CS511
Advanced
Database Systems

Special Topic

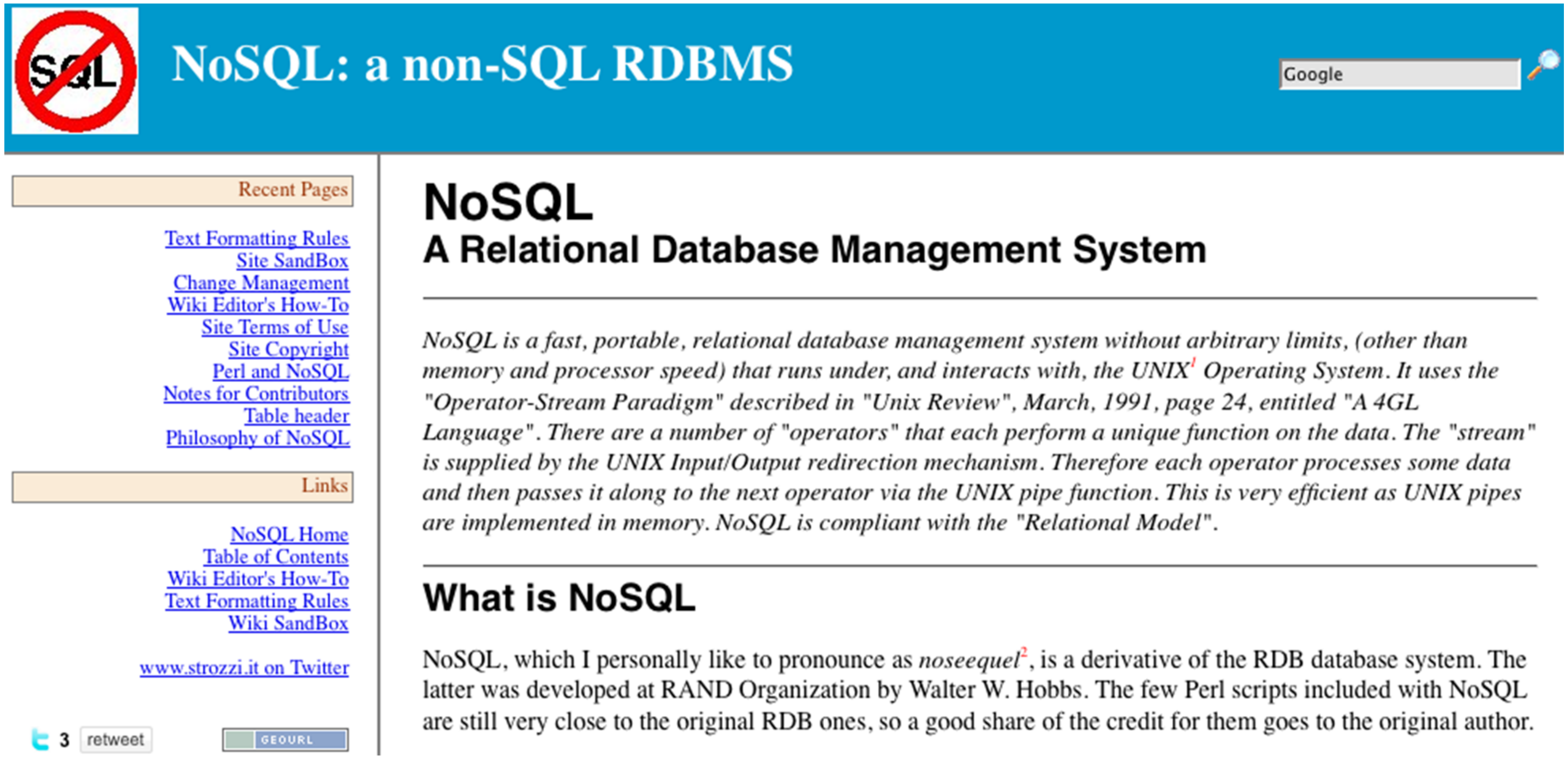
The Trends of NoSQL DBMS: What and Why.

Kevin C. Chang

What is NoSQL?

Not Only SQL

What is NoSQL:
A real system since 1998 by Carlo Strozzi



NoSQL: a non-SQL RDBMS

Google

Recent Pages

Text Formatting Rules
Site SandBox
Change Management
Wiki Editor's How-To
Site Terms of Use
Site Copyright
Perl and NoSQL
Notes for Contributors
Table header
Philosophy of NoSQL

Links

NoSQL Home
Table of Contents
Wiki Editor's How-To
Text Formatting Rules
Wiki SandBox

www.strozzi.it on Twitter

3 retweet

GEOURL

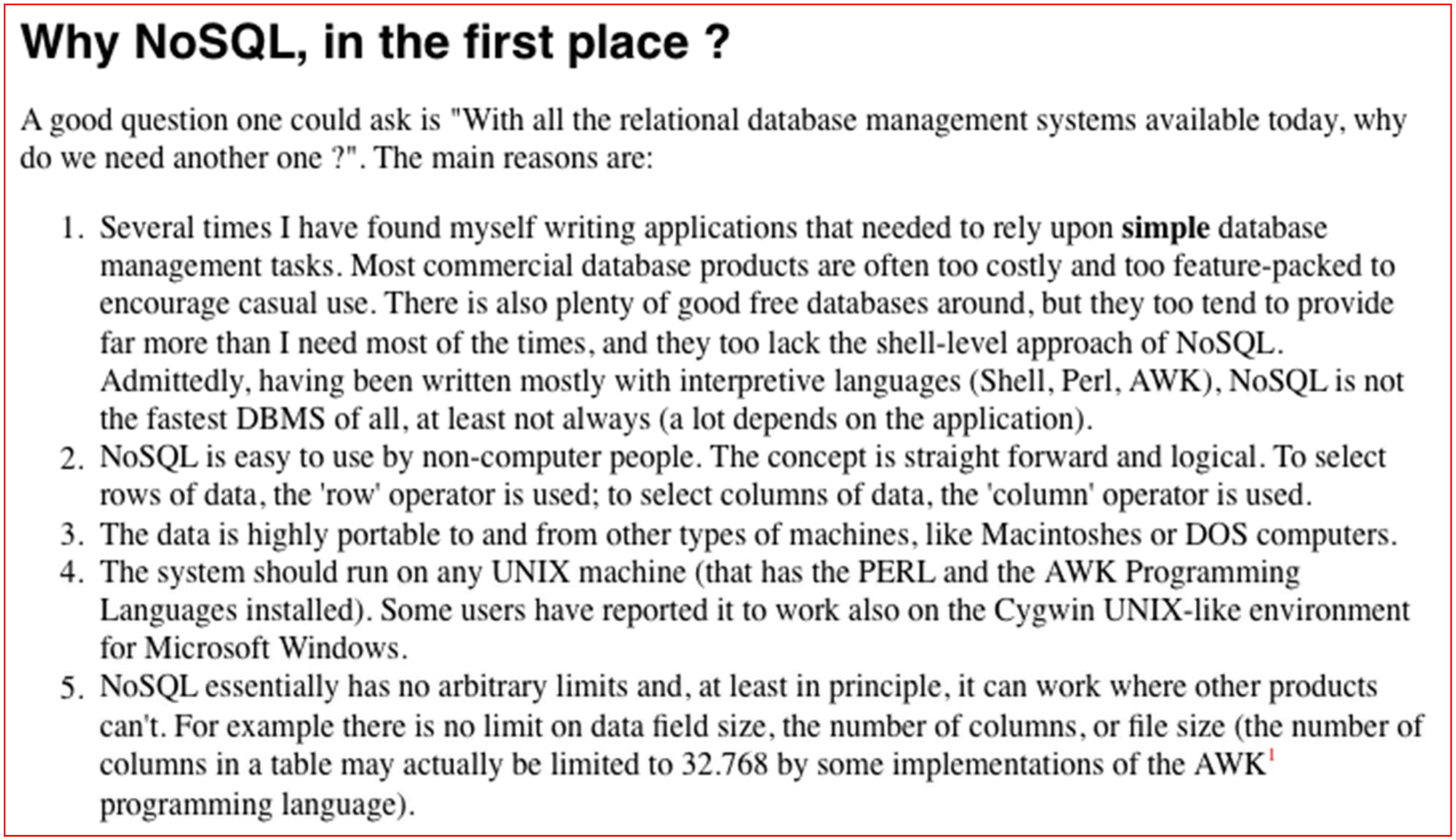
NoSQL
A Relational Database Management System

NoSQL is a fast, portable, relational database management system without arbitrary limits, (other than memory and processor speed) that runs under, and interacts with, the UNIX[1] Operating System. It uses the "Operator-Stream Paradigm" described in "Unix Review", March, 1991, page 24, entitled "A 4GL Language". There are a number of "operators" that each perform a unique function on the data. The "stream" is supplied by the UNIX Input/Output redirection mechanism. Therefore each operator processes some data and then passes it along to the next operator via the UNIX pipe function. This is very efficient as UNIX pipes are implemented in memory. NoSQL is compliant with the "Relational Model".

What is NoSQL

NoSQL, which I personally like to pronounce as *noseequel*[2], is a derivative of the RDB database system. The latter was developed at RAND Organization by Walter W. Hobbs. The few Perl scripts included with NoSQL are still very close to the original RDB ones, so a good share of the credit for them goes to the original author.

The Original NoSQL: Structured Storage + Shell Operation

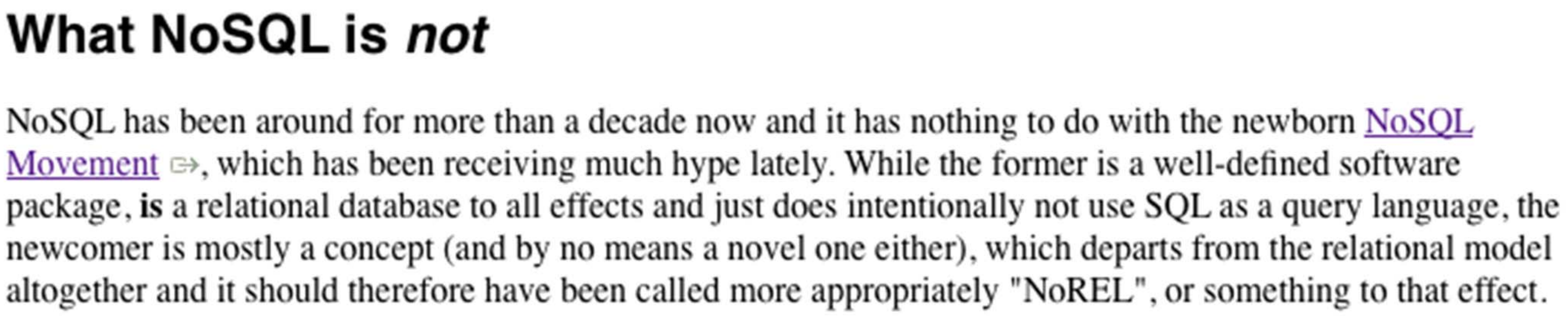


Why NoSQL, in the first place ?

A good question one could ask is "With all the relational database management systems available today, why do we need another one ?". The main reasons are:

1. Several times I have found myself writing applications that needed to rely upon **simple** database management tasks. Most commercial database products are often too costly and too feature-packed to encourage casual use. There is also plenty of good free databases around, but they too tend to provide far more than I need most of the times, and they too lack the shell-level approach of NoSQL. Admittedly, having been written mostly with interpretive languages (Shell, Perl, AWK), NoSQL is not the fastest DBMS of all, at least not always (a lot depends on the application).
2. NoSQL is easy to use by non-computer people. The concept is straight forward and logical. To select rows of data, the 'row' operator is used; to select columns of data, the 'column' operator is used.
3. The data is highly portable to and from other types of machines, like Macintoshes or DOS computers.
4. The system should run on any UNIX machine (that has the PERL and the AWK Programming Languages installed). Some users have reported it to work also on the Cygwin UNIX-like environment for Microsoft Windows.
5. NoSQL essentially has no arbitrary limits and, at least in principle, it can work where other products can't. For example there is no limit on data field size, the number of columns, or file size (the number of columns in a table may actually be limited to 32.768 by some implementations of the AWK[1] programming language).

Why NoSQL (orig) is Not…



What NoSQL is *not*

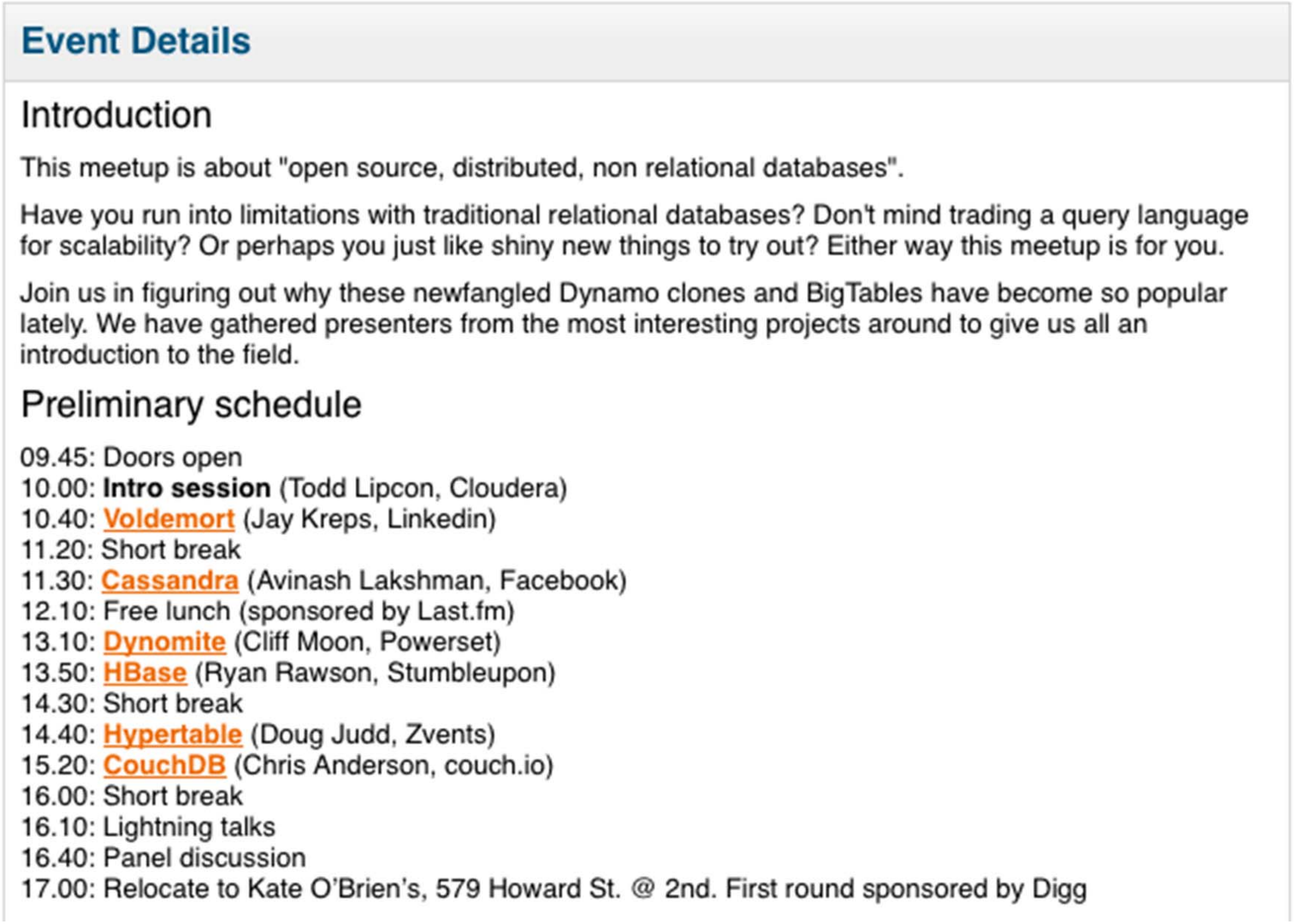
NoSQL has been around for more than a decade now and it has nothing to do with the newborn NoSQL Movement ⇨, which has been receiving much hype lately. While the former is a well-defined software package, **is** a relational database to all effects and just does intentionally not use SQL as a query language, the newcomer is mostly a concept (and by no means a novel one either), which departs from the relational model altogether and it should therefore have been called more appropriately "NoREL", or something to that effect.

What’s After NoSQL? The Newborn NoSQL Movement…

- NoSQL 2009 Conference.
  - An event to discuss open-source distributed databases.
  - It was said that Eric Evans, a Rackspace employee, came up the name for the conference.
- "a conference of non-relational data stores“
- its motto:

  select fun, profit **from** real_world **where** relational = false
- The most common interpretation of "NoSQL" is, thus, "non-relational".

The NoSQL 2009 Conference



Event Details

Introduction

This meetup is about "open source, distributed, non relational databases".

Have you run into limitations with traditional relational databases? Don't mind trading a query language for scalability? Or perhaps you just like shiny new things to try out? Either way this meetup is for you.

Join us in figuring out why these newfangled Dynamo clones and BigTables have become so popular lately. We have gathered presenters from the most interesting projects around to give us all an introduction to the field.

Preliminary schedule

09.45: Doors open
10.00: **Intro session** (Todd Lipcon, Cloudera)
10.40: Voldemort (Jay Kreps, Linkedin)
11.20: Short break
11.30: Cassandra (Avinash Lakshman, Facebook)
12.10: Free lunch (sponsored by Last.fm)
13.10: Dynomite (Cliff Moon, Powerset)
13.50: HBase (Ryan Rawson, Stumbleupon)
14.30: Short break
14.40: Hypertable (Doug Judd, Zvents)
15.20: CouchDB (Chris Anderson, couch.io)
16.00: Short break
16.10: Lightning talks
16.40: Panel discussion
17.00: Relocate to Kate O'Brien's, 579 Howard St. @ 2nd. First round sponsored by Digg

So, Why?

- Linkedin, Facebook, Powerset, Stumbleupon, Zvents, …

- What do you think are obstacles?

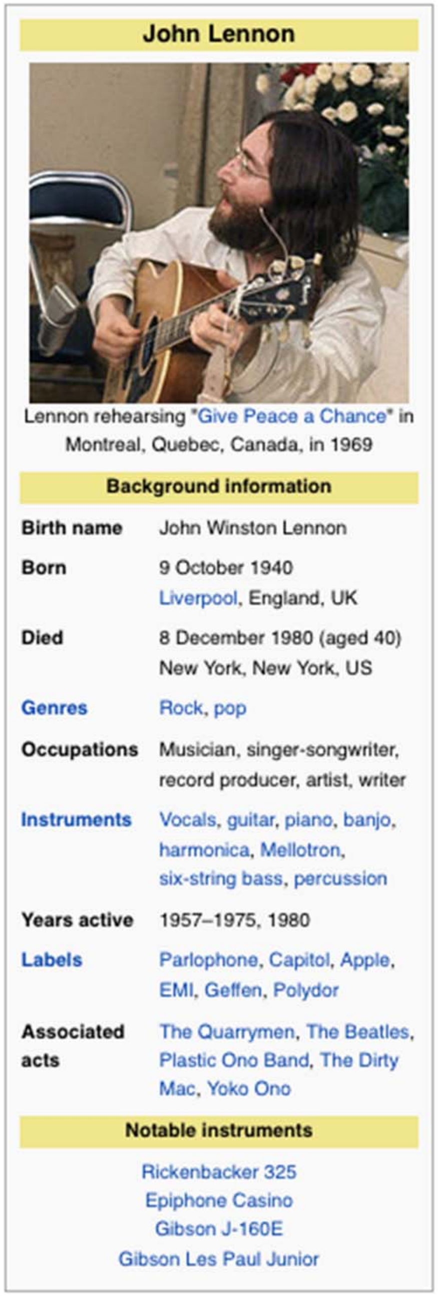
The Case:
Querying Wikipedia.

Querying Wikipedia: Issue 1 – Large Scale

- Large data:
  - As of November 25, 2010, there are 3,485,971 English articles.

- Many users: http://stats.grok.se/

Issue 2 – Flexible Attributes



John Lennon

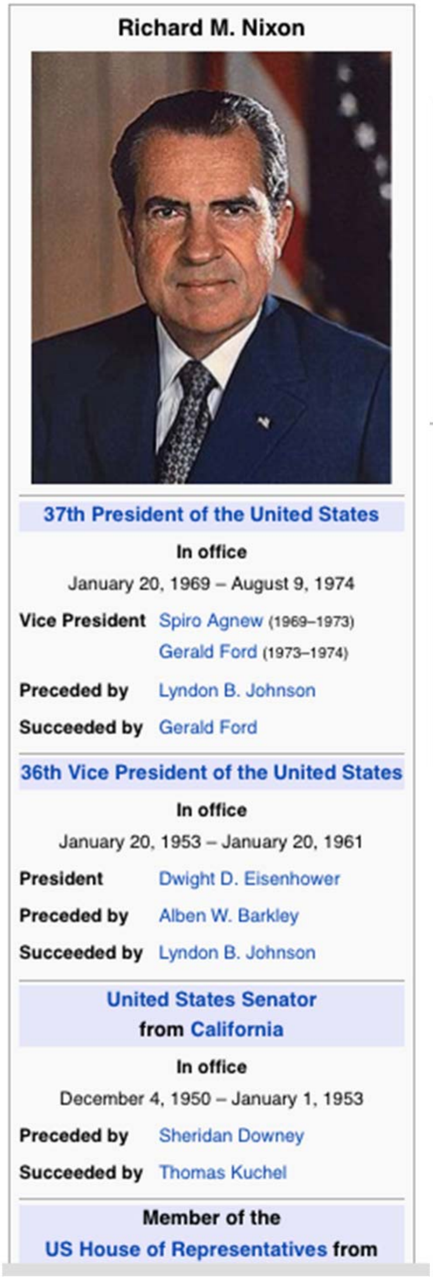
Lennon rehearsing "Give Peace a Chance" in Montreal, Quebec, Canada, in 1969

Background information

Birth name	John Winston Lennon
Born	9 October 1940 Liverpool, England, UK
Died	8 December 1980 (aged 40) New York, New York, US
Genres	Rock, pop
Occupations	Musician, singer-songwriter, record producer, artist, writer
Instruments	Vocals, guitar, piano, banjo, harmonica, Mellotron, six-string bass, percussion
Years active	1957–1975, 1980
Labels	Parlophone, Capitol, Apple, EMI, Geffen, Polydor
Associated acts	The Quarrymen, The Beatles, Plastic Ono Band, The Dirty Mac, Yoko Ono

Notable instruments

Rickenbacker 325
Epiphone Casino
Gibson J-160E
Gibson Les Paul Junior



Richard M. Nixon

37th President of the United States

In office
January 20, 1969 – August 9, 1974

Vice President	Spiro Agnew (1969–1973) Gerald Ford (1973–1974)
Preceded by	Lyndon B. Johnson
Succeeded by	Gerald Ford

36th Vice President of the United States

In office
January 20, 1953 – January 20, 1961

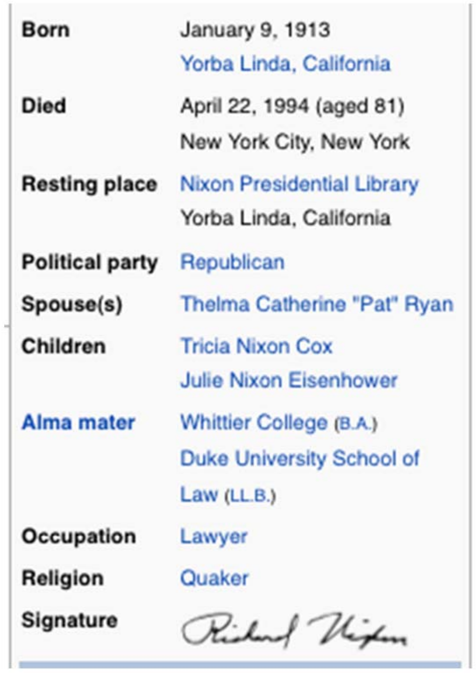
President	Dwight D. Eisenhower
Preceded by	Alben W. Barkley
Succeeded by	Lyndon B. Johnson

United States Senator from California

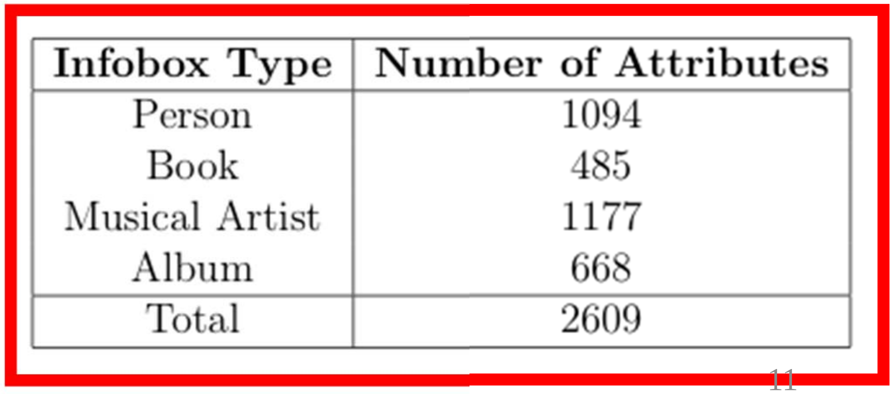
In office
December 4, 1950 – January 1, 1953

Preceded by	Sheridan Downey
Succeeded by	Thomas Kuchel

Member of the US House of Representatives from



Born	January 9, 1913 Yorba Linda, California
Died	April 22, 1994 (aged 81) New York City, New York
Resting place	Nixon Presidential Library Yorba Linda, California
Political party	Republican
Spouse(s)	Thelma Catherine "Pat" Ryan
Children	Tricia Nixon Cox Julie Nixon Eisenhower
Alma mater	Whittier College (B.A.) Duke University School of Law (LL.B.)
Occupation	Lawyer
Religion	Quaker
Signature	[illegible]

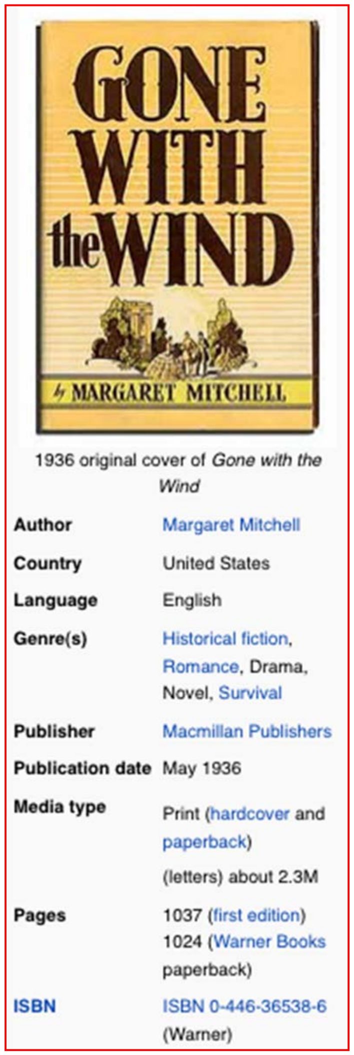


Infobox Type	Number of Attributes
Person	1094
Book	485
Musical Artist	1177
Album	668
Total	2609

Issue 3: Flexible Values

Inconsistent attribute types within an attribute.

- Consider Pages (page count) of books.



1936 original cover of *Gone with the Wind*

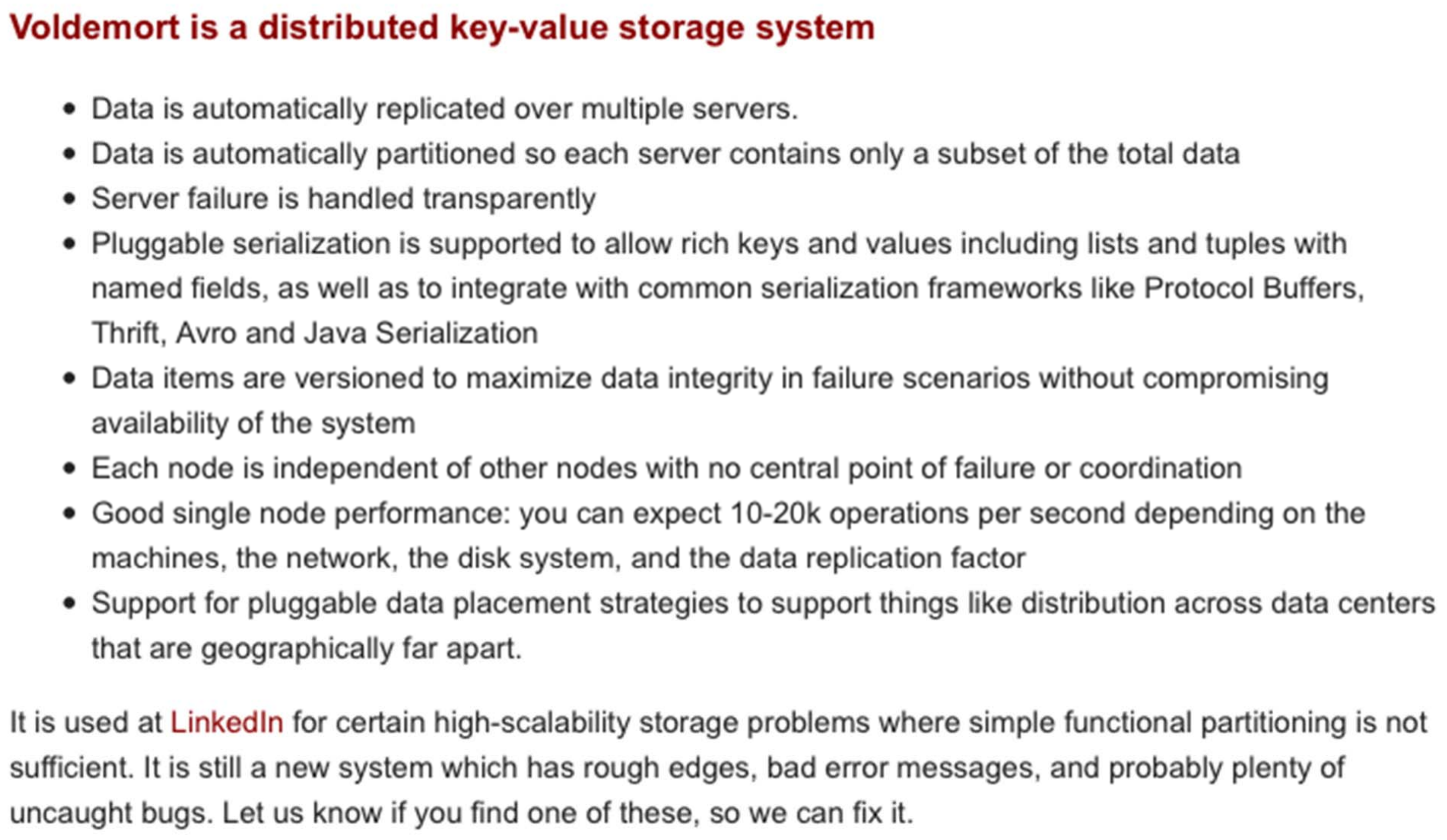
Author	Margaret Mitchell
Country	United States
Language	English
Genre(s)	Historical fiction, Romance, Drama, Novel, Survival
Publisher	Macmillan Publishers
Publication date	May 1936
Media type	Print (hardcover and paperback)
	(letters) about 2.3M
Pages	1037 (first edition) 1024 (Warner Books paperback)
ISBN	ISBN 0-446-36538-6 (Warner)

Out of 14301 values, only about 33% were in the expected digit-only for-mat. 52% of values had the number followed by pp, e.g. 402 pp for 402 pages. About half of the remaining 15% were split between the alternative suffixes pages, pp., and p.. A little below 2% were interpreted as a list of page counts, corresponding perhaps to different editions or formats of a book. Finally, a little over 6% of values do not fall in any of the above categories, and include values such as ”3 vols.” and ”272-287”.

Format	Count
”# pp”	7442
””# p.”	362
”# pp.”	335
”# pages”	372
#	4655
List	237
Others	898
Total	14301

Project Voldemort
A distributed database.

Voldemort

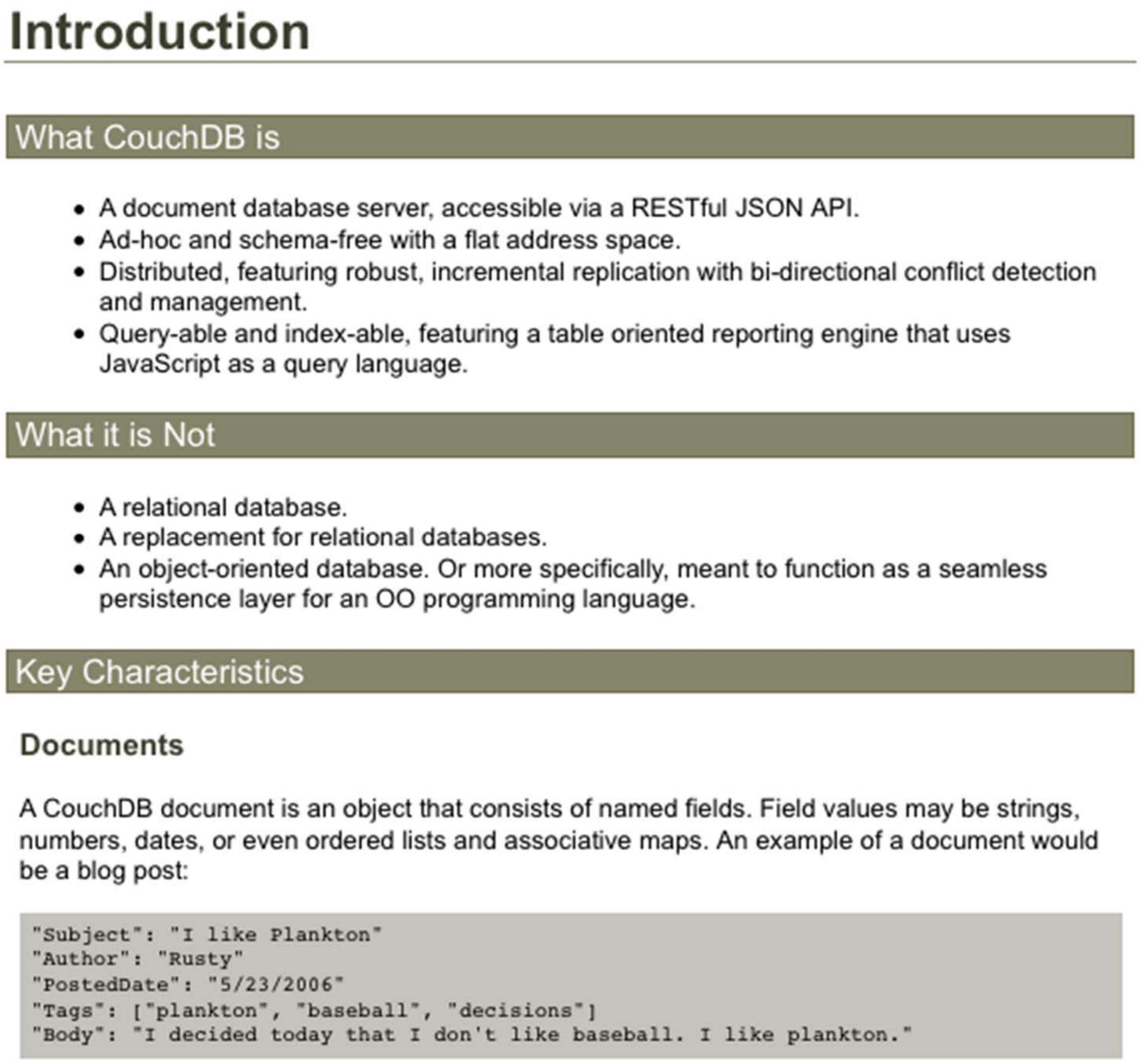


Voldemort is a distributed key-value storage system

- Data is automatically replicated over multiple servers.
- Data is automatically partitioned so each server contains only a subset of the total data
- Server failure is handled transparently
- Pluggable serialization is supported to allow rich keys and values including lists and tuples with named fields, as well as to integrate with common serialization frameworks like Protocol Buffers, Thrift, Avro and Java Serialization
- Data items are versioned to maximize data integrity in failure scenarios without compromising availability of the system
- Each node is independent of other nodes with no central point of failure or coordination
- Good single node performance: you can expect 10-20k operations per second depending on the machines, the network, the disk system, and the data replication factor
- Support for pluggable data placement strategies to support things like distribution across data centers that are geographically far apart.

It is used at LinkedIn for certain high-scalability storage problems where simple functional partitioning is not sufficient. It is still a new system which has rough edges, bad error messages, and probably plenty of uncaught bugs. Let us know if you find one of these, so we can fix it.

Other System: CouchDB



Introduction

What CouchDB is

- A document database server, accessible via a RESTful JSON API.
- Ad-hoc and schema-free with a flat address space.
- Distributed, featuring robust, incremental replication with bi-directional conflict detection and management.
- Query-able and index-able, featuring a table oriented reporting engine that uses JavaScript as a query language.

What it is Not

- A relational database.
- A replacement for relational databases.
- An object-oriented database. Or more specifically, meant to function as a seamless persistence layer for an OO programming language.

Key Characteristics

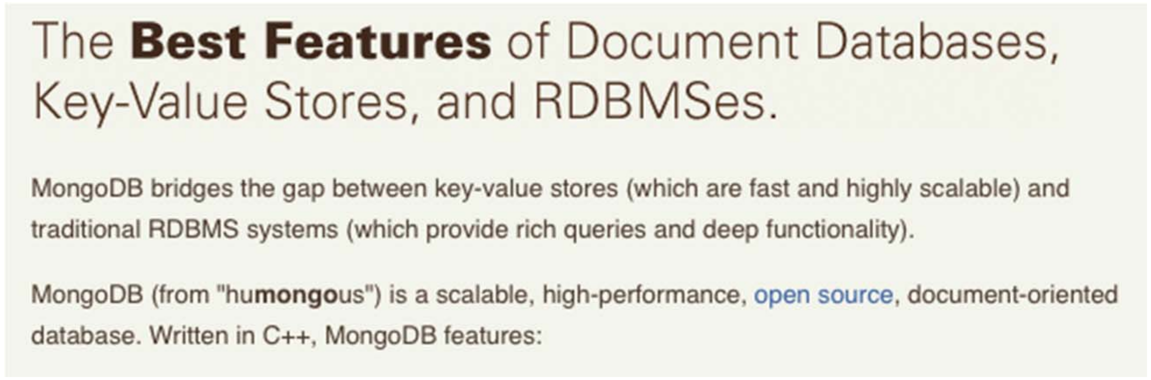
Documents

A CouchDB document is an object that consists of named fields. Field values may be strings, numbers, dates, or even ordered lists and associative maps. An example of a document would be a blog post:

```
"Subject": "I like Plankton"
"Author": "Rusty"
"PostedDate": "5/23/2006"
"Tags": ["plankton", "baseball", "decisions"]
"Body": "I decided today that I don't like baseball. I like plankton."
```

MongoDB

- What is MongoDB:



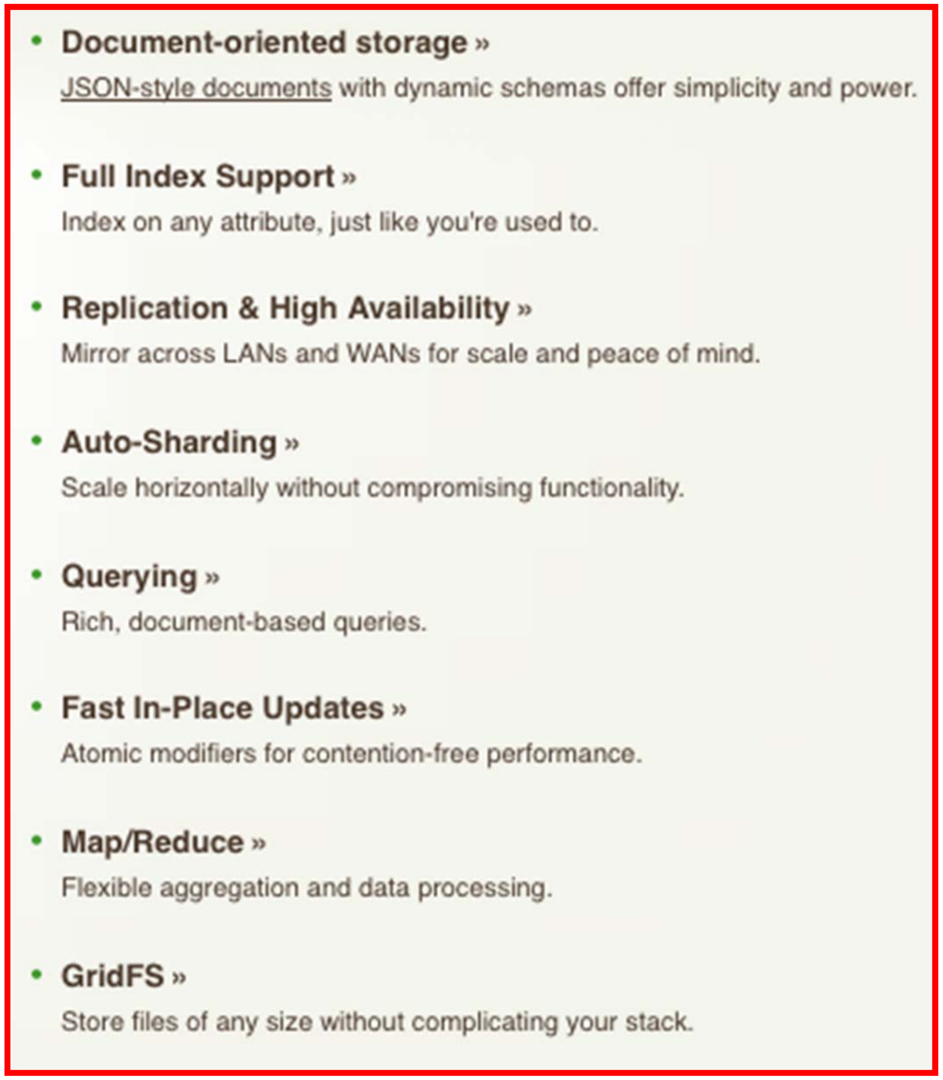
The **Best Features** of Document Databases, Key-Value Stores, and RDBMSes.

MongoDB bridges the gap between key-value stores (which are fast and highly scalable) and traditional RDBMS systems (which provide rich queries and deep functionality).

MongoDB (from "hu**mongo**us") is a scalable, high-performance, open source, document-oriented database. Written in C++, MongoDB features:

- Who uses it?
  - Shutterfly, Foursquare, Intuit, sourceforge, ShareThis, github, Chicago Tribune, …

MongoDB: Features



- **Document-oriented storage »**
  JSON-style documents with dynamic schemas offer simplicity and power.
- **Full Index Support »**
  Index on any attribute, just like you're used to.
- **Replication & High Availability »**
  Mirror across LANs and WANs for scale and peace of mind.
- **Auto-Sharding »**
  Scale horizontally without compromising functionality.
- **Querying »**
  Rich, document-based queries.
- **Fast In-Place Updates »**
  Atomic modifiers for contention-free performance.
- **Map/Reduce »**
  Flexible aggregation and data processing.
- **GridFS »**
  Store files of any size without complicating your stack.

Hands On– A Quick Tutorial

You can try: http://try.mongodb.org/

- Download/install: fromMongoDB.org
- Start server; start shell client
- 1. JavaScript Shell
- 2. Documents
- 3. Saving
- 4. Saving and Querying
- 5. Basic Queries
- 6. Query Operators
- 7. Updates
- 8. Update Operators for Partial Updates
- 9. Deleting Data

Observations? Comments?

- Limitations possibly?

- Advantages possibly?

Back to the Case of Wikipedia

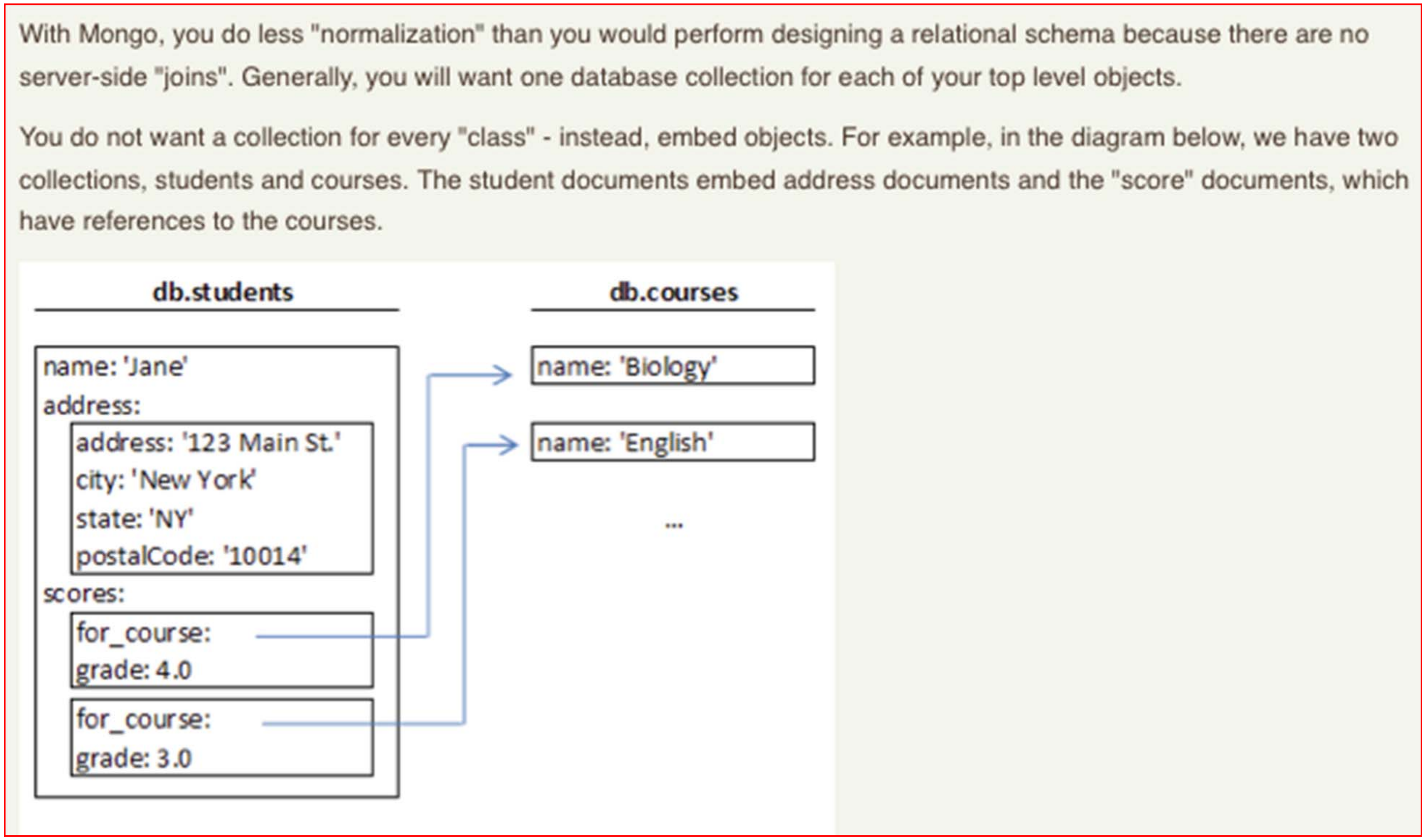
- One collection:
  - title: string
  - template_type: string
  - template: dictionary
  - links: list of string

- Indexed book, person, musical artist, album
- Try out: seven degrees; musical artists.
- Try out queries in shell

Comparing with SQL

- MongoDB code:
  - `query2 = {'template_type':'infobox musical artist', '$or': [{'template.currentmembers': member}, {'template.pastmembers': member}]}`
  - `member_pages = coll.find(spec=query2, fields={'title': 1, 'template.genre':1}, limit=30)`
- SQL code:
  - `query = "select b.title, b.key, b.value from template a, template b where (a.key == 'currentmembers' or a.key == 'pastmembers') and a.value in (%s) and a.title == b.title" % ','.join('?' for m in all_members)`

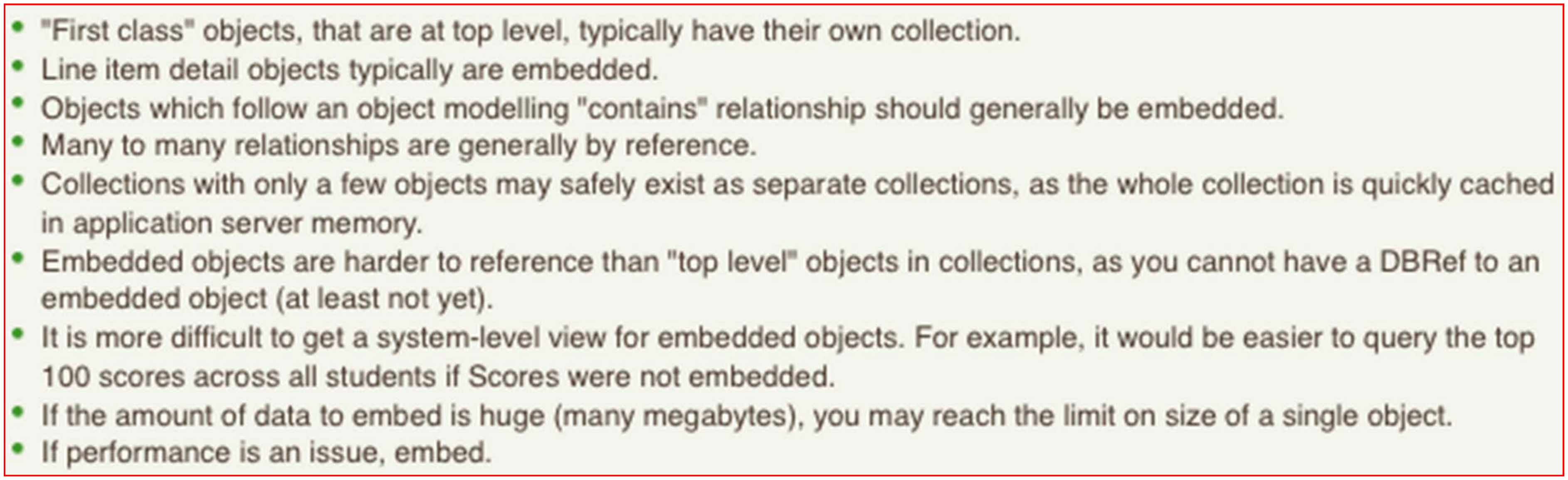
1. Schema Design



With Mongo, you do less "normalization" than you would perform designing a relational schema because there are no server-side "joins". Generally, you will want one database collection for each of your top level objects.

You do not want a collection for every "class" - instead, embed objects. For example, in the diagram below, we have two collections, students and courses. The student documents embed address documents and the "score" documents, which have references to the courses.

Embed vs. Reference: Some General Rules

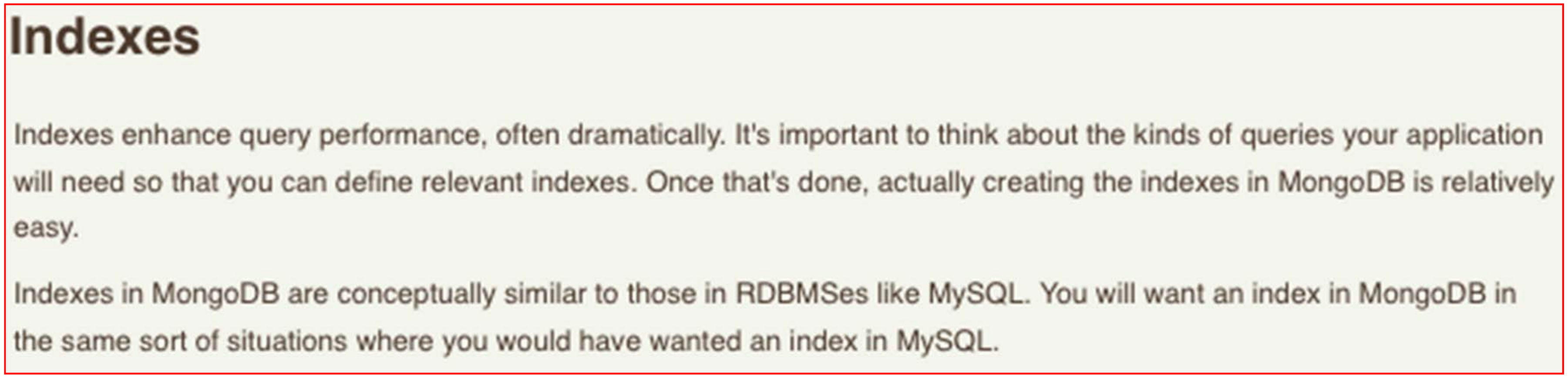


- "First class" objects, that are at top level, typically have their own collection.
- Line item detail objects typically are embedded.
- Objects which follow an object modelling "contains" relationship should generally be embedded.
- Many to many relationships are generally by reference.
- Collections with only a few objects may safely exist as separate collections, as the whole collection is quickly cached in application server memory.
- Embedded objects are harder to reference than "top level" objects in collections, as you cannot have a DBRef to an embedded object (at least not yet).
- It is more difficult to get a system-level view for embedded objects. For example, it would be easier to query the top 100 scores across all students if Scores were not embedded.
- If the amount of data to embed is huge (many megabytes), you may reach the limit on size of a single object.
- If performance is an issue, embed.

Research Issues

- Can we make this transparent?
- Can we still support joins, why not?

2. Data Indexing



Indexes

Indexes enhance query performance, often dramatically. It's important to think about the kinds of queries your application will need so that you can define relevant indexes. Once that's done, actually creating the indexes in MongoDB is relatively easy.

Indexes in MongoDB are conceptually similar to those in RDBMSes like MySQL. You will want an index in MongoDB in the same sort of situations where you would have wanted an index in MySQL.

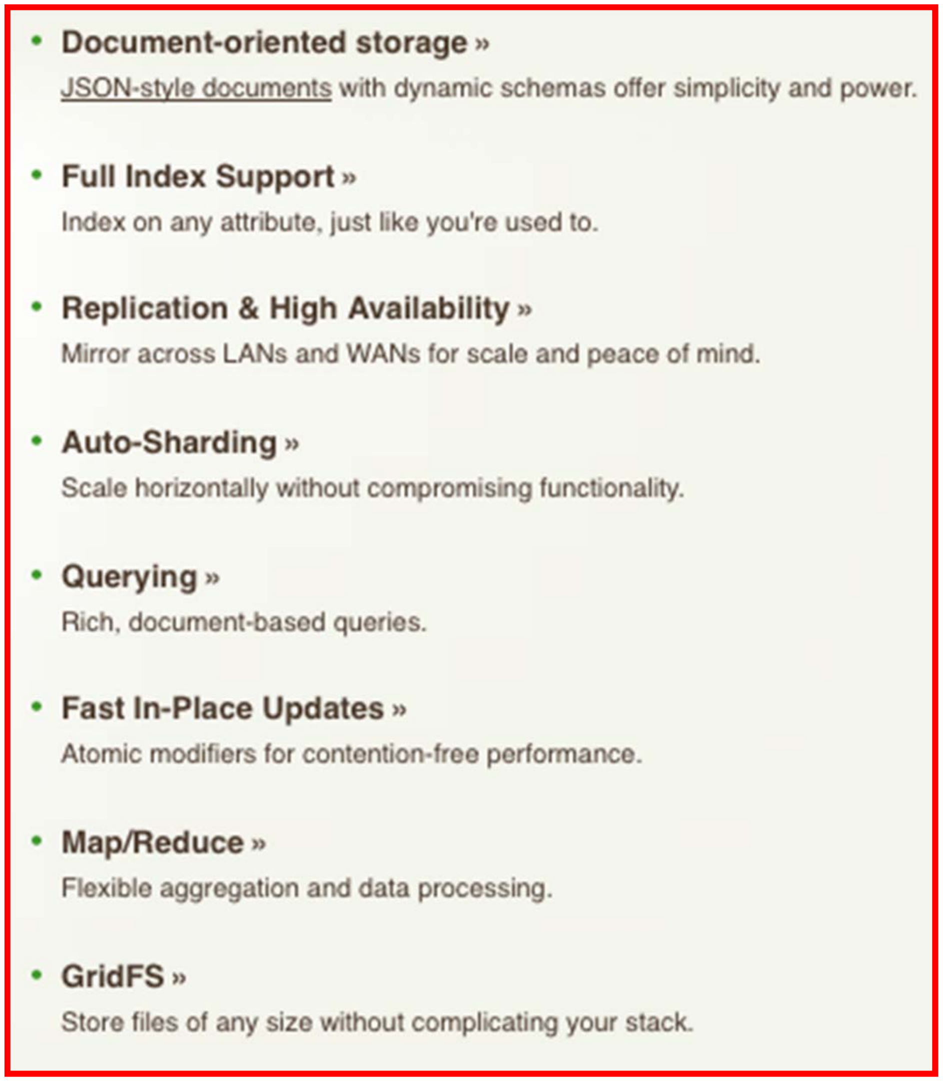
- `db.users.ensureIndex({"name":1});`
- `db.things.ensureIndex({"address.city": 1})`
- `db.factories.insert( { name: "xyz", metro: { city: "New York", state: "NY" } } );`
- `db.factories.ensureIndex( { metro : 1 } );`

Research Issues

- How to index with knowing attribute types?
- How to index complex documents?

The Rest are almost implied,

or at the cost of sacrificing full ACID for high performance



- **Document-oriented storage »**
  JSON-style documents with dynamic schemas offer simplicity and power.
- **Full Index Support »**
  Index on any attribute, just like you're used to.
- **Replication & High Availability »**
  Mirror across LANs and WANs for scale and peace of mind.
- **Auto-Sharding »**
  Scale horizontally without compromising functionality.
- **Querying »**
  Rich, document-based queries.
- **Fast In-Place Updates »**
  Atomic modifiers for contention-free performance.
- **Map/Reduce »**
  Flexible aggregation and data processing.
- **GridFS »**
  Store files of any size without complicating your stack.

Have fun!