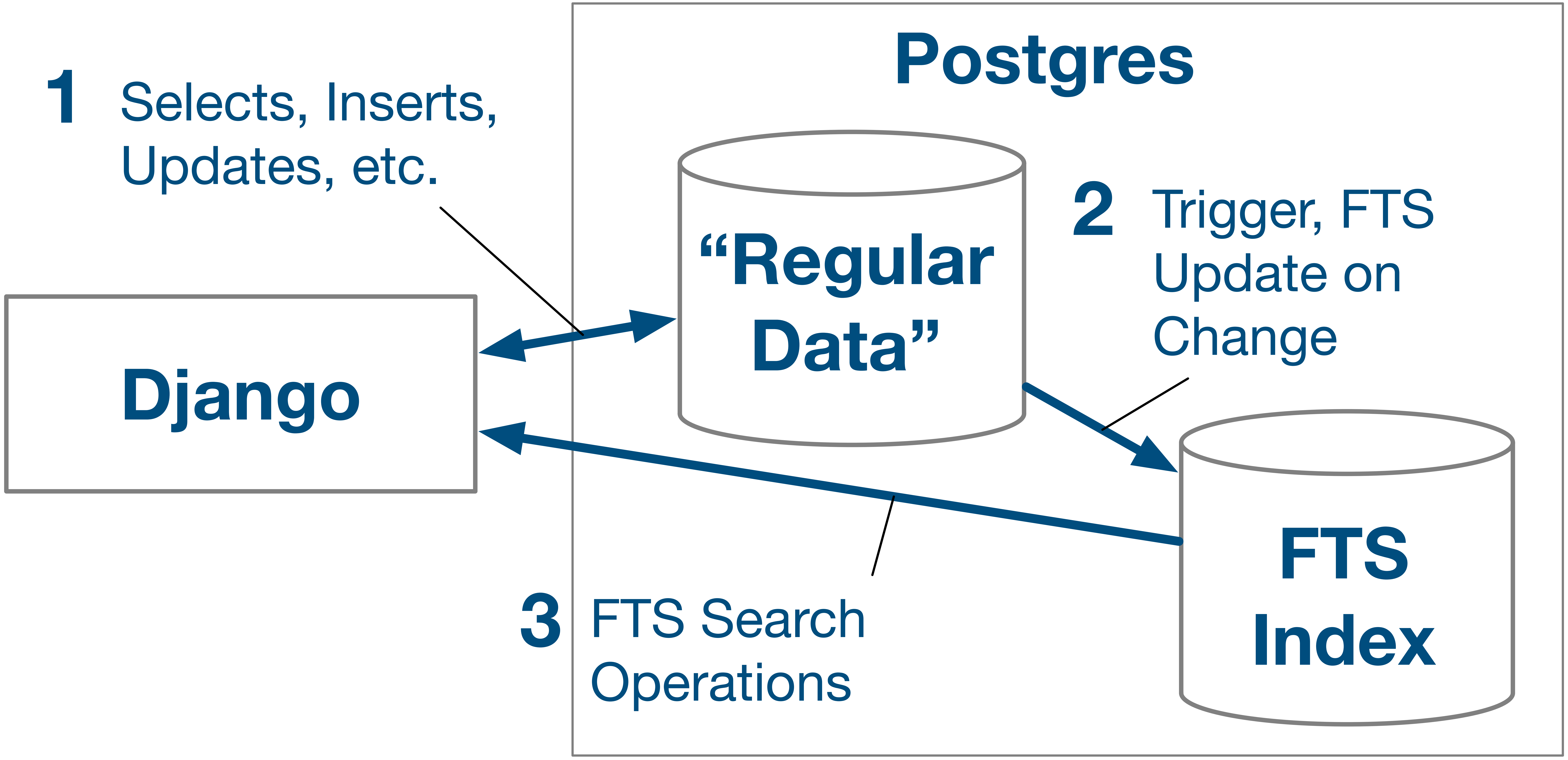
Klaatu / Water, water everywhere...

... but not a drop to drink. Doesn't just apply to sea water these days. What an eye-opening film which everyone should watch. Our water is no longer our own, …

David C. Oshel "grikdog" / Is it tea to the water, or water to the tea?

I can never remember. Julie Andrews sang a little song about "pouring out" when this first came out, but Disney cut most of the running tea gags on re-release -- …

Indexing Text

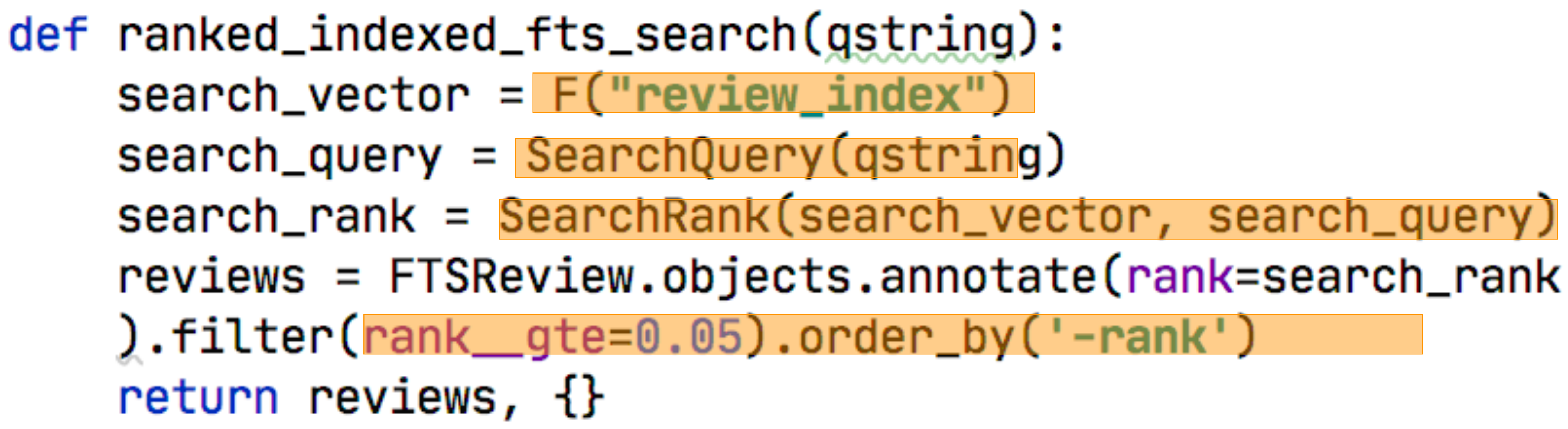

Postgres
1 Selects, Inserts, Updates, etc.
"Regular Data"
2 Trigger, FTS Update on Change
Django
FTS Index
3 FTS Search Operations

Indexing Text - Database

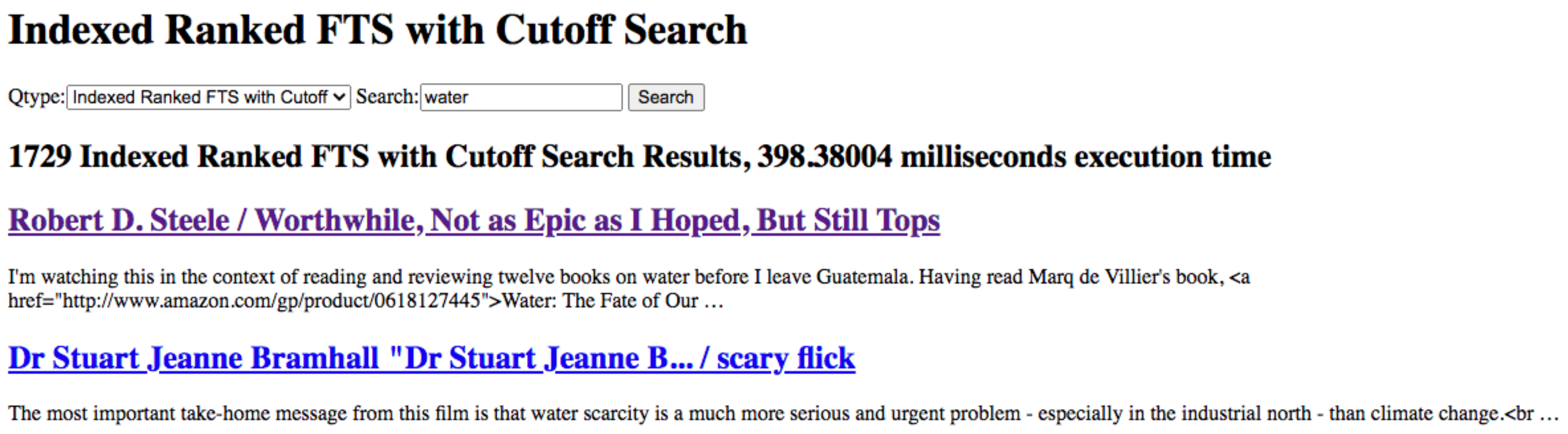
```
class FTSReview(models.Model):
    ...
    review_index = SearchVectorField(null=True)
    class Meta:
        indexes = [GinIndex(fields=["review_index"])]
```

```
class Migration(migrations.Migration):

    dependencies = [
        ('django_search_app', '0002_auto_20200716_0758'),
    ]

    migration = '''
        CREATE TRIGGER review_index_update BEFORE INSERT OR UPDATE
        ON django_search_app_ftsreview FOR EACH ROW EXECUTE FUNCTION
        tsvector_update_trigger(review_index, 'pg_catalog.english', review_summary, review_text);

        UPDATE django_search_app_ftsreview set ID = ID;
    '''

    reverse_migration = '''
        DROP TRIGGER review_index_update ON django_search_app_ftsreview;
    '''
```

Indexing Text - Query



```
def ranked_indexed_fts_search(qstring):
    search_vector = F("review_index")
    search_query = SearchQuery(qstring)
    search_rank = SearchRank(search_vector, search_query)
    reviews = FTSReview.objects.annotate(rank=search_rank
    ).filter(rank__gte=0.05).order_by('-rank')
    return reviews, {}
```



Indexed Ranked FTS with Cutoff Search

Qtype: Indexed Ranked FTS with Cutoff Search: water Search

1729 Indexed Ranked FTS with Cutoff Search Results, 398.38004 milliseconds execution time

Robert D. Steele / Worthwhile, Not as Epic as I Hoped, But Still Tops

I'm watching this in the context of reading and reviewing twelve books on water before I leave Guatemala. Having read Marq de Villier's book, <a href="http://www.amazon.com/gp/product/0618127445">Water: The Fate of Our …

Dr Stuart Jeanne Bramhall "Dr Stuart Jeanne B... / scary flick

The most important take-home message from this film is that water scarcity is a much more serious and urgent problem - especially in the industrial north - than climate change.<br …

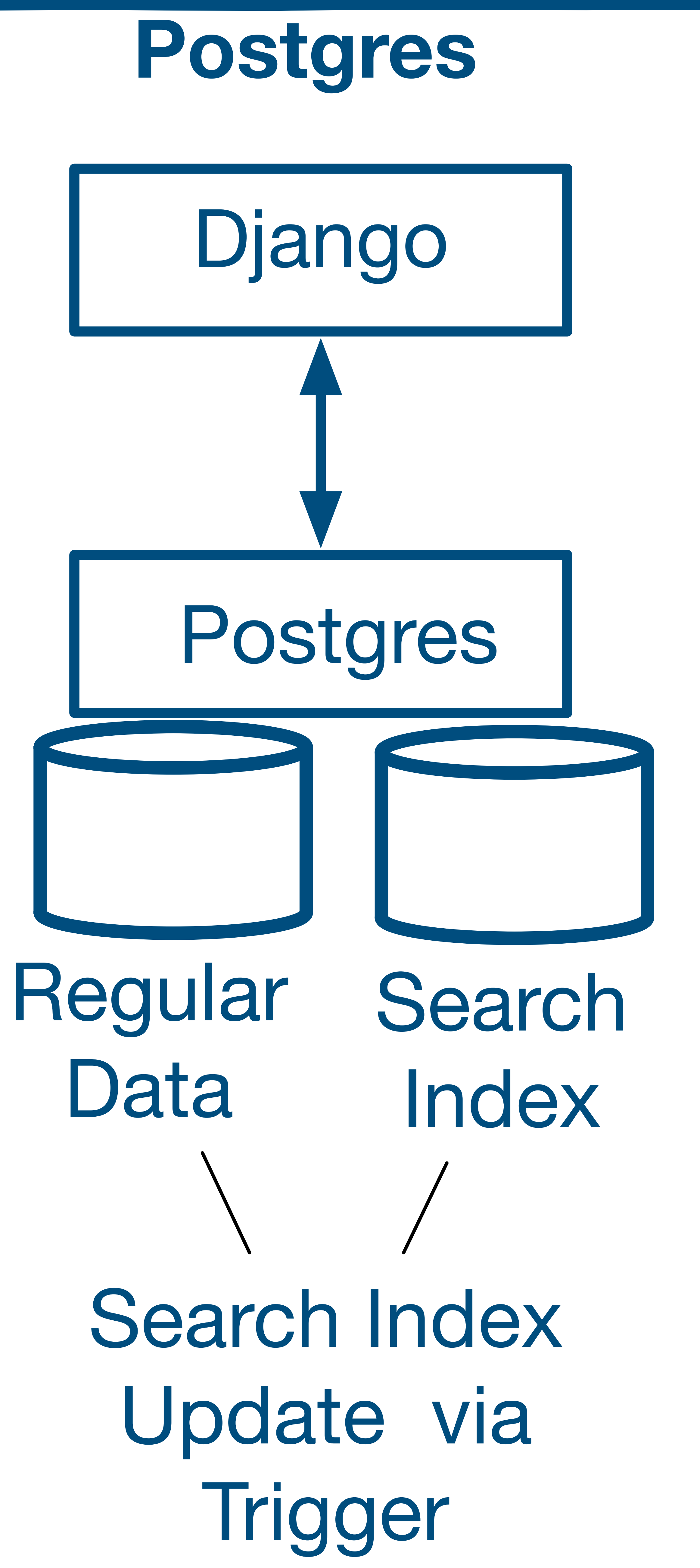
Using elasticsearch

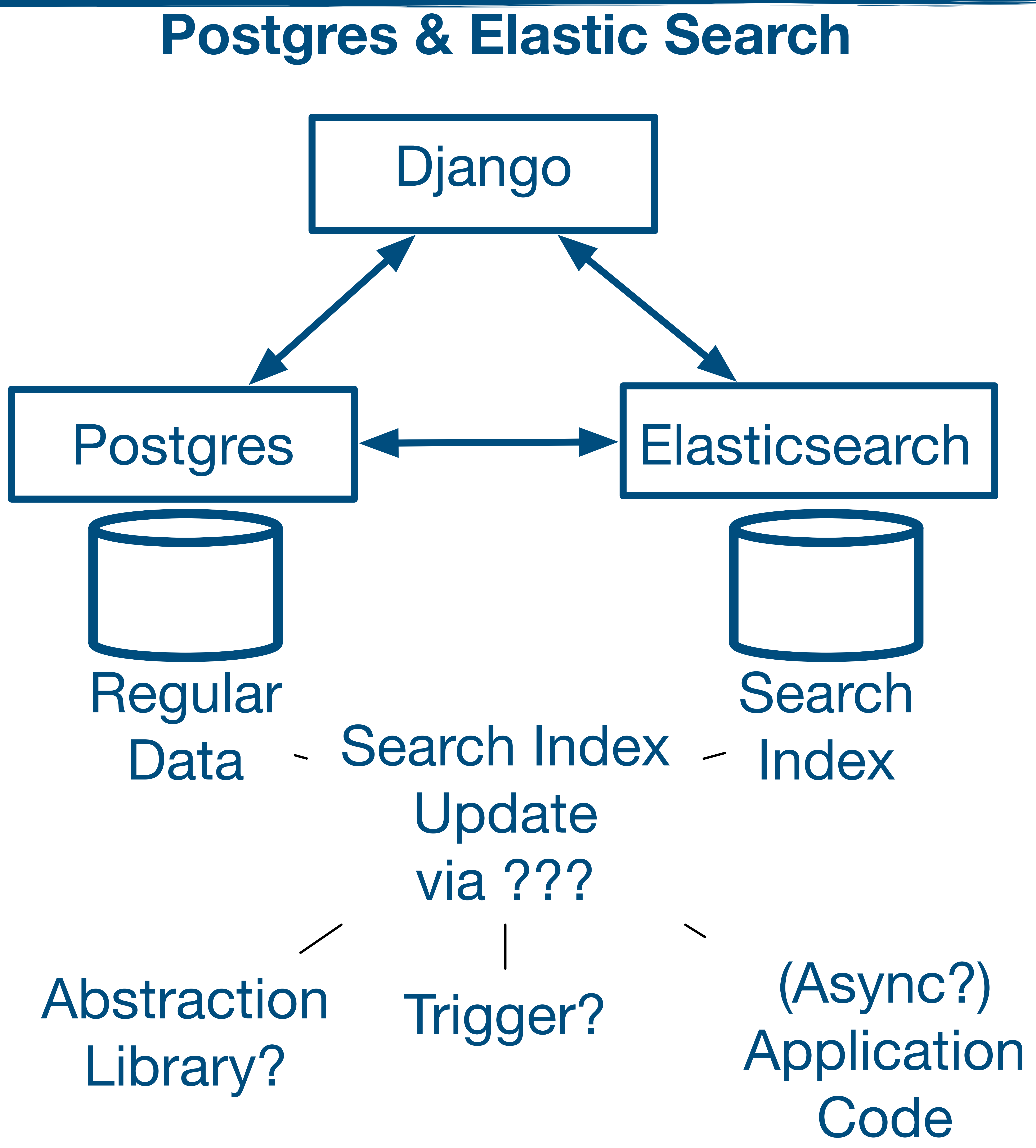
Search Engine…

- based on Lucene
- REST API
- Rich in features
- Scaleable
- Commercial and Open Source

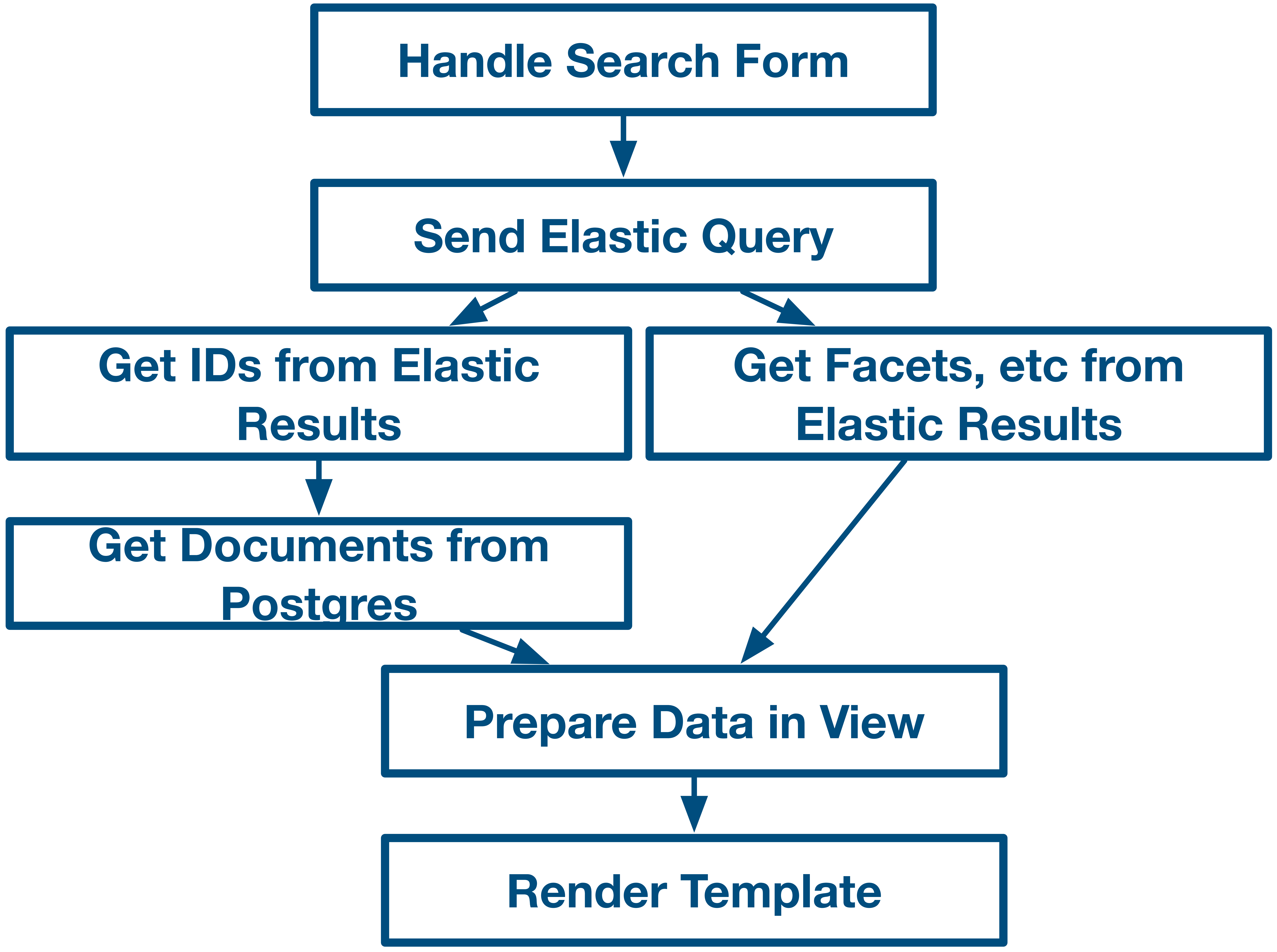
  for a pure Open Source Alternative, see Apache Solr

Elasticsearch with Django - Design Decisions





Search with Elasticsearch & Django


Handle Search Form
Send Elastic Query
Get IDs from Elastic Results
Get Facets, etc from Elastic Results
Get Documents from Postgres
Prepare Data in View
Render Template

Implementation Decisions

We there are different ways to add Elasticsearch to our Django Application

Official Python Elasticsearch Client

https://github.com/elastic/elasticsearch-py

Abstraction Libraries : Haystack

https://github.com/django-haystack/django-haystack

Direct Use of the REST API

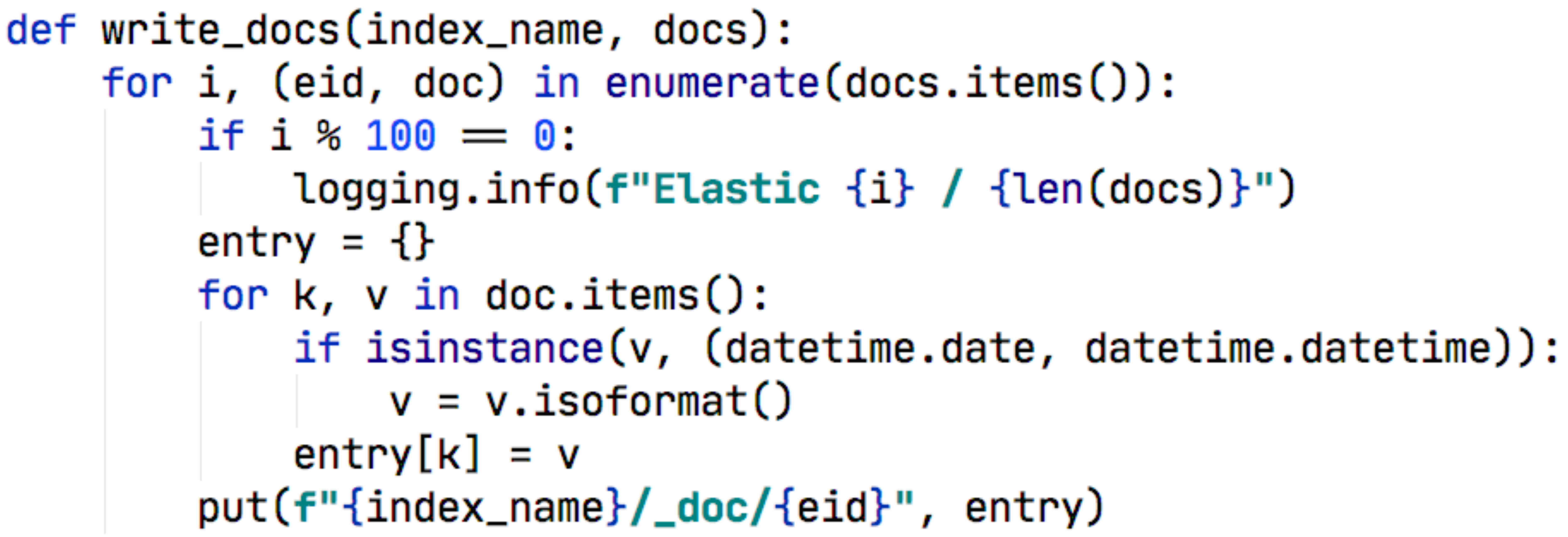
Used here

Index Definition in Elasticsearch

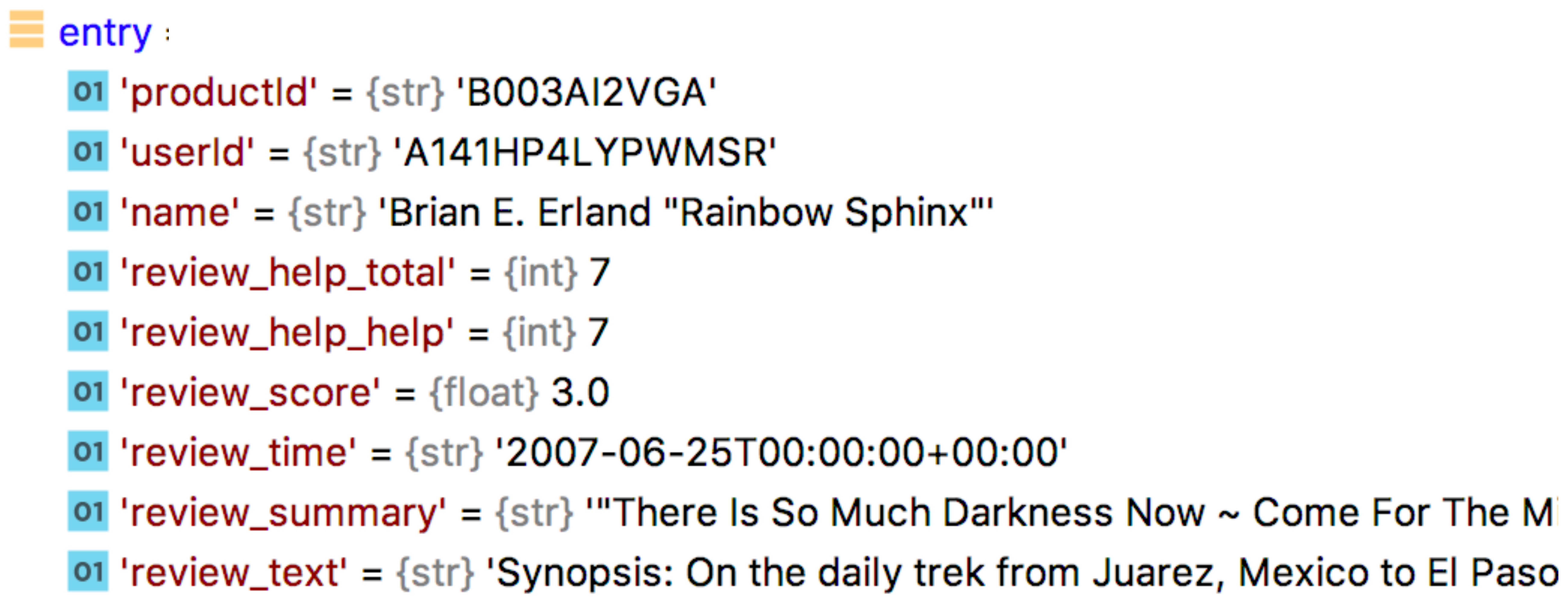
```python
def create_index(index_name):
    put(index_name)

    settings = {
        "analysis": {
            "filter": {
                "englishStopWords": {
                    "type": "stop",
                    "stopwords": "_english_"
                }
            },
            "analyzer": {
                "reviewAnalyzer": {
                    "tokenizer": "standard",
                    "filter": [
                        "lowercase",
                        "englishStopWords"
                    ]
                }
            }
        }
    }

    post(f"{index_name}/_close")
    put(f"{index_name}/_settings", settings)
    post(f"{index_name}/_open")
```

```python
mapping = {'properties':
               {'name': {'type': 'keyword'},
                'productId': {'type': 'keyword'},
                'review_help_help': {'type': 'long'},
                'review_help_total': {'type': 'long'},
                'review_score': {'type': 'float'},
                'review_summary': {
                    'type': 'text',
                    'analyzer': "reviewAnalyzer",
                    'search_analyzer': "reviewAnalyzer"
                },
                'review_text': {
                    'type': 'text',
                    'analyzer': "reviewAnalyzer",
                    'search_analyzer': "reviewAnalyzer"
                },
                'review_time': {'type': 'date'},
                'userId': {'type': 'keyword'}
                }
           }
put(f"{index_name}/_mapping", mapping)
```

Indexing Documents

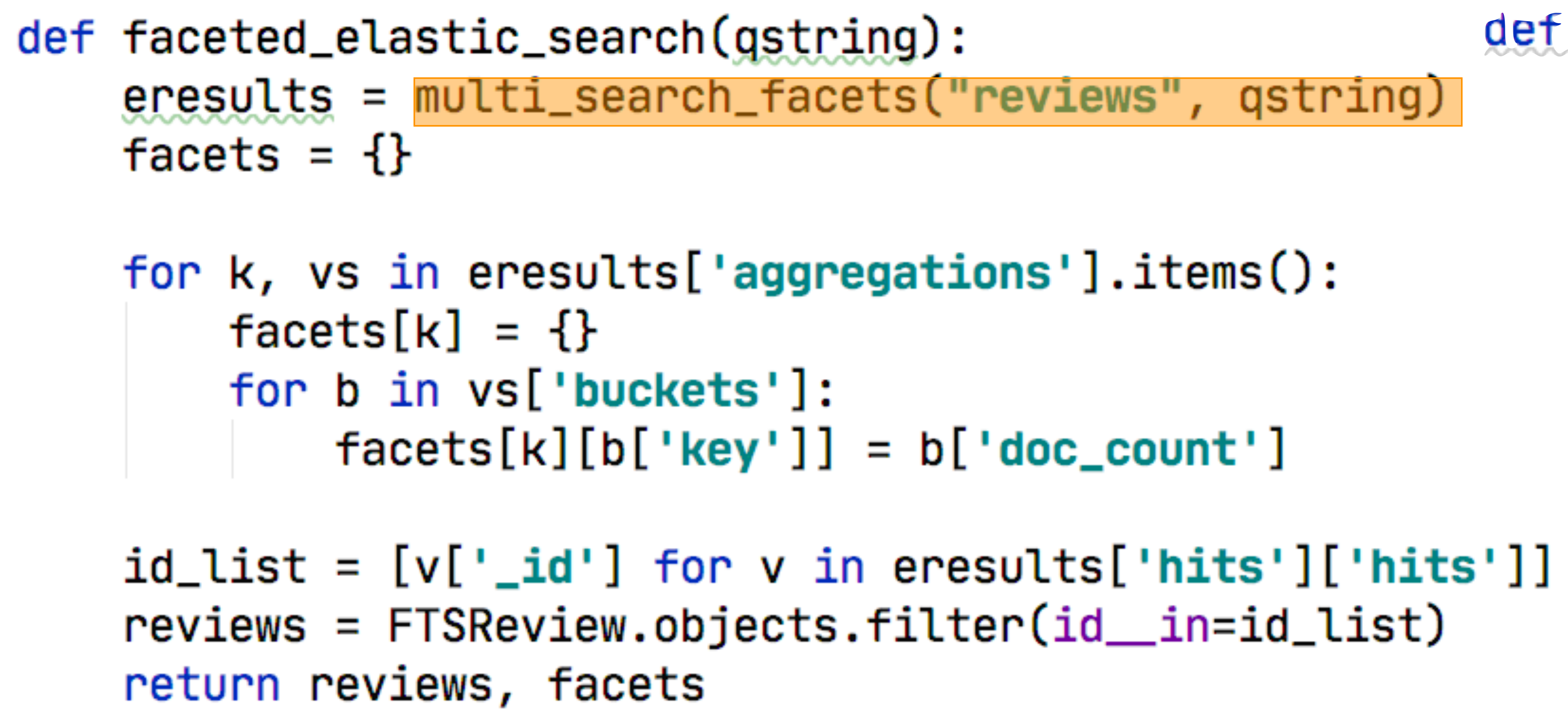


```python
def write_docs(index_name, docs):
    for i, (eid, doc) in enumerate(docs.items()):
        if i % 100 == 0:
            logging.info(f"Elastic {i} / {len(docs)}")
        entry = {}
        for k, v in doc.items():
            if isinstance(v, (datetime.date, datetime.datetime)):
                v = v.isoformat()
            entry[k] = v
        put(f"{index_name}/_doc/{eid}", entry)
```

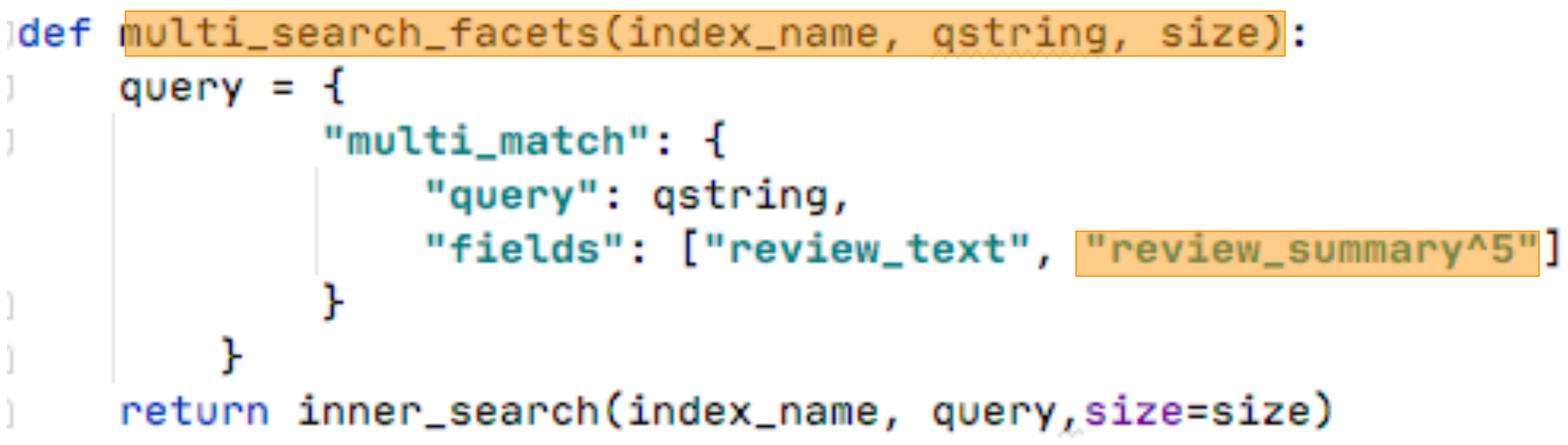


```
entry :
    01 'productId' = {str} 'B003AI2VGA'
    01 'userId' = {str} 'A141HP4LYPWMSR'
    01 'name' = {str} 'Brian E. Erland "Rainbow Sphinx"'
    01 'review_help_total' = {int} 7
    01 'review_help_help' = {int} 7
    01 'review_score' = {float} 3.0
    01 'review_time' = {str} '2007-06-25T00:00:00+00:00'
    01 'review_summary' = {str} '"There Is So Much Darkness Now ~ Come For The M
    01 'review_text' = {str} 'Synopsis: On the daily trek from Juarez, Mexico to El Paso
```

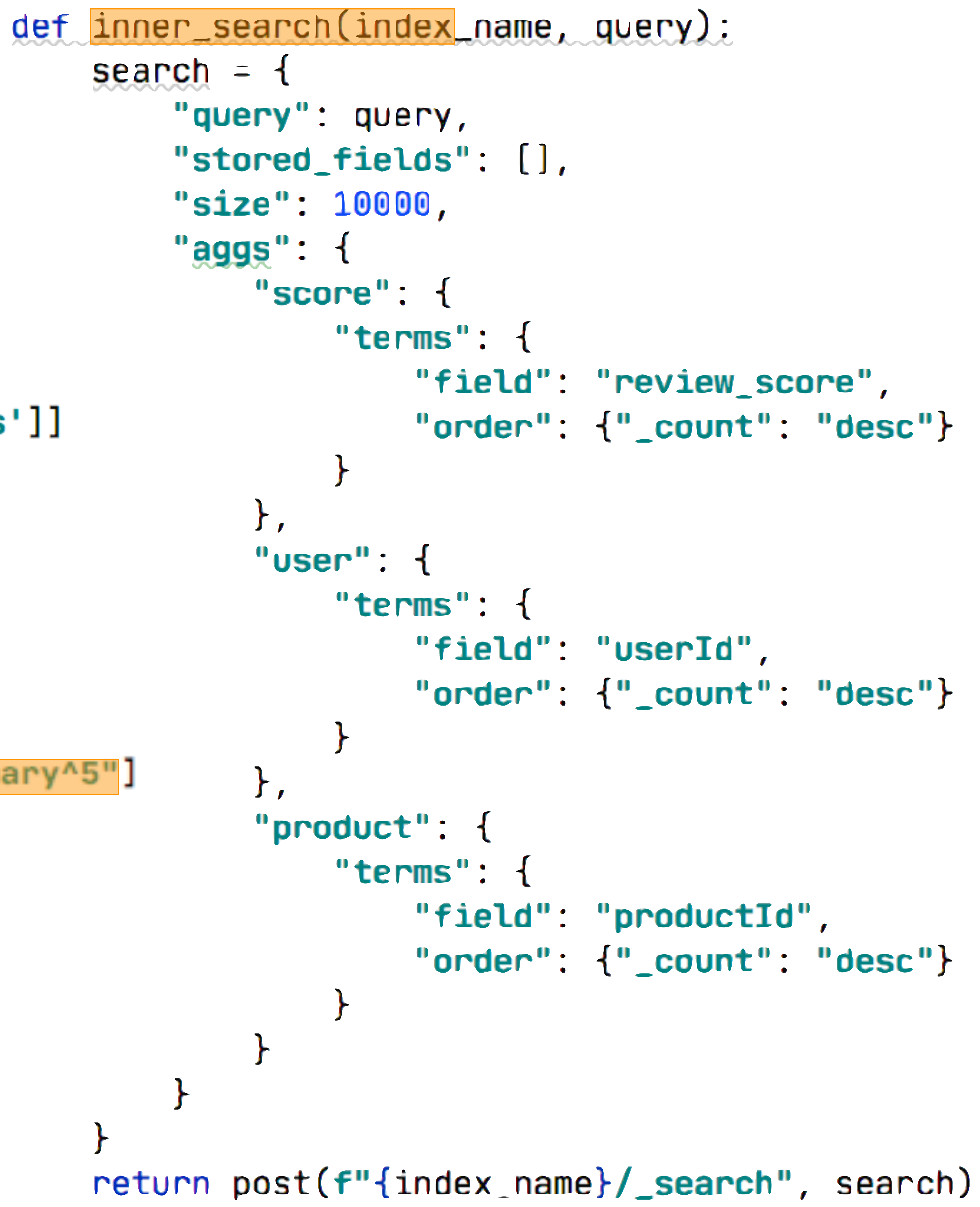
Search Example



```
def faceted_elastic_search(qstring):
    eresults = multi_search_facets("reviews", qstring)
    facets = {}

    for k, vs in eresults['aggregations'].items():
        facets[k] = {}
        for b in vs['buckets']:
            facets[k][b['key']] = b['doc_count']

    id_list = [v['_id'] for v in eresults['hits']['hits']]
    reviews = FTSReview.objects.filter(id__in=id_list)
    return reviews, facets
```

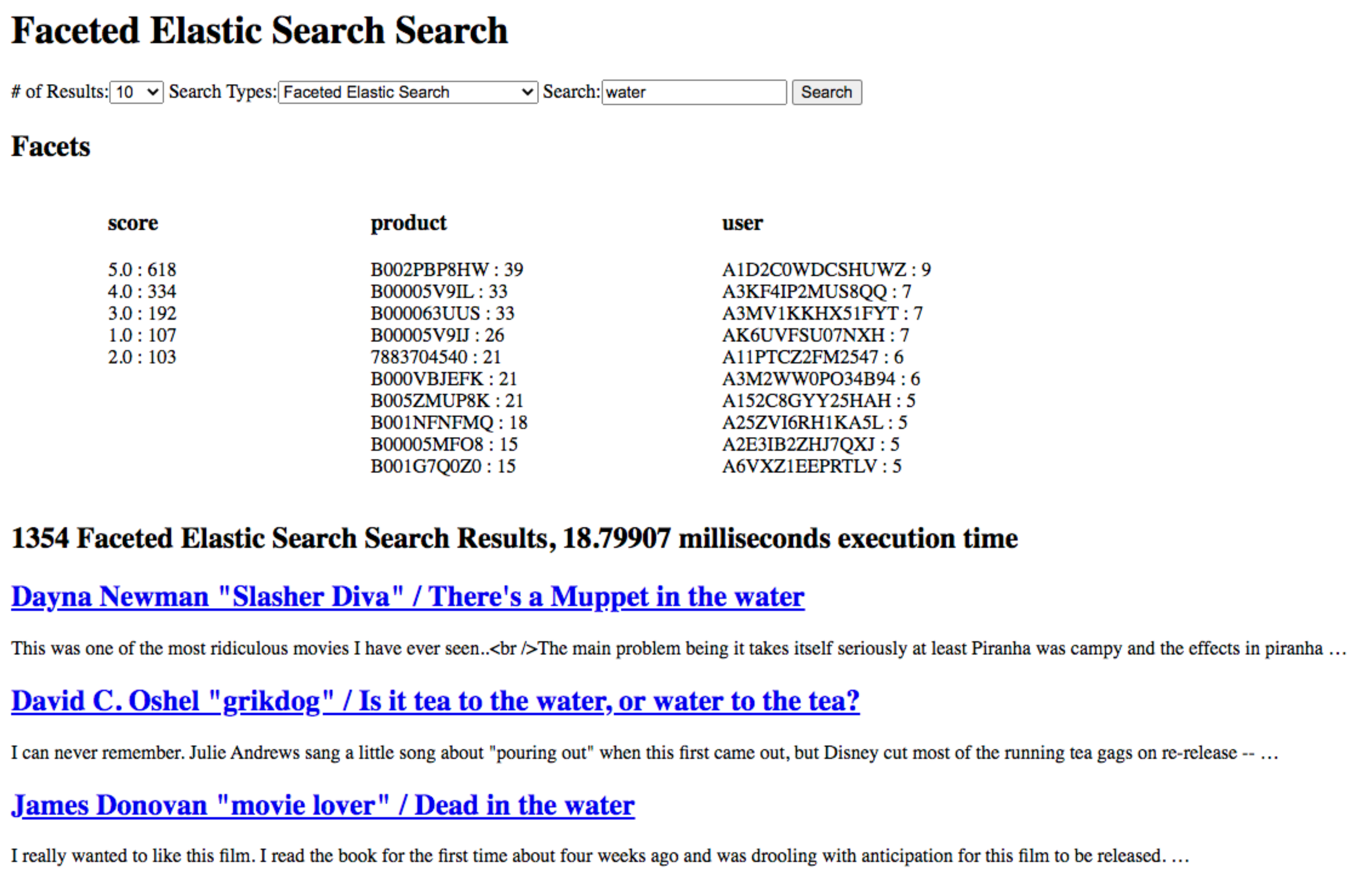


```
def multi_search_facets(index_name, qstring, size):
    query = {
            "multi_match": {
                "query": qstring,
                "fields": ["review_text", "review_summary^5"]
            }
        }
    return inner_search(index_name, query,size=size)
```



```
def inner_search(index_name, query):
    search = {
        "query": query,
        "stored_fields": [],
        "size": 10000,
        "aggs": {
            "score": {
                "terms": {
                    "field": "review_score",
                    "order": {"_count": "desc"}
                }
            },
            "user": {
                "terms": {
                    "field": "userId",
                    "order": {"_count": "desc"}
                }
            },
            "product": {
                "terms": {
                    "field": "productId",
                    "order": {"_count": "desc"}
                }
            }
        }
    }
    return post(f"{index_name}/_search", search)
```

Elasticsearch Results



Faceted Elastic Search Search

# of Results: 10 Search Types: Faceted Elastic Search Search: water Search

Facets

score	product	user
5.0 : 618	B002PBP8HW : 39	A1D2C0WDCSHUWZ : 9
4.0 : 334	B00005V9IL : 33	A3KF4IP2MUS8QQ : 7
3.0 : 192	B000063UUS : 33	A3MV1KKHX51FYT : 7
1.0 : 107	B00005V9IJ : 26	AK6UVFSU07NXH : 7
2.0 : 103	7883704540 : 21	A11PTCZ2FM2547 : 6
	B000VBJEFK : 21	A3M2WW0PO34B94 : 6
	B005ZMUP8K : 21	A152C8GYY25HAH : 5
	B001NFNFMQ : 18	A25ZVI6RH1KA5L : 5
	B00005MFO8 : 15	A2E3IB2ZHJ7QXJ : 5
	B001G7Q0Z0 : 15	A6VXZ1EEPRTLV : 5

1354 Faceted Elastic Search Search Results, 18.79907 milliseconds execution time

Dayna Newman "Slasher Diva" / There's a Muppet in the water

This was one of the most ridiculous movies I have ever seen..<br />The main problem being it takes itself seriously at least Piranha was campy and the effects in piranha …

David C. Oshel "grikdog" / Is it tea to the water, or water to the tea?

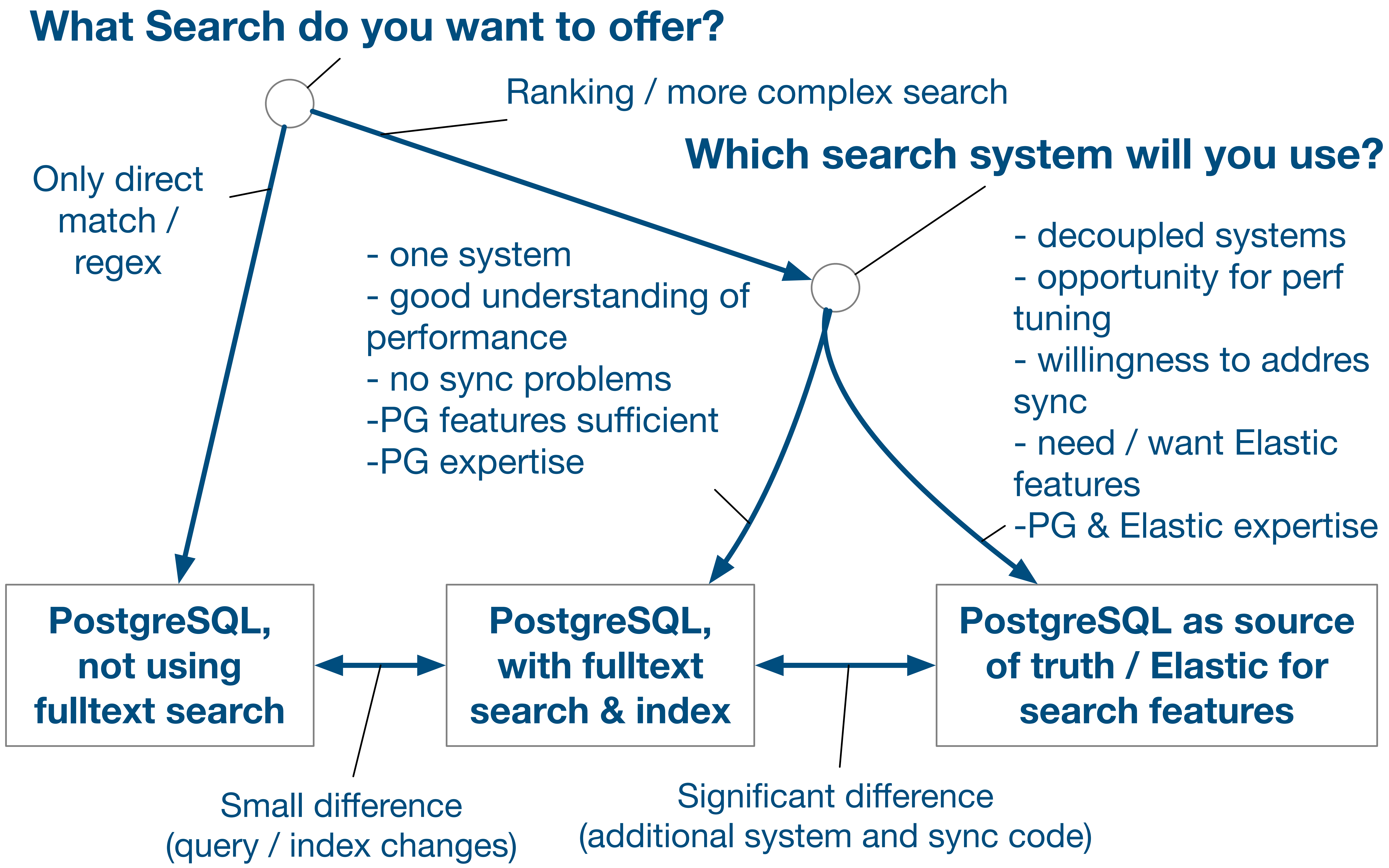
I can never remember. Julie Andrews sang a little song about "pouring out" when this first came out, but Disney cut most of the running tea gags on re-release -- …

James Donovan "movie lover" / Dead in the water

I really wanted to like this film. I read the book for the first time about four weeks ago and was drooling with anticipation for this film to be released. …

Summary

Deciding on a Search Solution


What Search do you want to offer?
Ranking / more complex search
Only direct match / regex
Which search system will you use?
- one system
- good understanding of performance
- no sync problems
-PG features sufficient
-PG expertise
- decoupled systems
- opportunity for perf tuning
- willingness to addres sync
- need / want Elastic features
-PG & Elastic expertise
PostgreSQL, not using fulltext search
PostgreSQL, with fulltext search & index
PostgreSQL as source of truth / Elastic for search features
Small difference (query / index changes)
Significant difference (additional system and sync code)

Personal Impressions (yours might differ)

		Postgres		Postgres & Elastic Search
Features	⬇	Search, Indexing, Preprocessing and Ranking Support	⬆	Larger Selection of Ranking and Preprocessing Options. Various related features (Aggregations / Facets)
Complexity	⬆	One, system, updates via trigger.	⬇	Need to keep two systems in sync
Performance	???	Depends on use case	⬆	Depends on use case, can be scaled independently from Postgres

Summary

Search is useful

Good search, relevance is hard.

Depends on tuning, know how, technology is ‘only’ a necessary enabler

We have good options available:

Postgres FTS, more or less out of the box

Elastic (or Solr, …) to build an independent search system

Thank you!

https://github.com/stbaercom/djangocon_eu_2020_searchoptions

Stefan Baerisch, stefan@stbaer.com, 2020-04-07