In the begining

Structured was it!

Structured Data

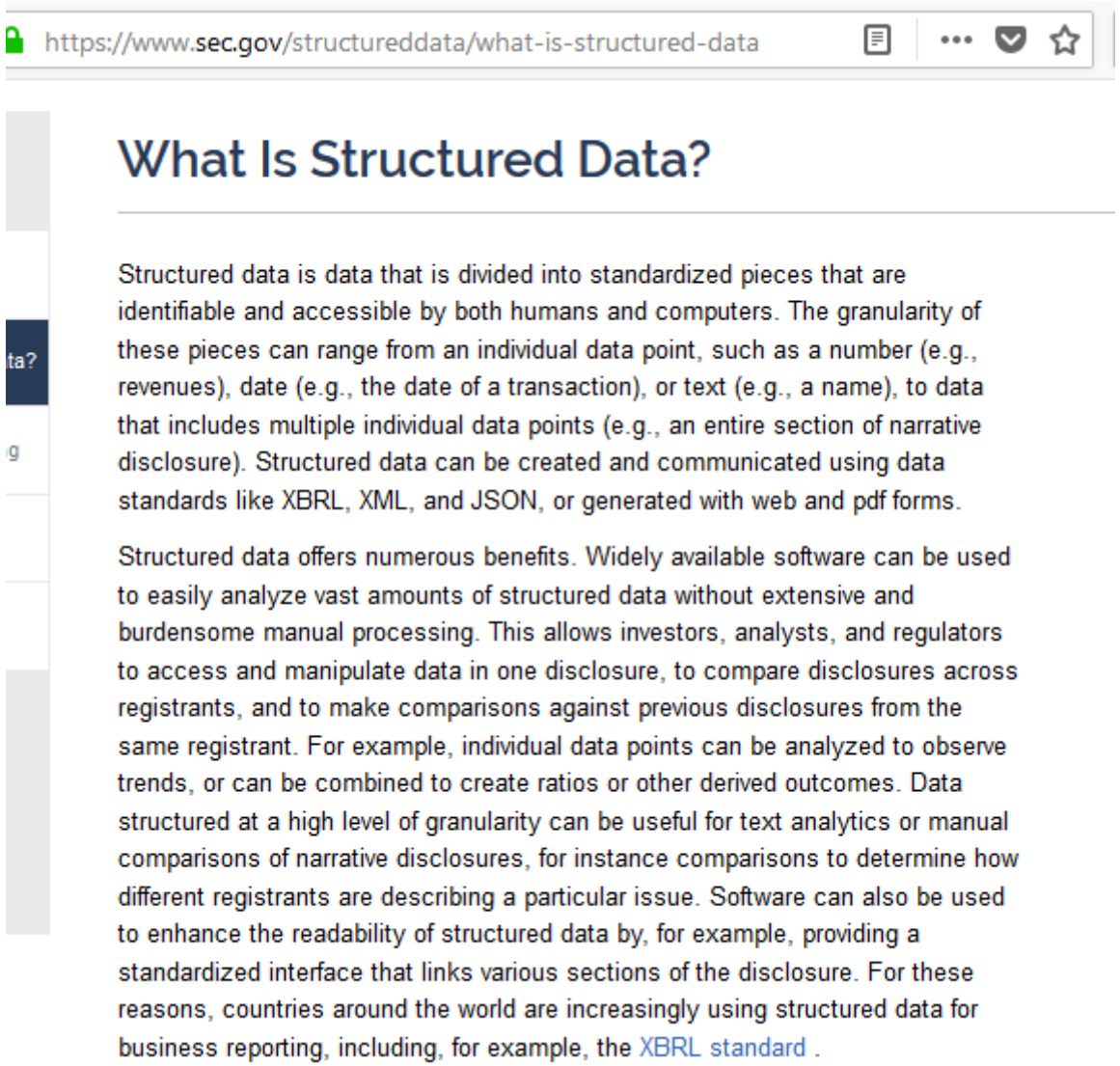
Why was it so?

Computers barely existed!

There were no networks!

No browsers!

Not considered social!
All data was generated by
Business applications and they had
the structure etc
PoS and reservation systems



Progression

Important Events in the Development of Computers

Other timelines: Semiconductor Industry Association, ComputerHistory.Org, History of Digital Storage, Micron [PDF],

Year	Event
1946	ENIAC, First Digital, Electronic Computer Completed. Price: $486,804.22. The project contracted by the Army for ballistic missle calculations to for use in World War II, but the project wasn't completed until the war was over.
1951	Univac, First Commercial Computer. Prices vary from $159,00 to 1,500,000. One was purchased in 1954 by John Hancock Insutrance for about $1,500,000.
1956	IBM Sells First Hard Drive, 5 MB for $50,000, $10,000 per megabyte. Developed at IBM labs in San Jose, CA.

http://www.computerbusinessresearch.com/Timeline

http://www.computerbusinessresearch.com/Home/database/network-database-model

Onto Relational Model

History

The network database model was invented by Charles Bachman in 1969 as an enhancement of the already existing database model, the hierarchical database model. Because the hierarchical database model was highly flaw, Bachman decided to create a database that is similar to the hierarchical database but with more flexibility and less defaults. The original and existing hierarchical database has one owner file linked strictly to one member file, creating a ladder affect that restricted the database to find relationships outside of its category.

Network Database vs. Hierarchical Database Model

Network Database Model	Hierarchical Database Model
Many-to-many relationship	One-to-many relationship
Easily accessed because of the linkage between the information	Difficult to navigate because of its strict owner to member connection
Great flexibility among the information files because the multiple relationships among the files	Less flexibility with the collection of information because of the hierarchical position of the files

Comparison

For relational databases
E/R is the essential piece.

Schema is essential.

Schema cannot change often
Expensive task.

Business evolves – structure and
Semantics of data changes…

Network Database vs. Hierarchical Database Model

Network Database Model	Hierarchical Database Model
Many-to-many relationship	One-to-many relationship
Easily accessed because of the linkage between the information	Difficult to navigate because of its strict owner to member connection
Great flexibility among the information files because the multiple relationships among the files	Less flexibility with the collection of information because of the hierarchical position of the files

Network Database vs. Hierarchical Database Model

Network Database Model	Relational Database Model
The files are greatly related	Information is stored on separate tables tied together with other clumps of information

Advantages of a Network Database Model

- Because it has the many-many relationship, network database model can easily be accessed in any table record in the database
- For more complex data, it is easier to use because of the multiple relationship founded among its data
- Easier to navigate and search for information because of its flexibility

Disadvantage of a Network Database Model

- Difficult for first time users
- Difficulties with alterations of the database because when information entered can alter the entire database

RDB -- ideal for transactional systems

Distributed
Concurrent

ACID

When the internet took hold, people to people communications emerged – social exchange….and that is not structured… unstructured data dominates today – structured data volume is still essential but a fraction…this will continue

FB etc faced extreme difficulties trying to work with RDB…

Comes along BASE and CAP → Schemaless dataStores without ACID

For the purpose of this discussion, we can define a transaction as a unit of work containing one or more operations involving one or more shared resources having ACID properties. ACID is an acronym for atomicity, consistency, isolation and durability, the four important properties of transactions. The meanings of these terms is:

- **Atomicity** : A transaction must be atomic. This means that either all the work done in the transaction must be performed, or none of it must be performed. Doing part of a transaction is not allowed.
- **Consistency** : When a transaction is completed, the system must be in a stable and consistent condition.
- **Isolation** : Different transactions must be isolated from each other. This means that the partial work done in one transaction is not visible to other transactions until the transaction is committed, and that each process in a multi-user system can be programmed as if it was the only process accessing the system.
- **Durability** : The changes made during a transaction are made persistent when it is committed. When a transaction is committed, its changes will not be lost, even if the server crashes afterwards.

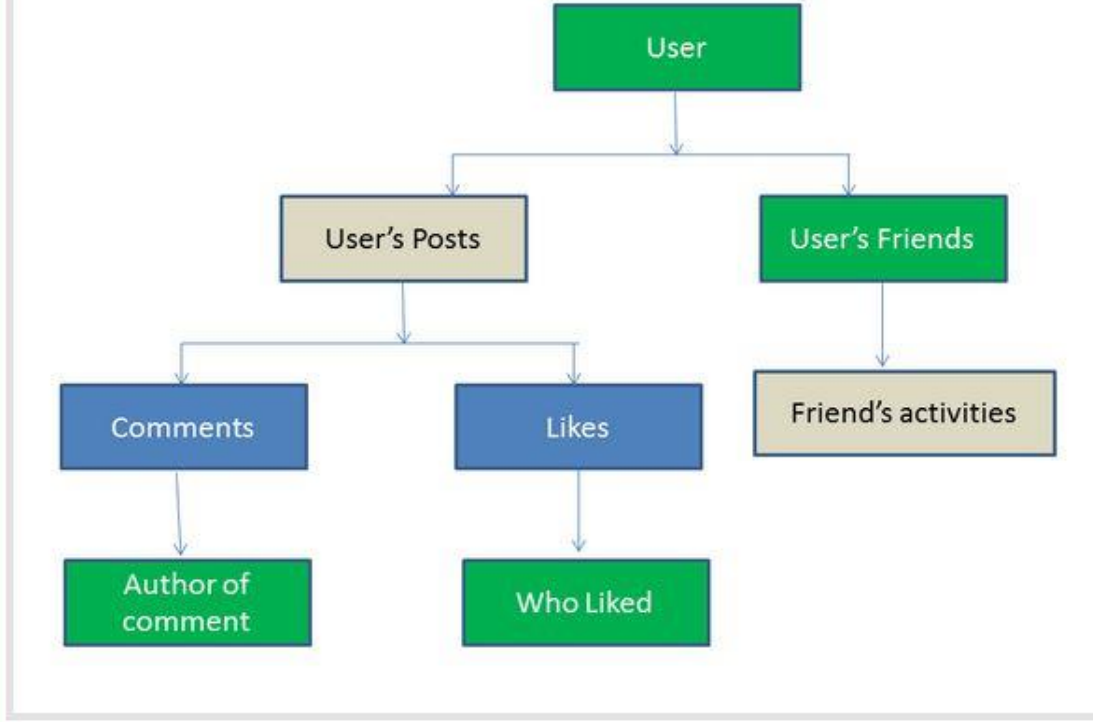
Inside a RDBMS Engine

https://www.youtube.com/watch?v=Z_cX3bzkExE

https://www.youtube.com/watch?v=C_Uv_4I9gus

https://www.youtube.com/watch?v=vAHZjOLmu1g

Issues With RDBMS



In the above diagram, same color box fall into same category object. For example the user, user's friends, who liked and Author of comments all are FB users. Now if we try to store the entire data in RDBMS, for executing a single query which can be just a response of opening home page, we need to join multiple tables with trillions of row together to find a combined table and then run algorithms to find the most relevant information for the user.

Issues Incompatibilities

One size does not fit ALL

World is NOT tabular

tabular to business object translation

Does not keep up with high velocity data

UnStructured

Putting them all together –
– Threading a narrative/story

Introduction to Unstructured data

NOSQL – Not Only SQL Engines

Introduction to mongo

Columnar

Intro to Columnar: http://www.youtube.com/watch?v=8KGVFB3kVHQ

https://www.youtube.com/watch?v=wYl_YxqTof4

https://www.youtube.com/watch?v=mRvkikVuojU

There is no benefit to Columnar approach to manipulate single row or operations one row at a time.

Writes (updating) are NOT trivial and not suitable for Columnar

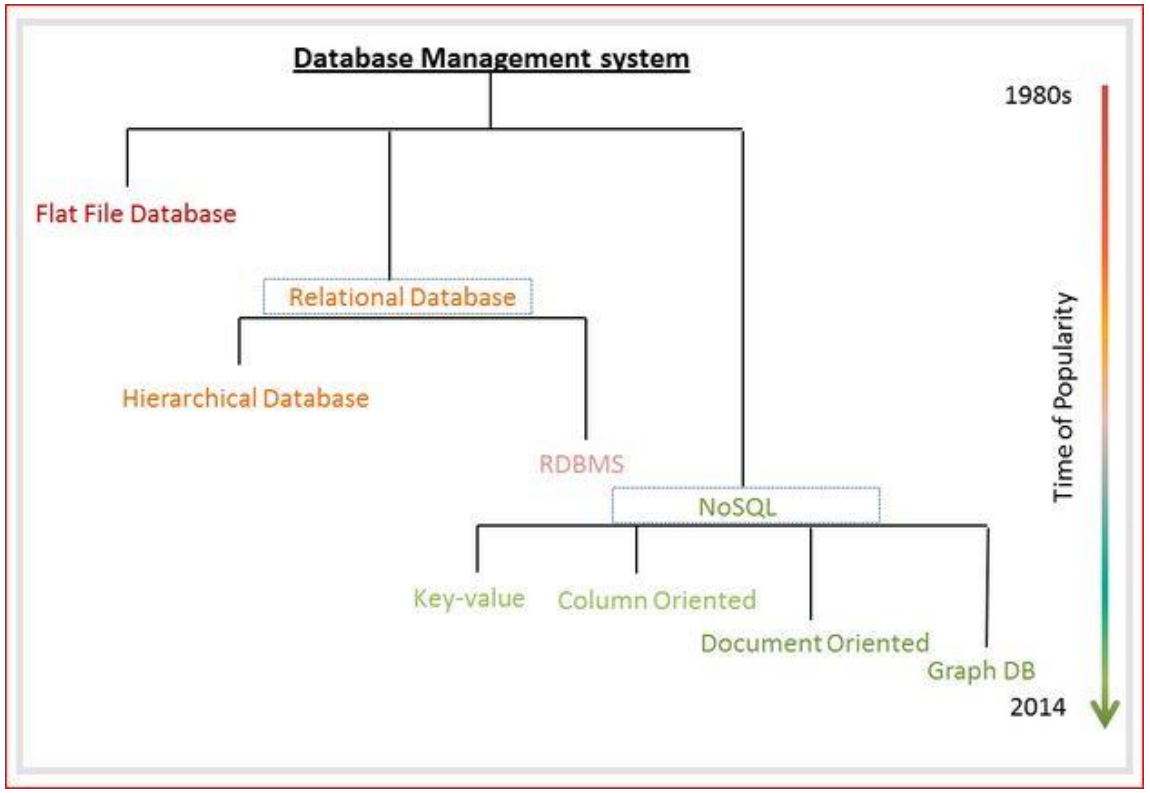
Columnar database is not suitable for OLTP

Structure of Data

Table/column – relational
Name/value pairs – ndbm ++
Document Oriented – mongodb
Graph databases – neo4j
Wide column – cassandra

TimeSeries
XML/json databases

History



Pre-WWW

Data is transactional
ACID/RDBMS rules

Advent of WWW
Email+FTP+Image
+MP3+A/V
Mostly p2p comm
Usenet Groups,
broadcast,list

Social
Communication
CSCW
Youtube
Facebook
twitter

Have we seen it all? Is this the end? Cognitive and contextual computing

Siesmic Shift

Transactional data used to be the bulk of the data
– impersonal, not people oriented, structured, text

Dynamic structure
Structure is not static
Changes
Unmanageable volumes
At speeds never
Anticipated
Interactions are long lived

→

RDBMS cannot handle.
Give up Schema
Give up Structure
Give up Transaction
trade ACID for CAP

Now, it is ALL unstructured, mostly human-oriented,
Unstructured, and variety multimedia stream

And Moore's Law effect → faster computer, faster network
Netflix is streaming movies on par with theater experience

Hello (n silent for hell no)

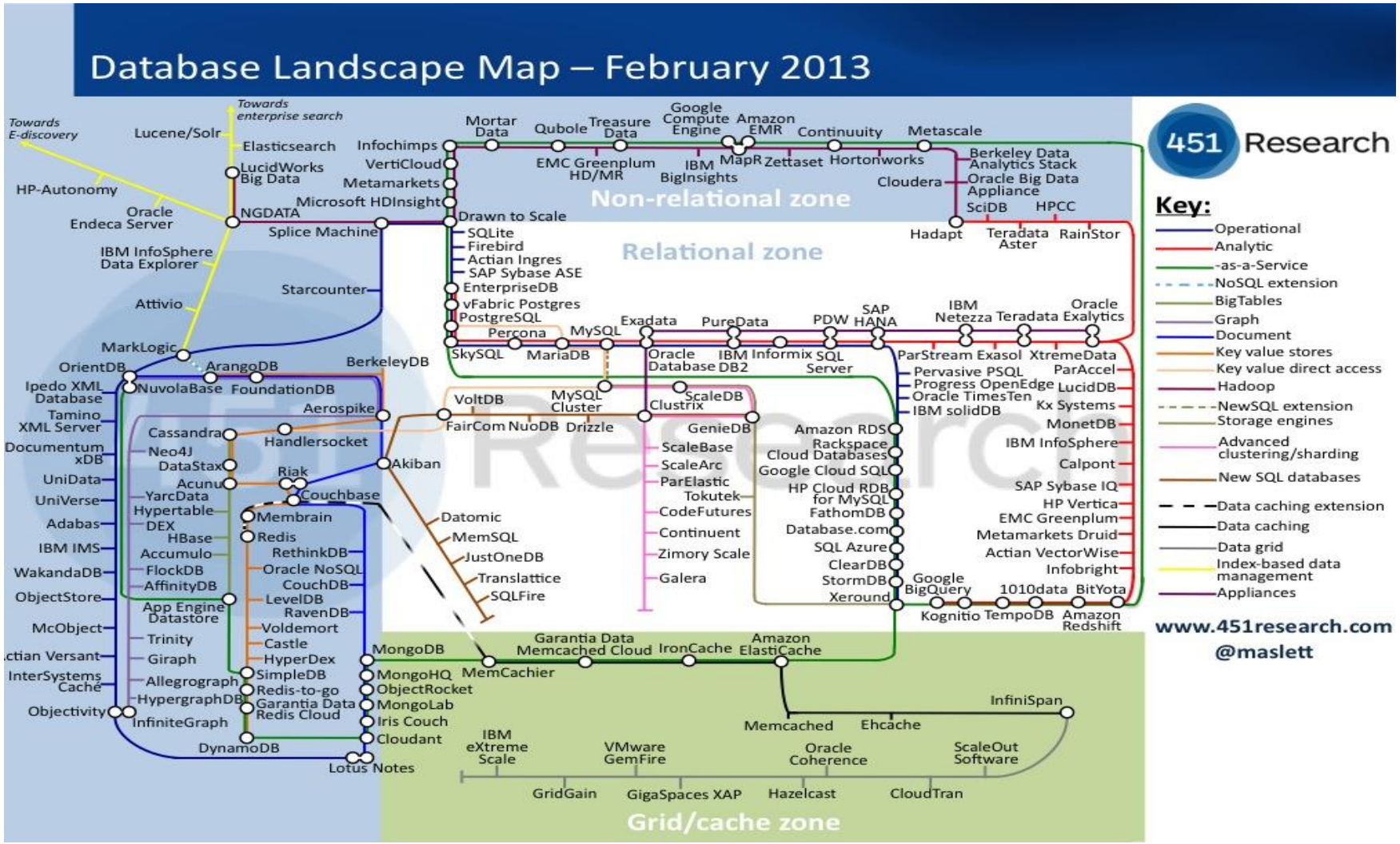
It just started – it will never be over –
pace of new technological innovation is accelerating

Imagine which
car said what to
which car
after a pile up
Insurance
companies
Where is the
black box?

Drones, geolocation, smart delivery
Skies will be crowded
Man made eclipse when there is a snarl
IoT internet of things
Talking humans → talking devices, cars
Nothing is listening everything is talking
Cognitive Computing
Intelligent human compatible computing

Paste is out of the tube..cannot start over.. it will never be the same

Crowded Place



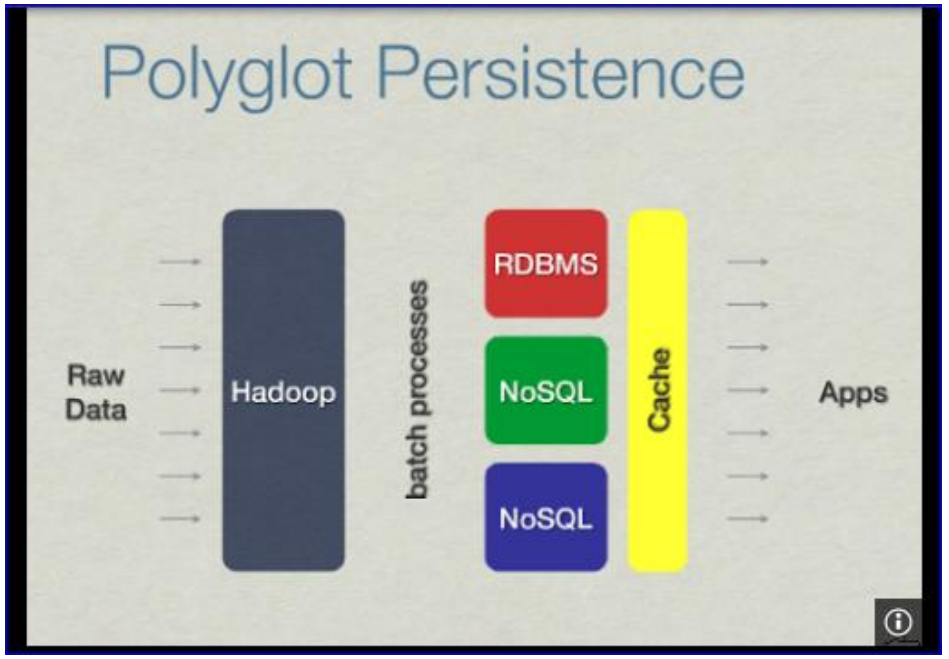
Polyglot Persistence

Your choice of data store really depends on two key factors:

1. the type of data you're working with, and
2. the sort of workload.

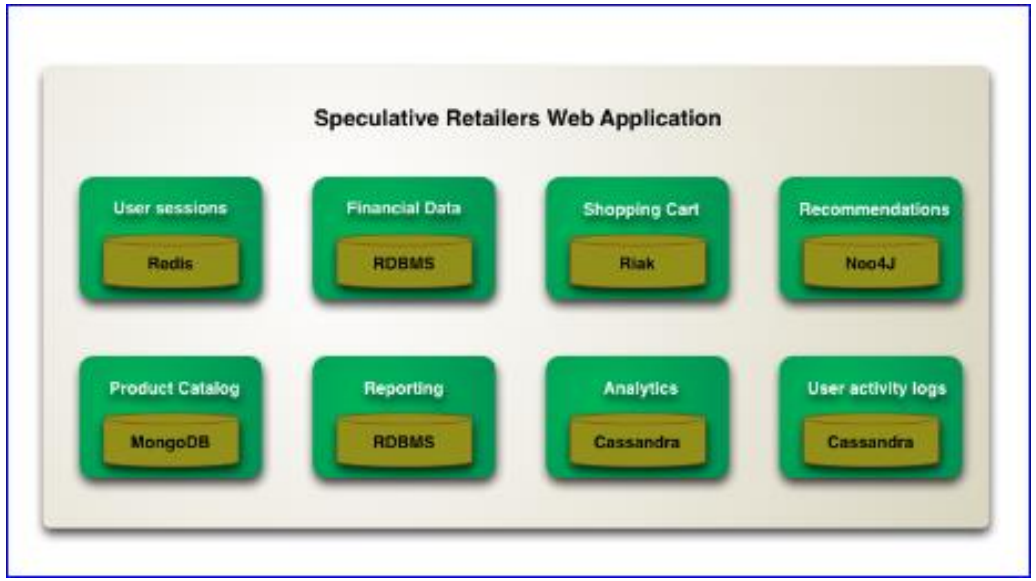
https://www.mapr.com/products/polyglot-persistence

Reference Architecture



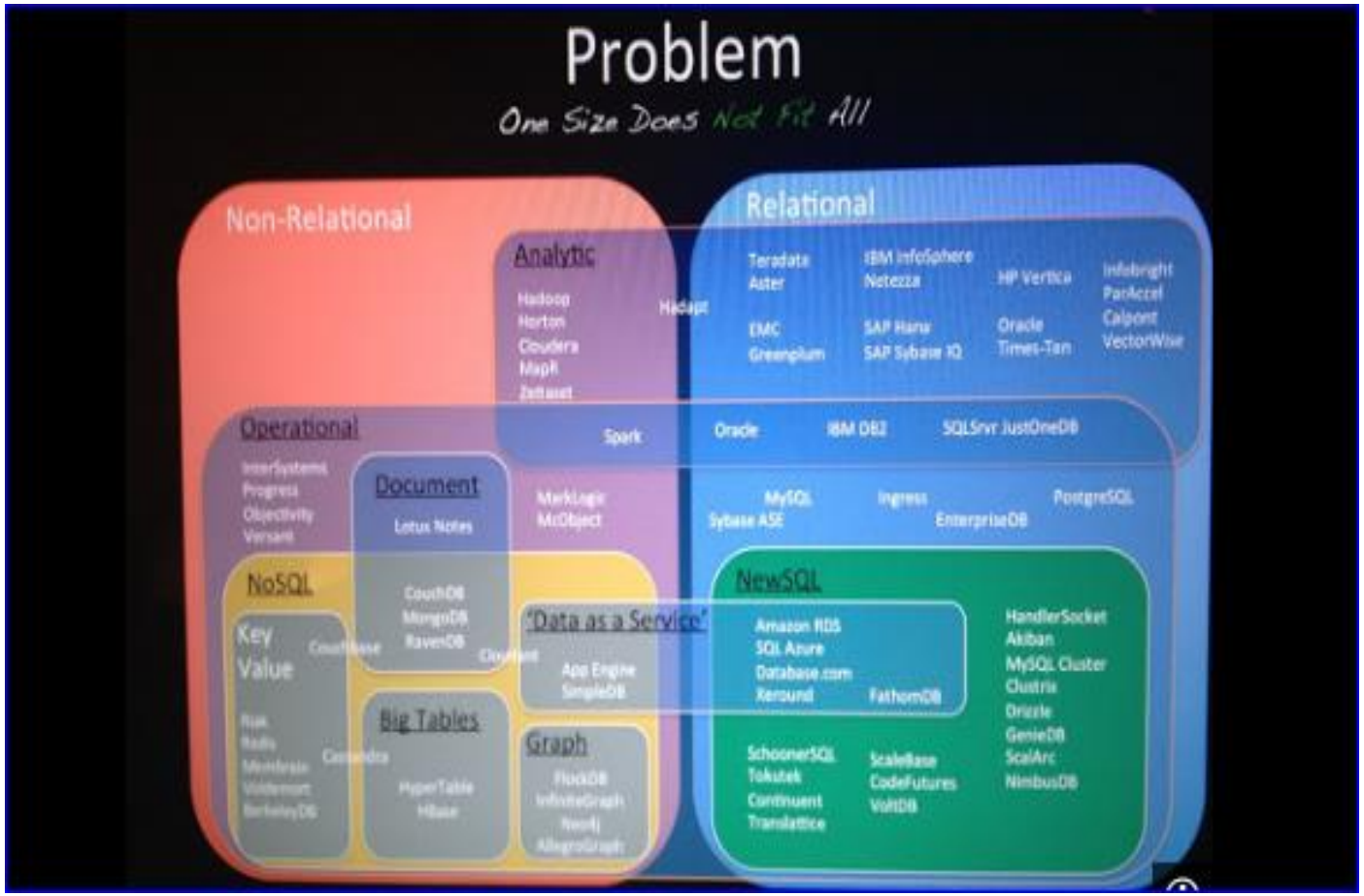
http://nosql.mypopescu.com/post/4017169246/polyglot-persistence-the-architecture-of-the

Polyglot Persistence



http://architects.dzone.com/articles/martin-fowler-polyglot

Thinking out of the box



Interesting read

http://www.infoworld.com/article/2616126/
Application-development/
10-things-never-to-do-with-a-relational-database.html

If you need a plane, stapling a flock of birds to a truck is not a strategy

http://arnoldit.com/wordpress/2008/06/28/
idcs-database-market-share-analysis/

RDBMS → 19 billion dollar ostrich (It aint dead yet..)

http://www.infoworld.com/article/2617876/database/
which-freaking-database-should-i-use-.html

http://download.microsoft.com/download/6/C/D/
6CDC123B-A159-438C-BBAD-7750EE5D0D67/
IDC-Server%20Workloads%20Forecast.pdf

What is NoSQL

- Not only SQL
- Many variant
  - Document oriented
  - Graph database
  - Key/value
  - Wide column
- Mongo – document oriented
  - Hu-***mongo***-us
  - No transaction / no acid / no schema / no relation

more

- CAP not ACID
- Eventually consistent
- Document --> record
- Operation on one document is atomic
  - No partial update to a single document

Resources

http://docs.mongodb.org/manual/tutorial/write-scripts-for-the-mongo-shell/
http://howtodoinjava.com/2014/05/26/introduction-to-mongodb-why-mongodb/
http://docs.mongodb.org/manual/core/introduction/
http://howtodoinjava.com/2014/05/29/mongodb-selectqueryfind-documents-examples/
http://jandiandme.blogspot.com/search/label/CAP%20Theorem
http://jandiandme.blogspot.de/2013/06/mongodb-and-cap-theorem.html
http://info.mongodb.com/rs/mongodb/images/10gen_Top_5_NoSQL_Considerations.pdf
http://css.dzone.com/articles/how-acid-mongodb

http://docs.mongodb.org/manual/tutorial/getting-started/

http://www.mkyong.com/mongodb/how-to-install-mongodb-on-mac-os-x/
http://docs.mongodb.org/manual/reference/operator/#comparison

http://www.mkyong.com/mongodb/mongodb-authentication-example/

ACID Gone with the Wind!

CAP

The CAP theorem by Brewer basically says that a distributed systems can only have two of the following three properties:

- Consistency i.e. each node has the same data
- Availability i.e. a node will always answer queries if possible
- Partition tolerance i.e. work despite a network failure so nodes cannot communicate with one another

CAP Theorem

What is CAP about?

The *(CAP) theorem* (**C**onsistency, **A**vailability and **P**artitioning tolerance) was given by Eric Brewer, a professor at the University of California, Berkeley and one of the founders of Google, in 2001 in the keynote of Principles of Distributed Computing.

The theorem states:

> ***Though its desirable to have Consistency, High-Availability and Partition-tolerance in every system, unfortunately no system can achieve all three at the same time.***

In other words a system can have at most two of three desirable properties at the same time **in presence of errors**.

http://ivoroshilin.com/2012/12/13/brewers-cap-theorem-explained-base-versus-acid/

Definitions

Consistency: A service that is consistent should follow the rule of ordering for updates that spread across all replicas in a cluster – “what you write is what you read”, regardless of location. For example, Client A writes 1 then 2 to location X, Client B cannot read 2 followed by 1. This rule has another name “Strong consistency”.

Availability: A service should be available. There should be a guarantee that every request receives a response about whether it was successful or failed. If the system is not available it can be still consistent. However, consistency and availability cannot be achieved at the same time. This means that one has two choices on what to leave. Relaxing consistency will allow the system to remain highly available under the partitioning conditions (see next definition) and strong consistency means that under certain conditions the system will not be available.

Partition tolerance: The system continues to operate despite arbitrary message loss or failure of part of the system. A simple example, when we have a cluster of N replicated nodes and for some reason a network is unavailable among some number of nodes (e.g. a network cable got chopped). This leads to inability to synchronize data. Thus, only some part of the system doesn’t work, the other one does. If you have a partition in your network, you lose either consistency (because you allow updates to both sides of the partition) or you lose availability (because you detect the error and shut down the system until the error condition is resolved).

BASE

A simple meaning of this theorem is “It is impossible for a protocol to guarantee both consistency and availability in a partition prone distributed system”.

Eventual consistency
(BASE)

The term eventual consistency or as it is called BASE (Basically Available, Soft state, Eventual consistency) is the opposite of ACID (Atomicity, Consistency, Isolation and Durability). Where ACID is pessimistic and requires consistency at the end of every operation, BASE is optimistic and accepts that the database consistency will be in a state of flux. The eventual consistency is simply an acknowledgement that there is an unbounded delay in propagating a change made on one machine to all the other copies which might lead to stale data.

Node/Read/Write

NRW (Node, Read, Write) allows to analyse and tune how a distributed database will trade off consistency, read / write performance.

N = the number of nodes that keep copies of a record distributed to.
W = the number of nodes that must successfully acknowledge a write to be successfully committed.
R = the number of nodes that must send back the same value of a unit of data for it to be accepted as read by the system.
The majority of NoSQL databases use N>W>1 – more than one write must complete, but not all nodes need to be updated immediately.
When:

W < N – high write availability
R < N – high read availability
W+R > N – is a strong consistency, read/write are fully overlapped
W+R <= N – is an eventual consistency, meaning that there is no overlap in the read and write set;

Off to the races

http://docs.mongodb.org/manual/tutorial/query-documents/

No relations → no joins

Object embedding

Object ID Reference

This needs to be resolved.

Lingo: JSON

Javascript object notation

http://json.org/ & http://stackoverflow.com
for doubts

http://www.freeformatter.com/json-formatter.html#json-explained

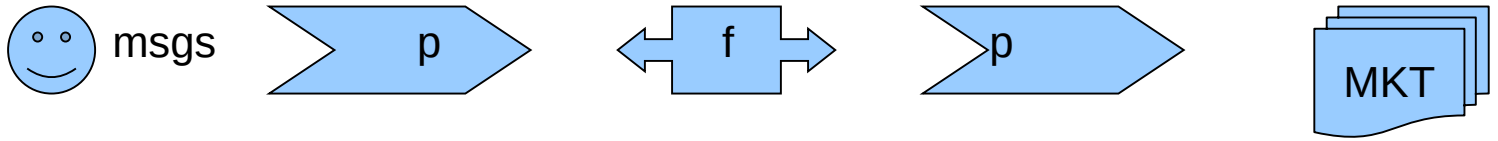
http://www.freeformatter.com/json-validator.html#json-explained

useless example


```
{
  "anObject": {
    "numericProperty": -122,
    "stringProperty": "An offensive \" is problematic",
    "nullProperty": null,
    "booleanProperty": true,
    "dateProperty": "2011-09-23"
  },
  "arrayOfObjects": [
    {
      "item": 1
    },
    {
      "item": 2
    },
    {
      "item": 3
    }
  ],
  "arrayOfIntegers": [
    1,
    2,
    3,
    4,
    5
  ]
}
```


{example:"value"} is invalid but {"example":"value"} is valid.

Useful examples




```
db.filters.insert({ class:"stateless", type:"BasicFilter", name:"BF01",
...  predicate: [ {T:"35", op:"EQ", value: [ "D" ]} ,
 {T:"55",op:"EQ", value: ["IBM","ABB","CSCO"]} ] } );

book1 = {“name”: "Understanding JAVA", “pages” : 100}

db.books.save(book1)
```

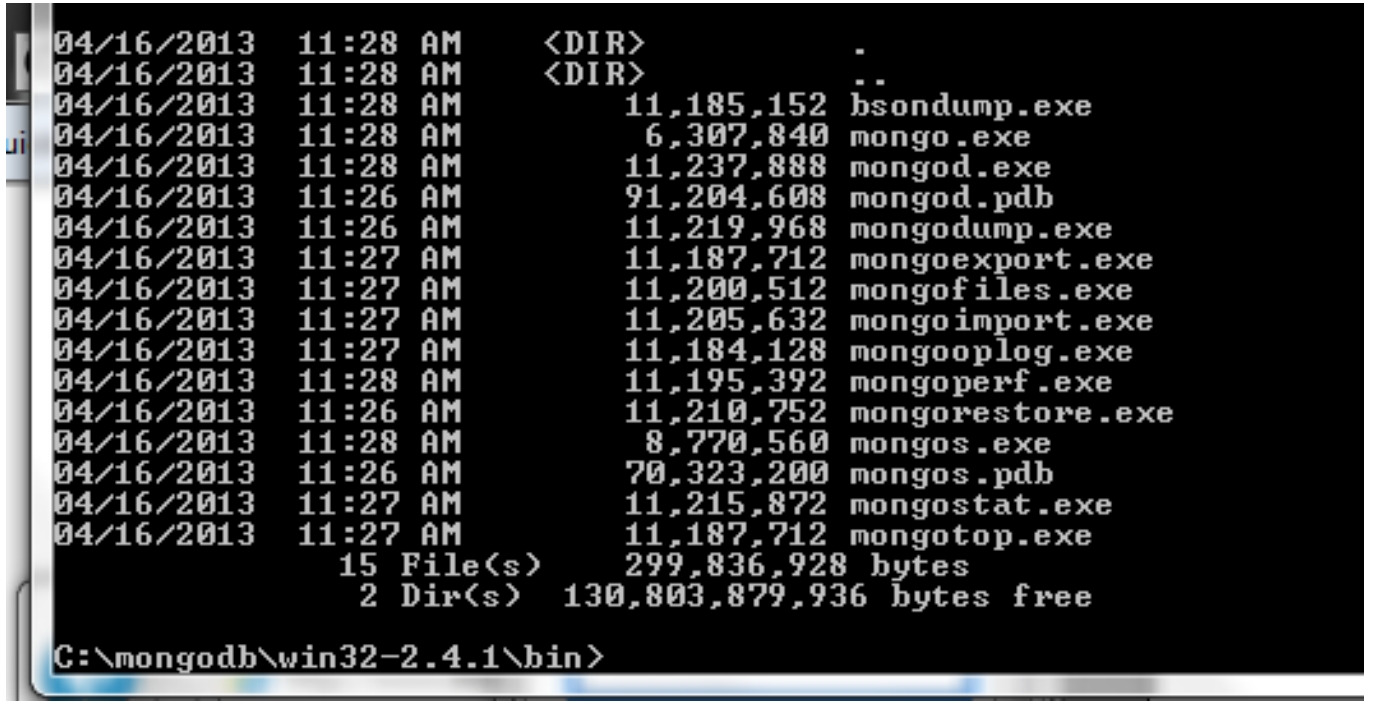
steps

http://docs.mongodb.org/manual/

Then (appraise yourself with install etc)
Minimal configuration
Starting and Stopping and
A few basic commands

http://www.mongodb.org/downloads

Setup: install and verify



Here mongod is the server and mongo is the shell client

Starting the server

Data is mine I dont want to loose if my windoooz goes down on me.

Data is in E drive and I can reinstall mongodb in C as many times as I want to without loosing my data

The default port on which mongod serves is data is 27017.
You can configure both these using a conf file...

Start mongod with the dbpath option

Stopping and admin db

```
C:\mongodb\win32-2.4.1\bin>mkdir e:\MONGODB\

C:\mongodb\win32-2.4.1\bin>mkdir e:\MONGODB\data

C:\mongodb\win32-2.4.1\bin>mkdir e:\MONGODB\data\db

C:\mongodb\win32-2.4.1\bin>mongod --dbpath e:\mongodo\data\db
Tue Apr 16 11:34:34.432
Tue Apr 16 11:34:34.437 warning: 32-bit servers don't have journaling enabled by
 default. Please use --journal if you want durability.
Tue Apr 16 11:34:34.439
Tue Apr 16 11:34:34.472 [initandlisten] MongoDB starting : pid=10464 port=27017
dbpath=e:\mongodo\data\db 32-bit host=vram
```

Create a db directory
Start mongod with the dbpath option

Stopping the server

```
> use admin
switched to db admin
> db.shutdownServer({timeoutSecs: 60});
Tue Apr 16 12:53:01.788 DBClientCursor::init ca
server should be down...
Tue Apr 16 12:53:01.808 trying reconnect to 127
```

use admin

db.shutdownServer({timeoutSecs: 60});

config

fork = true

bind_ip = 127.0.0.1

port = 27017

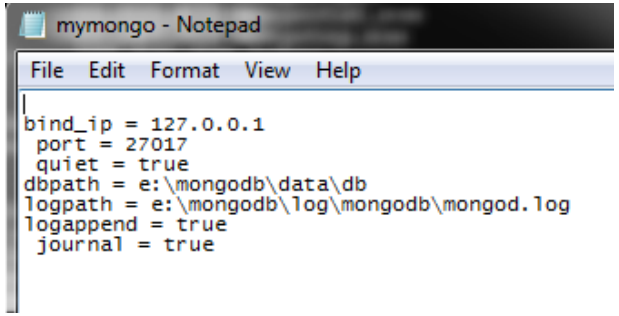
quiet = true

dbpath = e:\mongodb\data\db

logpath = e:\mongodb\log\mongodb\mongod.log

logappend = true

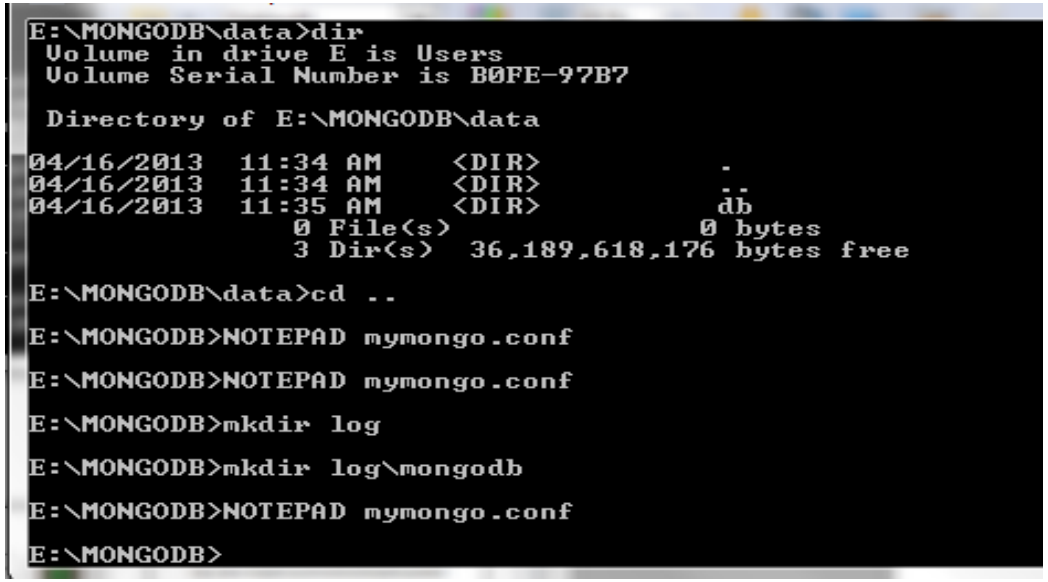
journal = true




```
fork = true
bind_ip = 127.0.0.1
port = 27017
quiet = true
dbpath = /srv/mongodb
logpath = /var/log/mongodb/mongod.log
logappend = true
journal = true
```

Working with PD Software is about patience and T/E.
You cannot give up.
Fork option did not work. My config does not have it

Preparing your DB



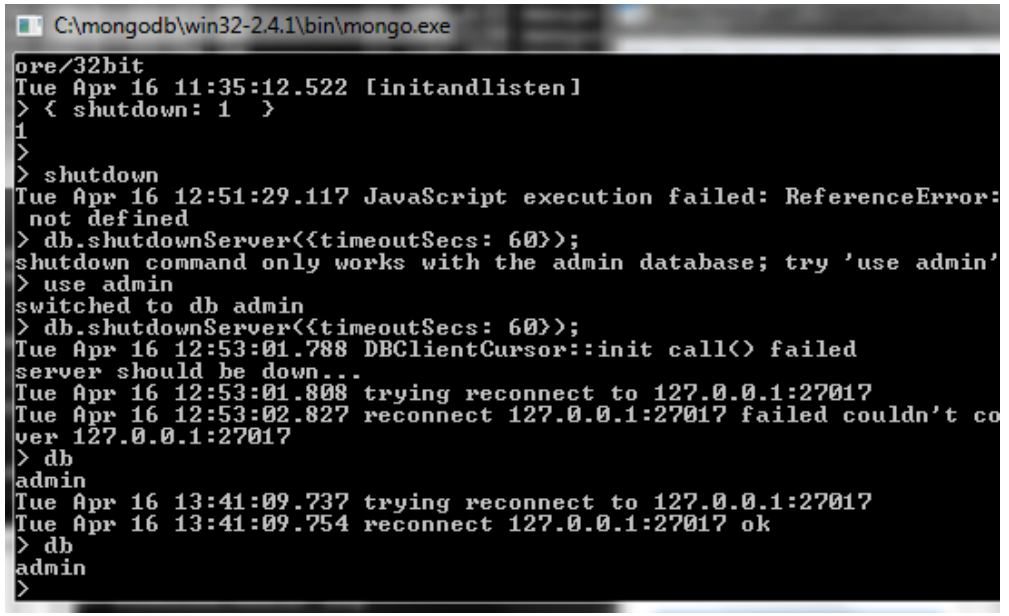
I have stored the config file in e:\mongodb – can be anywhere

I am directing all logs as specfied in the config file

reconnect

Start the server with the config option

```
C:\mongodb\win32-2.4.1\bin>mongod -f e:\MONGODB\mymongo.conf
all output going to: e:\mongodb\log\mongodb\mongod.log
```



Changing DB

```
E:\MONGODB>c:\mongodb\win32-2.4.1\bin\mongo
MongoDB shell version: 2.4.1
connecting to: test
Server has startup warnings:
Tue Apr 16 13:37:30.589 [initandlisten]
Tue Apr 16 13:37:30.590 [initandlisten] ** NOTE: This is a 32 bit MongoDB binary
.
Tue Apr 16 13:37:30.590 [initandlisten] **       32 bit builds are limited to le
ss than 2GB of data (or less with --journal).
Tue Apr 16 13:37:30.590 [initandlisten] **       See http://dochub.mongodb.org/c
ore/32bit
Tue Apr 16 13:37:30.590 [initandlisten]
> db
test
>
```

```
> use mydb
switched to db mydb
> use admin
switched to db admin
> use mydb
switched to db mydb
>
```

Same as mysql!!

C in CRUD

http://docs.mongodb.org/manual/reference/operators/#comparison
http://docs.mongodb.org/manual/core/shell-types/

{symbolname:"IPFF", date:"2013-04-12",open:"27.05",
high:"27.05",low:"26.81",close:"26.86",volume:"35300",
adjClose:"26.86"}

db.ticks.save({symbolname:"IPFF"})
db.ticks.save({symbolname:"IPFF", date:"2013-04-12",
open:"27.05",high:"27.05",low:"26.81",close:"26.86",
volume:"35300",adjClose:"26.86"})

More  Steps

```
db.ticks.find()
db.ticks.find({"high:{$gt:2}"}) -- WILL NOT WORK
db.ticks.find({high:"27.05"})
db.ticks.find({symbolname:"IPFF"})
db.ticks.save({symbolname:"IPFF", date:"2013-04-
12",open:27.05,high:27.05,low:26.81,close:26.86,
volume:35300,adjClose:26.86})
db.ticks.find({high:27.05})
db.ticks.find({high:"27.05"})
db.ticks.find({high:{$gt:27.05}})
db.ticks.find({high:{$gt:27.0}})
```

```
db.ticks.distinct("symbolname")
db.doctor.update([“symbolname”:"IPFF"],[$set:[“high”:
28.02]])
db.ticks.remove(“symbolname”:"IPFF")
```

CRUD

```
db.execs.find({},{_id:0})

{ "ticker" : "IBM", "qty" : "100", "px" : "202" }

{ "ticker" : "IBM", "qty" : "100", "px" : "203" }

{ "ticker" : "IBM", "qty" : "200", "px" : "198.50" }

{ "ticker" : "IBM", "qty" : "200", "px" : "199.50" }

{ "ticker" : "IBM", "qty" : "200", "px" : "199.90" }
```

```
db.execs.find({},{_id:0},{$sort:{px:1}})

{ "ticker" : "IBM", "qty" : "100", "px" : "202" }

{ "ticker" : "IBM", "qty" : "100", "px" : "203" }

{ "ticker" : "IBM", "qty" : "200", "px" : "198.50" }

{ "ticker" : "IBM", "qty" : "200", "px" : "199.50" }

{ "ticker" : "IBM", "qty" : "200", "px" : "199.90" }
```

Loading

mongoimport --db mydb --collection rktrades --type csv --headerline --file trades.js

Sales collection

```
 db.sales.insert({ Year:2000, Region:"USA",Sales:400})
> db.sales.insert({ Year:2001, Region:"USA",Sales:200})
> db.sales.insert({ Year:2002, Region:"USA",Sales:400})

> db.sales.insert({ Year:2002, Region:"Europe",Sales:300})

> db.sales.insert({ Year:2003, Region:"Europe",Sales:100})

> db.sales.insert({ Year:2008, Region:"USA",Sales:500})
> db.sales.insert({ Year:2003, Region:"Asia",Sales:400})

> db.sales.count({Region:"UDS"})
0

> db.sales.count({Region:"USA"})
4
```

aggregate

```
db.sales.group( {

key: { Region: 1 },

cond: { Year: { $lt: 2005 } },

reduce: function(cur, result) { result.total += cur.Sales },

initial: { total: 0 }

} )
```

Group By

```
db.rktrades.group({key:{TKR:1},reduce:function(cur,result){result.total+=
cur.QTY * cur.PX,result.totalQty+=cur.QTY},
initial:{total:0,totalQty:0}})

db.rktrades.group({key:{TKR:1,SIDE:1},reduce:function(cur,result)
{result.total+=cur.QTY * cur.PX,result.totalQty+=cur.QTY},
initial:{total:0,totalQty:0}})


db.runCommand( { distinct: "sales", key:"Region", query:{Sales: {$gt:200}}})
```

Unstructured Data Types

Unstructured data is raw and unorganized and organizations store it all. Ideally, all of this information would be converted into structured data however, this would be costly and time consuming. Also, not all types of unstructured data can easily be converted into a structured model. For example, an email holds information such as the time sent, subject, and sender (all uniform fields), but the content of the message is not so easily broken down and categorized. This can introduce some compatibility issues with the structure of a relational database system.

Here is a limited list of types of unstructured data:

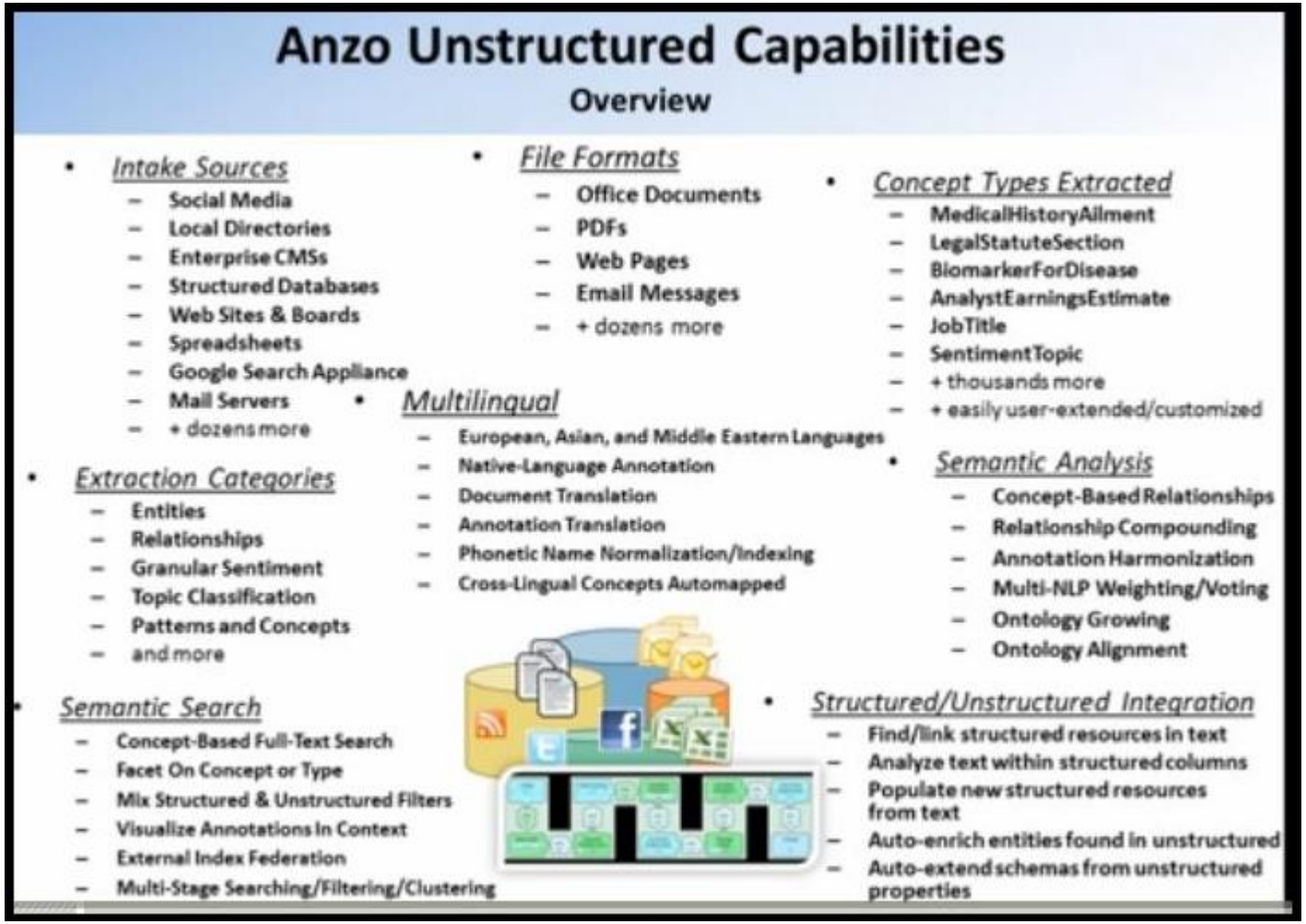
- Emails
- Word Processing Files
- PDF files
- Spreadsheets
- Digital Images
- Video
- Audio
- Social Media Posts

https://breakthroughanalysis.com/2008/08/01/unstructured-data-and-the-80-percent-rule/

[[https://www.bing.com/search?q=proportion+of+unstructured+to+structured+data]]
[[https://sherpasoftware.com/blog/structured-and-unstructured-data-what-is-it/]]

Looking at the list, you may be wondering what these files have in common. The files listed above can be stored and managed without the format of the file being understood by the system. This allows them to be stored in an unstructured fashion because the contents of the files are unorganized.

Unstructured Data



https://www.slideshare.net/camsemantics/introduction-to-anzo-unstructured

ACID →BASE

In Brewer's address he presented a simple table that outlined the essential traits of each of the two models:[8]

ACID	*BASE*
• Strong Consistency	• Weak Consistency – stale data OK
• Isolation	• Availability first
• Focus on "commit"	• Best effort
• Nested transactions	• Approximate answers OK
• Availability?	• Aggressive (optimistic)
• Conservative (pessimistic)	• Simpler!
• Difficult evolution (e.g. schema)	• Faster
	• Easier evolution

Mark Logic/DataVarsity primer

Eventual Consistency

A. Strong (Strict) Consistency – All read operations return the value from the last finalized write operation. It doesn't matter which replica the operation completed the write to; all replicas must be in the same state for the next operation to occur on those values.

B. Eventual Consistency – This has the greatest variability of potential values returned. At any given point readers will see some written value, but there is no guarantee that any two readers will see the exact same write. All replicas will eventually have the latest update; it's just a matter of time when that will happen.

C. Monotonic Read Consistency – This is also known as a session guarantee. Reads are similar to eventual consistency in that the data could still be stale; but monotonic read consistency guarantees that over time the client will get a more up-to-date read (or the same read) if they request a read from the same object.

D. Read Your Own Writes (or Read My Writes) – guarantees that the client always reads their most recent writes, but other may not see the same updates. It doesn't matter what replica the writes are going to, the client always sees their most updated one.

E. Causal Consistency – if a client reads one value (a) and then writes the next value (b), and another client then reads the value of (b) they will also see the value of (a) since they are connected to each other. Therefore, any writes that are causally related must be seen by all processes in the specific order they were written.[9]

Datavarsity

BASE/CAP

ACID constraints provide strong consistency, all the time, no matter what. Such requirements often have repercussions, though; especially regarding availability. If the system must always remain in a consistent state, so all parties see the same view of the data at the beginning and end of a transaction, then across thousands of nodes that data may not always be available.

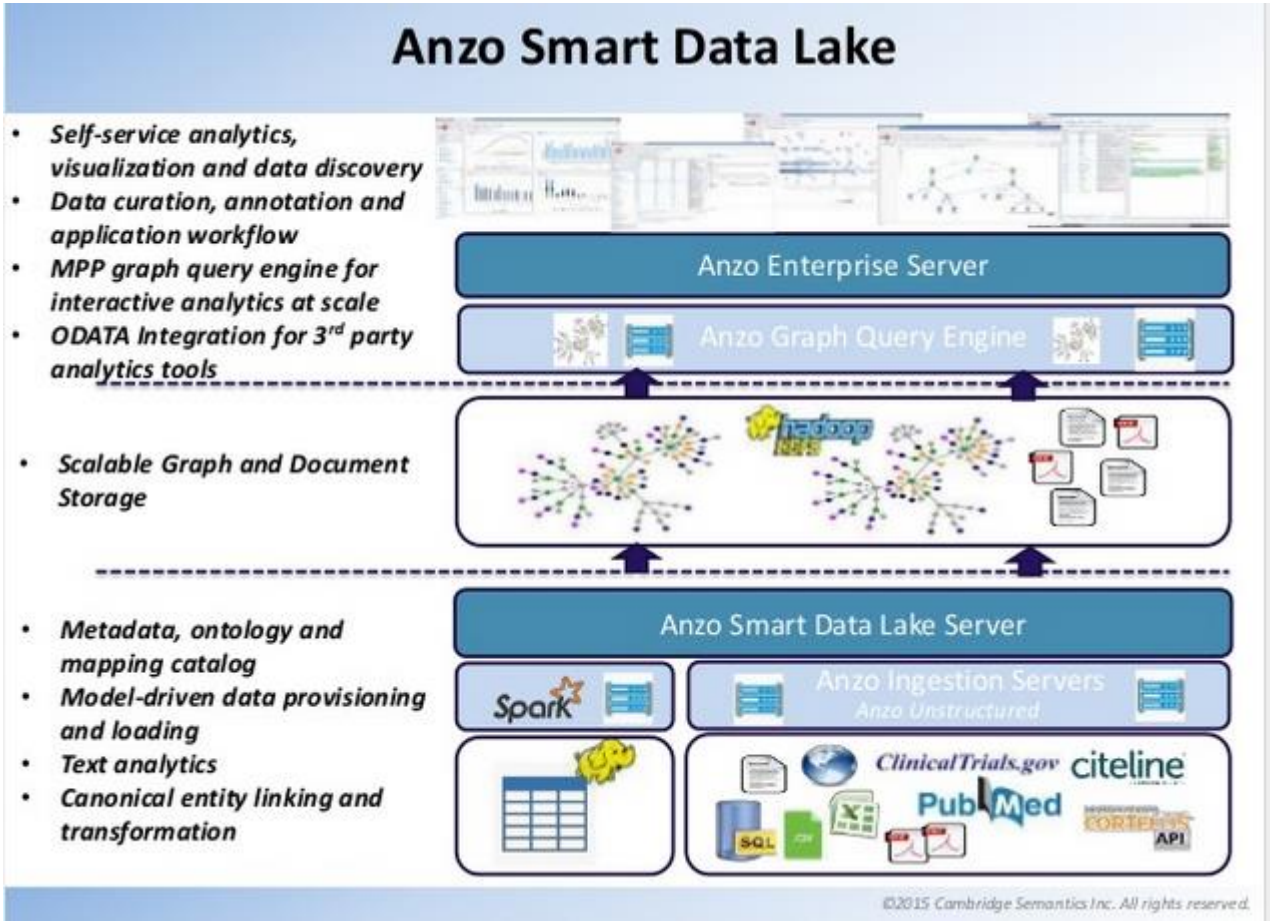
CONSISTENCY - HOW IS THE DATA PERCEIVED?

AVAILABILITY – HOW ACCESSIBLE IS THE DATA?

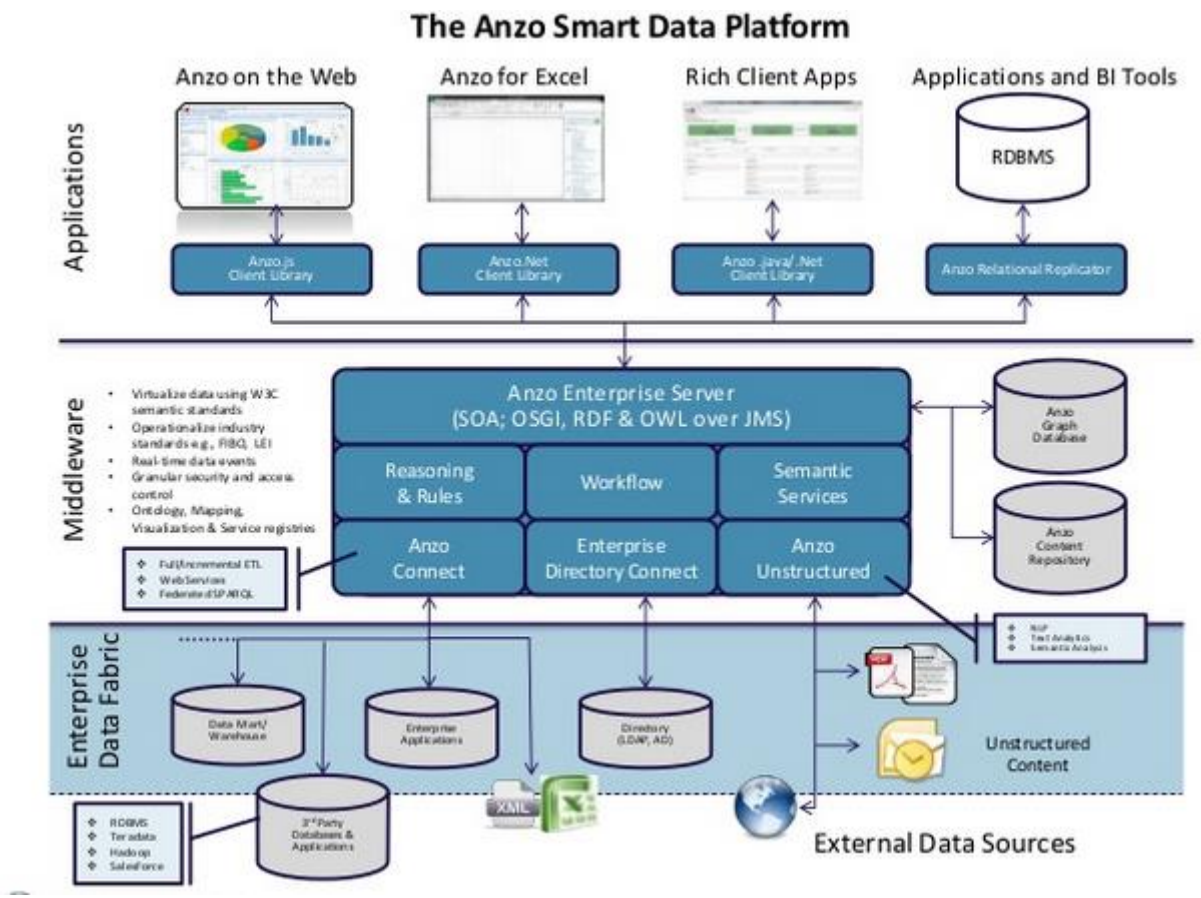
The IEEE 610 definition of availability is “the degree to which a system or component is operational and accessible when required for use by an authorized user.”1

PARTITION TOLERANCE – FUNCTION WITHOUT COMMUNICATION?

Where are we headed?



Anzo Platform?



Capability Matrix