My Path to NoSQL

CouchDB, Redis, Elasticsearch

Karel Minařík

Karel Minařík

→ Software developer at Elasticsearch.com

→ Previously: Adobe Flash Developer, Art Director, Information Architect, Rails programmer, ...

→ Graduated as Philosophy major

→ @karmiq at Twitter

→ karmi.cz

CouchDB

relax

Apache CouchDB is a distributed, fault-tolerant and schema-free document-oriented database accessible via a RESTful HTTP/JSON API.

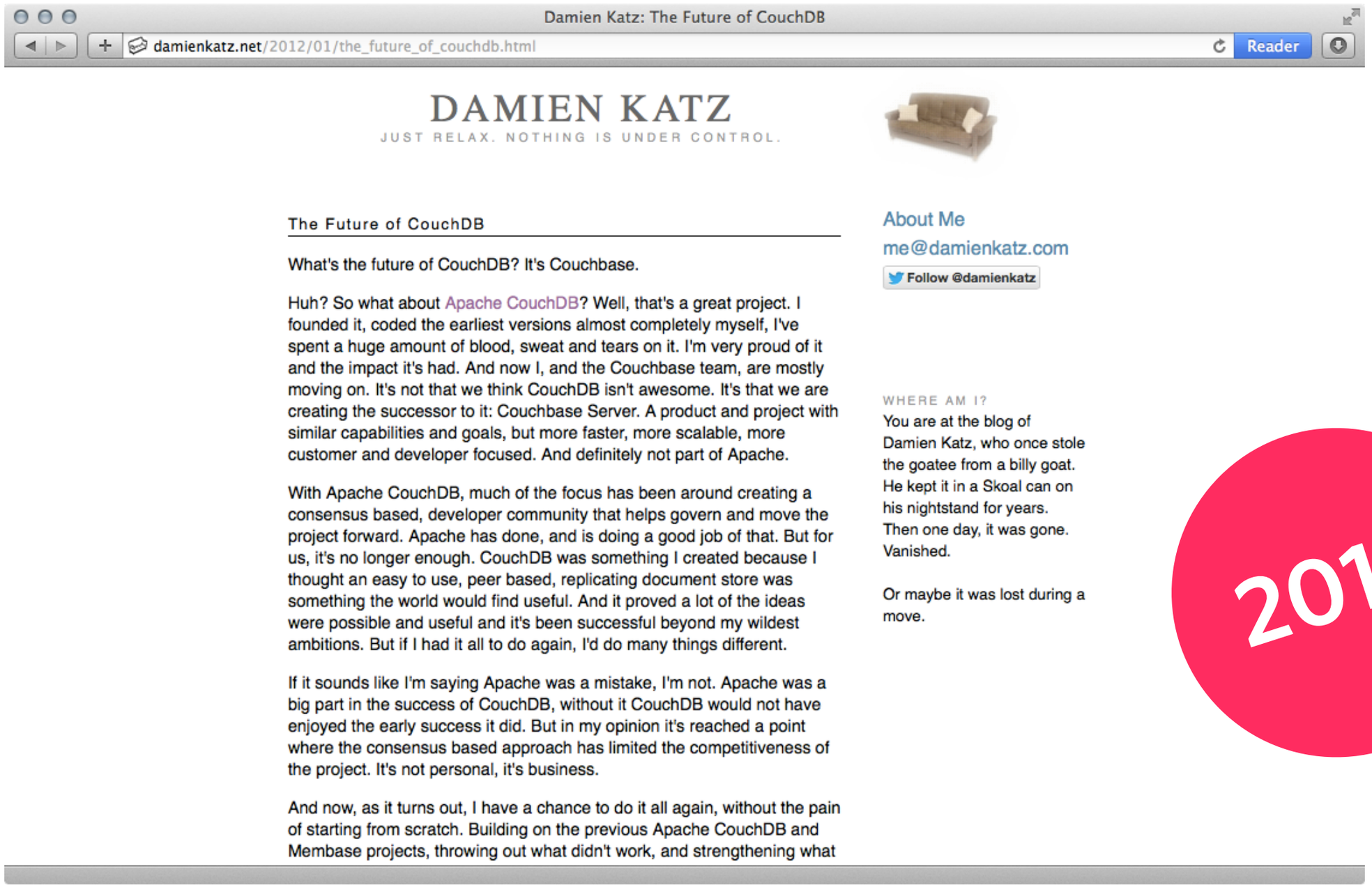
http://wiki.apache.org/couchdb

Apac hDB i ult-
tole -oriented
datab TTP/
JSON API.

http://wiki.apache.org/couchdb

DAMIEN KATZ

R.I.P, CouchDB



2012


NO SQL !!!1
Replication!
Decentralization!
Schema FREE!!!
JSON, not XML!
Clusters!
HTTP!!!!!
FAIL

Database denormalization at Digg

```
SELECT `digdate`, `id` FROM `Diggs`
WHERE `userid` IN (1, 2, 3, 4, ... 1000000)
AND itemid = 123 ORDER BY `digdate` DESC, `id` DESC;
```

"A full query can actually clock in at 1.5kb, which is many times larger than the actual data we want. With a cold cache, this query can take 14 seconds to execute."

"Non-relational data stores reverse this model completely, because they don't have the complex read operations of SQL. The model forces you to shift your computation to the writes, while reducing most reads to simple operations – the equivalent of `SELECT * FROM `Table`.`"

http://about.digg.com/blog/looking-future-cassandra

“Relational Data”

That does **not** mean the world conforms to the **third normal form**.

In fact, it's rather **the exact opposite.**

The Textbook Example

Design a **customer database.**

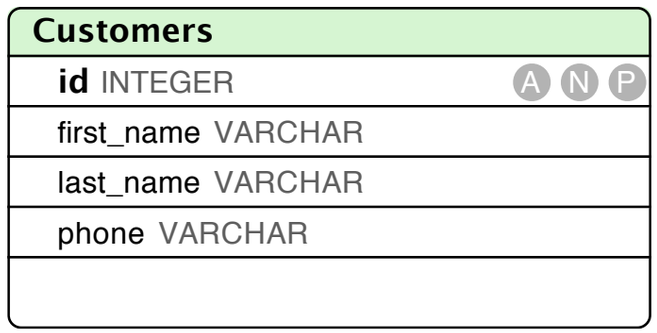
People have names, e-mail, phone numbers, ...

How many phone numbers?

The Textbook Example

Relational Databases 101

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659
789	Maria	Fernandez	555-808-9633

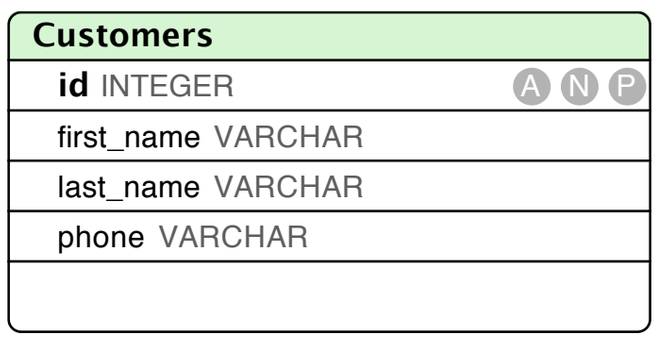


Now. What about multiple phone numbers?

The Textbook Example

The “solution”, Pt. 1

Customer ID	First Name	Surname	Telephone Numbers
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659, 555-776-4100
789	Maria	Fernandez	555-808-9633



“We will use the database only from the application, anyway.”

http://en.wikipedia.org/wiki/First_normal_form#Domains_and_values

The Textbook Example

The “solution”, Pt. 2

Customer ID	First Name	Surname	Tel. No. 1	Tel. No. 2	Tel. No. 3
123	Robert	Ingram	555-861-2025		
456	Jane	Wright	555-403-1659	555-776-4100	555-403-1659
789	Maria	Fernandez	555-808-9633		

Customers	
id INTEGER	A N P
first_name VARCHAR	
last_name VARCHAR	
phone_1 VARCHAR	
phone_2 VARCHAR	
phone_3 VARCHAR	

“This is clearly better design!”

Alright. Then, please answer these questions:

- How do you search for a customers given a phone number?
- Which customers have the same phone number?
- How many phone numbers a customer has?

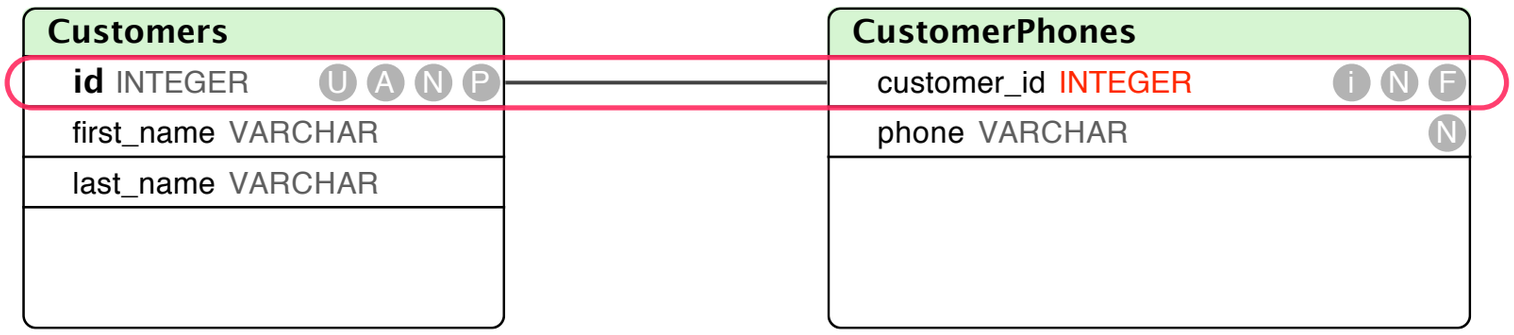
Then, please add the ability to store **four** phone numbers. Thanks.

The Textbook Example

The Right Solution

Customer ID	First Name	Surname
123	Robert	Ingram
456	Jane	Wright
789	Maria	Fernandez

Customer ID	Telephone Number
123	555-861-2025
456	555-403-1659
456	555-776-4100
789	555-808-9633



The Textbook Example

```
mysql> SELECT * FROM Customers LEFT JOIN CustomerPhones
                              ON Customers.id = CustomerPhones.customer_id;

+----+------------+-----------+-------------+-------+
| id | first_name | last_name | customer_id | phone |
+----+------------+-----------+-------------+-------+
|  1 | John       | Smith     | 1           | 123   |
|  1 | John       | Smith     | 1           | 456   |
+----+------------+-----------+-------------+-------+
```

The Textbook Example

```
mysql> SELECT * FROM Customers WHERE id = 1;

+----+------------+-----------+
| id | first_name | last_name |
+----+------------+-----------+
|  1 | John       | Smith     |
+----+------------+-----------+
```

```
mysql> SELECT phone FROM CustomerPhones WHERE customer_id IN (1);

+-------+
| phone |
+-------+
| 123   |
| 456   |
+-------+
```

Structured data

But, damn!, I want something like this:

```
{
  "id"         : 1,

  "first_name" : "Clara",
  "last_name"  : "Rice",

  "phones"     : ["0000 777 888 999", "0000 123 456 789", "0000 314 181 116"]
}
```

"No problem, you just iterate over the rows and build your object. That's the way it is!"

"If this would be too painful, we will put some cache there."

Ephemeral data

Not everything needs to be done “right”. Right?

```
class User < ActiveRecord::Base
  serialize :preferences
end
```

“Consistency”

Does the “Right Way“ sometimes fail?

Hell yeah.

EXAMPLE

When designing an invoicing application, you store the customer for the invoice the “right way”, via foreign keys.

Then, the customer address changes.

Did the address on the invoice also changed?

Documents in the Real World

Documents in the Real World


```
{
    "_id"          : "clara-rice",
    "_rev"         : "1-def456",

    "first_name"   : "Clara",
    "last_name"    : "Rice",

    "phones"       : {
      "mobile" : "0000 777 888 999"
      "home"   : "0000 123 456 789",
      "work"   : "0000 314 181 116"
    },

    "addresses"    : {
      "home"   : {
        "street"  : "Wintheiser Ports",
        "number"  : "789/23",
        "city"    : "Erinshire",
        "country" : "United Kingdom"
      },
    },

    "occupation" : "model",
    "birthday"   : "1970/05/01",

    "groups"     : ["friends", "models"],

    "created_at" : "2010/01/01 10:00:00 +0000"
}
```

Built “Of the Web”

Django may be built for the Web,
but CouchDB is built of the Web.

I’ve never seen software that so completely
embraces the philosophies behind HTTP.

Jacob Kaplan-Moss, Of the Web (2007)

http://jacobian.org/writing/of-the-web/

HTTP API

```
HOST=http://localhost:5984

curl -X GET $HOST
# {"couchdb":"Welcome","version":"0.11.0b22c551bb-git"}

curl -X GET $HOST/my-database
# {"error":"not_found","reason":"no_db_file"}

curl -X PUT $HOST/my-database
# {"ok":true}

curl -X PUT $HOST/my-database/abc123 -d '{"foo":"bar"}'
# {"ok":true,"id":"abc123","rev":"1-4c6114c65e295552ab1019e2b046b10e"}

curl -X GET $HOST/my-database/abc123
# {"_id":"abc123","_rev":"1-4c6114c65e295552ab1019e2b046b10e","foo":"bar"}

curl -X DELETE $HOST/my-database/abc123?rev=2-d179f665eb01834faf192153dc72dcb3
# {"ok":true,"id":"abc123","rev":"1-4c6114c65e295552ab1019e2b046b10e"}
```

Easy To Wrap

```
require 'rubygems'
require 'ostruct'

require 'restclient'
require 'json'

class Article < OpenStruct

  def self.db(path='')
    RestClient::Resource.new "http://localhost:5984/blog/#{path}",
                             :headers => { :content_type => :json, :accept => :json }
  end


  db.put '' rescue RestClient::PreconditionFailed

  def self.create(params={})
    new db.post(params.to_json)
  end

  def self.find(id)
    new JSON.parse( db(id).get )
  end

  def destroy
    self.class.db(self._id + "?rev=#{self._rev}").delete
  end

end
```

1 HTTP library

2 JSON library

What is “RESTful”?

REST is a set of principles that define how Web standards, such as HTTP and URIs, are supposed to be used. (...) In summary, the five key principles are:

- Give every “thing” an ID
- Link things together
- Use standard methods
- Resources with multiple representations
- Communicate statelessly

Stefan Tilkov, *A Brief Introduction to REST*

What is “RESTful”?

The basic idea is even more simple, though.

HTTP is not just a “transfer protocol”.

It is the interface for interacting with “things” themselves.

Playing Tricks with HTTP

Elasticsearch.org Playing HTTP Tricks With Nginx | Blog | Elasticsearch

www.elasticsearch.org/blog/playing-http-tricks-nginx/ Reader

elasticsearch.

OVERVIEW RESOURCES COMMUNITY CASE STUDIES BLOG COMPANY DOWNLOAD

playing http tricks with nginx

Karel Minařík October 7, 2014

One of the defining features of Elasticsearch is that it's exposed as a (loosely) RESTful service over HTTP.

The benefits are easy to spell out, of course: the API is familiar and predictable to all web developers. It's easy to use with "bare hands" via the `curl` command, or in the browser. It's easy to write API wrappers in various programming languages.

Nevertheless, the importance of the HTTP-based nature of Elasticsearch is rooted deeper: in the way it fits into the existing paradigm of software development and architecture.

http as an architectural paradigm

Display a menu

The Google Paper

Google Research Publication: MapReduce

http://labs.google.com/papers/mapreduce.html

map reduce google

Google Research Publications — Google Labs Home

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

Appeared in:
OSDI'04: Sixth Symposium on Operating System Design and Implementation,
San Francisco, CA, December, 2004.

Download: PDF Version

Slides: HTML Slides

The Concept

```
module Enumerable
  alias :reduce :inject unless method_defined? :reduce
end
```

```
(1..3).map    { |number| number * 2 }
# => [2, 4, 6]

(1..3).reduce(0) { |sum, number| sum += number}
# => 6
```

The Simplest View

```
function(doc) {
  if (doc.last_name && doc.first_name) {
    emit( doc.last_name + ' ' + doc.first_name, doc )
  }
}
```

The Simplest View

INPUT

```
function(doc) {
  if (doc.last_name && doc.first_name) {
    emit( doc.last_name + ' ' + doc.first_name, doc )
  }
}
```

OUTPUT KEY VALUE

The Result of Map

Key	Value

"Armstrong Lottie"	_id: "lottie-armstrong",
	_rev: "2-fcb71b26096957b3ff3ffd2970f3c933",
	addresses: {
	home: {
	city: "Murphyville"
	...
	}
	},
	first_name: "Lottie",
	last_name: "Armstrong",
	occupation: "programmer",

"Bailey Kaelyn"	_id: "kaelyn-bailey",
	_rev: "1-2e25e6c9448520fa796988894423a23b",
	addresses: {
	home: {
	city: "Lake Dedric"
	...
	}
	},
	first_name: "Kaelyn",
	last_name: "Bailey",
	occupation: "supermodel"

...	...

The Result of Map

http://localhost:5984/addressbook/_design/person/_view/by_name?reduce=false

http://localhost:5984/addressbook/_design/person/_view/by_name?reduce=false

Google

{delicious} [rack::bug] localhost:3000 localhost:4567 futon [doc] [GData API]

Apache CouchDB - Futon: Brow... | http://localhost:5984/address...


```
{
   total_rows: 10,
   offset: 0,
 - rows: [
     - {
         id: "lottie-armstrong",
         key: "Armstrong Lottie",
       - value: {
           _id: "lottie-armstrong",
           _rev: "2-fcb71b26096957b3ff3ffd2970f3c933",
         + addresses: { … },
           first_name: "Lottie",
           last_name: "Armstrong",
           occupation: "programmer",
         + phones: { … },
         + groups: [ … ],
           birthday: "1952/10/24",
         + _attachments: { … }
       }
     },
   - {
       id: "kaelyn-bailey",
       key: "Bailey Kaelyn",
     - value: {
           _id: "kaelyn-bailey",
           _rev: "1-2e25e6c9448520fa796988894423a23b",
         + addresses: { … },
           first_name: "Kaelyn",
           last_name: "Bailey",
           occupation: "supermodel",
         - phones: {
             cell: "1-758-583-3506 x57386",
             home: "(738)356-9599 x5321"
           },
         + groups: [ … ],
           birthday: "1970/07/09"
       }
     },
```


Done

Even Simpler View

```
function(doc) {
  emit(doc.occupation, 1);
}
```

Result of Even Simpler View

Key ▼ Grouping: exact	Value Reduce
"supermodel" ID: kaelyn-bailey	1
"supermodel" ID: jeramie-feest	1
"supermodel" ID: edd-mosciski	1
"supermodel" ID: desmond-stokes	1
"supermodel" ID: bertram-bogan	1
"supermodel" ID: andreanne-jakubowski	1
"programmer" ID: lottie-armstrong	1
"designer" ID: khalid-thiel	1
"designer" ID: jo-lubowitz	1
"designer" ID: dorothy-von	1

Showing 1-10 of 10 rows ← Previous Page | Rows per page: 50 | Next Page →

A Simple Reduce

```
function(keys, values) {
  return sum(values)
}
```

Result of a Simple Reduce

Key	Grouping: exact	Value	Reduce
"supermodel"		6	
"programmer"		1	
"designer"		3	

Showing 1-3 of unknown rows ← Previous Page | Rows per page: 50 | Next Page →

Group Levels

```
function(doc) {
  var date = new Date(doc.birthday)
  emit( [date.getFullYear(), date.getMonth()+1, date.getDate()], 1 )
}
```

COMPOSITE KEY (ARRAY)

```
_count
```

Group Level Exact

Key	Grouping: exact	Value	Reduce
[1990, 5, 9]		1	
[1990, 5, 3]		1	
[1988, 11, 20]		1	
[1986, 11, 9]		1	
[1983, 5, 4]		1	
[1983, 2, 27]		1	
[1979, 11, 21]		1	
[1978, 5, 29]		1	
[1975, 11, 24]		1	
[1971, 5, 5]		1	

Showing 1-10 of unknown rows ← Previous Page | Rows per page: 50 | Next Page →

Group Level 2

Key ▼ Grouping: level 2	Value ☑ Reduce
[1990, 5]	2
[1988, 11]	1
[1986, 11]	1
[1983, 5]	1
[1983, 2]	1
[1979, 11]	1
[1978, 5]	1
[1975, 11]	1
[1971, 5]	1

Showing 1-9 of unknown rows ← Previous Page | Rows per page: 50 | Next Page →

Group Level 1

Key ▼ Grouping: level 1	Value Reduce
[1990]	2
[1988]	1
[1986]	1
[1983]	2
[1979]	1
[1978]	1
[1975]	1
[1971]	1

Showing 1-8 of unknown rows ← Previous Page | Rows per page: 50 | Next Page →

Complex Queries

So… What if you need something like:

Show me all designers who live in Beckerborough.

Out of luck?

COMPLEX QUERIES

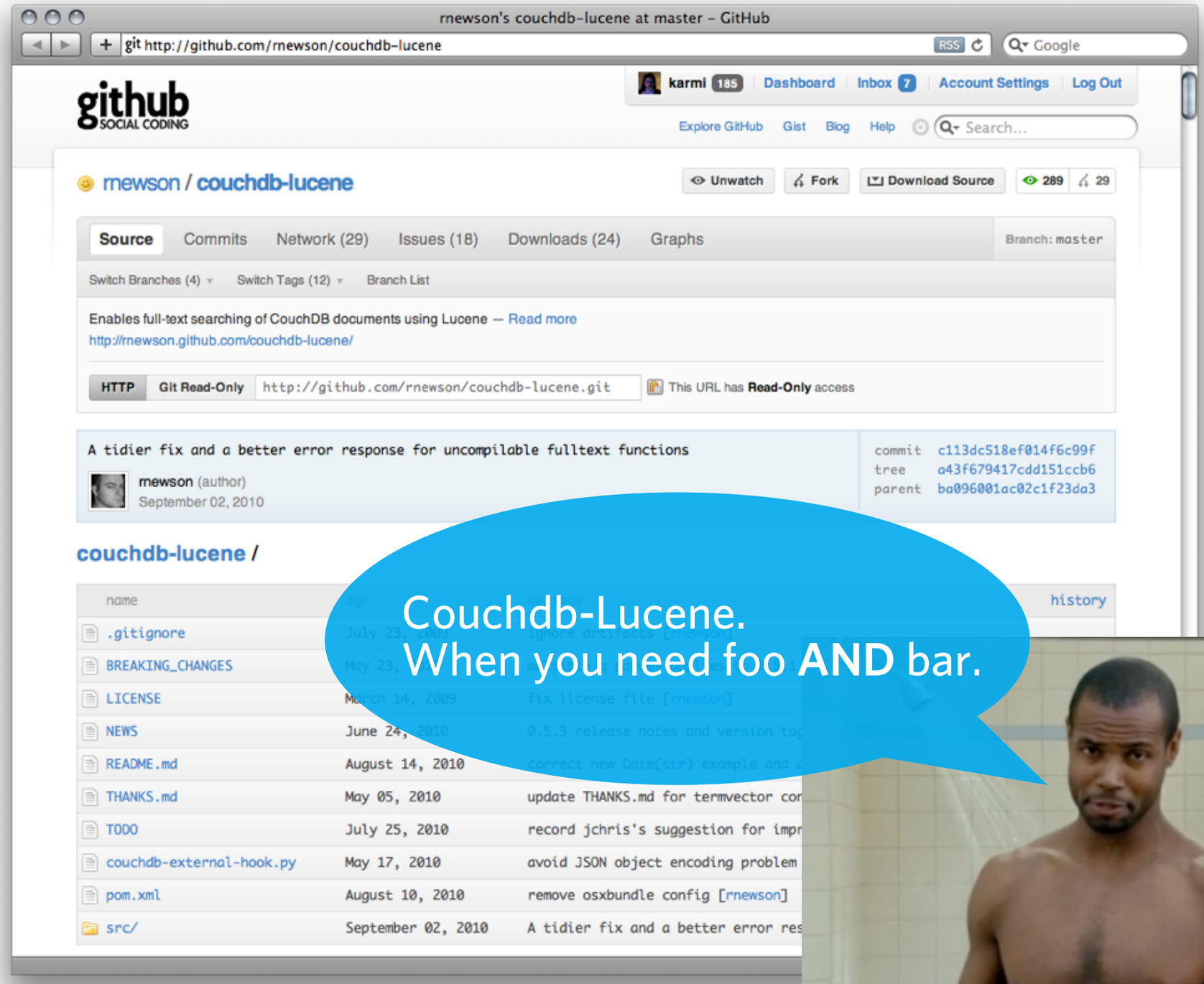
CouchDB-Lucene

Robert Newson

This guy knows.

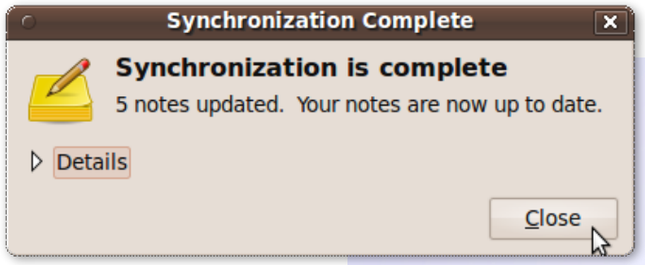
Show me all designers who live in Beckerborough.

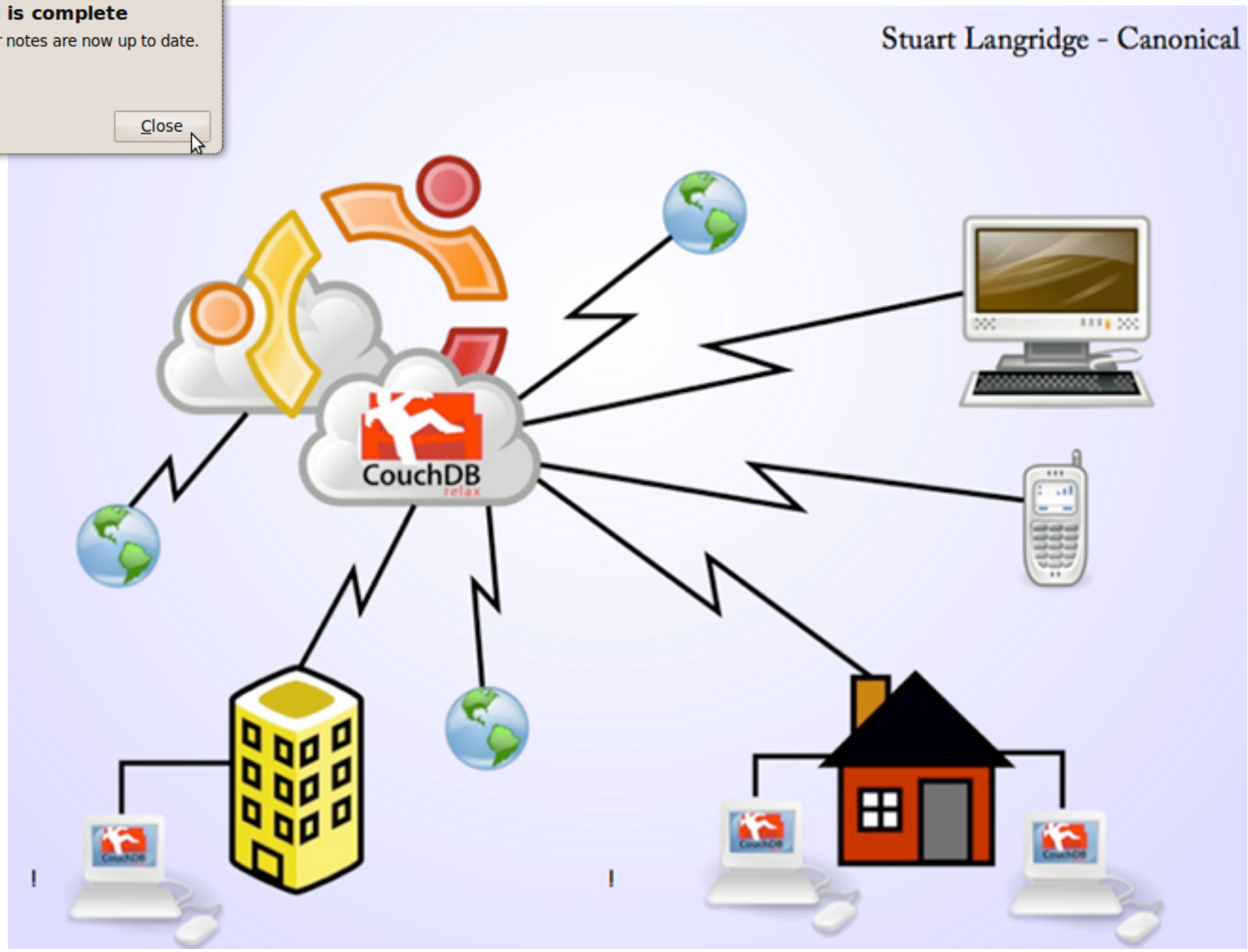
foo AND bar



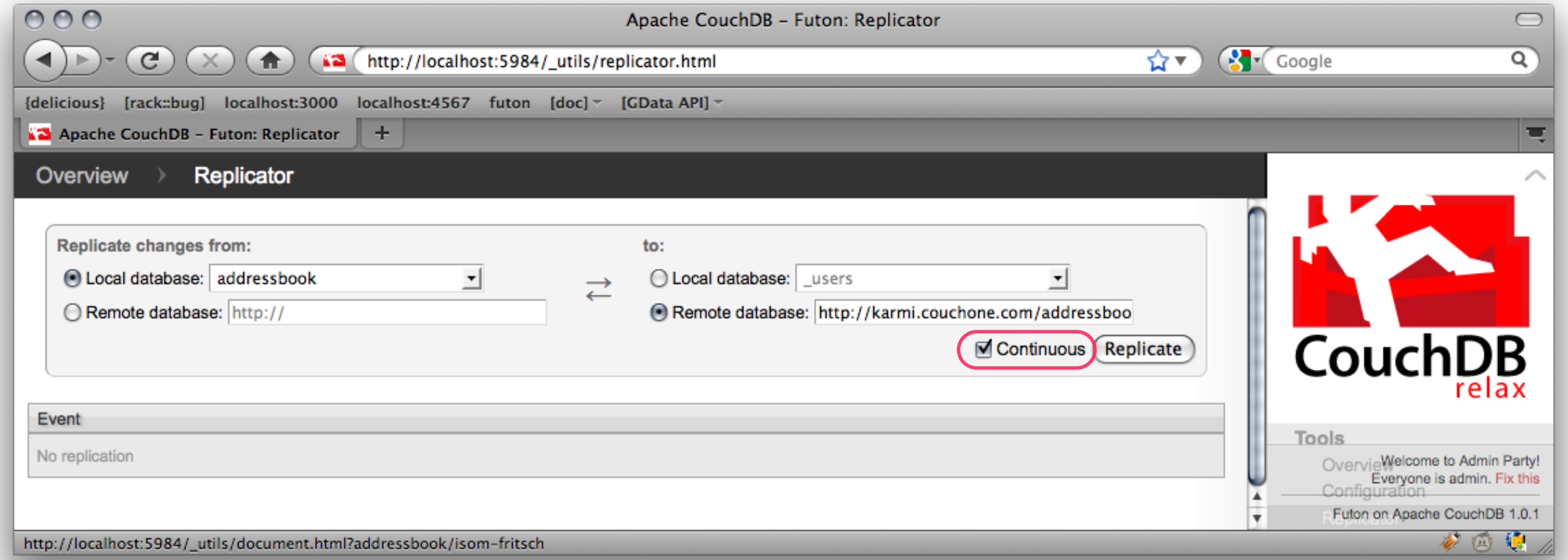
http://github.com/rnewson/couchdb-lucene

Ubuntu One

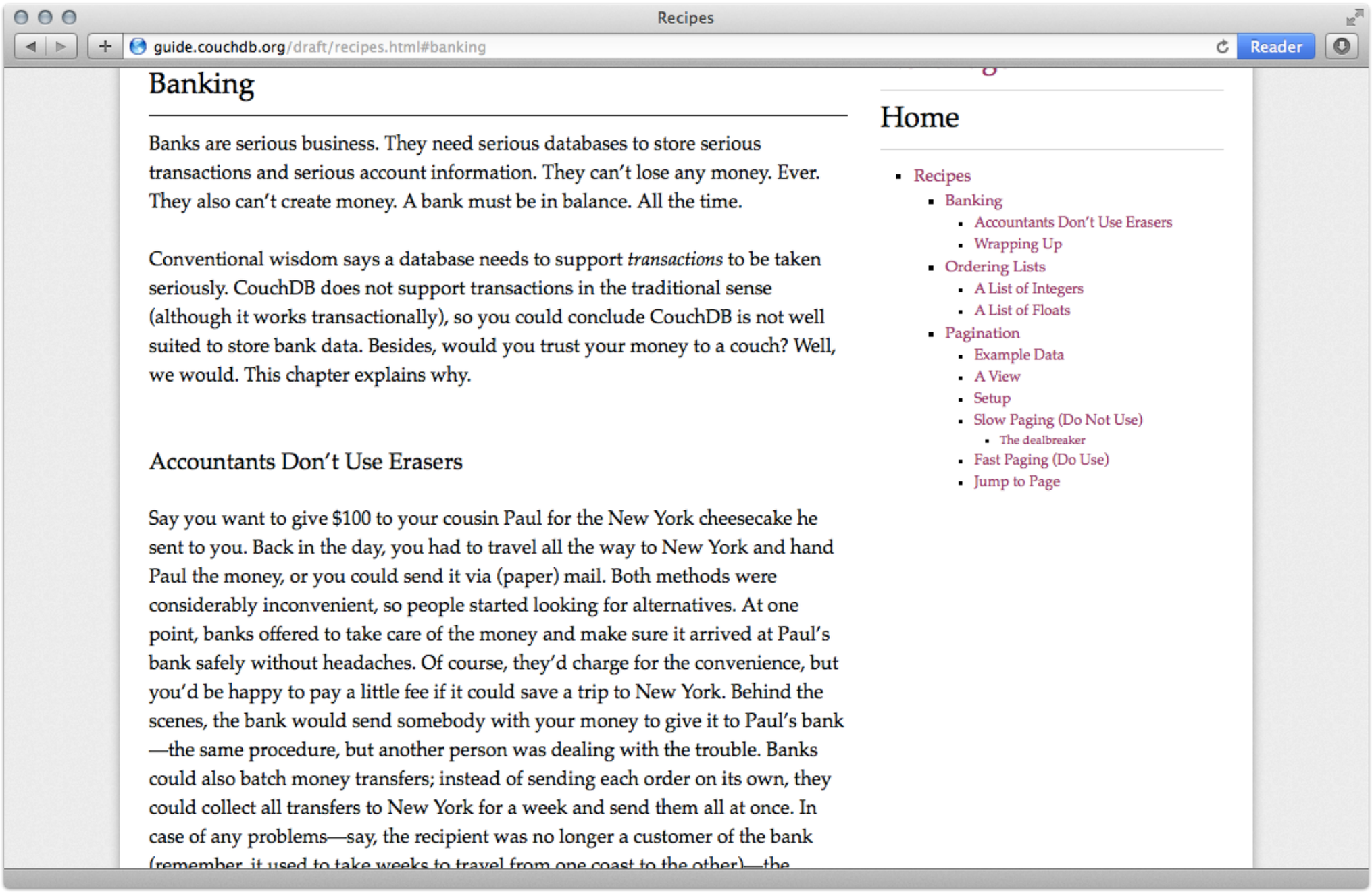

Synchronization Complete
Synchronization is complete
5 notes updated. Your notes are now up to date.
Details
Close


Stuart Langridge - Canonical
CouchDB
relax

Replication

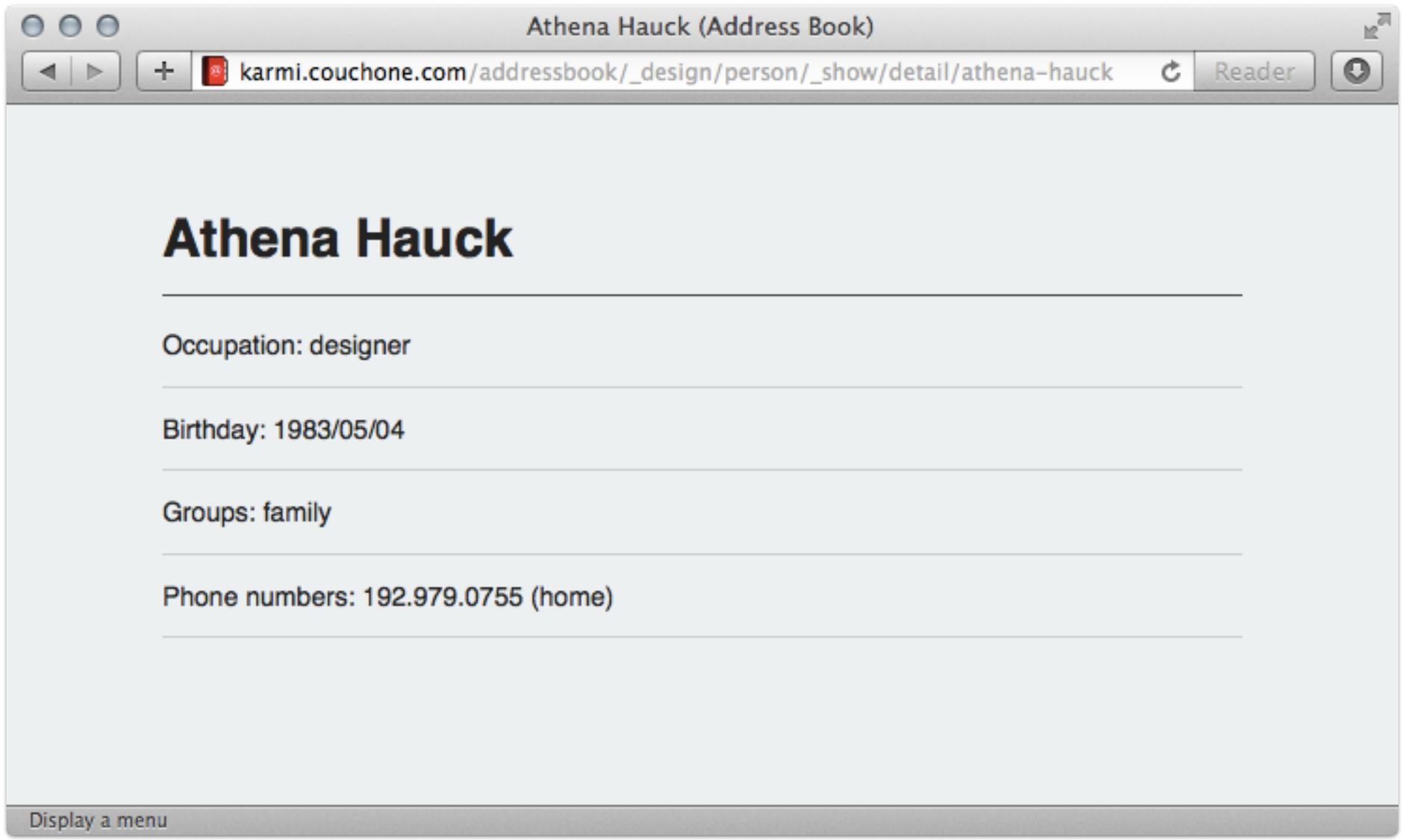


“Accountants don’t use erasers”

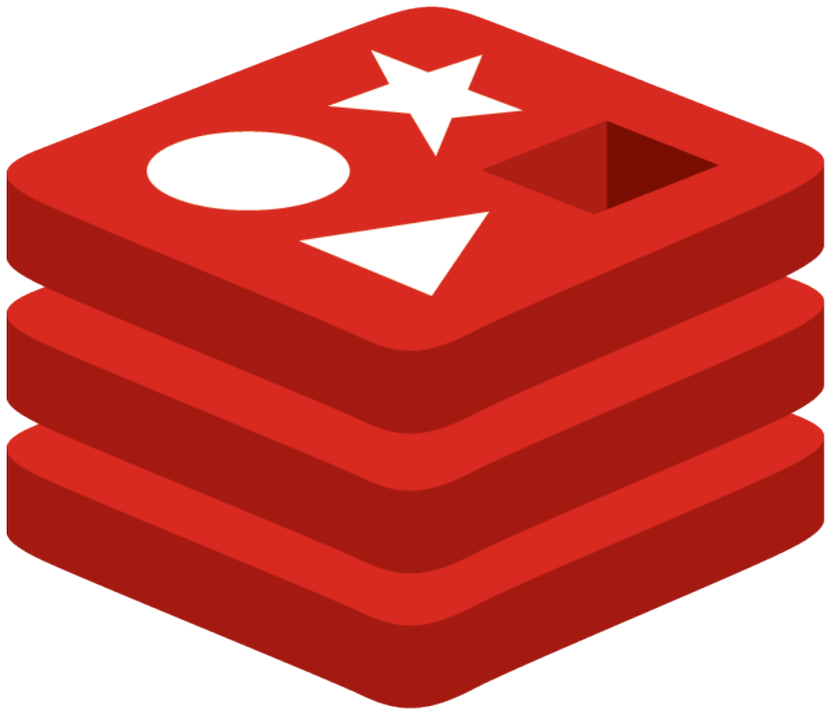


DEMO

Example “CouchApp”



SOURCE CODE: http://github.com/karmi/couchdb-showcase



redis

AK-47

noun
a type of assault rifle, originally manufactured in the Soviet Union.

ORIGIN acronym for Russian ***Avtomat Kalashnikova 1947,*** the designation of the original model, designed in 1947 by Mikhail T. Kalashnikov (b. 1919).

http://www.youtube.com/watch?v=J6c3DLlM9KA#t=8m32s

AK-47

Designed at the end of WWII by Mikhail Kalashnikov

Driven by changes in military strategy and tactics

Extremely simple design

Designed for mass production & low-quality manufacturing

Extreme **reliability**, at the cost of accuracy

Reliability

sim•plic•i•ty |simˈplisitē|
noun

Reliability

Speed

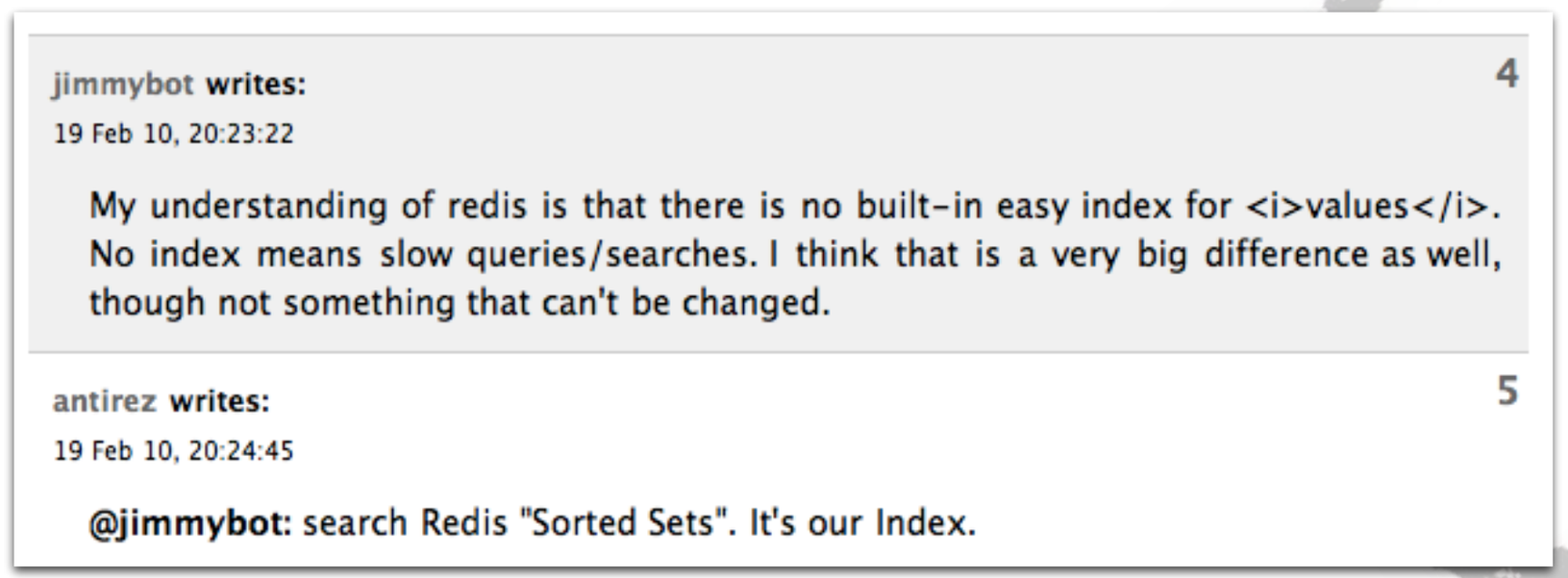
Simplicity

Versatility

Predictability

Low Footprint

The Zen of Redis



jimmybot **writes:** 4

19 Feb 10, 20:23:22

My understanding of redis is that there is no built-in easy index for values. No index means slow queries/searches. I think that is a very big difference as well, though not something that can't be changed.

antirez **writes:** 5

19 Feb 10, 20:24:45

@jimmybot: search Redis "Sorted Sets". It's our Index.

(...) what Redis provides are data structures (...)

Strings

```
SET key "value"  ~ 4 billions of keys
GET key
=> "value"
DEL key
```

Expiration

```
EXPIRE key 5
GET key
=> "value"
TTL key
=> 1
GET key
=> (nil)
```

Cache

http://antirez.com/post/redis-as-LRU-cache.html

Sessions

https://github.com/mattmatt/redis-session-store

Atomic Increments

```
GET key
=> nil
INCR key
=> 1
INCR key
=> 2
GET key
=> 2
```

Counters (downloads, hits, votes, ...)

```
$ curl http://example.com/downloads/file1.mpg
```

```
INCR downloads:total
INCR downloads:total:today
INCR downloads:total:2011-05-10
INCR downloads:/downloads/file1.mpg:total
INCR downloads:/downloads/file1.mpg:today
INCR downloads:/downloads/file1.mpg:2011-05-10
```

Usage

Counters (downloads, hits, votes, ...)

```
# Total downloads for server, all time
GET downloads:total

# Total downloads for server, today
GET downloads:total:today

# Total downloads for file
GET downloads:/downloads/file1.mpg:total

# Total downloads for file today
GET downloads:/downloads/file1.mpg:today
...
```

Counters (downloads, hits, votes, ...)

```
# Expire at 2011-05-10 23:59:59
EXPIREAT downloads:total:today 1305064799
```

All this runs at super-sonic speed, with minimal overhead and resource consumption.

See implementation for Rubygems.org: https://gist.github.com/296921

However, you'll hit *denormalization bloat* once you start adding metrics (eg. downloads per country, per category, ...)

Variations: **Rate limiting**

```
$ curl http://api.example.com/list.json
```

```
INCR api:<TOKEN>:hits
=> 1
```

```
if INCR('api:abc123:hits') > LIMIT
  return 420 Enhance Your Calm
end
```

```
# Every hour...
DEL api:<TOKEN>:hits
```

Lists

```
LPUSH key 1
=> 1
LPUSH key 2
=> 2
LPUSH key 3
=> 3
LRANGE key 0 -1
=> "3"
=> "2"
=> "1"
```

```
RPOP key
=> "1"
LRANGE key 0 -1
=> "3"
=> "2"
LLEN key
=> 2
LTRIM key 0 1
=> OK
```

Usage

Indexes (list of comments, ...)

```
LPUSH article:comments:<ID> “New comment"
```

Timelines (of all sorts: messages, logs, ...)

```
LPUSH user:<ID>:inbox "Message from Alice"
LPUSH user:<ID>:inbox "Message from Bob"
# Limit the messages to 100
LTRIM user:<ID>:inbox 0 99
# Get last 10 messages
LRANGE user:<ID>:inbox 0 9
# Get next 10 messages
LRANGE user:<ID>:inbox 10 19
```

Queues

```
# Publisher
RPUSH queue "task-1"
RPUSH queue "task-2"
```

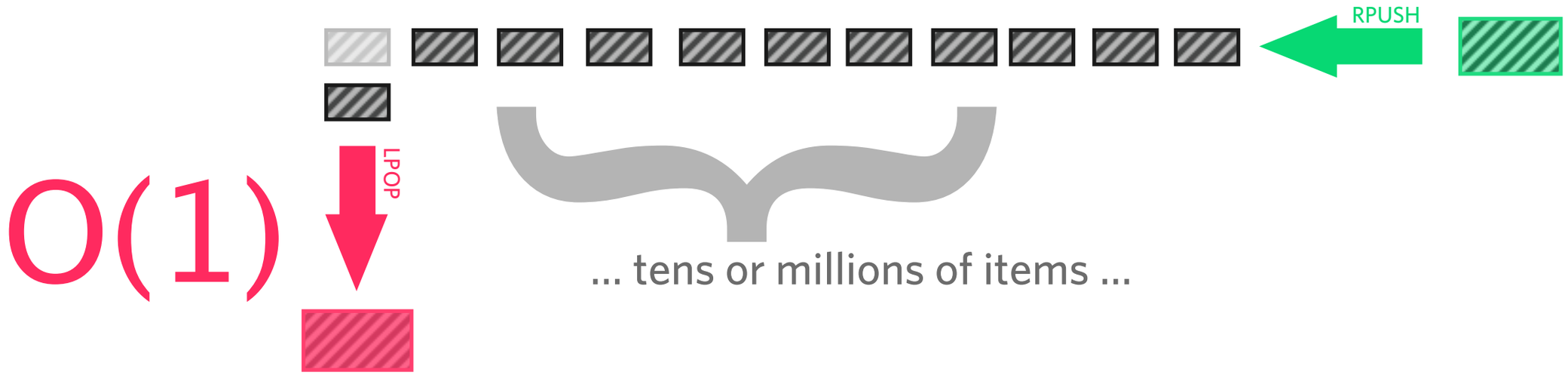
```
# Worker (blocks and waits for tasks)
BLPOP queue 0
```

Queues

```
# publisher.sh
for i in {1..10}; do
  redis-cli RPUSH "queue" "task-$i"
done
```

```
# worker.sh
while true; do
  redis-cli BLPOP "queue" 0
done
```

Resque: Background Processing from Github

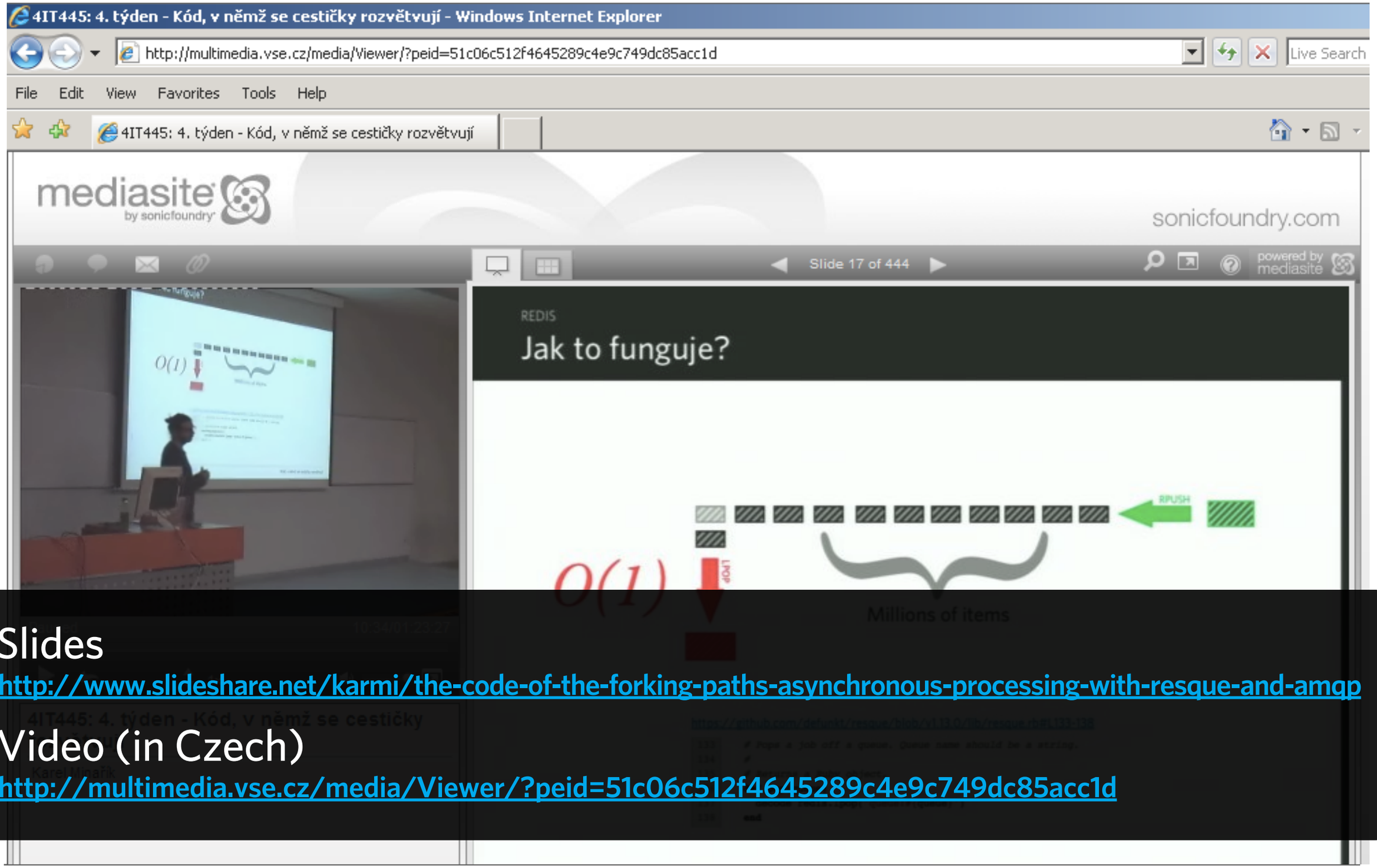


https://github.com/defunkt/resque/blob/v1.13.0/lib/resque.rb#L133-138

```
133     # Pops a job off a queue. Queue name should be a string.
134     #
135     # Returns a Ruby object.
136     def pop(queue)
137       decode redis.lpop("queue:#{queue}")
138     end
```

TALK ON ASYNCHRONOUS PROCESSING

The Code of the Forking Paths



Slides
http://www.slideshare.net/karmi/the-code-of-the-forking-paths-asynchronous-processing-with-resque-and-amqp

Video (in Czech)
http://multimedia.vse.cz/media/Viewer/?peid=51c06c512f4645289c4e9c749dc85acc1d

Redis

Sets

```
SADD key 1
=> 1
SADD key 2
=> 2
SADD key 3
=> 3
SMEMBERS key
=> "3"
=> "1"
=> "2"
```

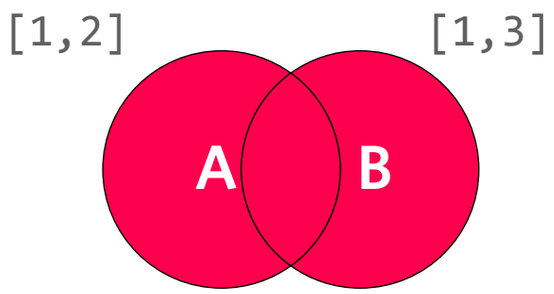
```
SISMEMBER key 1
=> "1"
SISMEMBER key 5
=> "0"
SRANDMEMBER key
=> "<RAND>"
SREM key 3
=> 1
```

Set Operations

```
SADD A 1
SADD A 2

SMEMBERS A
=> "1"
=> "2"
```

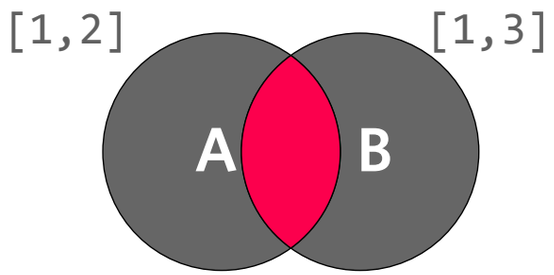
```
SADD B 1
SADD B 3

SMEMBERS B
=> "1"
=> "3"
```

Set Operations



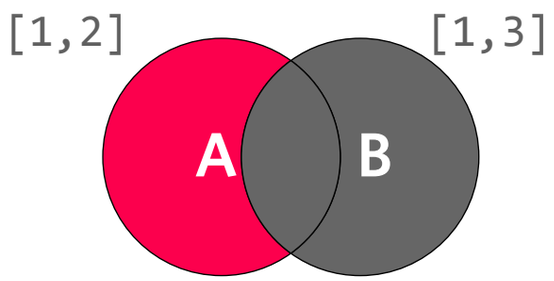
Union

```
SUNION A B
=> 1
=> 2
=> 3
```



Intersection

```
SINTER A B
=> 1
```



Difference

```
SDIFF A B
=> 2
```

http://en.wikipedia.org/wiki/Set_(mathematics)#Basic_operations

Ad serving

```
SADD ads:cars    "Check out Toyota!"
SADD ads:cars    "Check out Ford!"
...
SADD ads:movies "Check out Winter's Bone!"
...

SRANDMEMBER ads:cars
SRANDMEMBER ads:movies
```

Note: Time complexity is `O(1)`. “Check out ODER BY RAND()!”

Relations (Friends/followers)

```
SADD users:A:follows B

SADD users:B:follows C
SADD users:B:follows D

SADD users:C:follows A
SADD users:C:follows D
```

Relations (Friends/followers)

```
# Joint network of A and B

SUNION users:A:follows users:B:follows
1) "C"
2) "D"
3) "B"
```

Relations (Friends/followers)

```
# Common for A and B
SINTER users:A:follows users:B:follows
[]
# Common for B and C
SINTER users:B:follows users:C:follows
1) "D"
# Unique to B compared to C
SDIFF users:B:follows users:C:follows
1) "C"
```

Relations (Friends/followers)

```
# Whom I follow...
SADD friends A                  SMEMBERS friends
SADD friends B                  1) "A"
                                2) "B"

# Who follows me...
SADD followers B                SMEMBERS followers
SADD followers C                1) "C"
                                2) "B"
```

Relations (Friends/followers)

```
# Mutual relationships
SINTER friends followers
1) "B"
# Who does not follow me back?
SDIFF friends followers
1) "A"
# Who am I not following back?
SDIFF followers friends
1) "C"
```

Relations (Article tags/categories)

```
SADD tags:ruby article-1

SADD tags:java article-2

SADD tags:web  article-1
SADD tags:web  article-2
```

Relations (Article tags/categories)

```
# ruby OR java
SUNION tags:ruby tags:java
1) "article-2"
2) "article-1"
```

```
# ruby AND java
SINTER tags:ruby tags:java
[]
```

```
# web AND NOT ruby
SDIFF tags:web tags:ruby
1) "article-2"
```

Friends Online

```
# My friends
SADD friends A
SADD friends B
SADD friends C

# Friend A performs an action on the site...
SADD online:fresh A
SUNIONSTORE online online:fresh online:stale

# Who's online now?
SINTER friends online
[A]
```

Friends Online

```
# Every minute, rename the "fresh" to "stale" ...
RENAME online:fresh online:stale
# ... and update the "online" set
SUNIONSTORE online online:fresh online:stale

# Friend B performs an action on the site...
SADD online:fresh B
SUNIONSTORE online online:fresh online:stale

# Who's online now?
SINTER friends online
```

[A,B]

Friends Online

```
# Time passes ...

# Rename the "fresh" to "stale", every minute ...
RENAME online:fresh online:stale
# ... and update the "online" set
SUNIONSTORE online online:fresh online:stale

# Who's online now?
SINTER friends online
[B]
```

Sorted Sets

```
ZADD key 100 A
ZADD key 10  C
ZADD key 80  B
```

```
ZRANGE key 0 -1
1) "C"
2) "B"
3) "A"
```

```
ZREVRANGE key 0 -1
1) "A"
2) "B"
3) "C"
```

```
ZINCRBY key 10 C
"20"
```

Sorted Sets

```
ZREVRANGE key 0 -1
WITHSCORES
1) "A"
2) "100"
3) "B"
4) "80"
5) "C"
6) "20"
```

```
ZREVRANGEBYSCORE
key 100 50
1) "A"
2) "B"
```

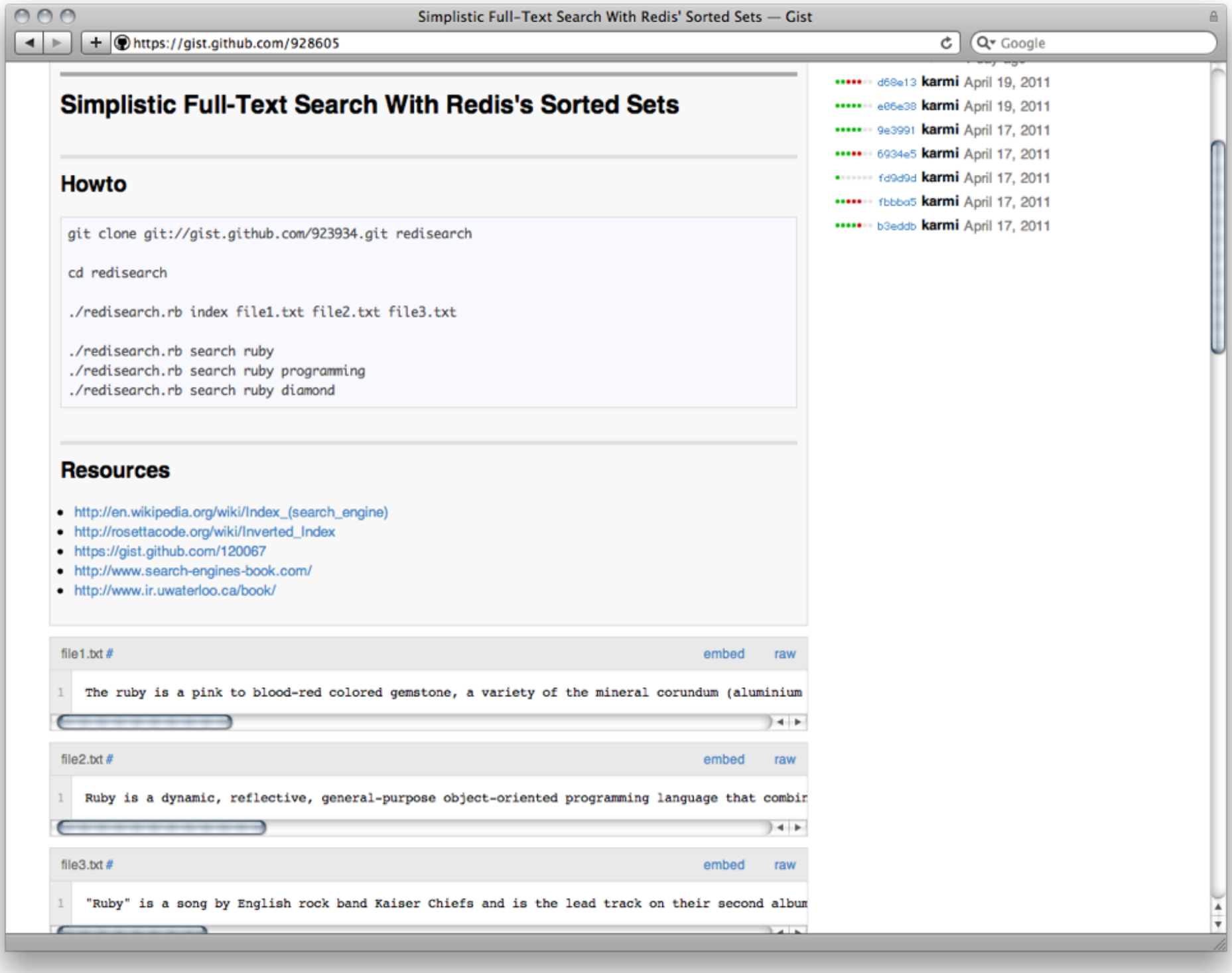
Leaderboards

```
# User A got 10 points
ZINCRBY scores 10 A
# User B got 15 points
ZINCRBY scores 15 B
# User A got another 10 points
ZINCRBY scores 10 A
# Display scores
ZREVRANGE scores 0 -1 WITHSCORES
1) "A"
2) "20"
3) "B"
4) "15"
```

Inverted Index

```
# Index document A
ZINCRBY index:foo 1 document-A
ZINCRBY index:foo 1 document-A
ZINCRBY index:bar 1 document-A

# Index document B
ZINCRBY index:foo 1 document-B
ZINCRBY index:baz 1 document-B
# Search for token foo, sort by occurences
ZREVRANGE index:foo 0 -1 WITHSCORES
1) "document-A"
2) "2"
3) "document-B"
4) "1"
```

Inverted Index



https://gist.github.com/928605

Usage

Counters (downloads, hits, votes, ...)

```
# Total downloads for server, all time
GET downloads:total

# Total downloads for server, today
GET downloads:total:today

# Total downloads for file
GET downloads:/downloads/file1.mpg:total

# Total downloads for file today
INCR downloads:/downloads/file1.mpg:today
...
```

Mm'kay. But I need the "leaderboard" of downloads!

Counters (downloads, hits, votes, ...)

```
INCR downloads:total
INCR downloads:total:today
...
```


```
# > Individual files
#   GET /downloads/file1.mpg
ZINCRBY downloads:files:total 1 /downloads/file1.mpg
ZINCRBY downloads:files:2011-05-10 1 /downloads/file1.mpg
#   GET /downloads/file2.mpg
ZINCRBY downloads:files:total 1 /downloads/file2.mpg
ZINCRBY downloads:files:2011-05-10 1 /downloads/file2.mpg
...
```

Usage

Counters (downloads, hits, votes, ...)

```
# 10 most downloaded, all the time
ZREVRANGE downloads:files:total 0 10 WITHSCORES

# 10 most downloaded on 2011-05-10
ZREVRANGE downloads:files:2011-05-10 0 10 WITHSCORES

# 10 most downloaded between 2011-05-10 and <OTHER DATE>
ZUNIONSTORE downloads:timeline 2 \
            downloads:files:2011-05-10 \
            <OTHER DATE>

ZREVRANGE downloads:timeline 0 10 WITHSCORES
```

Hashes

```
HMSET users:1 username j name John
HMSET users:2 username m name Mary

HGETALL users:1
1) "username"
2) "j"
...

HKEYS users:1
1) "username"
2) "name"

HSET users:1 score 100

HGET users:1 score
1) "100"
```

Structured data (Articles, users, ...)

```
HMSET articles:1 title     "Redis is cool!" \
                 content   "I recently ..." \
                 published "2011-05-10"

HGETALL articles:1
1) "title"
2) "Redis is cool!"
3) "content"
4) "I recently ..."
5) "published"
6) "2011-05-10"

HSET articles:1 title "Redis is very cool!"

HGET articles:1 title
"Redis is very cool!"
```

User Preferences (No login)

```
# Save preferences from <FORM>
HMSET prefs:<COOKIE HASH> background #ccc color #333

# Keep it for one year
EXPIRE prefs:<COOKIE HASH> 31556926

# Retrieve preferences
HGETALL prefs:<COOKIE HASH>
```

Publish/Subscribe

```
SUBSCRIBE   log.error
PUBLISH     log.error "ERROR"

PSUBSCRIBE log.*
PUBLISH     log.error "ERROR"
=> "ERROR"
=> "ERROR"

PUBLISH     log.info "INFO"
=> "INFO"
```

Scripting

```
SET myscript "return 'HELLO'"

EVAL "return loadstring(redis.call('get', KEYS[1]))()" 1 myscript
=> "HELLO"
```

http://redis.io/commands/EVAL

elasticsearch.

Search is the **primary interface** for getting information today.

Google

http://www.google.com/

Google

Web Images Videos Maps News Shopping Gmail more Sign in

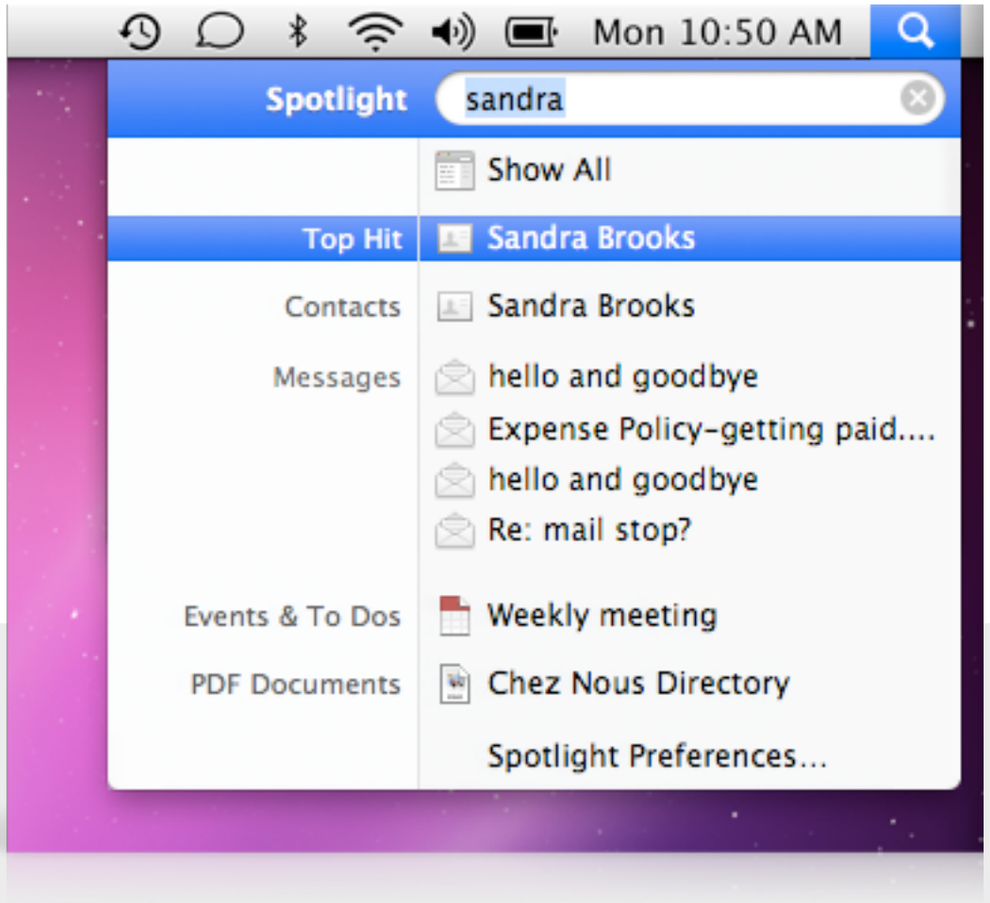
Google

Advanced search
Language tools

Google Search I'm Feeling Lucky

Advertising Programs Business Solutions About Google
Go to Google Česká republika

Change background image



http://www.apple.com/macosx/what-is-macosx/spotlight.html

http://www.apple.com/iphone/features/search.html

How does search work?

A collection of documents

file_1.txt

The ruby is a pink to blood-red colored gemstone ...

file_2.txt

Ruby is a dynamic, reflective, general-purpose object-oriented programming language ...

file_3.txt

"Ruby" is a song by English rock band Kaiser Chiefs ...

How do you search documents?

```
File.read('file_1.txt').include?('ruby')
File.read('file_2.txt').include?('ruby')
...
```

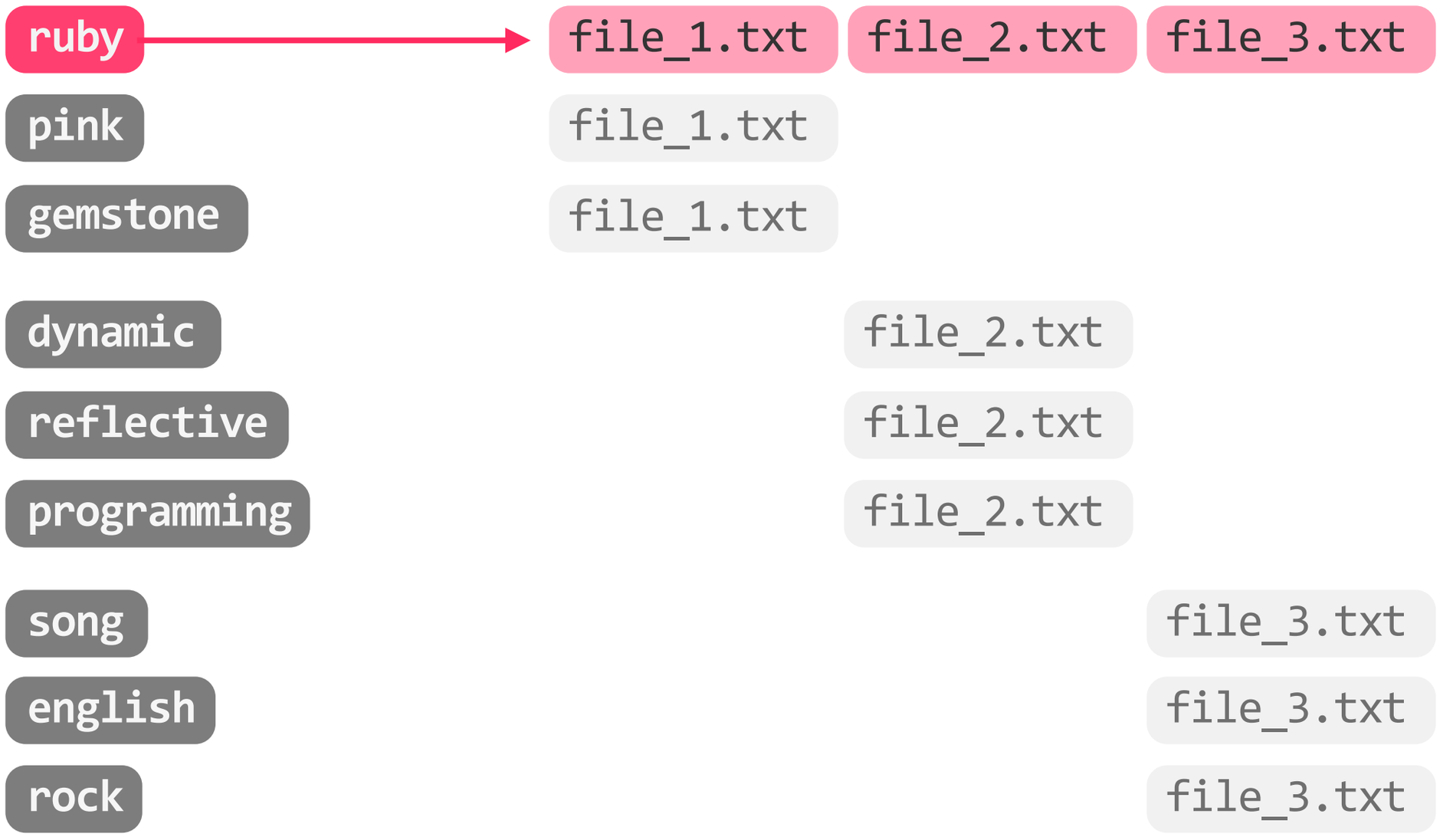
The inverted index

TOKENS	POSTINGS
ruby	file_1.txt, file_2.txt, file_3.txt
pink	file_1.txt
gemstone	file_1.txt
dynamic	file_2.txt
reflective	file_2.txt
programming	file_2.txt
song	file_3.txt
english	file_3.txt
rock	file_3.txt

http://en.wikipedia.org/wiki/Index_(search_engine)#Inverted_indices

The inverted index

```
MySearchLib.search "ruby"
```



http://en.wikipedia.org/wiki/Index_(search_engine)#Inverted_indices

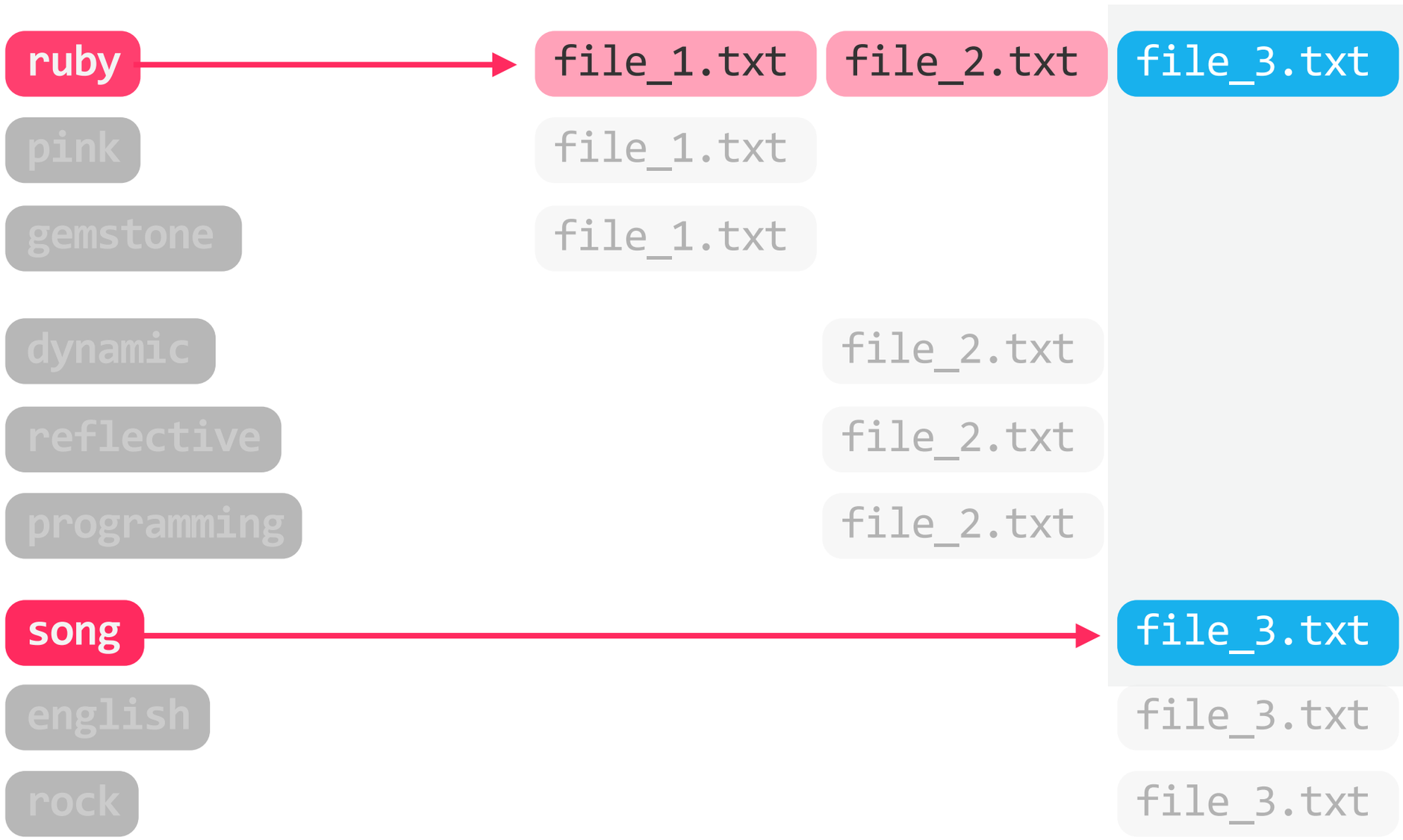
The inverted index

```
MySearchLib.search "song"
```

http://en.wikipedia.org/wiki/Index_(search_engine)#Inverted_indices

The inverted index

```
MySearchLib.search "ruby AND song"
```



http://en.wikipedia.org/wiki/Index_(search_engine)#Inverted_indices

A *naïve* Ruby implementation

```ruby
module SimpleSearch

  def index document, content
    tokens = analyze content
    store document, tokens
    puts "Indexed document #{document} with tokens:", tokens.inspect, "\n"
  end

  def analyze content
    # >>> Split content by words into "tokens"
    content.split(/\W/).
    # >>> Downcase every word
    map    { |word| word.downcase }.
    # >>> Reject stop words, digits and whitespace
    reject { |word| STOPWORDS.include?(word) || word =~ /^\d+/ || word == ''  }
  end

  def store document_id, tokens
    tokens.each do |token|
      # >>> Save the "posting"
      ( (INDEX[token] ||= []) << document_id ).uniq!
    end
  end

  def search token
    puts "Results for token '#{token}':"
    # >>> Print documents stored in index for this token
    INDEX[token].each { |document| "  * #{document}" }
  end

  INDEX = {}
  STOPWORDS = %w|a an and are as at but by for if in is it no not of on or that the then there

  extend self

end
```

Indexing documents

```
SimpleSearch.index "file1", "Ruby is a language. Java is also a language.
SimpleSearch.index "file2", "Ruby is a song."
SimpleSearch.index "file3", "Ruby is a stone."
SimpleSearch.index "file4", "Java is a language."
```

```
Indexed document file1 with tokens:
["ruby", "language", "java", "also", "language"]

Indexed document file2 with tokens:
["ruby", "song"]

Indexed document file3 with tokens:
["ruby", "stone"]

Indexed document file4 with tokens:
["java", "language"]
```

Words downcased, stopwords removed.

The index

```
puts "What's in our index?"
p SimpleSearch::INDEX

{
  "ruby"     => ["file1", "file2", "file3"],
  "language" => ["file1", "file4"],
  "java"     => ["file1", "file4"],
  "also"     => ["file1"],
  "stone"    => ["file3"],
  "song"     => ["file2"]
}
```

Search the index

```
SimpleSearch.search "ruby"

Results for token 'ruby':
* file1
* file2
* file3
```

The inverted index

TOKENS	POSTINGS
ruby 3	file_1.txt file_2.txt file_3.txt
pink 1	file_1.txt
gemstone	file_1.txt
dynamic	file_2.txt
reflective	file_2.txt
programming	file_2.txt
song	file_3.txt
english	file_3.txt
rock	file_3.txt

http://en.wikipedia.org/wiki/Index_(search_engine)#Inverted_indices

It is very **practical** to know how search works.

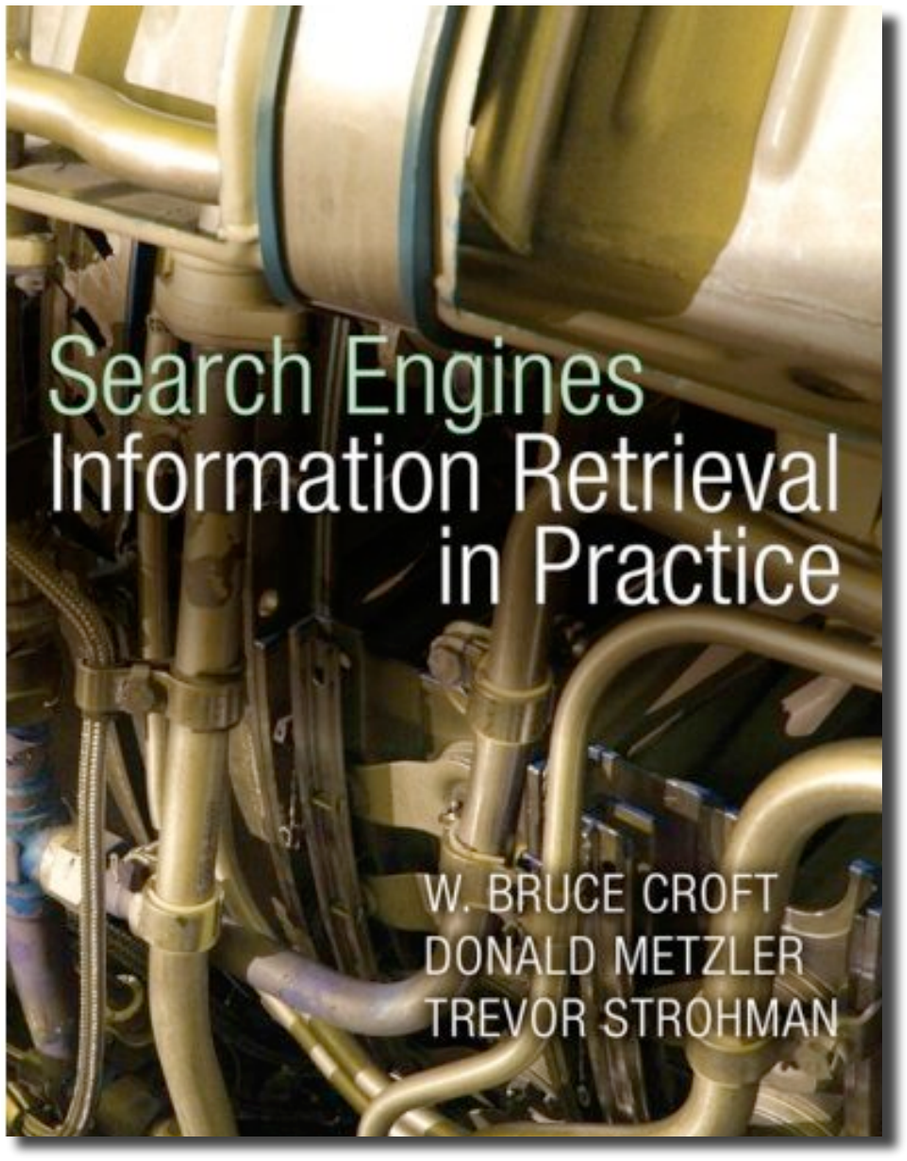
For instance, now you know that
the **analysis** step **is very important.**

It's **more** important than the **“search”** step.

```ruby
module SimpleSearch

  def index document, content
    tokens = analyze content
    store document, tokens
    puts "Indexed document #{document} with tokens:", tokens.inspect, "\n"
  end

  def analyze content
    # >>> Split content by words into "tokens"
    content.split(/\W/).
    # >>> Downcase every word
    map    { |word| word.downcase }.
    # >>> Reject stop words, digits and whitespace
    reject { |word| STOPWORDS.include?(word) || word =~ /^\d+/ || word == ''  }
  end

  def store document_id, tokens
    tokens.each do |token|
      # >>> Save the "posting"
      ( (INDEX[token] ||= []) << document_id ).uniq!
    end
  end

  def search token
    puts "Results for token '#{token}':"
    # >>> Print documents stored in index for this token
    INDEX[token].each { |document| "  * #{document}" }
  end

  INDEX = {}
  STOPWORDS = %w|a an and are as at but by for if in is it no not of on or that the then there

  extend self

end
```

A *naïve* Ruby implementation

The Search Engine Textbook



Search Engines
Information Retrieval in Practice

Bruce Croft, Donald Metzler and Trevor Strohma

Addison Wesley, 2009

http://search-engines-book.com

```
$ curl -X GET "http://localhost:9200/_search?q=<YOUR QUERY>"
```

Terms	`apple` `apple iphone`
Phrases	`"apple iphone"`
Proximity	`"apple safari"~5`
Fuzzy	`apple~0.8`
Wildcards	`app*` `*pp*`
Boosting	`apple^10 safari`
Range	`[2011/05/01 TO 2011/05/31]` `[java TO json]`

Boolean	`apple AND NOT iphone`
	`+apple -iphone`
	`(apple OR iphone) AND NOT review`

Fields	`title:iphone^15 OR body:iphone` `published_on:[2011/05/01 TO "2011/05/27 10:00:00"]`

http://lucene.apache.org/java/3_1_0/queryparsersyntax.html

JSON-based Query DSL

```
curl -X GET localhost:9200/articles/_search -d '{
  "query" : {
    "filtered" : {
      "query" : {
        "bool" : {

          "must" : {
            "match" : {
              "author.first_name" : {
                "query" : "claire",
                "fuzziness"  : 0.1
              }
            }
          },

          "must" : {
            "multi_match" : {
              "query"  : "elasticsearch",
              "fields" : ["title^10", "body"]
            }
          }
        }

      },

      "filter": {
        "and" : [
          { "terms" : { "tags" : ["search"] } },
          { "range" : { "published_on": {"from": "2013"} } },
          { "term"  : { "featured" : true } }
        ]
      }
    }
  }
}'
```

JSON-based Query DSL

```
curl -X GET loca
  "query
    "filtered"
      "query" :
        "bool"

          "must
            "mat
              "author.first_name" : {
                "query" : "claire",
                "fuzziness"  : 0.1
              }
            }
          },

          "must" : {
            "multi_match" : {
              "query"  : "elasticsearch",
              "fields" : ["title^10", "body"]
            }
          }
        }

      },

      "filter": {
        "and" : [
          { "terms" : { "tags" : ["search"] } },
          { "range" : { "published_on": {"from": "2013"} } },
          { "term"  : { "featured" : true } }
        ]
      }
    }
  }
}'
```

```
{
  "query" : {
    "filtered" : {
      "query" : {
        "bool" : {
```

JSON-based Query DSL

```
curl -X GET localhost:9200/articles/_search -d '{
  "query" : {
    "filtered" : {
      "query" : {
        "bool" : {

          "must" : {
            "match" : {
              "author.first
                "query" : "
                "fuzziness"
              }
            }
          },

          "must" : {
            "multi_match" :
              "query"  : "e
              "fields" : ["ti
            }
          }
        }

      },

      "filter": {
        "and" : [
          { "terms" : { "tags" : ["search"] } },
          { "range" : { "published_on": {"from": "2013"} } },
          { "term"  : { "featured" : true } }
        ]
      }
    }
  }
}'
```

```
"must" : {
  "match" : {
    "author.first_name" : {
      "query" : "claire",
      "fuzziness"  : 0.1
    }
  },
```

JSON-based Query DSL

```
curl -X GET localhost:9200/articles/_search -d '{
  "query" : {
    "filtered" : {
      "query" : {
        "bool" : {

          "must" : {
            "match" : {
              "author.first_name" : {
                "query" : "claire",
                "fuzziness"  : 0.1
              }
            }
          },
          "must" : {
            "multi_match" :
              "query"  : "e
              "fields" : ['
            }
          }
        }

      },

      "filter": {
        "and" : [
          { "terms" : { "tag
          { "range" : { "published_on" : { "from" : "2013" } } },
          { "term"  : { "featured" : true } }
        ]
      }
    }
  }
}'
```

```
    },
    "must" : {
      "multi_match" : {
        "query"  : "elasticsearch",
        "fields" : ["title^10", "body"]
      }
    }
  }
```

JSON-based Query DSL

```
curl -X GET localhost:9200/articles/_search -d '{
  "query" : {
    "filtered" : {
      "query" : {
        "bool" : {

          "must" : {
            "match" : {
              "author.first_name" : {
                "query" : "claire",
                "fuzziness"  : 0.1
              }
            }
          },

          "must" : {
            "multi_match" : {
              "query"  : "elasticsearch",
              "fields" : ["title^10", "body"]
            }
          }
        }

      },

      "filter": {
        "and" : [
          { "terms" : { "
          { "range" : { "
          { "term"  : { "
        ]
      }
    }
  }
}'
```

```
    },

    "filter": {
      "and" : [
        { "terms" : { "tags" : ["search"] } },
        { "range" : { "published_on": {"from": "2013"} } },
        { "term"  : { "featured" : true } }
      ]
    }
  }
```

Full-text Search

"Find all articles with 'search' in their title or body, give matches in titles higher score"

Structured Search

"Find all articles from year 2013 tagged 'search'"

Custom Scoring

See `custom_score` and `custom_filters_score` queries

The _analyze API

`_analyze?text=...&tokenizer=X&filters=A,B,C`

```
[žluťoučký:0->9:<ALPHANUM>]\n
\n2: \n[kůň:10-
>13:<ALPHANUM>]\n\n3:
\n[skákal:14->20:<ALPHANUM>]
\n\n4: \n[přes:21-
>25:<ALPHANUM>]\n\n5:
\n[potok:26->31:<ALPHANUM>]
```

`_analyze?pretty&format=text&text=Žluťoučký+kůň+skákal+přes+potok`

`_analyze?pretty&format=text&text=Žluťoučký+kůň+skákal+přes+potok&analyzer=czech`

```
[žluťoučk:0->9:<ALPHANUM>]\n
\n2: \n[koň:10-
>13:<ALPHANUM>]\n\n3:
\n[skákal:14->20:<ALPHANUM>]
\n\n5: \n[potok:26-
>31:<ALPHANUM>]\n
```

Query

Facets

Account Type: Basic

Welcome, Esteban Kozak · **Add Connections** · Settings · Help · Sign Out

LinkedIn® Home Profile Contacts Groups Jobs Inbox (9) More... Search People linkedin

Search

linkedin

Keep refine selections

Search Show more...

Refine By BETA

Location

- [x] **All Locations**
- [ ] United States (20762)
- [ ] San Francisco Bay Area (2444)
- [ ] India (2303)
- [ ] United Kingdom (2101)
- [ ] Greater New York City Area (2079)

Show more...

Enter location name

Relationship

- [x] **All LinkedIn Members**
- [ ] 1st Connections (192)
- [ ] 2nd Connections (2706)
- [ ] Group Members (860)
- [ ] 3rd + Everyone Else (30390)

Industry

- [x] **All Industries**
- [ ] Information Technology and Services (4219)
- [ ] Marketing and Advertising (2709)
- [ ] Staffing and Recruiting (2301)
- [ ] Computer Software (1947)
- [ ] Internet (1782)

Show more...

Enter industry name

Current Company

Past Company

School

Groups

Sort by: **Relationship** View: **Basic**

33,778 results for **linkedin** [Save this search]

Reid Hoffman 1st
Partner at Greylock
San Francisco Bay Area | Internet
In Common: 120 shared connections · 2 shared groups

David Hahn 1st
Director of Product Management, LinkedIn
San Francisco Bay Area | Internet
In Common: 83 shared connections

Candy Chastain Mielke 1st
Consultant at LinkedIn
Colorado Springs, Colorado Area | Internet
In Common: 128 shared connections · 2 shared groups

Wade Burgess 1st
Business Leader
Greater Omaha Area | Internet
In Common: 44 shared connections · 2 shared groups

Alex Vauthey 1st
Director of Engineering, Monetization at LinkedIn
San Francisco Bay Area | Information Technology and Services
In Common: 103 shared connections

Ruslan Belkin 1st
Sr. Director of Engineering at LinkedIn
San Francisco Bay Area | Computer Software
In Common: 95 shared connections

Elizabeth Reaves 1st
Product Manager at LinkedIn
San Francisco Bay Area | Internet
In Common: 92 shared connections · 4 shared groups

Riccardo Ferretti 1st
Principal Engineer at LinkedIn
San Francisco Bay Area | Computer Software
In Common: 121 shared connections

Shirley Xu 1st
Data Scientist at LinkedIn

Search tools

Advanced search

Saved Searches

Ads by LinkedIn Members

S&OP Summit

CE group

January 20/21 2010, The Palms Hotel Sales & Operations Planning Innovat

events.linkedin.com

From: Richard Bracchi

Channel Not Performing?

relayware

On demand solutions for channel performance optimization.

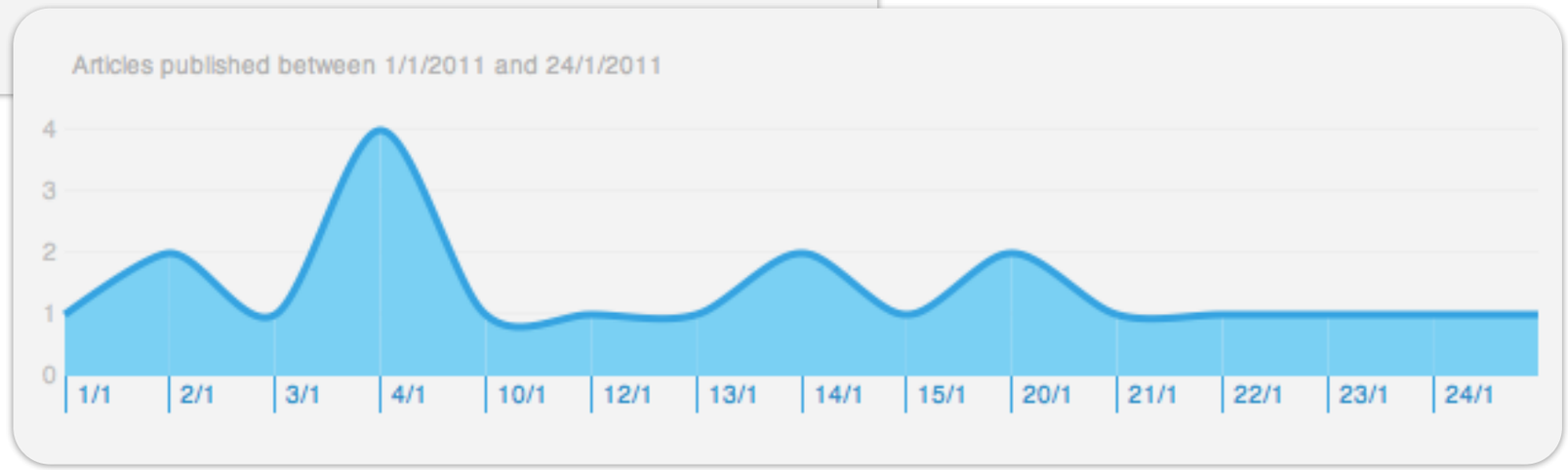
www.relayware.com

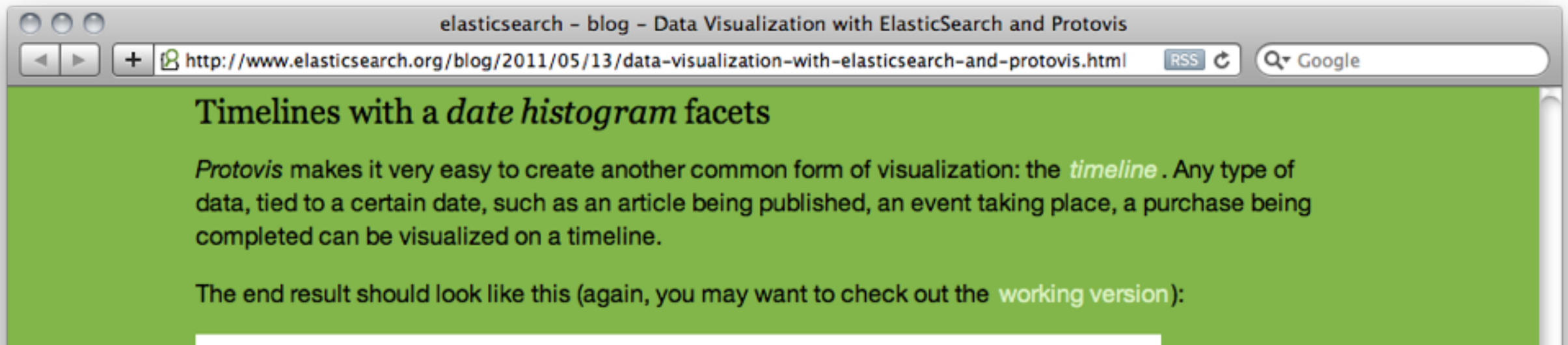
From: Mike Morgan

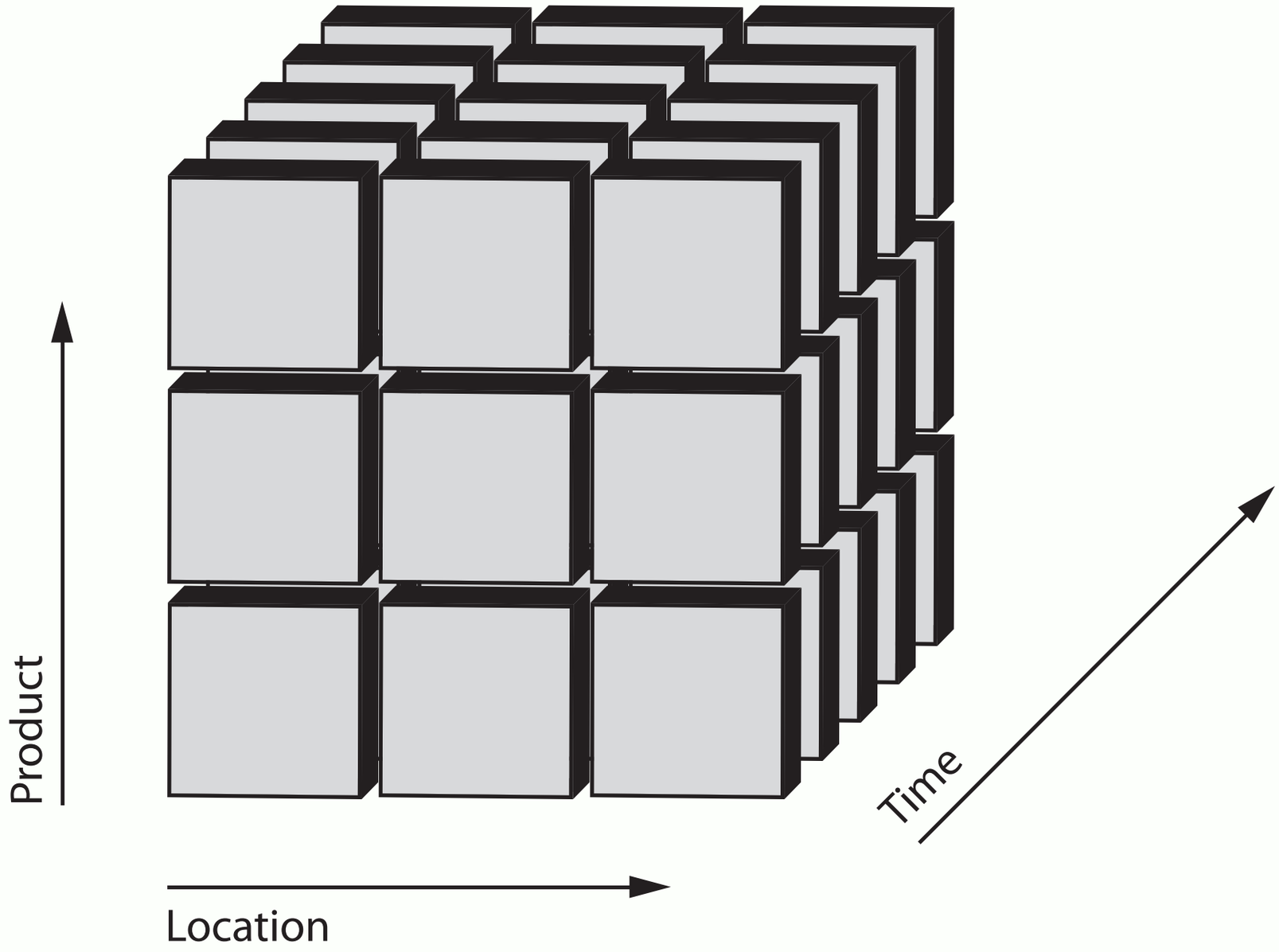
What's this?

Using facets as analytics

```
curl -X POST "http://localhost:9200/articles/_search?pretty=true" -d '
{
  "facets" : {
    "published_on" : {
      "date_histogram" : {
        "field"    : "published",
        "interval" : "day"
      }
    }
  }
}'
```

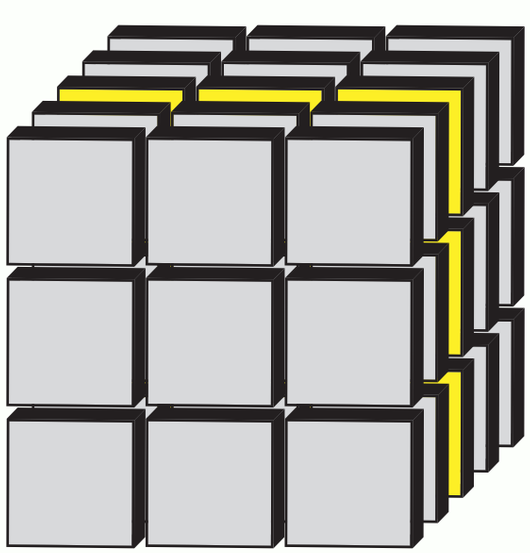






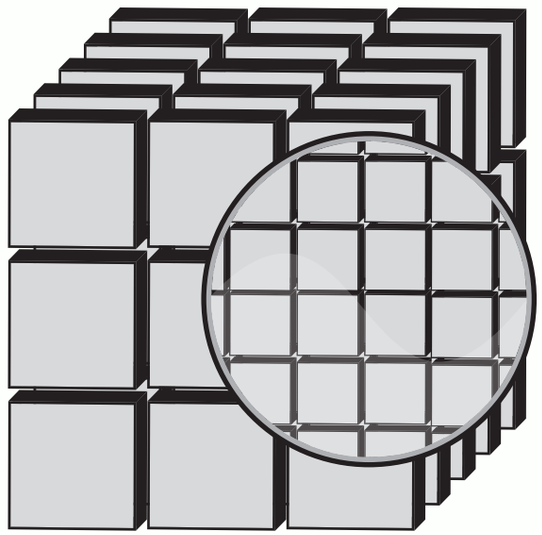
OLAP Cube

Dimensions, measures, aggregations



Slice

Dice



Drill Down / Roll Up

Show me sales numbers for all products across all locations in year 2013

Show me product A sales numbers across all locations over all years

Show me products sales numbers in location X over all years

Category Count With the `terms` Aggregation

Simplest "map/reduce" / `GROUP_BY` aggregation: document count per tag

```
curl -X POST 'localhost:9200/articles/_search?search_type=count&pretty' -d '{
  "aggregations": {
    "categories": {
      "terms" : {
        "field" : "category"
      }
    }
  }
}'
```

```
"facets" : {
    "tag-cloug" : {
      "buckets" : [ {
        "key" : "ruby",
        "doc_count" : 3
      }, {
        "key" : "java",
        "doc_count" : 2
      },
      ...
      } ]
    }
  }
```

Inner Aggregation within Category

Aggregating number of clicks per category

```
curl -X GET 'localhost:9200/articles/_search/?search_type=count&pretty' -d
  "aggregations": {
    "categories": {
      "terms" : {
        "field" : "category"
      },
      "aggregations" : {
        "clicks" : {
          "sum" : { "field" : "clicks" }
        }
      }
    }
  }
}'
```

```
"aggregations" : {
  "categories" : {
    "buckets" : [ {
      "key" : "ruby",
      "doc_count" : 3,
      "clicks" : {
        "value" : 420
      }
    },
    ... ]
  }
}
```

Limit the Aggregation Scope With a Fulltext Query

Realtime filtering with queries and filters

```
curl -X GET 'localhost:9200/articles/_search/?search_type=count&pretty' -d
  "query" : {
    "match" : {
      "body" : "Twitter"
    }
  },
  "aggregations": {
    ...
  }
}'
```

Aggregations

Terms

Significant Terms

Stats

Percentiles

Cardinality

Top Hits

Scripted Metric

Filter

Range

Histogram

http://www.elasticsearch.org/guide/en/elasticsearch/guide/current/aggregations.html

Thanks!