Databases 1

Daniel POP

Week 14

Course Outline

1. ~~Introduction to database approach~~
2. ~~The database environment~~
3. ~~Introduction to The Relational Model~~
4. ~~Views~~
5. ~~Transactions~~
6. ~~SQL Constraints~~
7. ~~Relational Database Design. Theory and practice~~
8. ~~An Introduction to Database Performance. Indexing~~
9. ~~JSON Support in Relational Database Management Systems~~
10. **NoSQL Databases**

Introduction to NoSQL databases

Agenda

1. Introduction to NoSQL databases
2. Key-value database
3. Document database
4. Column-family database
5. Graph database
6. Wrapping-up and few advices

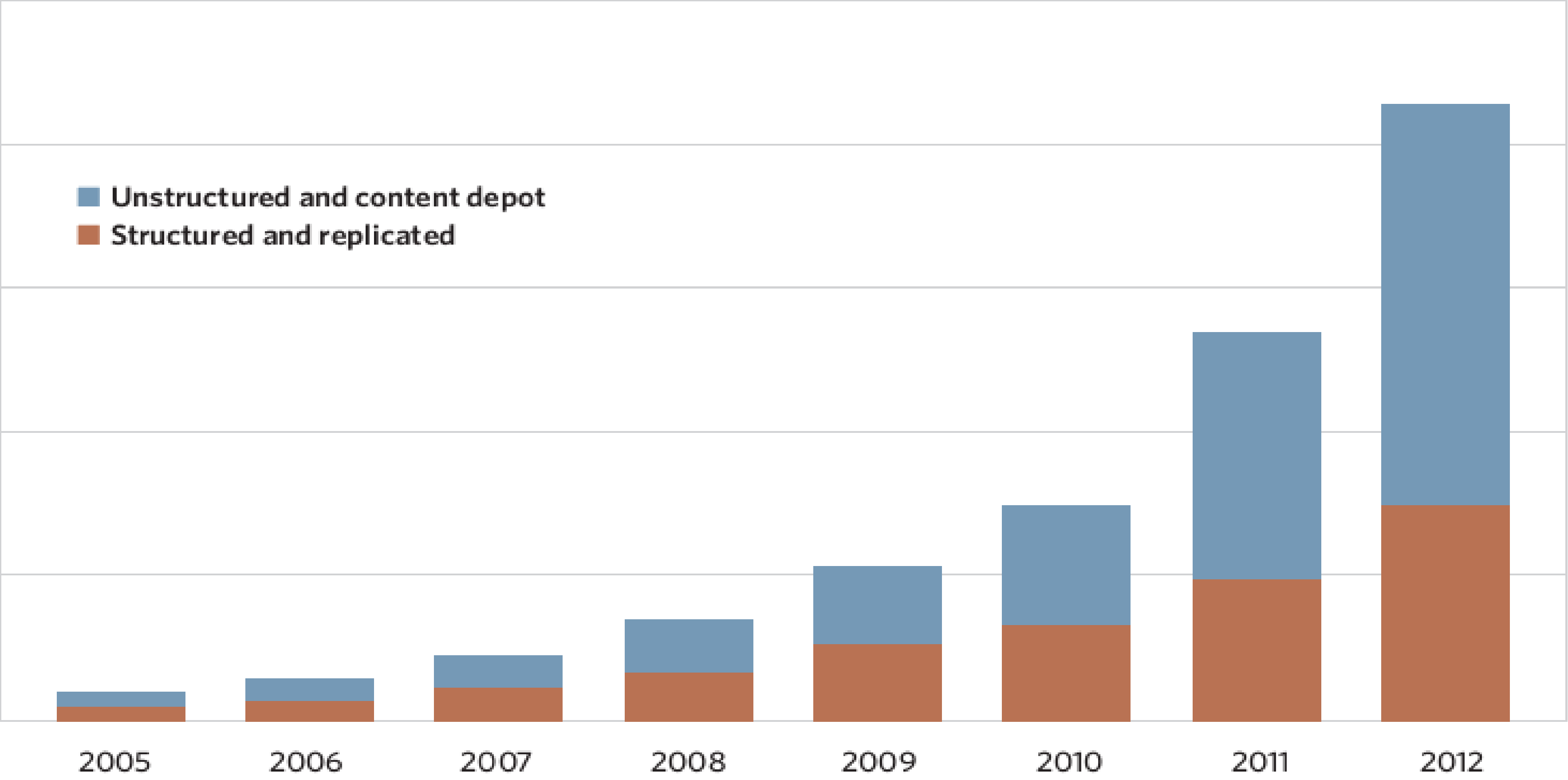
Relational databases advantages

1. Reliability
2. Safety
3. ACID: Atomicity, Consistency, Isolation, Durability
4. Easy, standardized query language (SQL)

Relational databases limitations

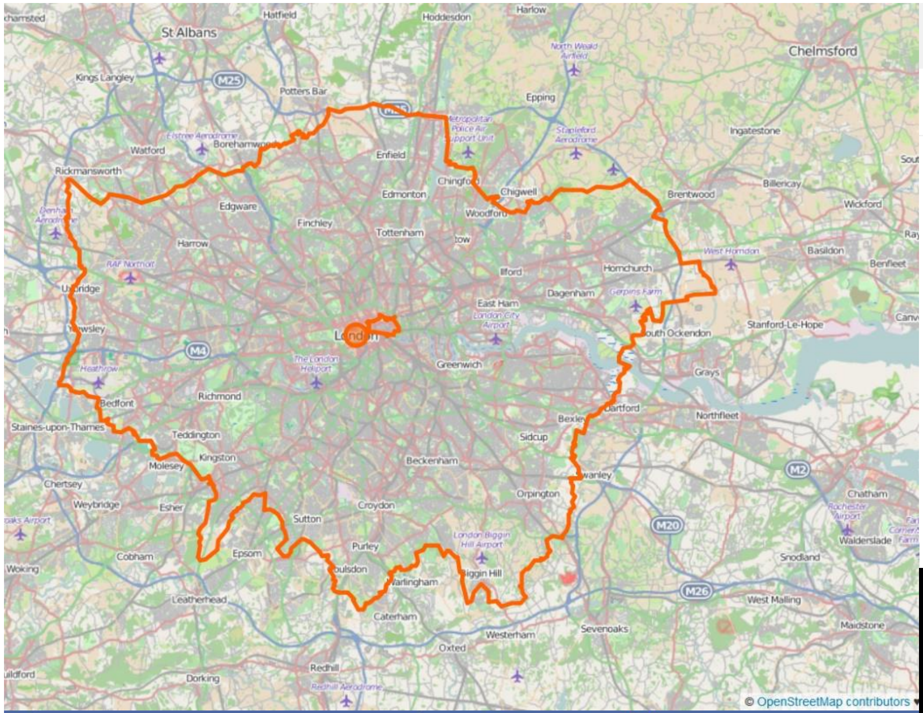
1. Scalability
   - usually costly scale-up approach
   - problems with distributed databases (difficult to join distributed tables)
2. Impedance mismatch
   - different representations in memory and database
3. Complexity
   - data as tables, schema-bound
4. Query language (SQL) only for structured data
5. Dealing with semi-/un- structured data

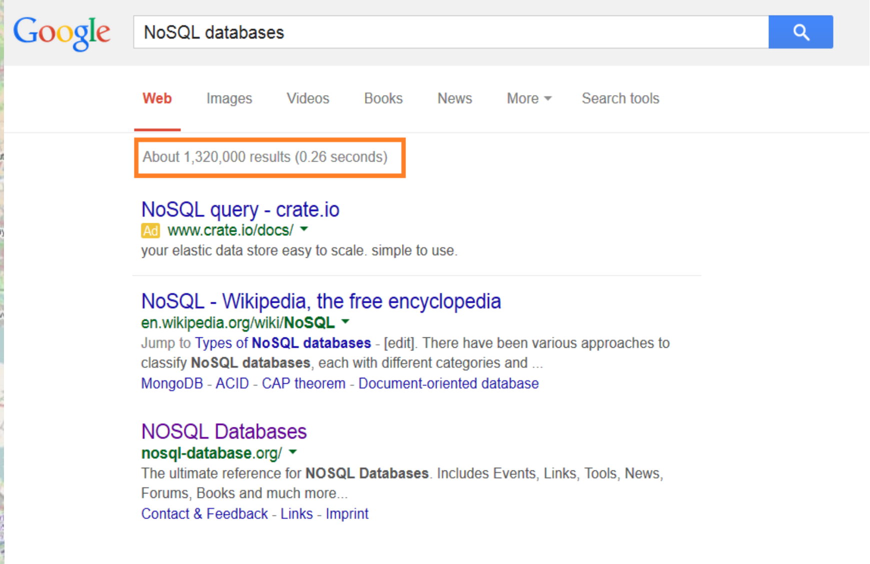
Unstructured data explosition



SOURCE: IDC DIGITAL UNIVERSE 2009: WHITE PAPER, SPONSORED BY EMC, 2009.

Semi-structured data examples



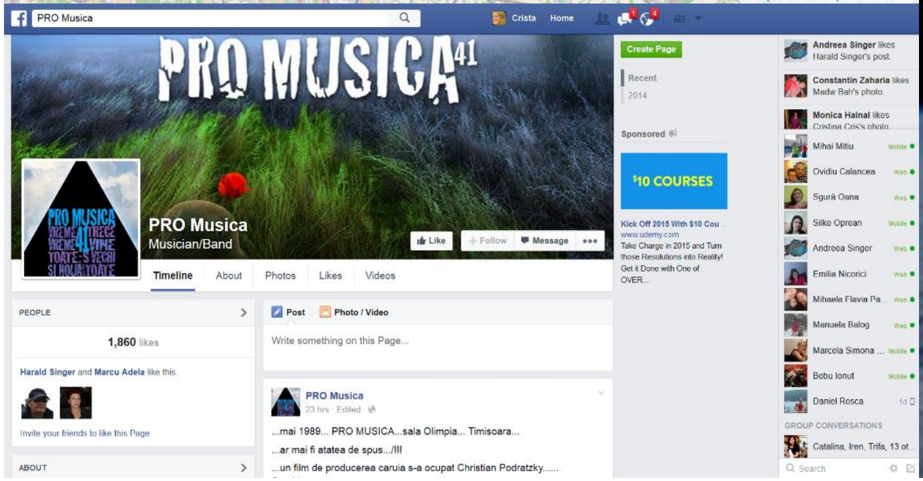


About 1,320,000 results (0.26 seconds)

NoSQL query - crate.io
Ad www.crate.io/docs/
your elastic data store easy to scale. simple to use.

NoSQL - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/NoSQL
Jump to Types of NoSQL databases - [edit]. There have been various approaches to classify NoSQL databases, each with different categories and ...
MongoDB - ACID - CAP theorem - Document-oriented database

NOSQL Databases
nosql-database.org/
The ultimate reference for NOSQL Databases. Includes Events, Links, Tools, News, Forums, Books and much more...
Contact & Feedback - Links - Imprint



PRO Musica
Musician/Band

1,860 likes

Harald Singer and Marcu Adela like this.

PRO Musica
23 hrs · Edited

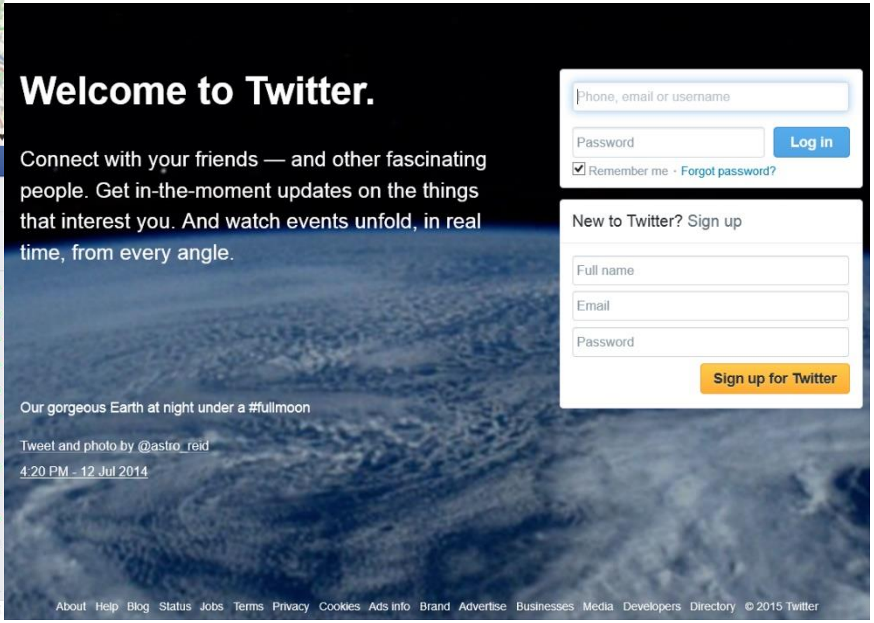
...mai 1989... PRO MUSICA...sala Olimpia... Timisoara...
...ar mai fi atatea de spus.../!!!
...un film de producerea caruia s-a ocupat Christian Podratzky......

Kick Off 2015 With $10 Cou...
www.udemy.com
Take Charge in 2015 and Turn those Resolutions into Reality! Get it Done with One of OVER...

Andreea Singer likes Harald Singer's post.
Constantin Zaharia likes Madw Bah's photo.
Monica Hainal likes Cristina Cris's photo.

Mihai Mitiu	Mobile ●
Ovidiu Calancea	Web ●
Sgură Oana	Web ●
Silke Oprean	Mobile ●
Andreea Singer	Web ●
Emilia Nicorici	Web ●
Mihaela Flavia Pa...	Web ●
Manuela Balog	Web ●
Marcela Simona ...	Mobile ●
Bobu Ionut	Mobile ●
Daniel Rosca	1d

Catalina, Iren, Trifa, 13 ot...



Phone, email or username
Password
Log in
☑ Remember me · Forgot password?
New to Twitter? Sign up
Full name
Email
Password
Sign up for Twitter
About Help Blog Status Jobs Terms Privacy Cookies Ads info Brand Advertise Businesses Media Developers Directory © 2015 Twitter

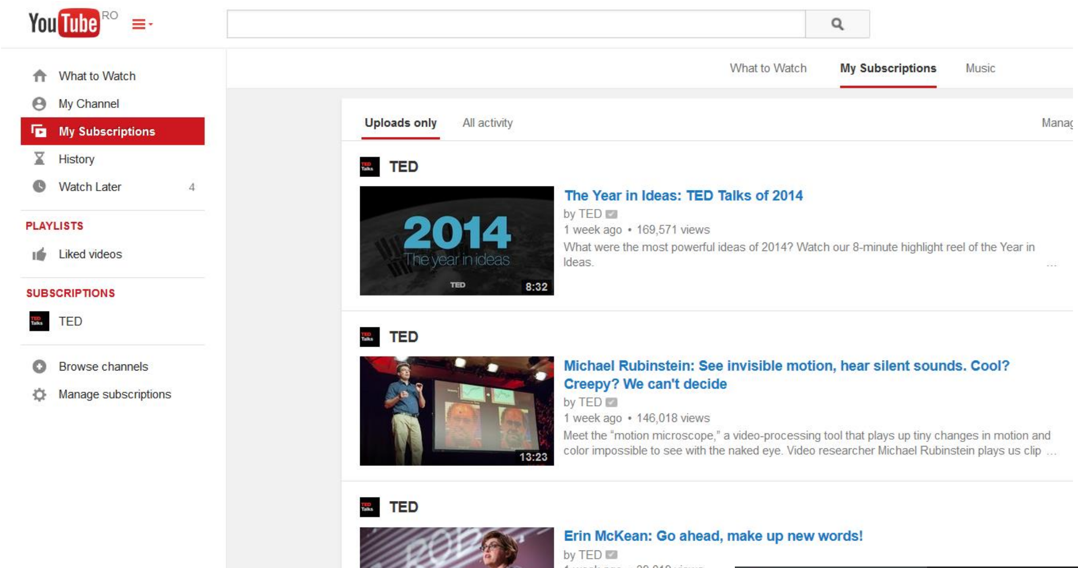
Welcome to Twitter.

Connect with your friends — and other fascinating people. Get in-the-moment updates on the things that interest you. And watch events unfold, in real time, from every angle.

Our gorgeous Earth at night under a #fullmoon

Tweet and photo by @astro_reid
4:20 PM - 12 Jul 2014

Un-structured data examples



YouTube RO

What to Watch | My Subscriptions | Music

- What to Watch
- My Channel
- My Subscriptions
- History
- Watch Later 4

PLAYLISTS

- Liked videos

SUBSCRIPTIONS

- TED
- Browse channels
- Manage subscriptions

Uploads only | All activity | Manag

TED

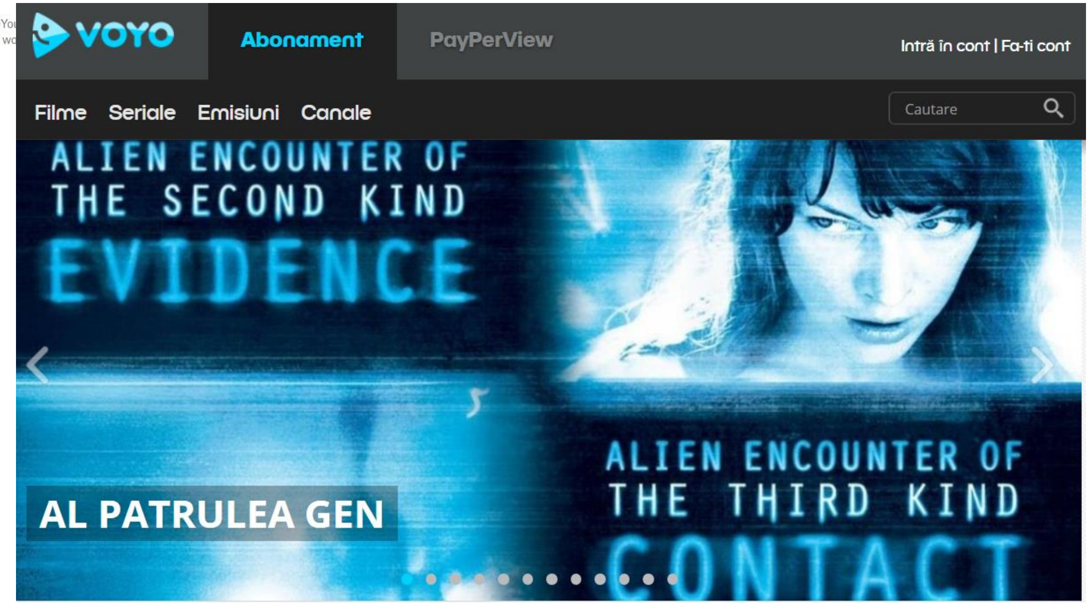
The Year in Ideas: TED Talks of 2014
by TED
1 week ago • 169,571 views
What were the most powerful ideas of 2014? Watch our 8-minute highlight reel of the Year in Ideas.
8:32

TED

Michael Rubinstein: See invisible motion, hear silent sounds. Cool? Creepy? We can't decide
by TED
1 week ago • 146,018 views
Meet the "motion microscope," a video-processing tool that plays up tiny changes in motion and color impossible to see with the naked eye. Video researcher Michael Rubinstein plays us clip ...
13:23

TED

Erin McKean: Go ahead, make up new words!
by TED
1 week ago • 39,019 views
In this fun, short talk from TEDYo
— her audience to create new wo
6:57



VOYO
Abonament
PayPerView
Intră în cont | Fa-ti cont
Filme
Seriale
Emisiuni
Canale
Cautare
ALIEN ENCOUNTER OF THE SECOND KIND
EVIDENCE
ALIEN ENCOUNTER OF THE THIRD KIND
CONTACT
AL PATRULEA GEN

What is Big Data?

From a technology perspective, Big Data is defined as those data sets whose size, type, and speed-of-creation make them impractical to process and analyse with traditional database technologies and related tools in a **cost-** or **time-effective way**.

Source: wikibon.org

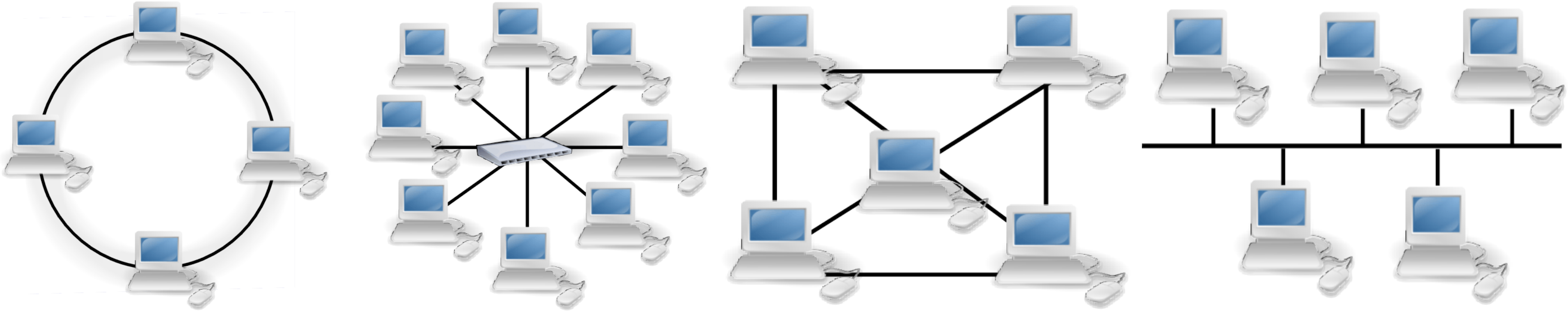
4 Vs of BigData

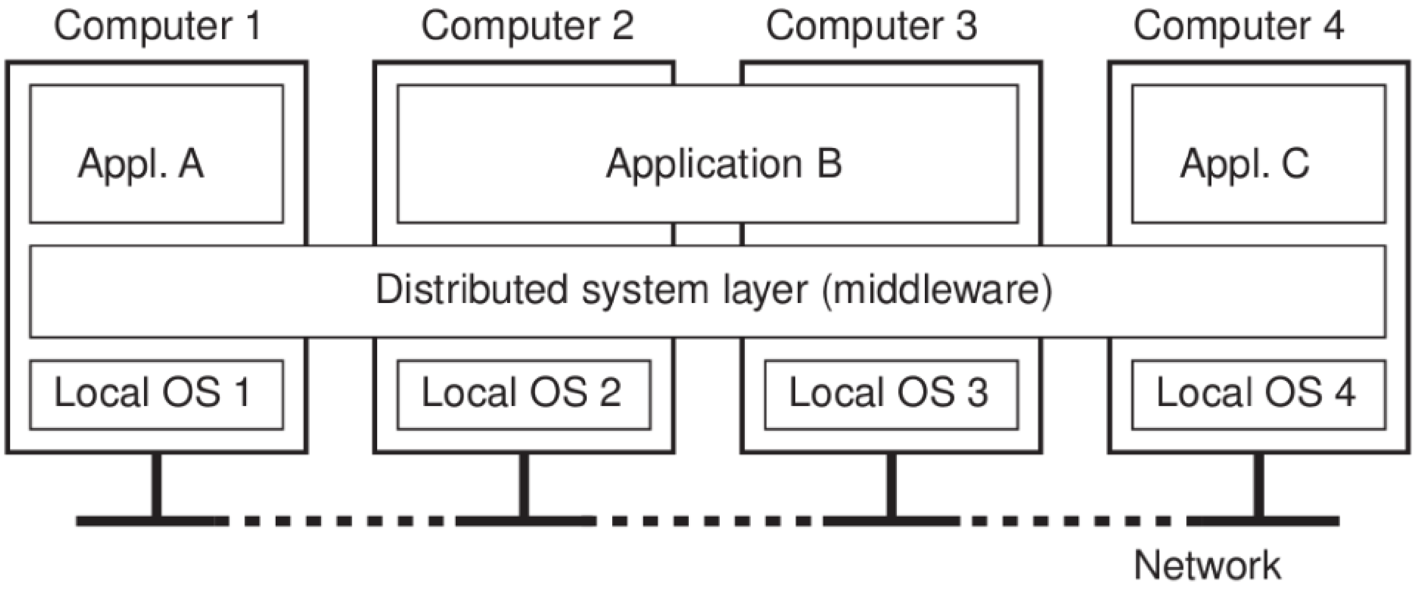
- Volume:
  - 12TB of tweets / day => product sentiment analysis
- Velocity:
  - fraud detection
  - predict customer churn faster (anlyse 500 million daily calls in real-time)
- Variety:
  - exploit 80% growth of un&semi-structured data for customer satisfaction
- Variability / Veracity:
  - variance in meaning (1 in 3 business leader don‘t trust the information they use to make decisions)

Source: Brian Hopkins (Forrester)

Distributed systems

Distributed system is a “**collection of independent computers that appear to the users of the system as a single computer**” (Tanenbaum)





Requirements of distributed applications

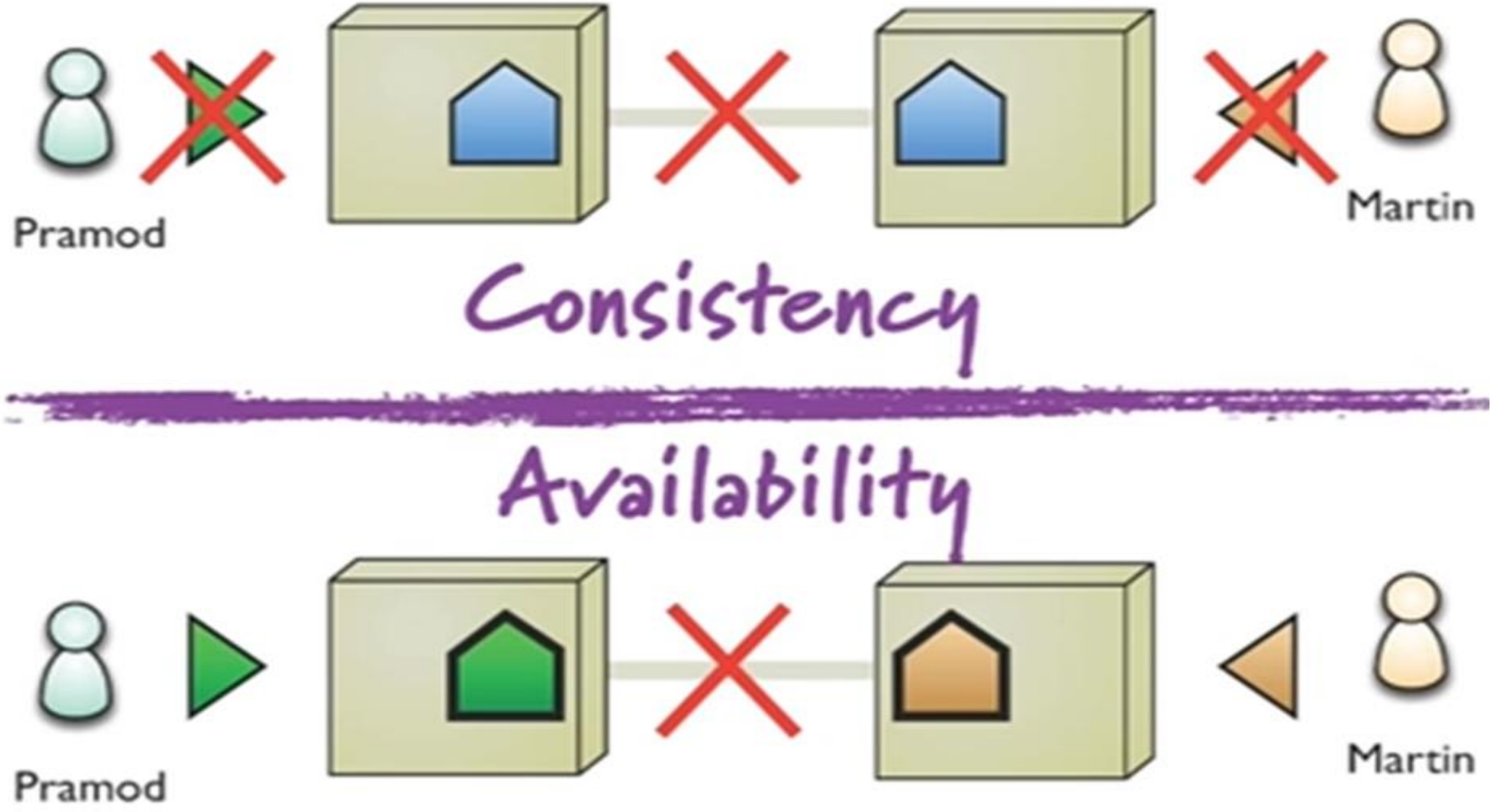
1. Consistency: all nodes ‘see’ the same data at the same time
2. Availability: guarantee that every request receives a response about whether it succeeded or failed
3. Partition tolerance: the system continues to operate despite arbitrary message loss or failure of part of the system

Brewer’s CAP Theorem

1. **Consistency:** all nodes ‘see’ the same data at the same time
2. **Availability:** guarantee that every request receives a response about whether it succeeded or failed
3. **Partition tolerance:** the system continues to operate despite arbitrary message loss or failure of part of the system

Brewer’s (CAP) Theorem: it is impossible for a distributed computer system to simultaneously provide all three requirements.

Consistency vs. Availability



Source: Martin Fowler

More at https://www.youtube.com/watch?v=ASiU89Gl0F0

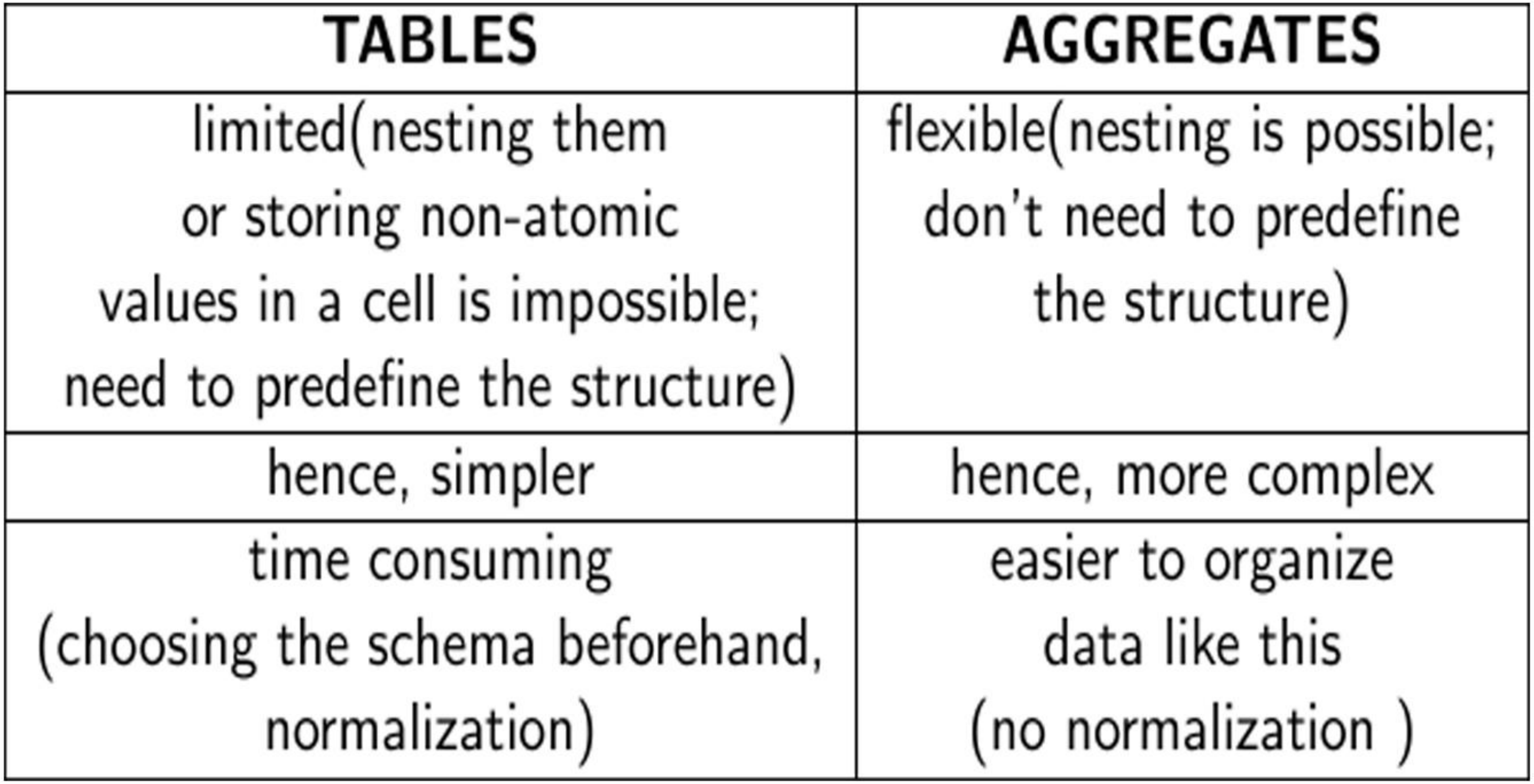
BASE instead of ACID

1. Basic Availability
2. Soft-state
3. Eventual consistency: informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value

NoSQL = Not Only SQL

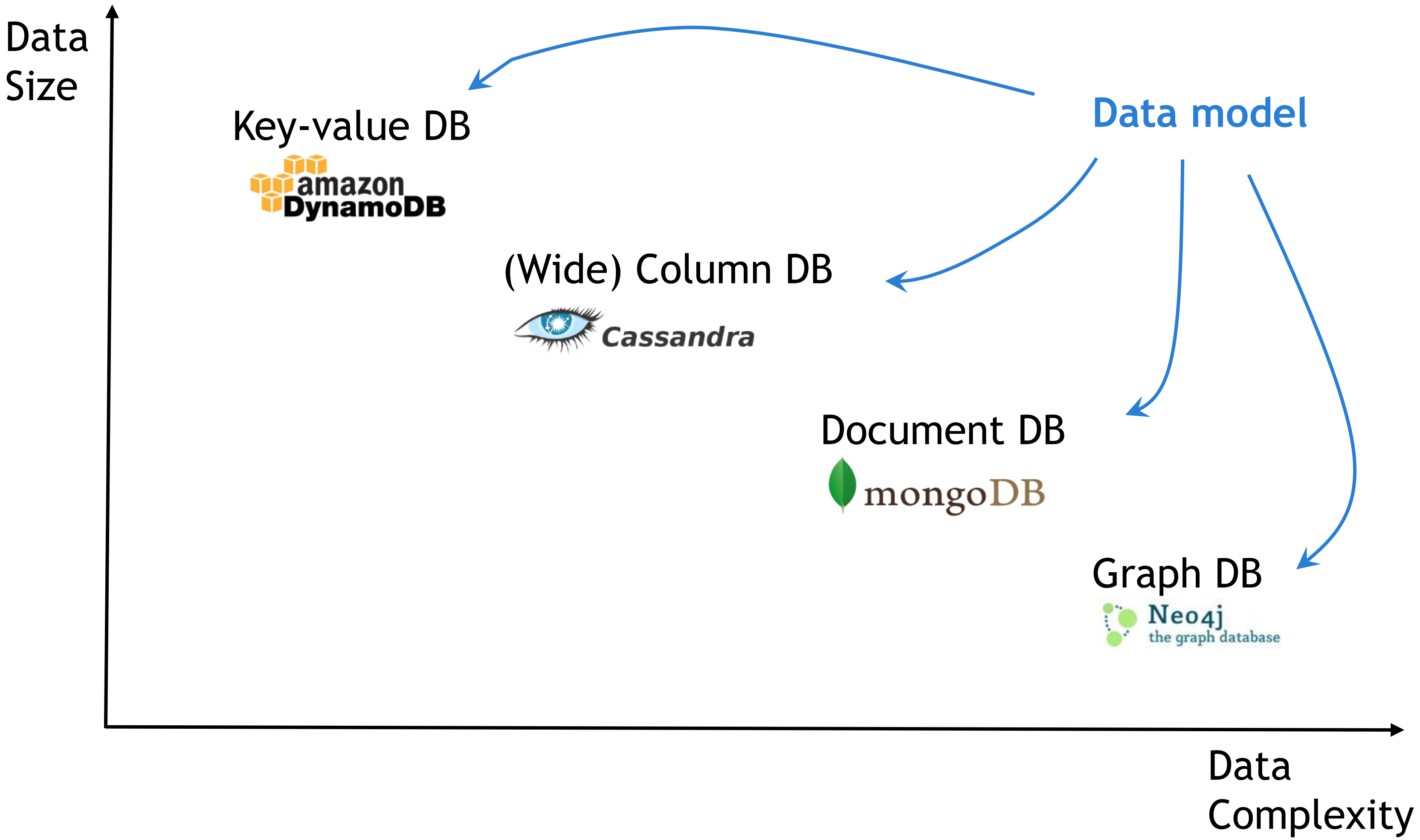
- Not based on relational model, not using SQL
- No fixed schema, new attributes can be added to data items at any time (schema-less)
- Designed for distributed, large clusters (scale-out), including support for distributed processing (MapReduce)
- Deliver eventual consistency
- Data-model specific query languages

Aggregate Data Model



TABLES	AGGREGATES
limited(nesting them or storing non-atomic values in a cell is impossible; need to predefine the structure)	flexible(nesting is possible; don't need to predefine the structure)
hence, simpler	hence, more complex
time consuming (choosing the schema beforehand, normalization)	easier to organize data like this (no normalization)

NoSQL: classification and usage pattern



Data Size

Key-value DB

amazon DynamoDB

(Wide) Column DB

Cassandra

Document DB

mongoDB

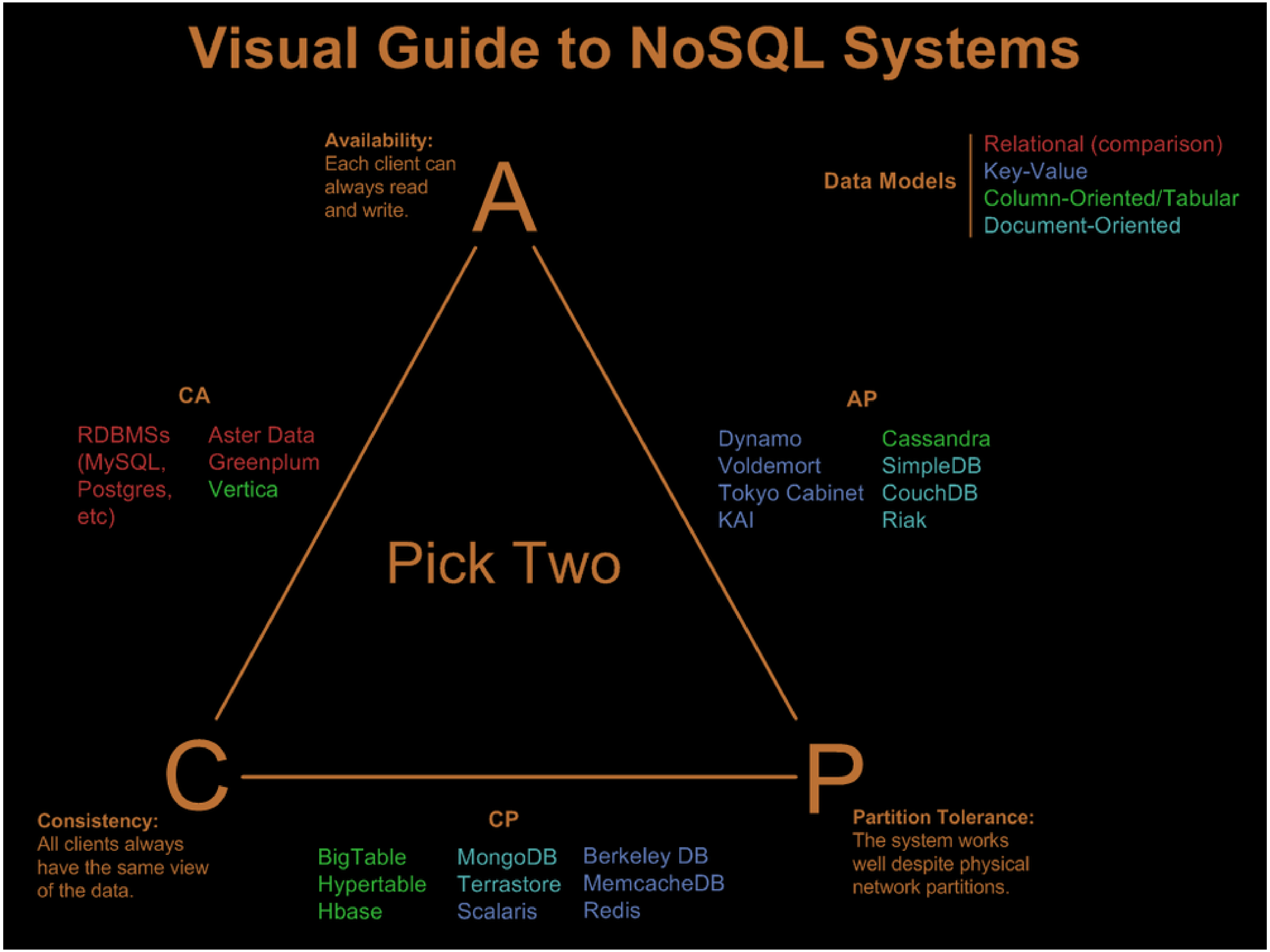
Graph DB

Neo4j the graph database

Data model

Data Complexity

Visual guide to NoSQL systems



Visual Guide to NoSQL Systems

Availability:
Each client can always read and write.

A

Data Models

- Relational (comparison)
- Key-Value
- Column-Oriented/Tabular
- Document-Oriented

CA

RDBMSs (MySQL, Postgres, etc)

Aster Data
Greenplum
Vertica

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI

Cassandra
SimpleDB
CouchDB
Riak

Pick Two

C

P

Consistency:
All clients always have the same view of the data.

CP

BigTable
Hypertable
Hbase

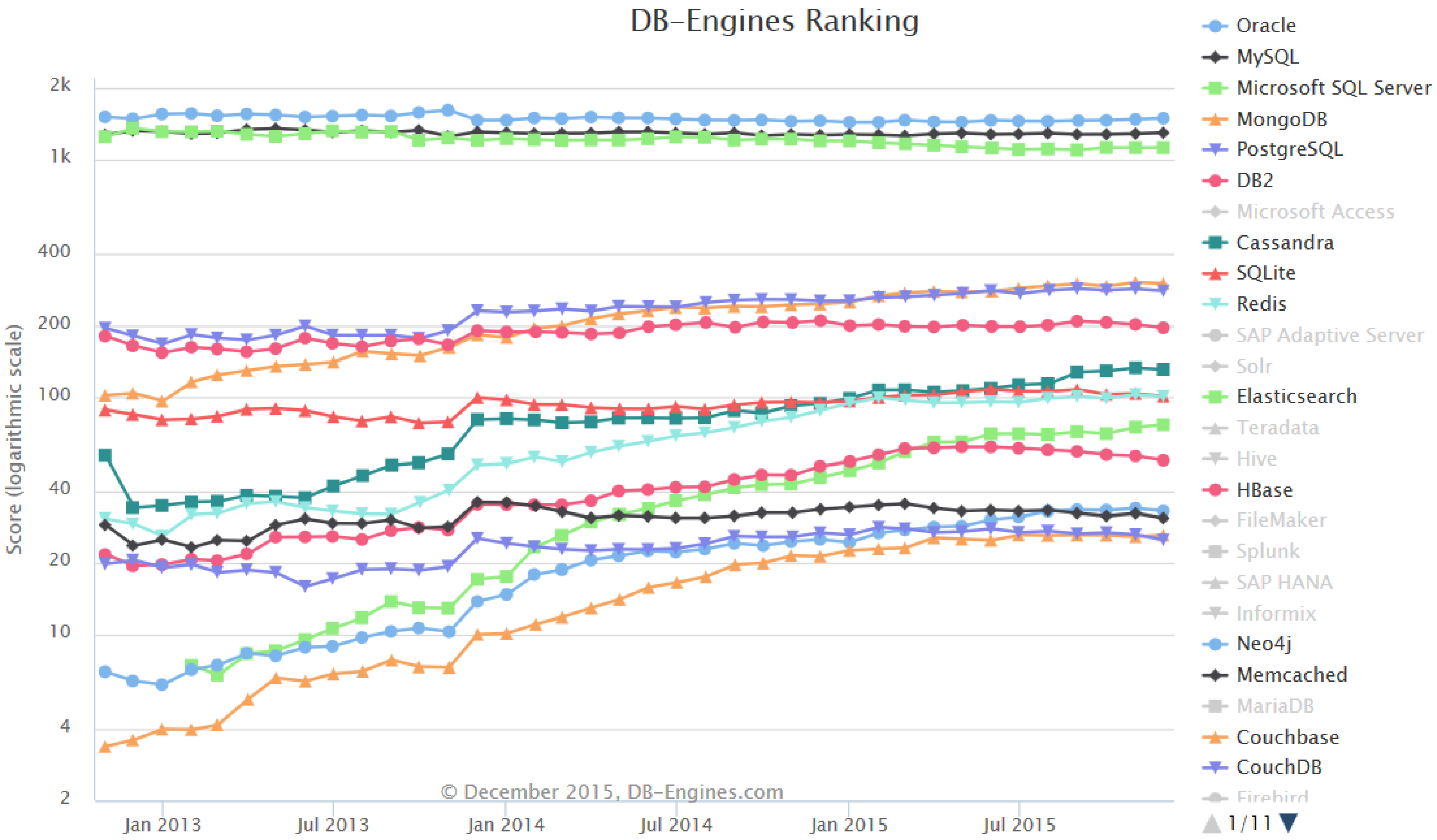
MongoDB
Terrastore
Scalaris

Berkeley DB
MemcacheDB
Redis

Partition Tolerance:
The system works well despite physical network partitions.

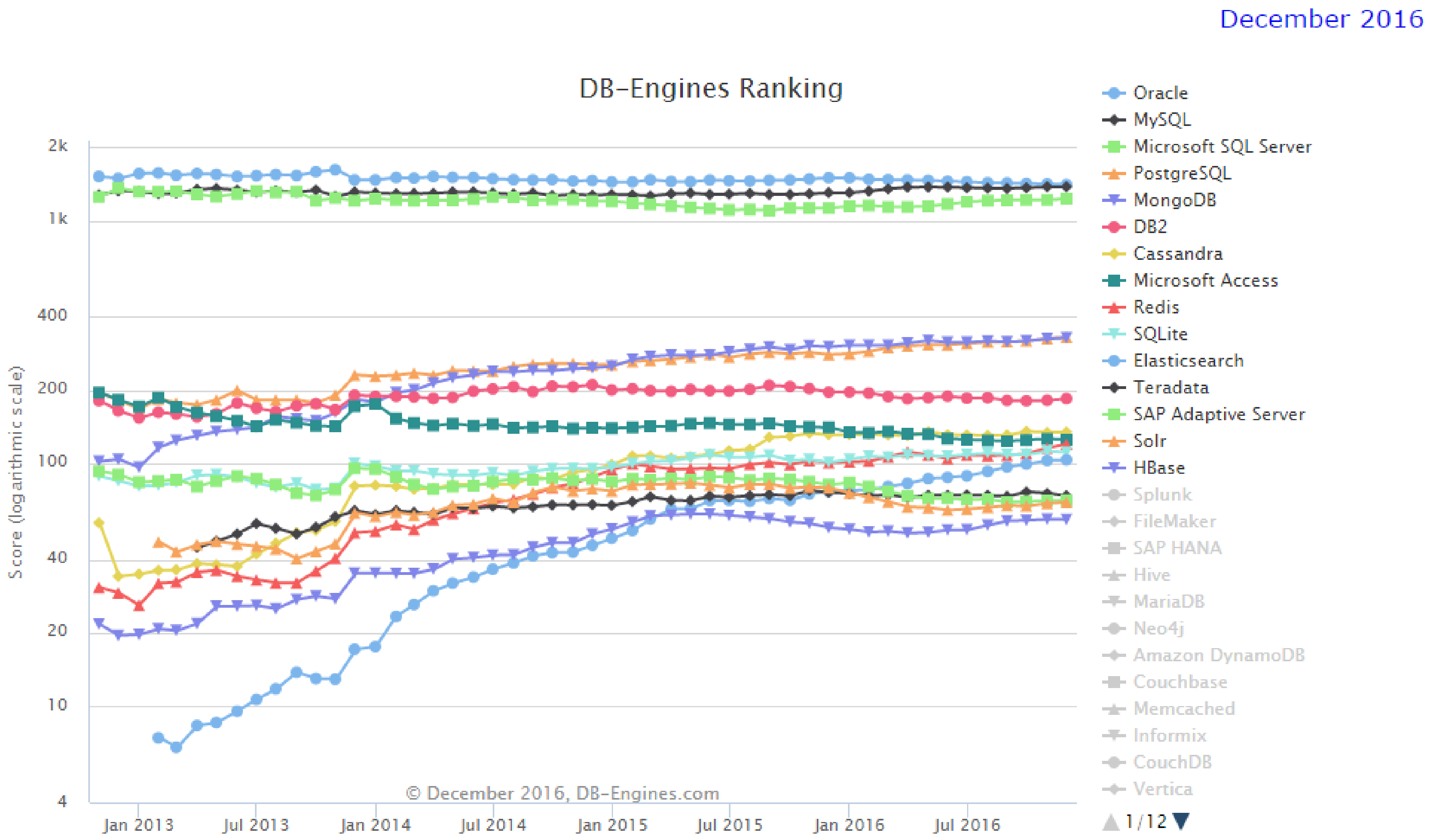
Trends

December 2015



Source: http://db-engines.com/en/ranking_trend

Trends



Source: http://db-engines.com/en/ranking_trend

NoSQL brief history

- Carlo Strozzi (1998) – to name his lightweight, open-source, relational database, but without SQL interface

- Johan Oskarsson and Eric Evans (2009) - meetup on distributed structured data storage; they used `#nosql` as Twitter hashtag for this

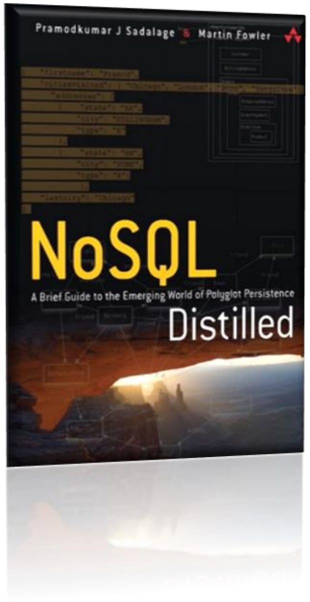
- Google Bigtable (2006)

- Amazon DynamoDB (2007)

- ....

- Used on large scale at Amazon, Facebook, Google, Twitter etc.

Bibliography


Pramodkumar J Sadalage & Martin Fowler
NoSQL
A Brief Guide to the Emerging World of Polyglot Persistence
Distilled

Pramod J. Sadalage, Martin Fowler. NoSQL Distilled. Addison Wesley, 2012

Watch M. Fowler @ NoSQL matters Conference in Cologne, Germany 2013

http://martinfowler.com/nosql.html

1. Lorenzo Alberton. NoSQL Databases: why, what and when
2. Yousof Alsatom. Introduction to NoSQL
3. Wikipedia
4. Introduction to NoSQL on w3resource.com
5. http://www.nosql-database.org/ - list, by data model, of NoSQL databases

Key-value database

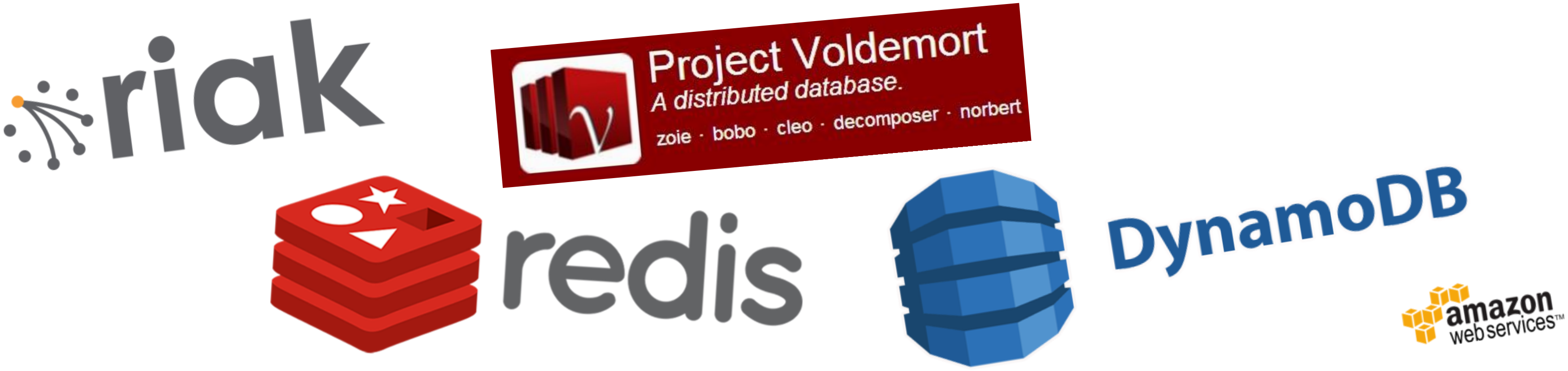
Simplest data model:

Key → **Value**

What Is a Key-Value Store ?

- The **simplest** NoSQL data stores
- Operations: **get** the value for the key, **put** a value for a key, or **delete** a key
- **The value** is a blob that the data store **just stores** without caring or knowing what’s inside.

- There are many key-value databases:

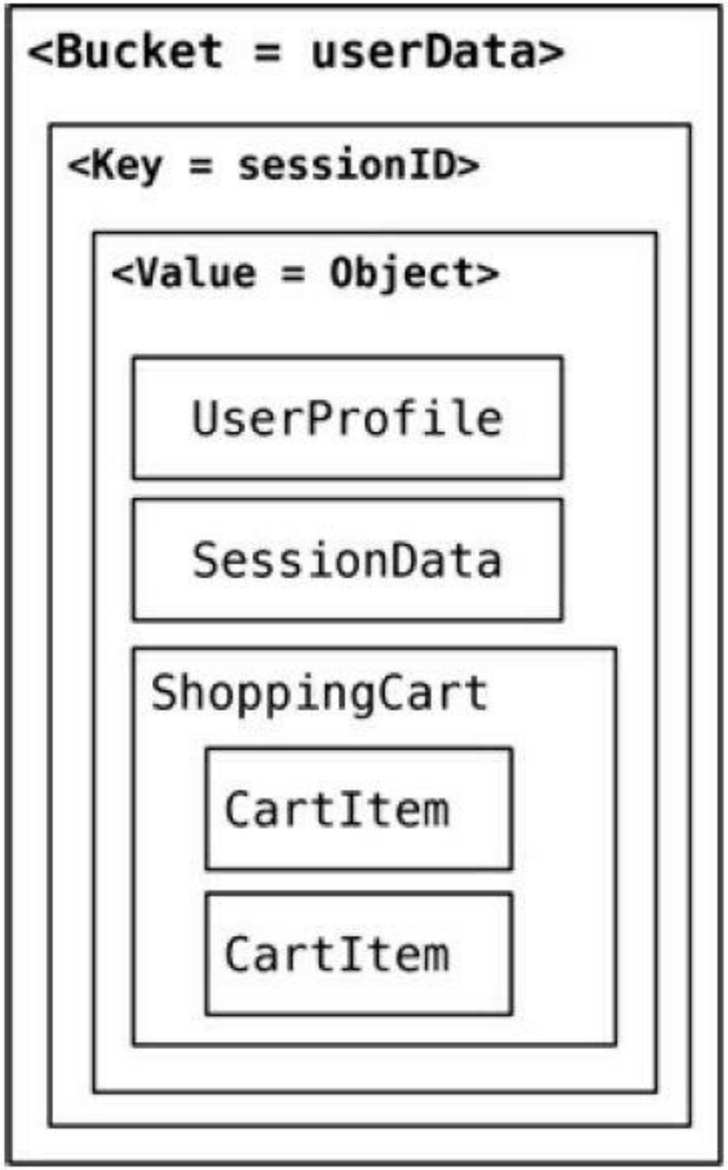


What Is a Key-Value Store ?

- Example:

  riak

- Riak lets us **store keys into buckets.**
- To store user session data, shopping cart information, user preferences in Riak, we could just store all of them in the **same bucket** with a **single key** and **single value** for all of these objects.



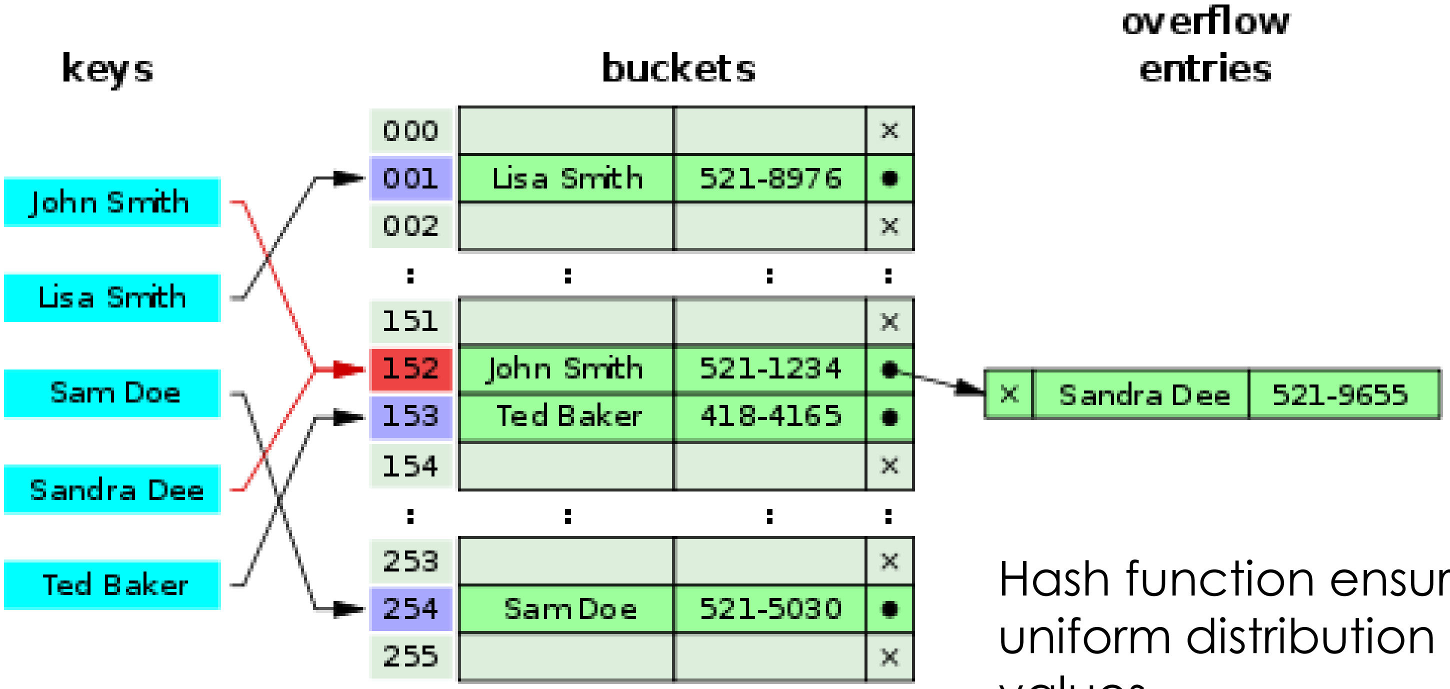
Characteristics

- Scalability

- Efficiency – able to handle huge number of inserts/updates/deletes per second

- Key design – in a way to allow segmentation of data in *domain buckets*

Hash tables



Hash function ensures a uniform distribution of hash values.

Hash collision by separate chaining with head records in the bucket array.

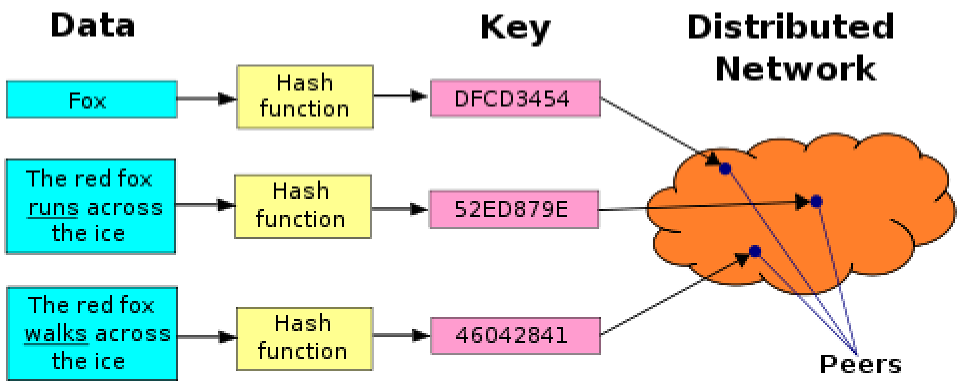
Source: http://en.wikipedia.org/wiki/Hash_table

Distributed hash tables

- Hash-function SHA-1, MD5, etc.
- Keyspace partitioning
  - **Distance function** between two keys k1 and k2; each node is assigned a single key called its Identifier (ID)
  - **Consistent hashing** has the essential property that removal or addition of one node changes only the set of keys owned by the nodes with adjacent IDs, and leaves all other nodes unaffected
  - DHT protocols: CAN, Tapestry, Pastry, Chord, Apache Cassandra etc.
- Overlay network

Topology ⇧

  - Each node maintains a set of links to other nodes (its *neighbors* or routing table)
  - For any key k, each node either has a node ID that owns k or has a link to a node whose node ID is *closer* to k in terms of the keyspace distance
  - Key-based routing algorithm



Source: http://en.wikipedia.org/wiki/distributed_Hash_table

Key-Value Store Features

1. Consistency:

is applicable only for operations on a single key, since these operations are either a get,put, or delete

In **distributed key-value store** implementations like Riak:

- Since the value may have already been replicated to other nodes,
- Riak has **two ways of resolving** update conflicts: either the ***newest write wins*** and ***older writes loose***,
- Or ***both (all) values are returned*** allowing the client to resolve the conflict.

Key-Value Store Features

2. Transactions:

Different products of the key-value store kind have different specifications of transactions.

Many **data stores do** implement transactions in **different ways**.

- Ex: We have a ***Riak cluster*** with a replication factor of 5 and we supply the W value of 3.
- When writing, the write is reported as successful only when it is written and reported as a ***success*** on ***at least three of the nodes***.
- So, the cluster can ***tolerate N - W = 2*** nodes being down for write operations.

Key-Value Store Features

3. Query Features:

All key-value stores can query by the key

What if we don't know the key ?

- Some key-value databases providing the ability to search inside the value.

Can the key be generated using some algorithm?

- Storing session data (with the session ID as the key), shopping cart data, user profiles, and so on.
- Riak for example provides an HTTP-based interface, so that all operations can be performed from the web browser or on the command line.

Key-Value Store Features

4. Structure of Data *and* Scaling

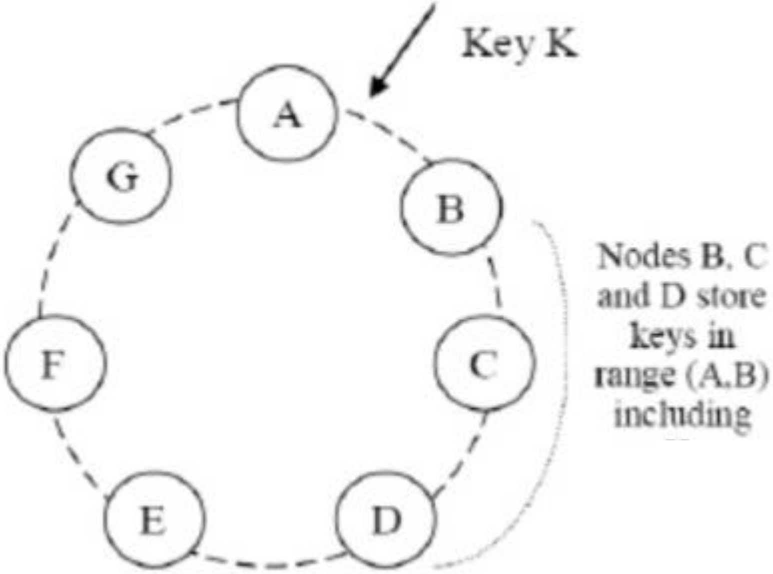
- Key-value databases **don’t care what is stored** in the value part of the key-value pair.
- The ***value can be*** a blob, text, JSON, XML, and so on.

- Many key-value ***stores scale by using sharding***.
- With sharding, the ***value of the key*** determines on which node the key is stored.
- Ex: if the key is f4b19d79587d, which **starts with an f**, it will be sent to **different node** than the key ad9c7a396542. This kind of sharding setup can **increase performance** as **more nodes** are added to the cluster.

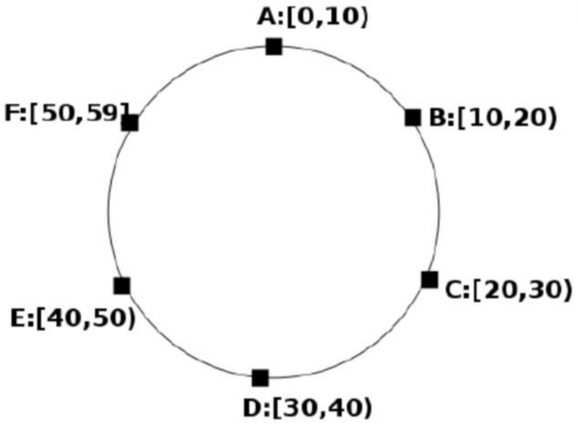
Amazon DynamoDB

- G. DeCandia of Amazon paper at SOSP 2007
- Proprietary
- Offered as a Service through Amazon Web Services: http://aws.amazon.com/dynamodb/
- Targets to achieve high availability and scalability, with less consistency
- Data is partitioned and replicated using **consistent hashing**
- MD5 algorithm as hash function
- Lately, it supports document data model as well

Amazon DynamoDB



Handling errors - Sloppy quorum
Read/write operations are performed on the first N healthy nodes from the preference list, which may not always be the first N nodes encountered while walking the consistent hashing ring



N=3

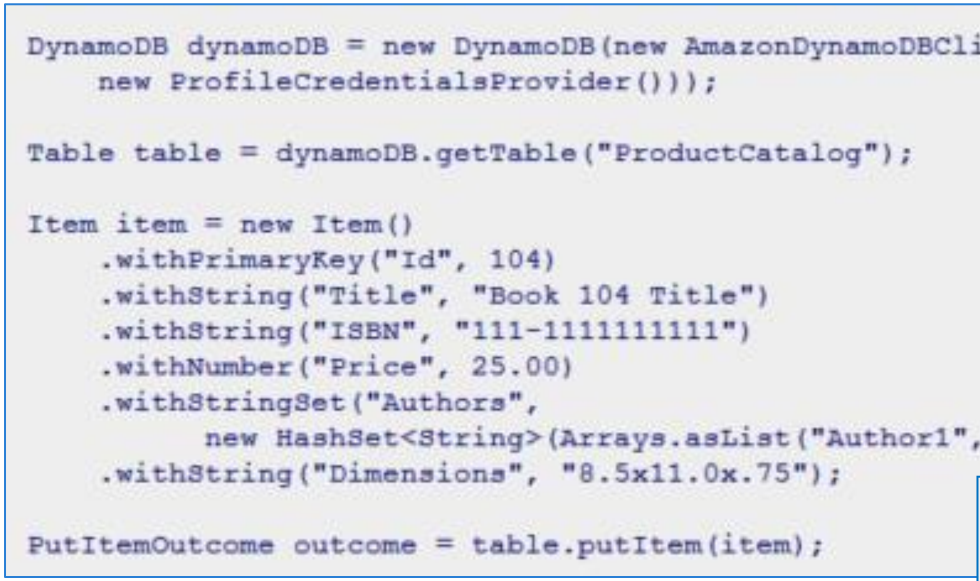
Hashwert	Knoten	Replika
3	A	B,C
12	B	C,D
19	B	C,D
20	C	D,E
37	D	E,F
40	E	F,A
54	F	A,B

Source : http://de.wikipedia.org/wiki/Amazon_Dynamo

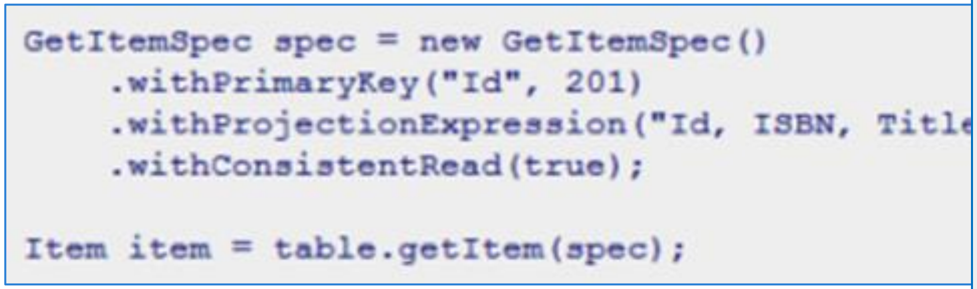
Amazon DynamoDB - Usage example

Using Amazon Web Services (AWS) SDK

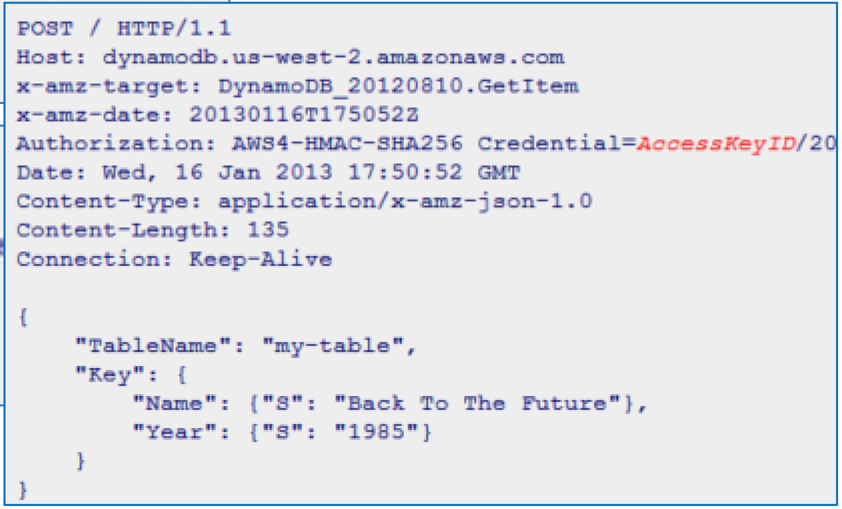
- It uses JSON as transport data model; it stores data differently on disk.
- APIs for Java, .NET, PHP, Python
- Operations over HTTP using POST request method



```
DynamoDB dynamoDB = new DynamoDB(new AmazonDynamoDBClient(
    new ProfileCredentialsProvider()));

Table table = dynamoDB.getTable("ProductCatalog");

Item item = new Item()
    .withPrimaryKey("Id", 104)
    .withString("Title", "Book 104 Title")
    .withString("ISBN", "111-1111111111")
    .withNumber("Price", 25.00)
    .withStringSet("Authors",
          new HashSet<String>(Arrays.asList("Author1","Author2")))
    .withString("Dimensions", "8.5x11.0x.75");

PutItemOutcome outcome = table.putItem(item);
```



```
GetItemSpec spec = new GetItemSpec()
    .withPrimaryKey("Id", 201)
    .withProjectionExpression("Id, ISBN, Title
    .withConsistentRead(true);

Item item = table.getItem(spec);
```



```
POST / HTTP/1.1
Host: dynamodb.us-west-2.amazonaws.com
x-amz-target: DynamoDB_20120810.GetItem
x-amz-date: 20130116T175052Z
Authorization: AWS4-HMAC-SHA256 Credential=AccessKeyID/20
Date: Wed, 16 Jan 2013 17:50:52 GMT
Content-Type: application/x-amz-json-1.0
Content-Length: 135
Connection: Keep-Alive

{
    "TableName": "my-table",
    "Key": {
        "Name": {"S": "Back To The Future"},
        "Year": {"S": "1985"}
    }
}
```

Source: http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/JavaDocumentAPIItemCRUD.html

Comparison to relational model

Consistency

- applicable only for operations on a single key
- eventually consistent model
  - since a value may have already been replicated to other nodes, there are two ways of resolving update conflicts: newest write wins vs. older writes loose, or both (all) values are returned allowing the client to resolve the conflict

Transactions

- Atomic (single-item) level
- Quorum – number of nodes to respond to write (replication factor) API call

Query features

- Query by key
- Search function (Lucene indexes)

Comparison to relational model

Structure of data

- Value can be everything: text (JSON, XML), binary
- Use of *content-type* attribute to specify type

Scaling

- Using sharding
- Tunning scalability using
  - cluster size (number of nodes),
  - write replication factor,
  - number of nodes to respond to read

Suitable Use Cases

1. **Storing Session Information**

everything about the session can be stored by a single PUT request or retrieved using GET.

2. **User Profiles, Preferences**

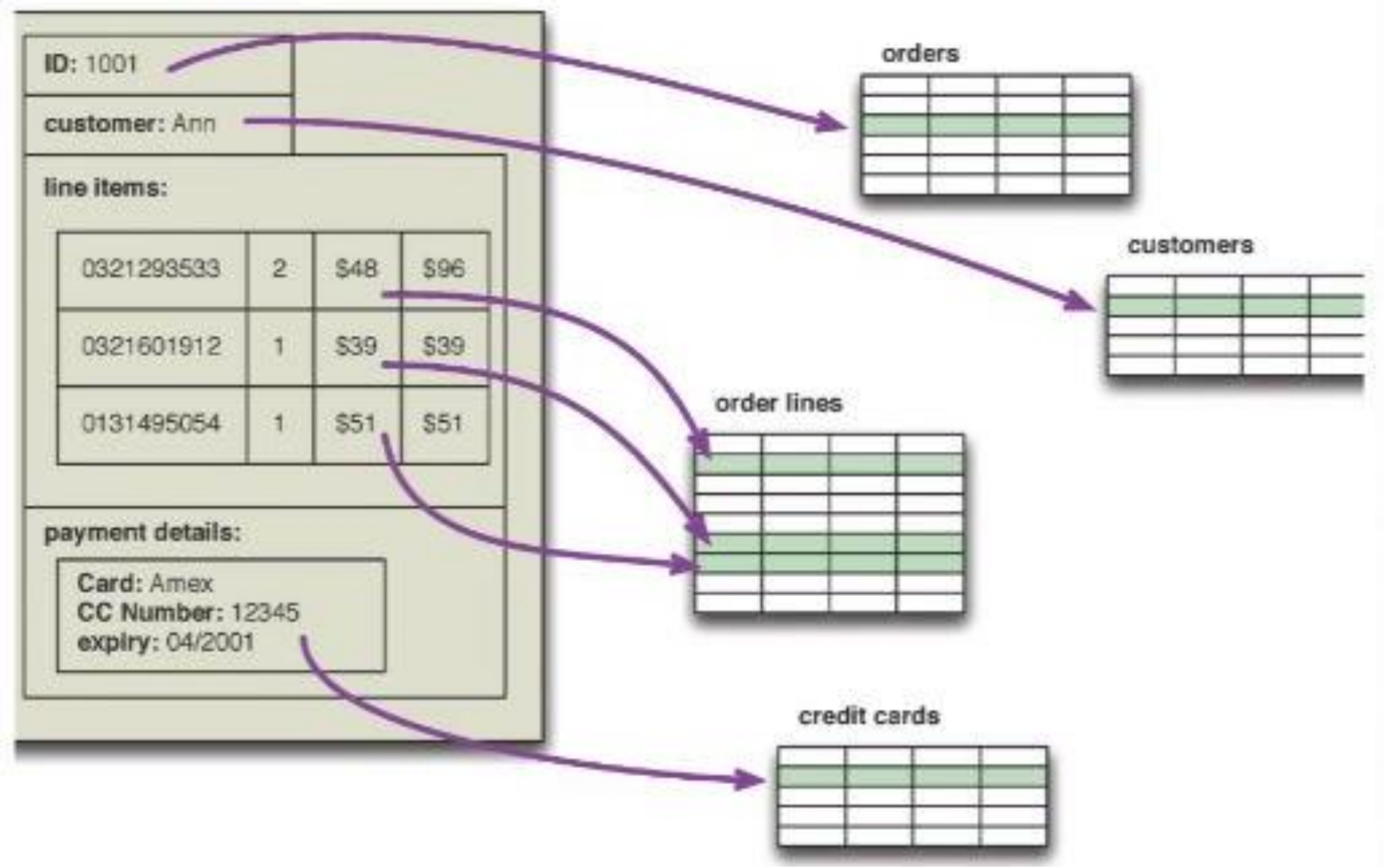
Almost every user has a unique userId, username, or some preferences such as language, etc. This can all be put into an object **so getting** preferences of a user takes ***a single GET operation***.

2. **Shopping Cart Data**

all the shopping information can be put into the value where the key is the userid.

Suitable Use Cases

Example:



ID: 1001

customer: Ann

line items:

0321293533	2	$48	$96
0321601912	1	$39	$39
0131495054	1	$51	$51

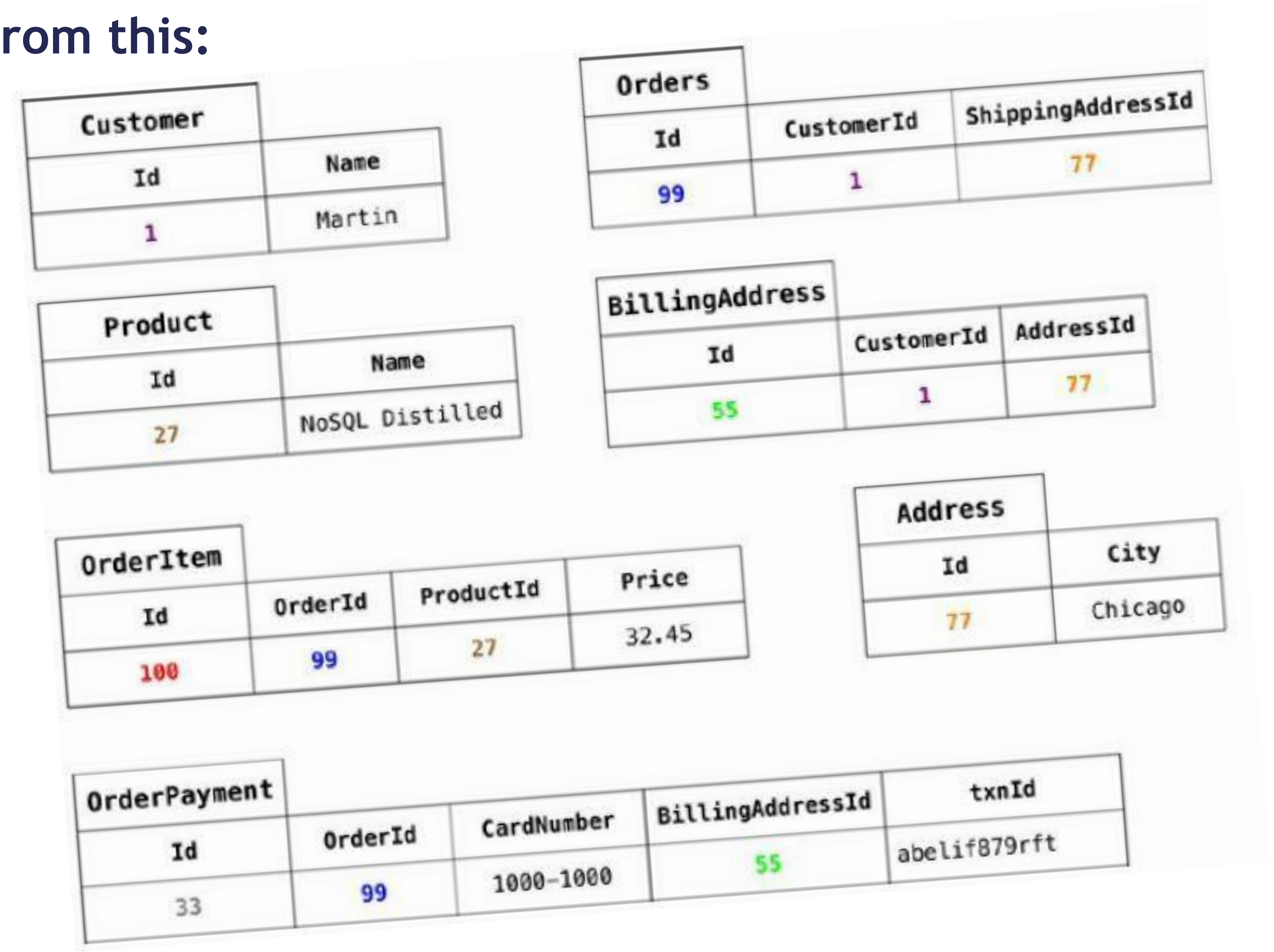
payment details:

Card: Amex
CC Number: 12345
expiry: 04/2001

Source:Martin Fowler https://www.youtube.com/watch?v=ASiU89Gl0F0&t=377s

Suitable Use Cases

From this:



Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

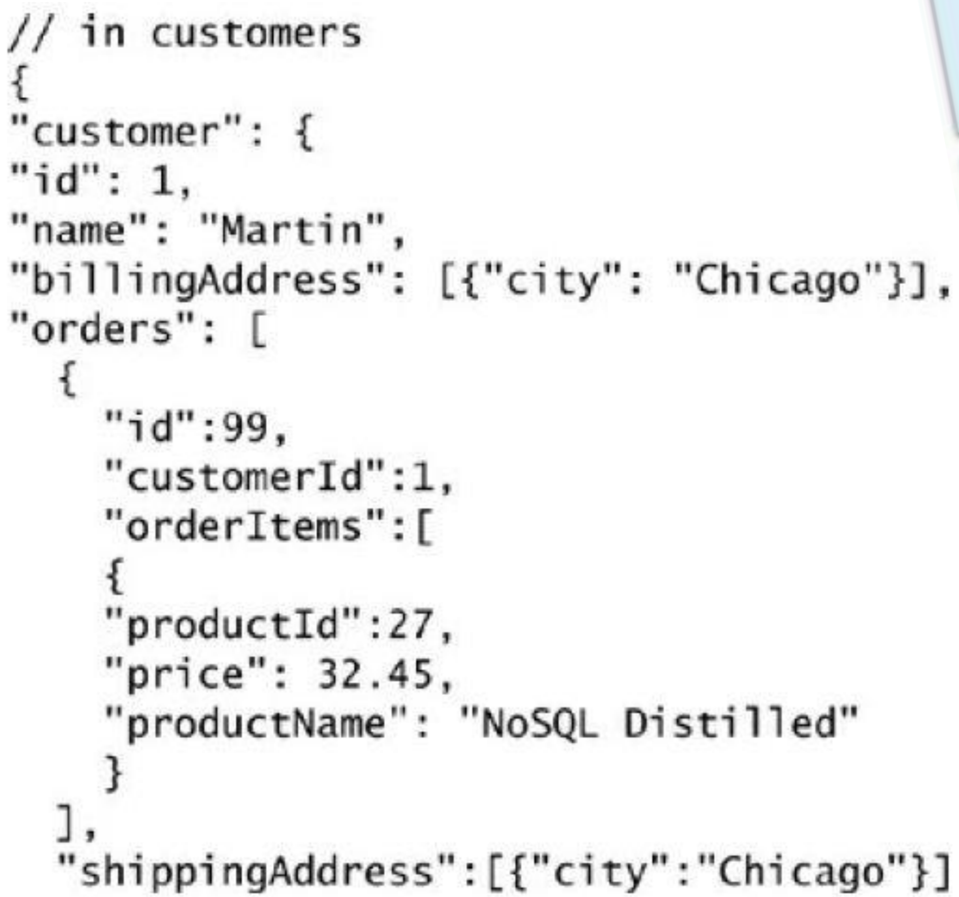
Id	City
77	Chicago

OrderPayment

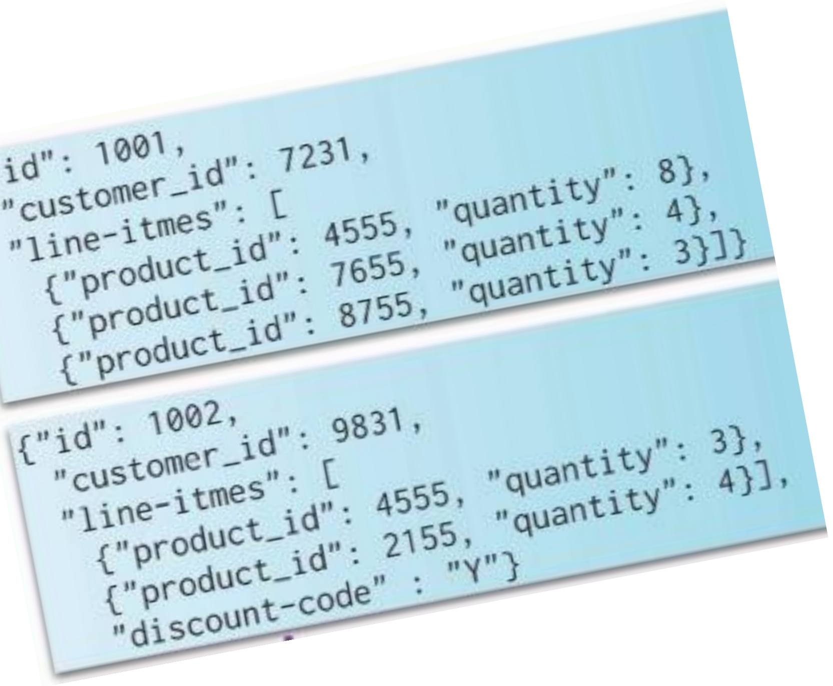
Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

Suitable Use Cases

To this:



```
// in customers
{
"customer": {
"id": 1,
"name": "Martin",
"billingAddress": [{"city": "Chicago"}],
"orders": [
  {
    "id":99,
    "customerId":1,
    "orderItems":[
    {
    "productId":27,
    "price": 32.45,
    "productName": "NoSQL Distilled"
    }
  ],
  "shippingAddress":[{"city":"Chicago"}]
```



```
{"id": 1001,
 "customer_id": 7231,
 "line-itmes": [
  {"product_id": 4555, "quantity": 8},
  {"product_id": 7655, "quantity": 4},
  {"product_id": 8755, "quantity": 3}]}
```

```
{"id": 1002,
 "customer_id": 9831,
 "line-itmes": [
  {"product_id": 4555, "quantity": 3},
  {"product_id": 2155, "quantity": 4}],
  "discount-code" : "Y"}
```

Source:Martin Fowler https://www.youtube.com/watch?v=ASiU89Gl0F0&t=377s

When Not to Use

Relationships among Data - When we have **relationships between different sets of data**, or correlate the data between different sets of keys.

2. Multioperation Transactions - Saving multiple keys and there is a failure to save any one of them, and you want to **revert** or **roll back** the rest of the operations

3. Query by Data - If you need to **search** the keys based on something found **in the value part** of the key-value pairs

4. Operations by Sets - Since operations are **limited to one key** at a time, there is **no way** to operate upon **multiple keys.**

Other key-value databases

MS Azure
Table Storage

ORACLE
BERKELEY DB

riak

MEMCACHED

redis

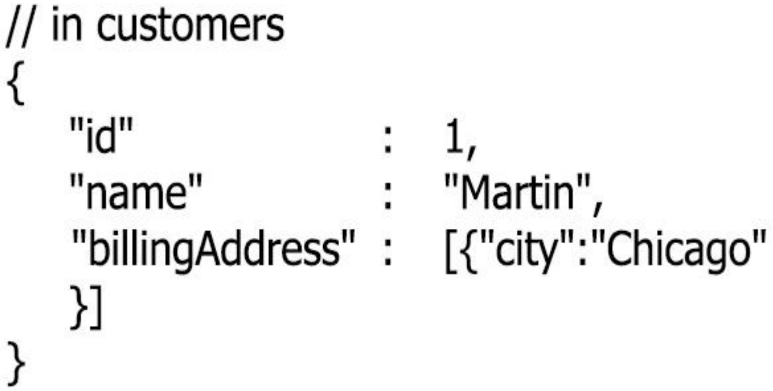
Document database

Document database

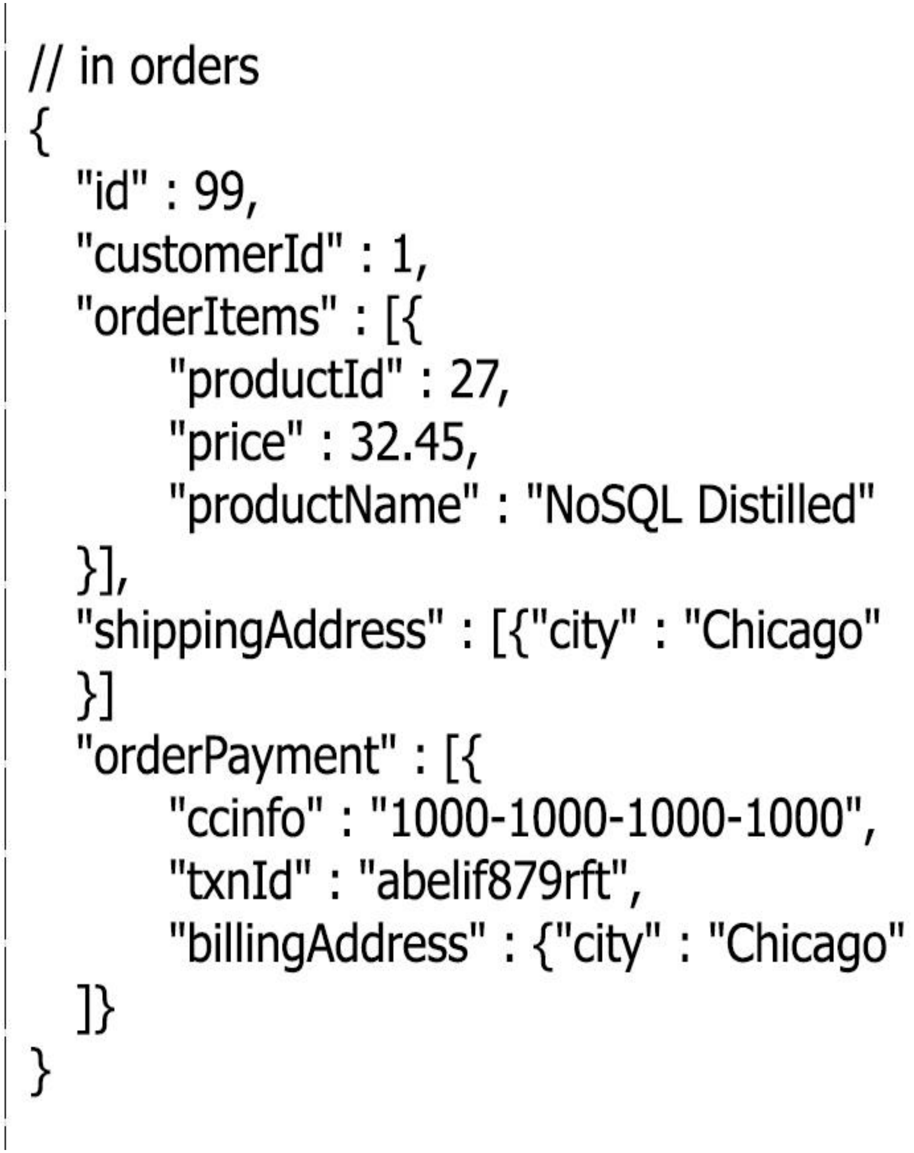
- Store, retrieve document-oriented, semi-structured information
- A document encodes data in some standard formats, such as XML, JSON, YAML, BSON, or binary formats (PDF, XLS, DOC etc)
- Documents are **self-describing hierarchical tree data structures**, which can consists of collections, maps or scalar values
- Documents are stored as (key, value) pairs, but it is a value that's examinable, "transparent"; Each document has a unique id, hence the document is the value on (key, value) pair
- Embedding child documents is possible and provide better performance
- Structure of documents may differ from one object to another
  - New attributes may be created without the need to set them for existing ones

```
{
  "Authors": [{"Name": "Jeffrey Ullman"}, {"Name": "Jennifer Widom"}] ,
  "Title" : "A first course in Database systems",
  "ISBN" : "0-13-713526-2"     <= ID
}
```

Examples



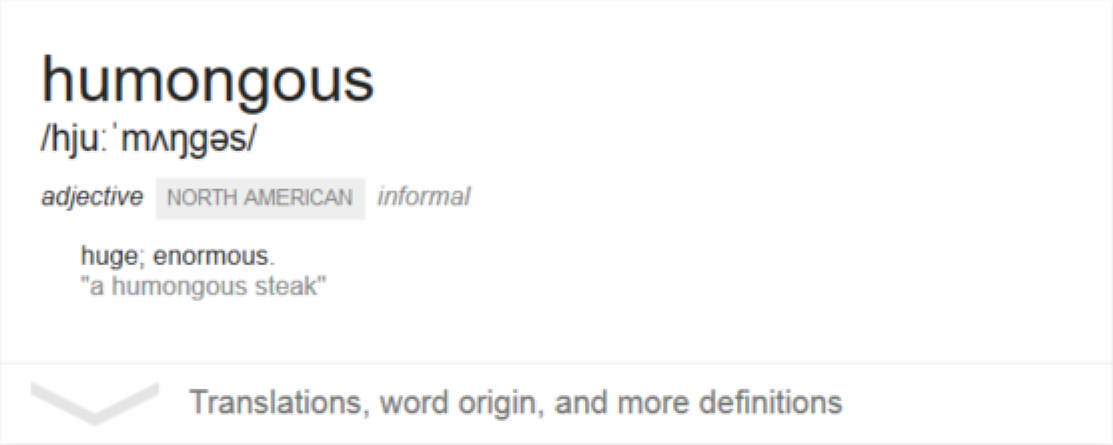
```
// in customers
{
    "id"              :   1,
    "name"            :   "Martin",
    "billingAddress"  :   [{"city":"Chicago"
    }]
}
```



```
// in orders
{
   "id" : 99,
   "customerId" : 1,
   "orderItems" : [{
         "productId" : 27,
         "price" : 32.45,
         "productName" : "NoSQL Distilled"
   }],
   "shippingAddress" : [{"city" : "Chicago"
   }]
   "orderPayment" : [{
         "ccinfo" : "1000-1000-1000-1000",
         "txnId" : "abelif879rft",
         "billingAddress" : {"city" : "Chicago"
   }]}
}
```

MongoDB

- Mongo – “humongous”



humongous
/hjuːˈmʌŋɡəs/
adjective NORTH AMERICAN *informal*

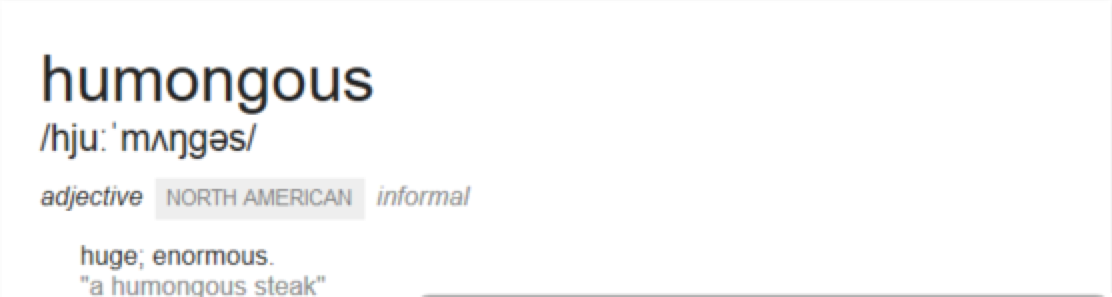
huge; enormous.
"a humongous steak"

Translations, word origin, and more definitions

- MongoDB is a scalable, high-performance, open source, document database
- https://www.mongodb.org/, http://www.mongodb.com/
- http://github.com/mongodb - source code repository (GNU AGPL license)
- First version in 2007 (part of a PaaS), in 2009 went open source

MongoDB

- Mongo – "humongous"

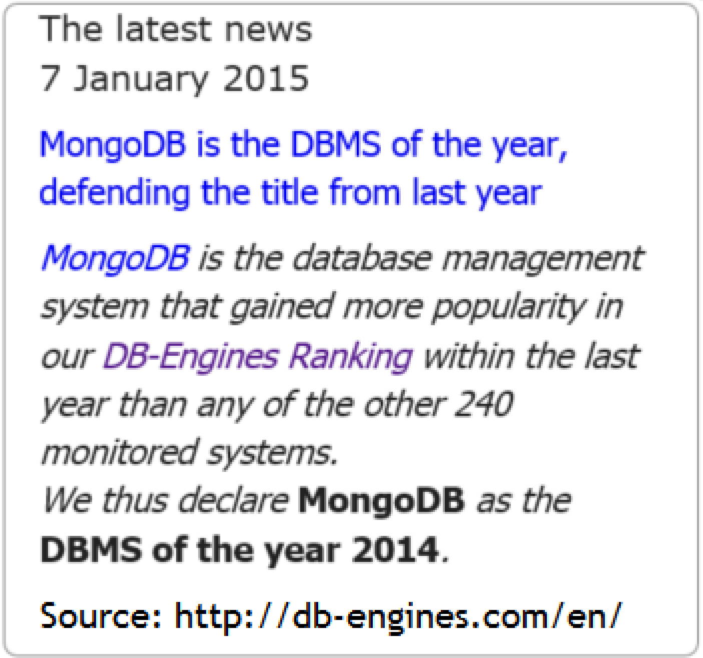


humongous

/hjuːˈmʌŋɡəs/

adjective NORTH AMERICAN *informal*

huge; enormous.
"a humongous steak"



The latest news

7 January 2015

MongoDB is the DBMS of the year, defending the title from last year

MongoDB is the database management system that gained more popularity in our DB-Engines Ranking within the last year than any of the other 240 monitored systems.
We thus declare ***MongoDB*** *as the* **DBMS of the year 2014***.*

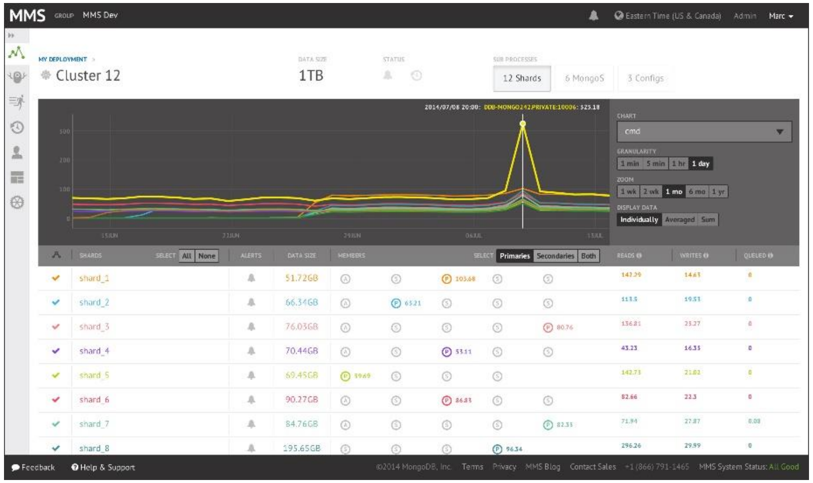
Source: http://db-engines.com/en/

- MongoDB is a s ce, open source, document dat
- https://www.mo ngodb.com/
- http://github.co pository (GNU AGPL license)
- First version in 2007 (part of a PaaS), in 2009 went open source

MongoDB: Features

- JSON documents
- Indexing
- Querying using JSON Query
- Replication and sharding
- Journaling
- Write concern
- Stored procedures (in JavaScript)
- Supports GeoSpatial
- Monitoring

- Security and privacy
  - Authentication (LDAP, AD, Kerberos)
  - Authorization (roles)
  - Auditing
  - Encryption (data encryption on disk, SSL connections)
- Backup and recovery with MMS (MongoDB Management Service, a Web app)



MY DEPLOYMENT > Cluster 12

DATA SIZE: 1TB

STATUS

SUB PROCESSES: 12 Shards | 6 MongoS | 3 Configs

2014/07/08 20:00: DDB-MONGO242.PRIVATE:10006: 525.18

CHART: cmd

GRANULARITY: 1 min | 5 min | 1 hr | **1 day**

ZOOM: 1 wk | 2 wk | **1 mo** | 6 mo | 1 yr

DISPLAY DATA: **Individually** | Averaged | Sum

SELECT All | None — SELECT **Primaries** | Secondaries | Both

	SHARDS	ALERTS	DATA SIZE	MEMBERS	READS	WRITES	QUEUED
✓	shard_1	🔔	51.72GB	Ⓐ Ⓢ Ⓟ 103.68 Ⓢ Ⓢ	142.29	14.63	0
✓	shard_2	🔔	66.34GB	Ⓐ Ⓟ 63.21 Ⓢ Ⓢ Ⓢ	113.5	19.53	0
✓	shard_3	🔔	76.03GB	Ⓐ Ⓢ Ⓢ Ⓢ Ⓟ 80.76	136.81	23.27	0
✓	shard_4	🔔	70.44GB	Ⓐ Ⓢ Ⓟ 53.11 Ⓢ Ⓢ	43.23	16.35	0
✓	shard_5	🔔	69.45GB	Ⓟ 59.69 Ⓢ Ⓢ Ⓢ	142.73	21.02	0
✓	shard_6	🔔	90.27GB	Ⓐ Ⓢ Ⓟ 86.83 Ⓢ Ⓢ	82.66	22.3	0
✓	shard_7	🔔	84.76GB	Ⓐ Ⓢ Ⓢ Ⓢ Ⓟ 82.35	71.94	27.87	0.08
✓	shard_8	🔔	195.65GB	Ⓢ Ⓢ Ⓢ Ⓟ 96.34	296.26	29.99	0

MongoDB

- MongoDB Architecture Guide

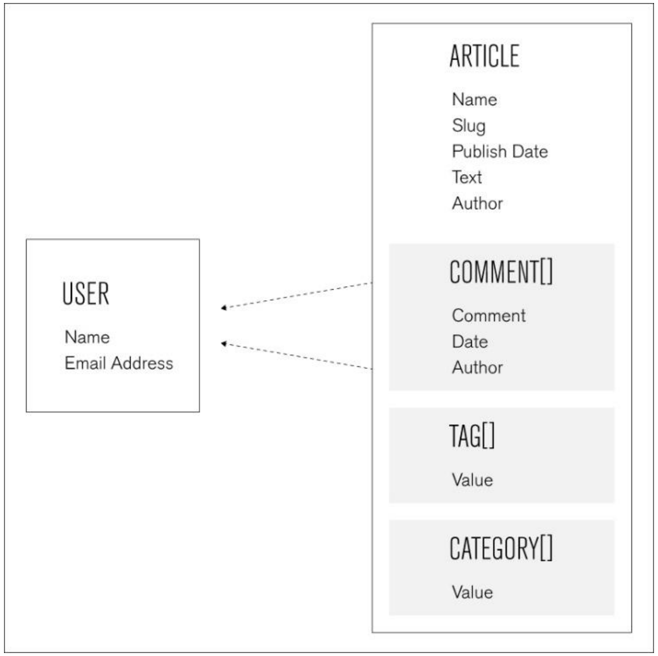
- JSON document format (no PDF or other binary formats)
  - actually is BSON – Binary JSON – binary serialization of JSON-like documents
  - BSON extends JSON with new data types (Date, BinData)
  - BSON is Lightweight (little overhead), Traversable and Efficient (quick to encode/decode data)

- Think of MongoDB documents as objects in OOP

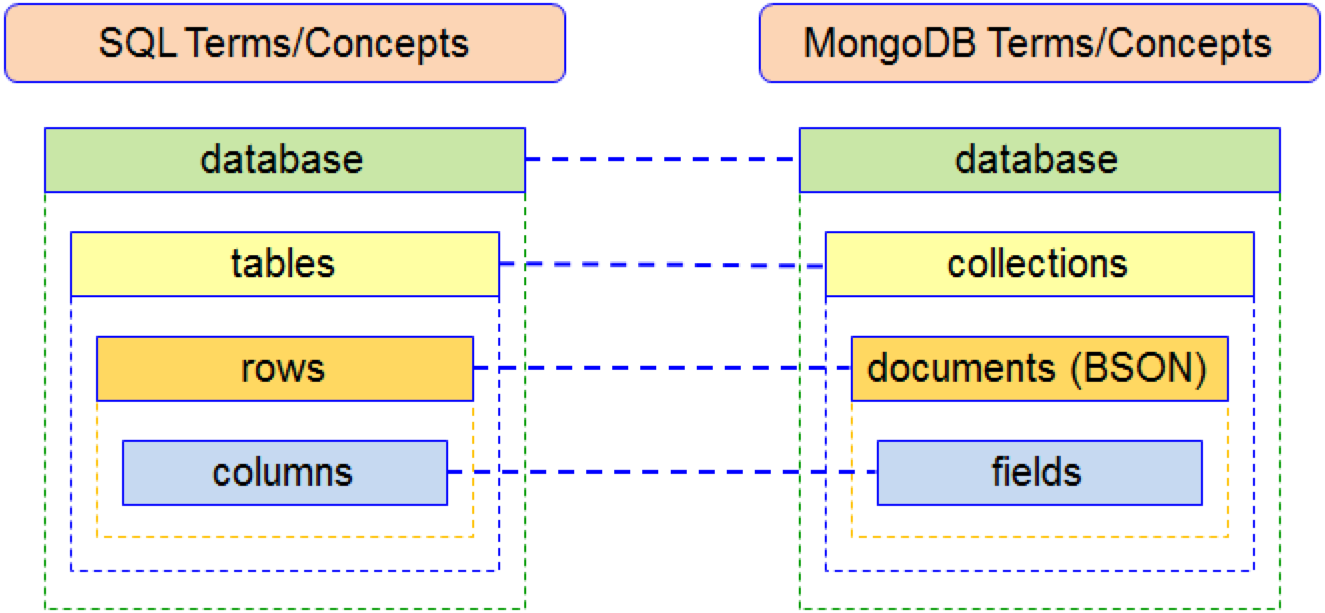
- Supports array of objects

- Dynamic schema
  - Fields can vary from document to document

- Document size <= 16MB



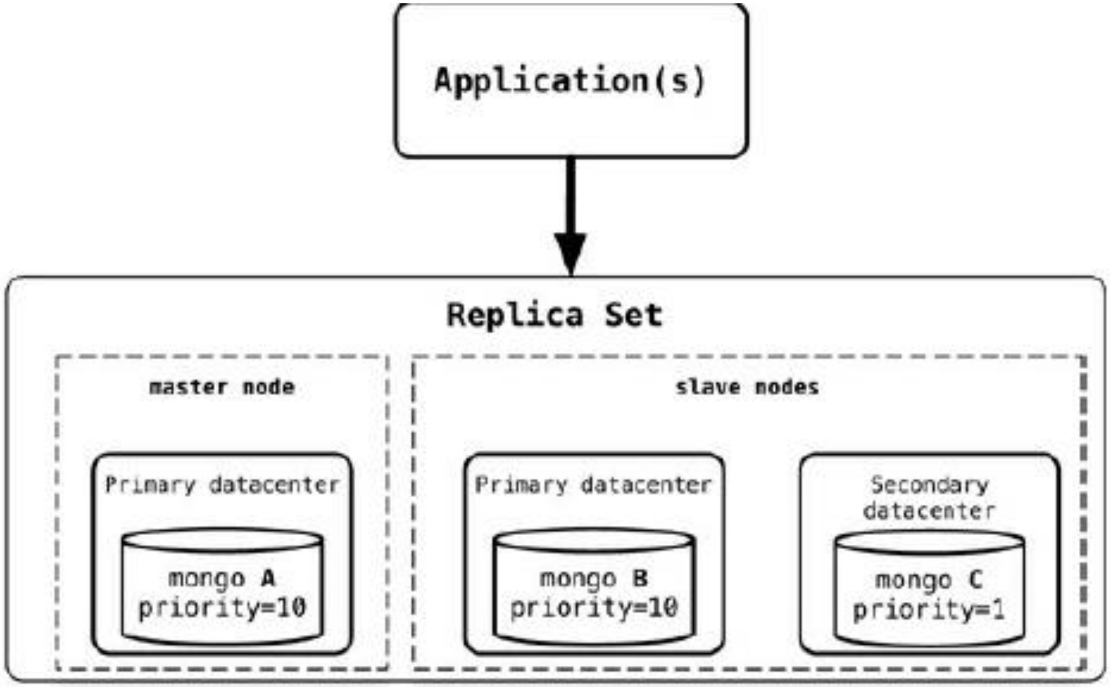
MongoDB


SQL Terms/Concepts
MongoDB Terms/Concepts
database
database
tables
collections
rows
documents (BSON)
columns
fields

MongoDB

- C++ implementation; memory mapped files; recommended on 64-bit platforms
- Per database write lock (+2.2)
- Indexing
  - Each document gets automatically an _id that is indexed on
  - Secondary indexes can be created on any field of the document
- MongoDB Query Optimizer; users can review and optimize the plans using *explain* method and index filters
- ACID properties at document level; write concerns (similar to consistency levels) can be specified at operation level

MongoDB

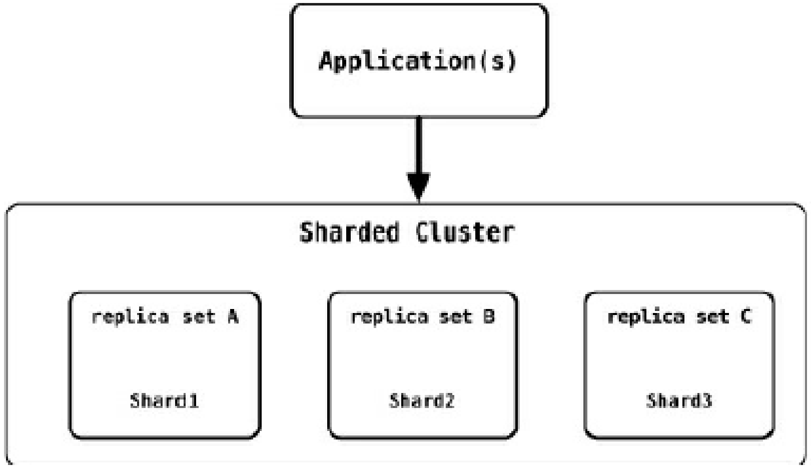


Source: P.J. Sadalage, M. Fowler – NoSQL Distilled, Addison Wesley, 2012

MongoDB

- Sharding: documents are partitioned
  - split data by value in a specific field
  - each shard is setup as one replica set
  - data is actually *moved* across shards

- Sharding options:
  - range based – based on shard key value (optimal for range-based queries)
  - hash based – based on MD5 hash of shard key value (optimal for uniform data distribution)
  - location based – based on a user-specified configuration



Source: P.J. Sadalage, M. Fowler – NoSQL Distilled, Addison Wesley, 2012

Consistency

- Consistency = All clients have the same view of the data at any given moment.
- Achieved by using REPLICA SETS.
- Def: In MongoDB, a replica set is a group of mongod processes that maintain the same data set.
- Replica set properties:

  -single master;

  -maintains backup copies of your database instance.

Availability

- Availability = Each client can read and write at any time
- Document databases try to improve on availability by replicating data using the master-slave setup. The same data is available on multiple nodes and the clients can get to the data even when the primary node is down.
- All requests go to the master node, and the data is replicated to the slave nodes. If the master node goes down, the remaining nodes in the replica set vote among themselves to elect a new master; all future requests are routed to the new master, and the slave nodes start getting data from the new master. When the node that failed comes back online, it joins in as a slave and catches up with the rest of the nodes by pulling all the data it needs to get current.

Transactions

- In RDBMS's, one can execute a set of commands containing inserts, updates, or deletes, over multiple tables, and only then decide whether to keep them(commit) or not(rollback).

- These constructs are generally not available in NoSQL—a write either succeeds or fails.

- Transactions at the single-document level are known as atomic transactions. Transactions involving more than one operation are not possible, although thereare products such as RavenDB that do support transactions across multiple operations.

Indexing in MongoDB

- Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a collection scan(scan every document in a collection, to select those documents that match the query statement). If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect.
- Indexes are special data structures that store a small portion of the collection’s data set in an easy to traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field. In addition, MongoDB can return sorted results by using the ordering in the index.
- MongoDB creates a unique index on the _id field during the creation of a collection. The _id index prevents clients from inserting two documents with the same value for the _id field. You cannot drop this index on the _id field.

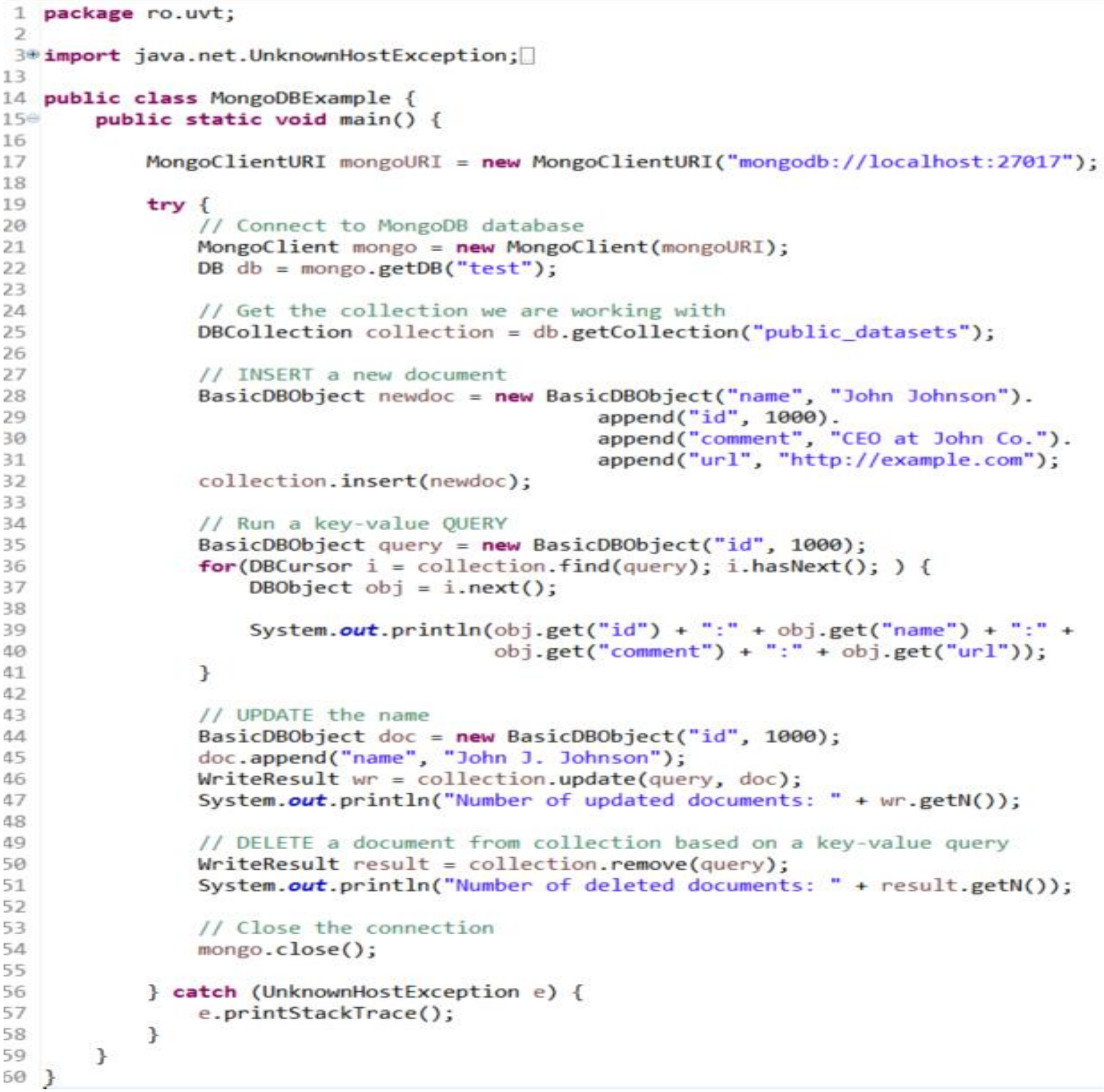
Scaling

- For reading: simply add the new node(s) to the cluster.
- For writing: SHARDING.
- Partition the collections based on a certain field. These partitions are called "shards" and each node in the database system will store one of these.
- In order to retrieve or write information, the application communicates with a mongos instance, a routing service provided by MongoDB. This instance locates the data in the sharded cluster, with the aid of 3 configuration servers, then performs the desired operations.
- Config Servers – Config servers store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards.

Query features

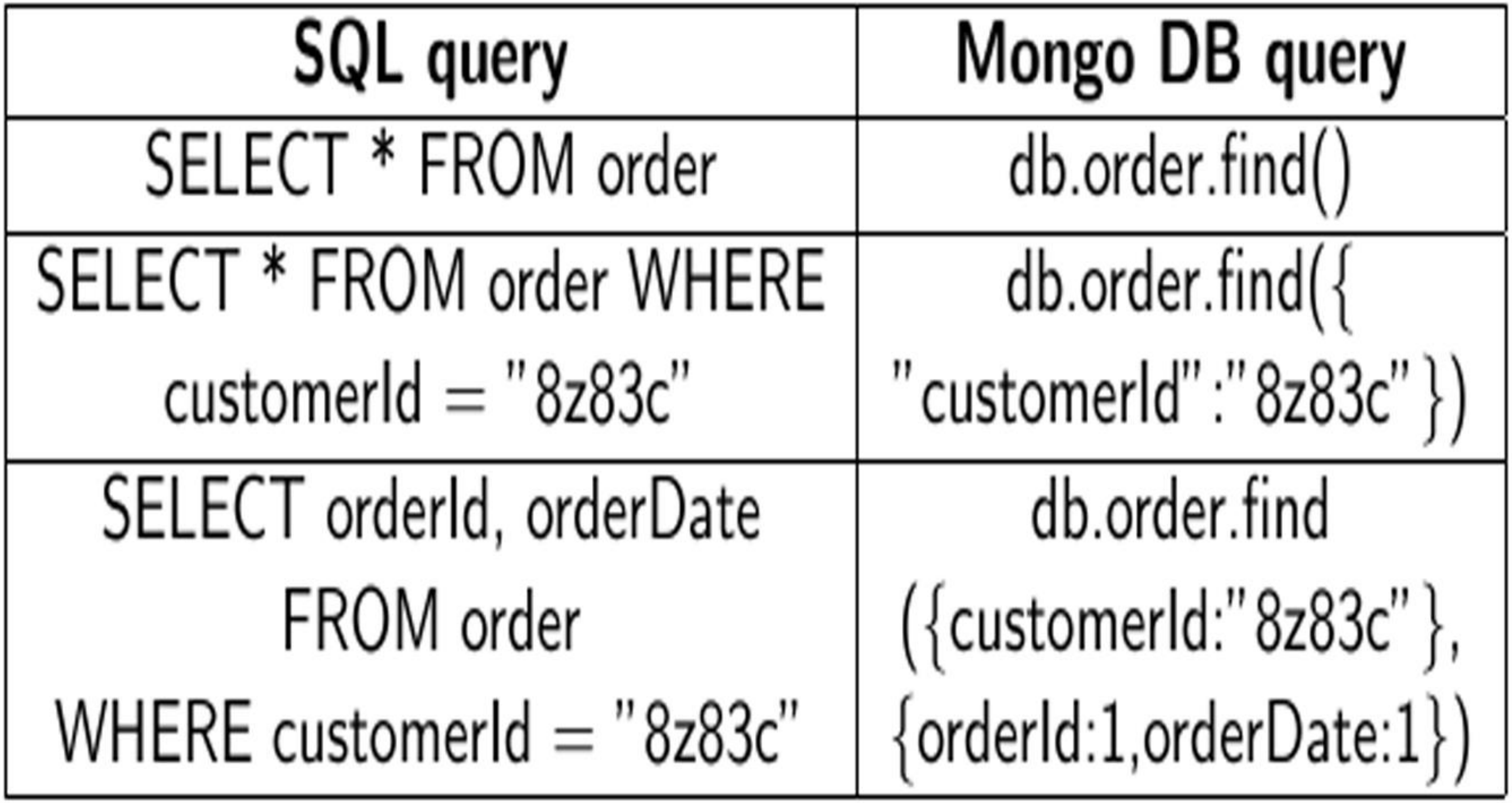
- Document databases provide different query features.
- One of the good features of document databases, as compared to key-value stores, is that we can query the data inside the document without having to retrieve the whole document by its key and then introspect the document. This feature brings these databases closer to the RDBMS query model.
- MongoDB has a query language which is expressed via JSON and has constructs such as *query*
- for the *where* clause, *orderby* for sorting the data, or *explain* to show the execution plan of the query.

MongoDB



```java
package ro.uvt;

import java.net.UnknownHostException;□

public class MongoDBExample {
    public static void main() {

        MongoClientURI mongoURI = new MongoClientURI("mongodb://localhost:27017");

        try {
            // Connect to MongoDB database
            MongoClient mongo = new MongoClient(mongoURI);
            DB db = mongo.getDB("test");

            // Get the collection we are working with
            DBCollection collection = db.getCollection("public_datasets");

            // INSERT a new document
            BasicDBObject newdoc = new BasicDBObject("name", "John Johnson").
                                   append("id", 1000).
                                   append("comment", "CEO at John Co.").
                                   append("url", "http://example.com");
            collection.insert(newdoc);

            // Run a key-value QUERY
            BasicDBObject query = new BasicDBObject("id", 1000);
            for(DBCursor i = collection.find(query); i.hasNext(); ) {
                DBObject obj = i.next();

                System.out.println(obj.get("id") + ":" + obj.get("name") + ":" +
                                   obj.get("comment") + ":" + obj.get("url"));
            }

            // UPDATE the name
            BasicDBObject doc = new BasicDBObject("id", 1000);
            doc.append("name", "John J. Johnson");
            WriteResult wr = collection.update(query, doc);
            System.out.println("Number of updated documents: " + wr.getN());

            // DELETE a document from collection based on a key-value query
            WriteResult result = collection.remove(query);
            System.out.println("Number of deleted documents: " + result.getN());

            // Close the connection
            mongo.close();

        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

- APIs for most used PLs (C++, Java, Python, PHP etc)
- A query may return an entire document or a subset of fields of a doc
- Query types: key-value, range, geospatial, text search, aggregation, MapReduce
- NO JOIN!
- MongoDB shell



SQL query	Mongo DB query
SELECT * FROM order	db.order.find()
SELECT * FROM order WHERE customerId = "8z83c"	db.order.find({ "customerId":"8z83c" })
SELECT orderId, orderDate FROM order WHERE customerId = "8z83c"	db.order.find ({customerId:"8z83c" }, {orderId:1,orderDate:1})

Comparison to relational model

Consistency

- Replica sets – number of nodes to respond to write (configurable per write operation)

Transactions

- Only atomic transactions supported (single-document level)
- Use `WriteConcern.REPLICAS_SAFE` to ensure that document is written to more than one node

Comparison to relational model

Query features

- Complex queries on documents are supported
- Ex: `db.orders.find({"items.product.name":/Refactoring/})` – all orders contining an item whose name contains ‘Refactoring’

Comparison to relational model

Structure of data

- Maybe different from one document to another (attributes missing), but generally similar

Scaling

- Adding more read-slaves nodes to a replica set (no downtime, no restart)
- Sharding

Availability

- Replica sets

Usages

- Event logging
- Content Management Platforms (blogging platforms)
- E-Commerce applications
- Web analytics

To avoid

- Multiple-operations transactions
- Query against varying aggregate structure

Other document databases

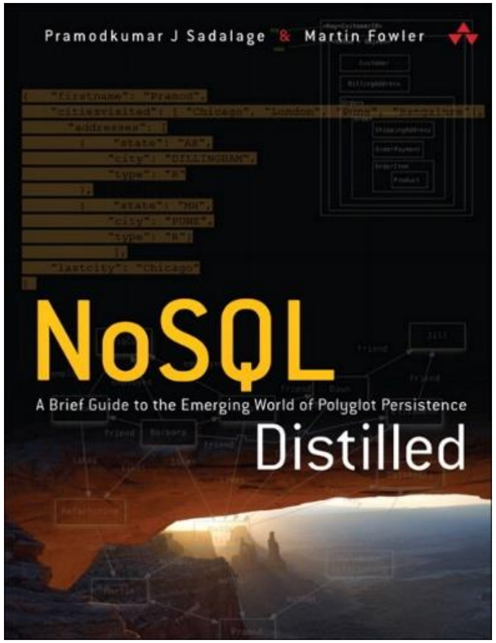
CouchDB
relax

terrastore
Scalable, elastic, consistent document store.

Couchbase

SequoiaDB

Bibliography

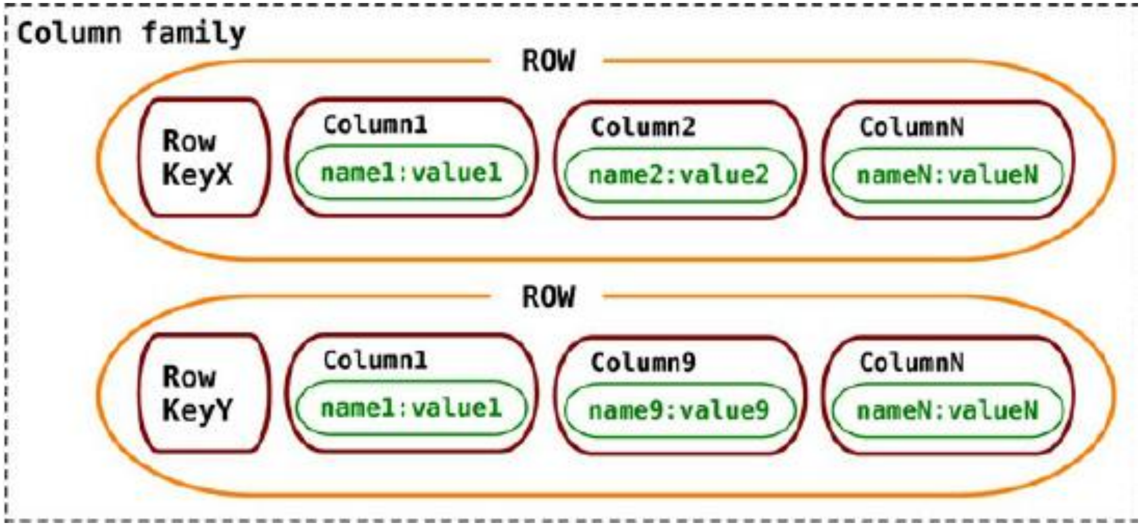

Pramodkumar J Sadalage & Martin Fowler
NoSQL
A Brief Guide to the Emerging World of Polyglot Persistence
Distilled

NoSQL Distilled by Pramod J. Sadalage and Martin Fowler, Addison Wesley, 2012 **Chapter 8,9**

Column-family database

Column-family databases

- Store data with keys mapped to values; values grouped into multiple column families, each column family being a map of data
- Column families are groups of related data that is often accessed together (e.g. Customer has Profile, Orders, etc)
- A row is a collection of columns attached or linked to a key;
- A collection of similar rows makes a column family.
- A column family is similar to a relational table, but the difference is that various rows do not have to have the same columns, and columns can be added to any row at any time without having to add it to other rows.



Source: P.J. Sadalage, M. Fowler – NoSQL Distilled, Addison Wesley, 2012

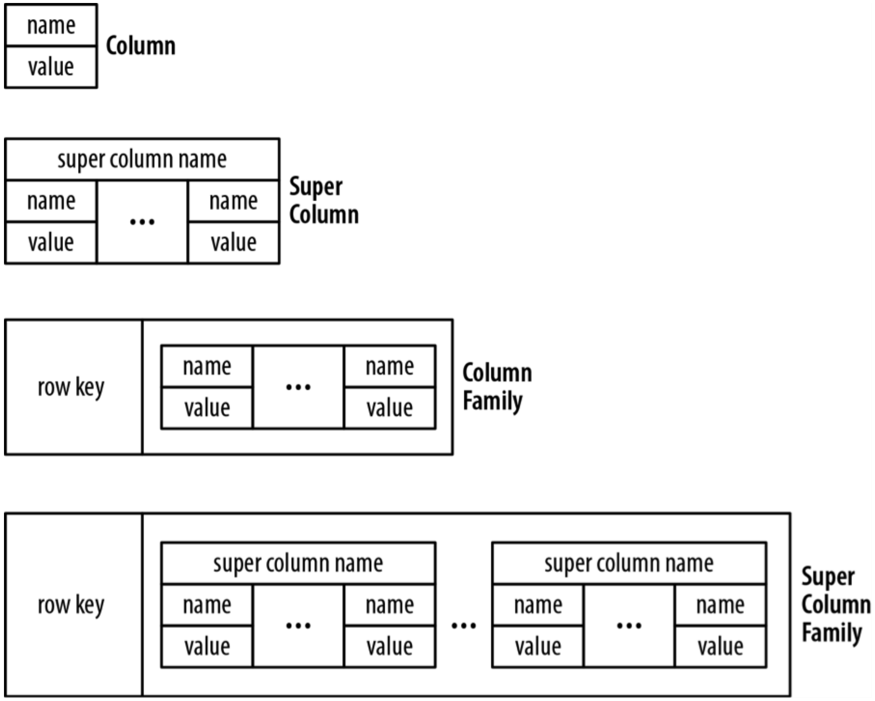
Features

On a structural level, the most basic unit is a **column**, which consists of a pair name-value with the name also acting as a key. A column is always stored along with a timestamp value used to expire the data.

A collection of columns grouped by a specific key is a **row**. Collection of similar rows is what makes a column-family.[1]

When a column contains a map to another column, it's called a **super column.**[2]

Both kinds of columns are stored into **keyspaces** of a specific application, which is similar to a database in a RDBMS.



Apache Cassandra

- Open Source @ http://cassandra.apache.org

- Distributed database system

- Massively scalable: “In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes in all experiments” (R. Tilmann et. al "Solving Big Data Challenges for Enterprise Application Performance Management“, VLDB 2012)

- Processing and analysis using MapReduce framework

- Customizable replication

- Query language: CQL (Cassandra Query Language)

Apache Cassandra

Brief history

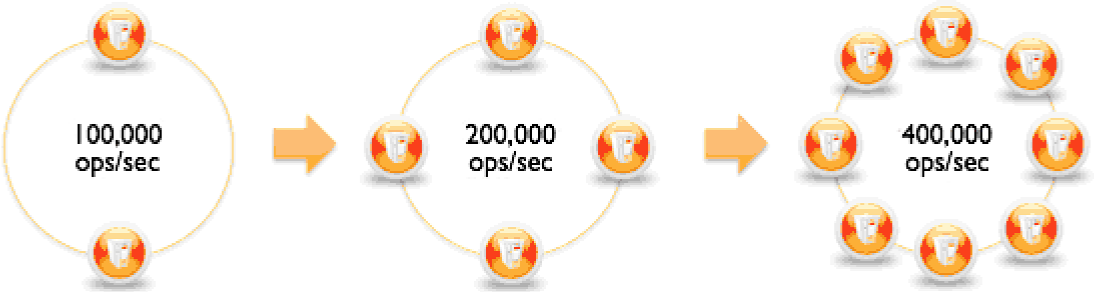
- Introduced in Cassandra paper (2008) by A. Lakshman and P. Malik (Facebook)

- 2010 promoted as top-level Apache project

DATASTAX - Enterprise distribution of Apache Cassandra

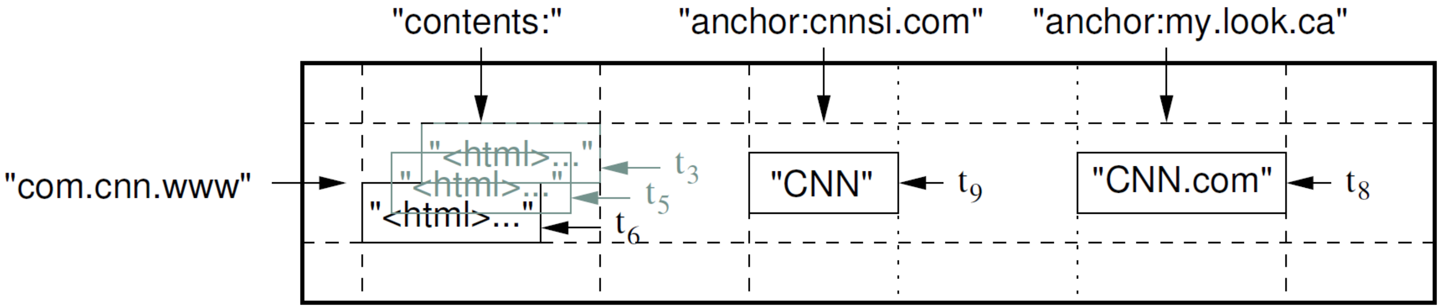
Apache Cassandra

- Borrows concepts from Amazon Dynamo, specifically the distributed architecture (ring topology, consistent hashing, gossip protocol) and from Google Bigtable, specifically the column-oriented data model and storage data structures

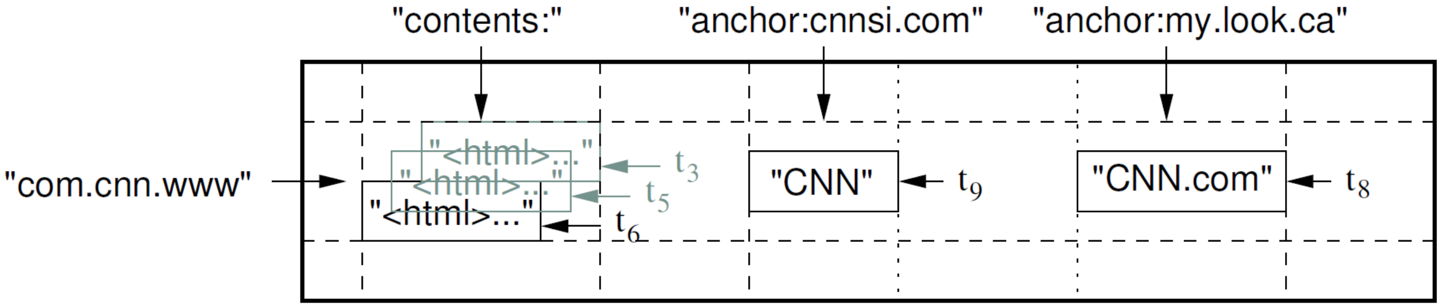


- Google Bigtable introduced by F. Chang et al. @ 7th Symposium on Operating System Design and Implementation (2006)

=> Column oriented data model



Apache Cassandra



“Sparse, distributed persistent multi-dimensional sorted map” (F. Chang et al.)

The map is indexed on **row key, column key** and **timestamp**; each value is a string (array of bytes)

Rows are ordered lexicographically and grouped in ranges called **tablet** => data locality

Column keys are grouped into sets called **column families - the basic unit of access control** => allows different application to add new data, create derived column families, access restricted column families

Timestamp (64-bit integers) allow versioning for cell values; can be managed by the system or by the application

No fixed number of columns per rows in a table

Apache Cassandra

- How does it work (watch Patrick McFadin @ Cerner's tech talk)
- WRITES
  - Writes first to the Log file on disk (append only)
  - Updates the memtable in memory
  - Replies to the user
  - When the memtable is full => flushes it to the disk (SSTable in the data directory)
  - There are multiple values at different timestamps for one cell; **last write wins**
  - Compactions => *clears* the data (writes a new SSTable and deletes old files)
- READS
  - Coordinated reads
  - A client 'asks' any node about data, which in turn asks all the nodes and receives the data from the one that owns it

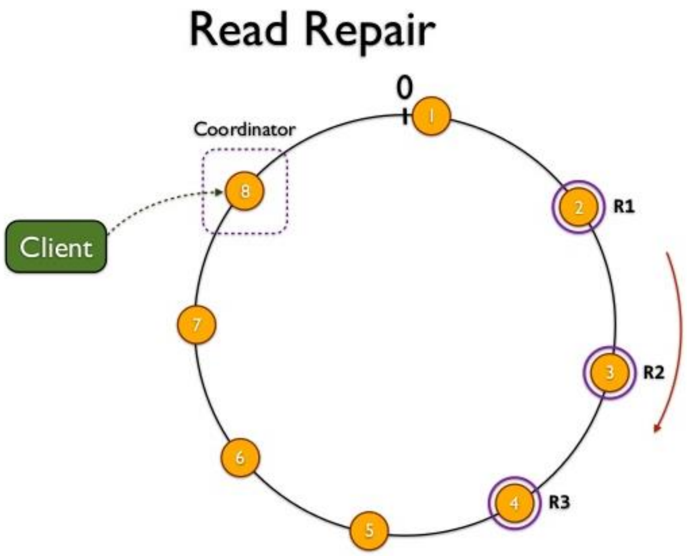
Sequential I/O operations => increased performance

Consistency

During the creation of the keyspace, the DBM must address the data locality issue, configuring what node gets a piece a data, and how many replicas are stored.

Consistency levels:

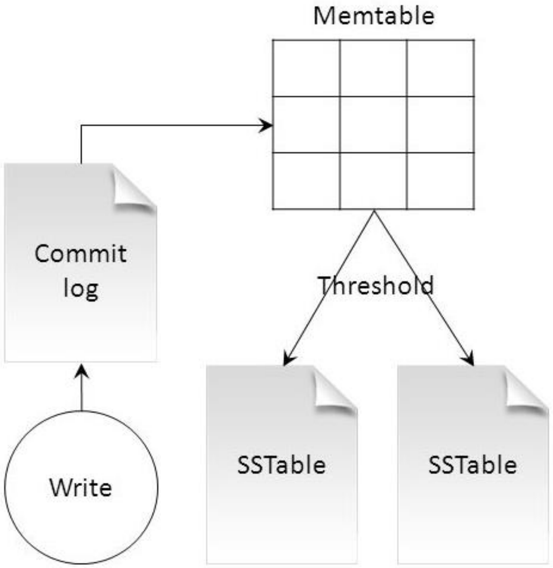
- **ONE:**
  - Write: OK with one node response
  - Read: Return first replica
- **QUORUM:**
  - Write: OK with majority response
  - Read: Return newest timestamp
- **ALL:**
  - Write; OK with all nodes response
  - Read: Returns after read repair[1]



Transactions

While not having transactions in a traditional sense, they can be inserted with *external libraries*, but just to synchronize writes and reads.

Write operations are considered **atomic** at row level, meaning that each insert or update of a column is considered as a single operation with just two possible results, success or failure. When a node goes down, the data stored in the commit log is used to restore it later.



Availability

By design, column-family stores are highly available, as every node is a **peer** in the cluster. This availability can be further increased by reducing the consistency. It’s regulated according to the following formula:

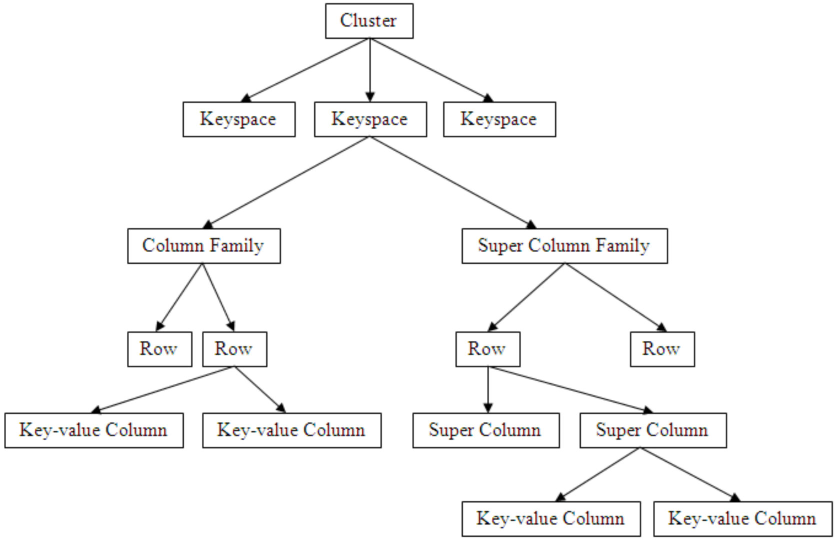
$$(R + W) > N$$

Where ***R*** is the minimum number of nodes that must return a successful read, ***W*** the minimum number of nodes that must be successfully written and ***N*** is the number of nodes participating in the replication of data.

Scaling

Scaling a cluster of n odes in this type of data store is just a matter of *adding more nodes* . More nodes improve the capacity of the cluster to support more R&W operations.

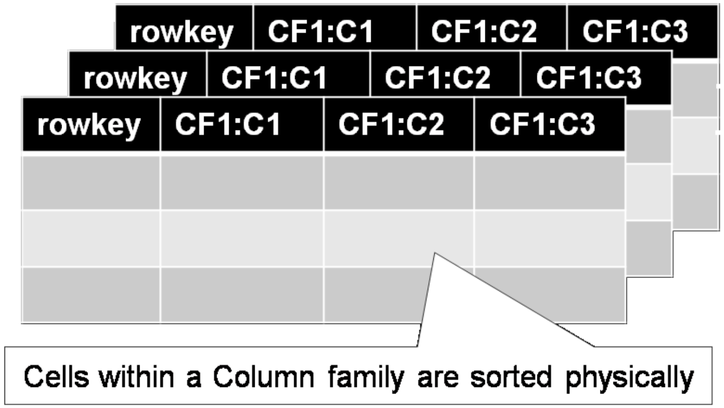
It’s a type of **horizontal scaling**.



Query Features

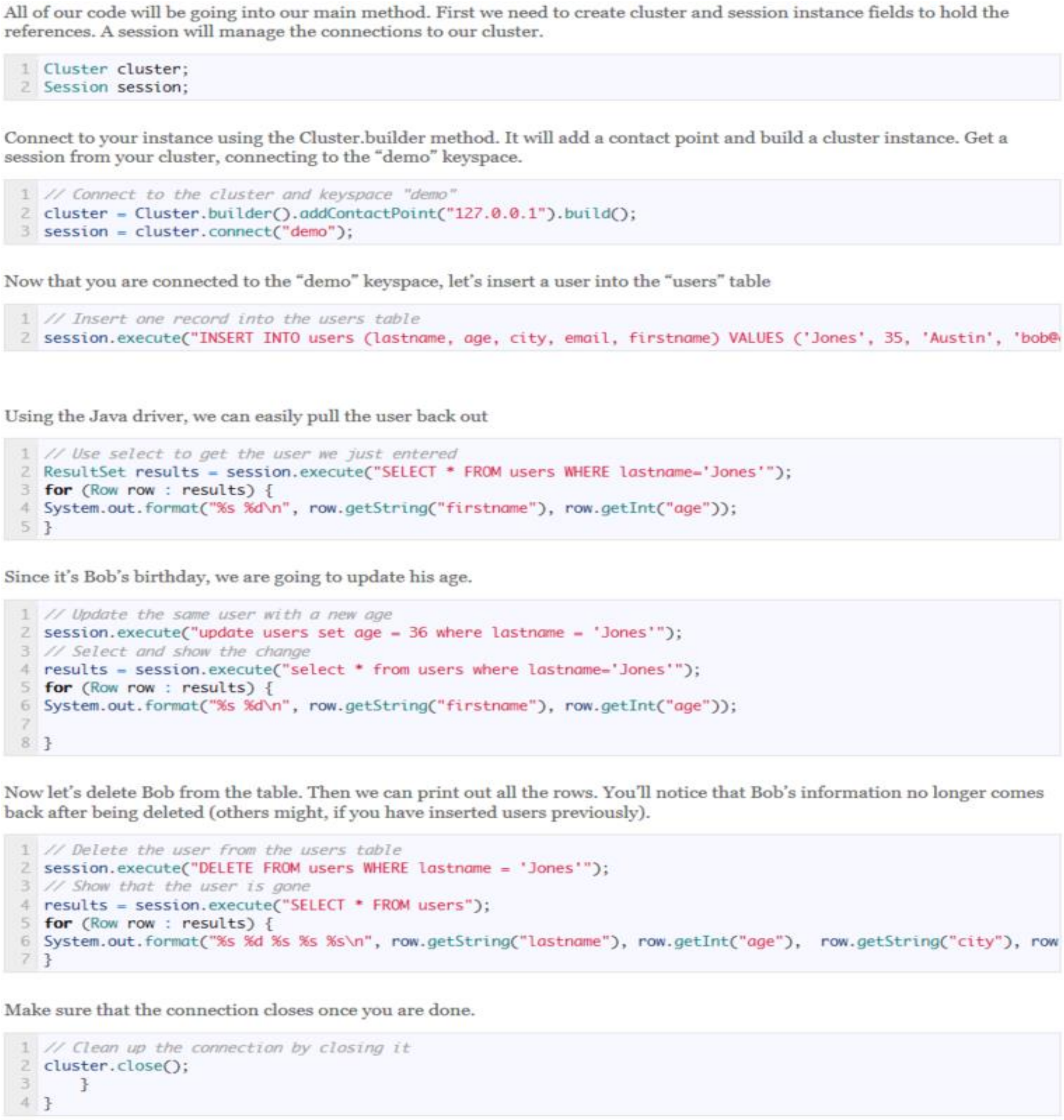
Column-family stores doesn’t have a rich query language, which means it falls upon the database designer to **optimize** the data for reading. The data is usually sorted alphabetically by column names.

The queries run in a specific keyspace, and needs to have a **well defined** column family before trying to run any query.



Cells within a Column family are sorted physically

Apache Cassandra Query Language (CQL)



All of our code will be going into our main method. First we need to create cluster and session instance fields to hold the references. A session will manage the connections to our cluster.

```
Cluster cluster;
Session session;
```

Connect to your instance using the Cluster.builder method. It will add a contact point and build a cluster instance. Get a session from your cluster, connecting to the "demo" keyspace.

```
// Connect to the cluster and keyspace "demo"
cluster = Cluster.builder().addContactPoint("127.0.0.1").build();
session = cluster.connect("demo");
```

Now that you are connected to the "demo" keyspace, let's insert a user into the "users" table

```
// Insert one record into the users table
session.execute("INSERT INTO users (lastname, age, city, email, firstname) VALUES ('Jones', 35, 'Austin', 'bob@
```

Using the Java driver, we can easily pull the user back out

```
// Use select to get the user we just entered
ResultSet results = session.execute("SELECT * FROM users WHERE lastname='Jones'");
for (Row row : results) {
System.out.format("%s %d\n", row.getString("firstname"), row.getInt("age"));
}
```

Since it's Bob's birthday, we are going to update his age.

```
// Update the same user with a new age
session.execute("update users set age = 36 where lastname = 'Jones'");
// Select and show the change
results = session.execute("select * from users where lastname='Jones'");
for (Row row : results) {
System.out.format("%s %d\n", row.getString("firstname"), row.getInt("age"));

}
```

Now let's delete Bob from the table. Then we can print out all the rows. You'll notice that Bob's information no longer comes back after being deleted (others might, if you have inserted users previously).

```
// Delete the user from the users table
session.execute("DELETE FROM users WHERE lastname = 'Jones'");
// Show that the user is gone
results = session.execute("SELECT * FROM users");
for (Row row : results) {
System.out.format("%s %d %s %s %s\n", row.getString("lastname"), row.getInt("age"),  row.getString("city"), row
}
```

Make sure that the connection closes once you are done.

```
// Clean up the connection by closing it
cluster.close();
    }
}
```

CQL = Cassandra Query Language

- similar to SQL
- **create** a column-family, **insert** data into it, **read** the whole column-family or just specific columns, and index them for later queries
- NO JOIN! No Subqueries!
- First item in PRIMARY KEY definition is the **partition key**
- Supports SQL data types (timestamp, integer, varchar, list<>)
- No sizes for strings (varchar)
- Supports TTL (TimeToLive)
- cqlsh – CQL shell

Source: http://planetcassandra.org/getting-started-with-apache-cassandra-and-java/

Comparison to relational model

Consistency

- Different levels: ONE, QUROM, ALL per read / write
- Replication factor set at Keyspace level

Transactions

- Atomic at the row level
- External libraries (ZooKeeper) for multi read/write transactions

Query features

- See CQL

Comparison to relational model

Structure of data

- Rows have similar structure, but not identical

Scaling

- Horizontal scaling supported

Availability

- High availability by design
- Controlled by $R + W > N$

Usages

- Event logging
- CMS, Blogging platforms
- Web analytics
- Expiring usage (as TTL is an internal concept)

To avoid

- When ACID transactions for write / read
- Ad-hoc queries
- Schema is not stable

Other column databases

Google
Bigtable

APACHE
HBASE

amazon
web services™

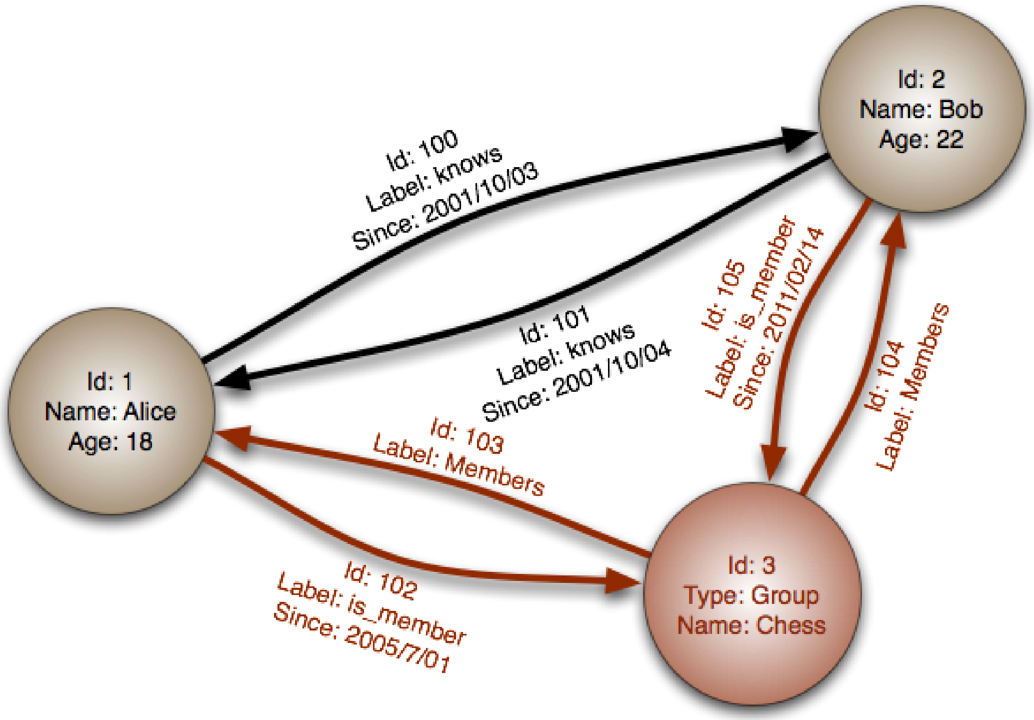
Amazon SimpleDB

HYPERTABLE INC

Graph database

Graph database

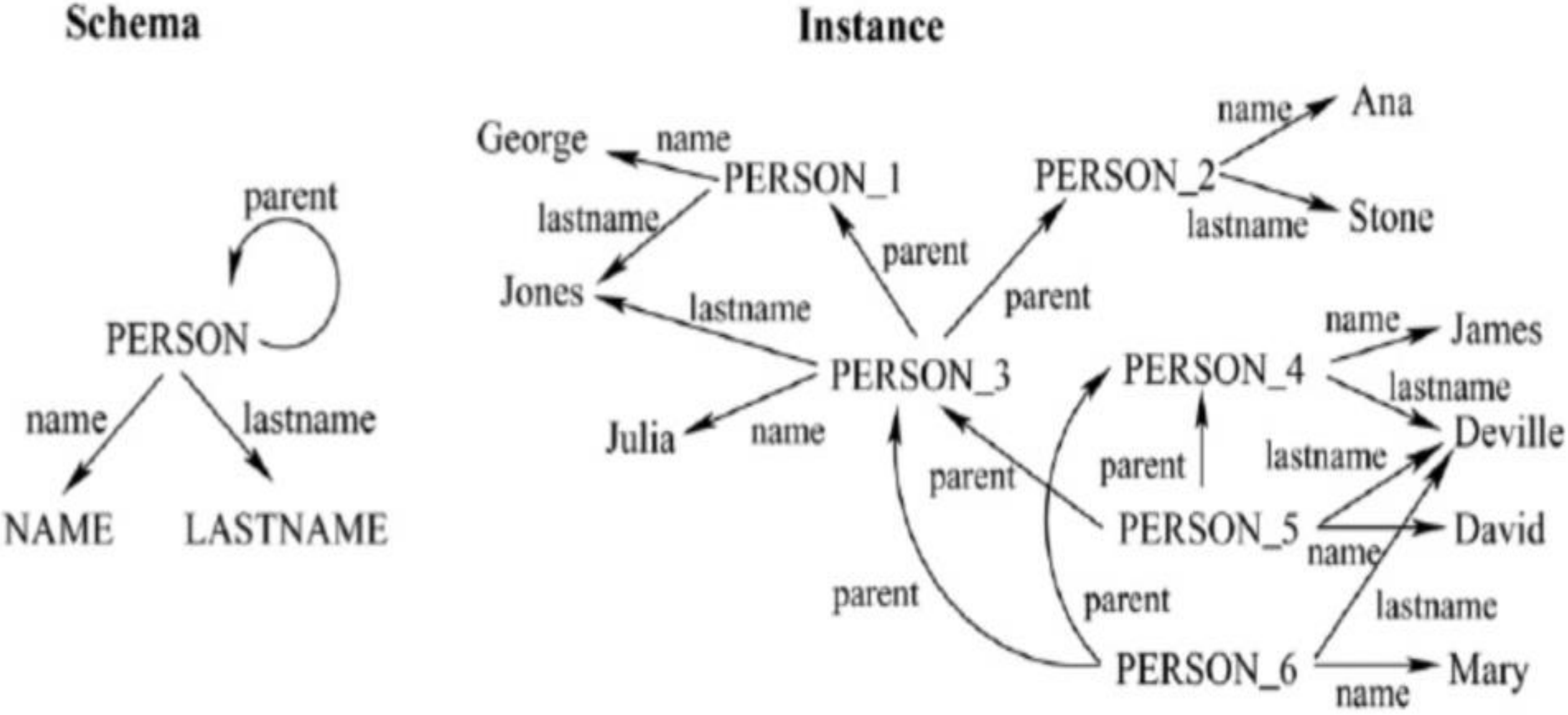
- A **Graph Database** is a database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data.
- A graph database is any storage system that provides index-free adjacency. This means that every element contains a direct pointer to its adjacent elements and no index lookups are necessary. (Source: http://en.wikipedia.org/wiki/Graph_database)



Graph database

- Graph databases are faster for associative datasets
- Don’t require JOIN operations
- Suitable for data with no (or changing) schema
- Natural choice for apps that require graph-like queries:
  - Give me the shortest path between 2 nodes of the graph
  - Detect all nodes linked to a specified node (community detection)
- Not particular good at scaling on large clusters (still, better comparing to relational)
- They are ACID-oriented databases (comparing to other NoSQL db that are BASE)

Graph database

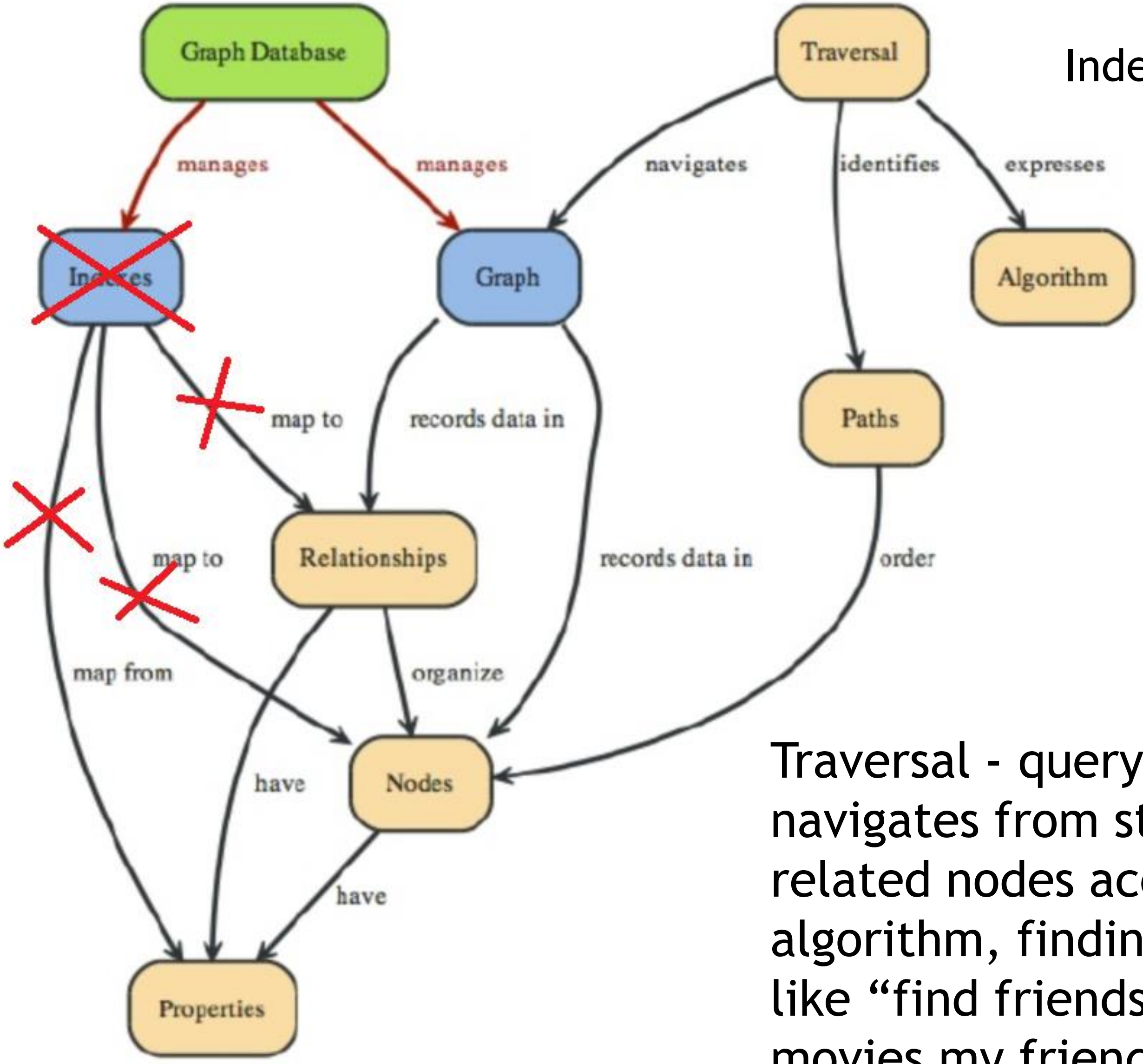

Schema

Instance

Neo4J

- Open source graph-database implemented in …… Java by Neo Technologies
- V1.0 in 2010, V2.0 in 2013, V2.3 in 2015
- Everything is stored as node, edge or attribute
- A node or edge can have any number of **attributes** and can be **labelled**

Neo4J

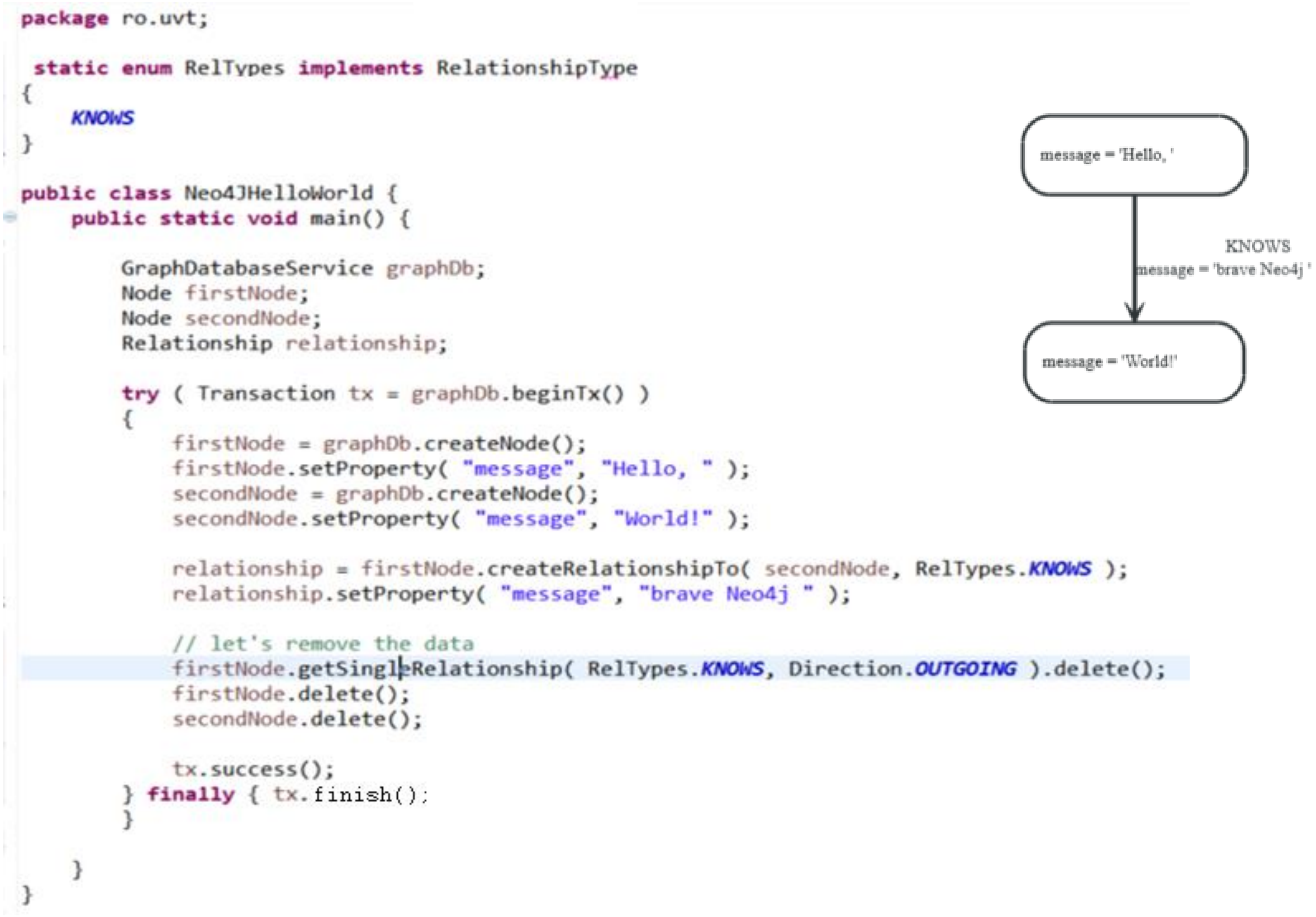


Indexes supported in V1.0 only

Traversal - query processing: it navigates from starting nodes to related nodes according to an algorithm, finding answers to questions like "find friends of my friends", "what movies my friends like" etc.

Neo4J

Neo4J Hello World Example in Java



```
package ro.uvt;

 static enum RelTypes implements RelationshipType
{
    KNOWS
}

public class Neo4JHelloWorld {
    public static void main() {

        GraphDatabaseService graphDb;
        Node firstNode;
        Node secondNode;
        Relationship relationship;

        try ( Transaction tx = graphDb.beginTx() )
        {
            firstNode = graphDb.createNode();
            firstNode.setProperty( "message", "Hello, " );
            secondNode = graphDb.createNode();
            secondNode.setProperty( "message", "World!" );

            relationship = firstNode.createRelationshipTo( secondNode, RelTypes.KNOWS );
            relationship.setProperty( "message", "brave Neo4j " );

            // let's remove the data
            firstNode.getSingleRelationship( RelTypes.KNOWS, Direction.OUTGOING ).delete();
            firstNode.delete();
            secondNode.delete();

            tx.success();
        } finally { tx.finish();
        }

    }
}
```

Neo4J

Cypher Query Language

http://neo4j.com/docs/stable/cypher-query-lang.html

- Declarative graph query language that allows for expressive and efficient querying and updating of the graph store.

- Its constructs are based on English prose and neat iconography which helps to make queries more self-explanatory.

- Focuses on the clarity of expressing ***what*** to retrieve from a graph, ***not*** on ***how*** to retrieve it.

- Inspired by SQL and SPARQL

Neo4J

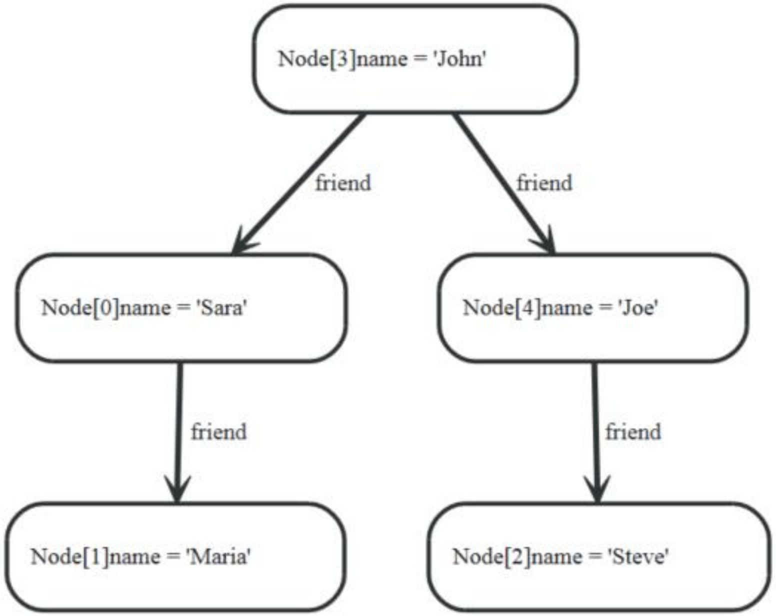
Clauses (examples):

- MATCH: The graph pattern to match. This is the most common way to get data from the graph.

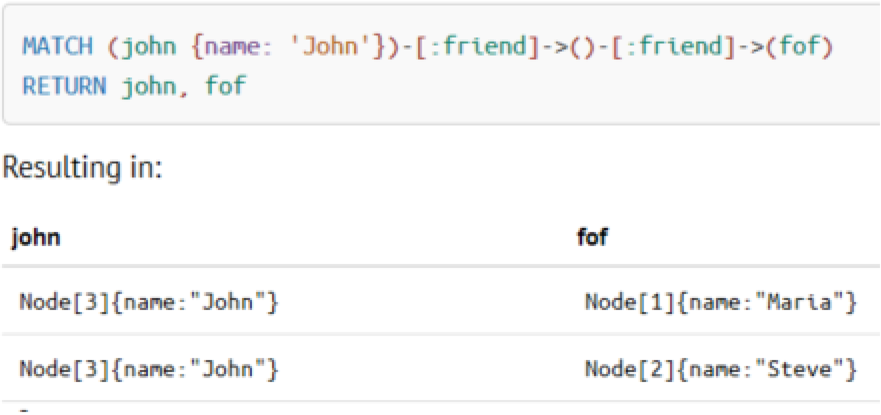
- WHERE: Not a clause in it’s own right, but rather part of MATCH, OPTIONAL MATCH and WITH. Adds constraints to a pattern, or filters the intermediate result passing through WITH.

- RETURN: What to return.

Neo4J



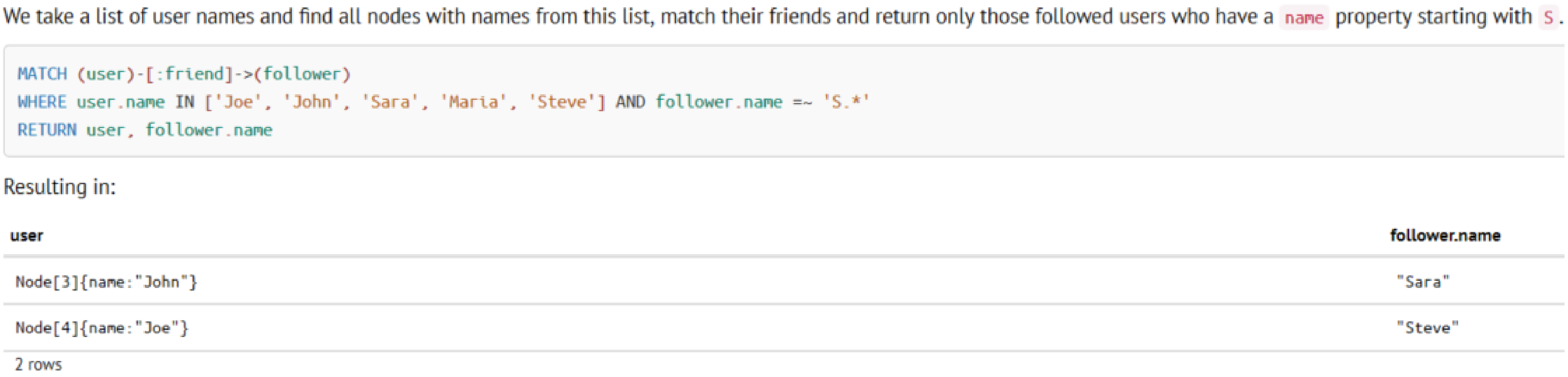
Finds a user called John and John's friends (though not his direct friends) before returning both John and any friends-of-friends that are found.



```
MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof)
RETURN john, fof
```

Resulting in:

john	fof
Node[3]{name:"John"}	Node[1]{name:"Maria"}
Node[3]{name:"John"}	Node[2]{name:"Steve"}



We take a list of user names and find all nodes with names from this list, match their friends and return only those followed users who have a `name` property starting with `S`.

```
MATCH (user)-[:friend]->(follower)
WHERE user.name IN ['Joe', 'John', 'Sara', 'Maria', 'Steve'] AND follower.name =~ 'S.*'
RETURN user, follower.name
```

Resulting in:

user	follower.name
Node[3]{name:"John"}	"Sara"
Node[4]{name:"Joe"}	"Steve"

2 rows

Comparison to relational model

Consistency

- Usually, single-server setup, data is rarely distributed
- Write to master is eventually synchronized with servers; slaves are always available for read

Transactions

- Neo4J is ACID compliant

Query features

- Neo4J Cypher
- Supports indexing using Lucene
- Finding paths between nodes

Comparison to relational model

Scaling

- Storing all connected nodes on the same server is better for performance -> poor scaling
- Sharding data using domain-specific knowledge (e.g. all nodes related to Europe on a server in Europe)
- Read-scaling is possible, adding more read nodes

Availability

- High availability by replicated slaves
- Neo4J uses Apache ZooKeeper to keep track of communication between nodes

Usages

- Social networks, connected data in general
- Routing, dispatch and location-based services
- Recommendation engines

To avoid

- When all entities need to be updated

Other graph databases

AllegroGraph
Franz Inc.

InfiniteGraph

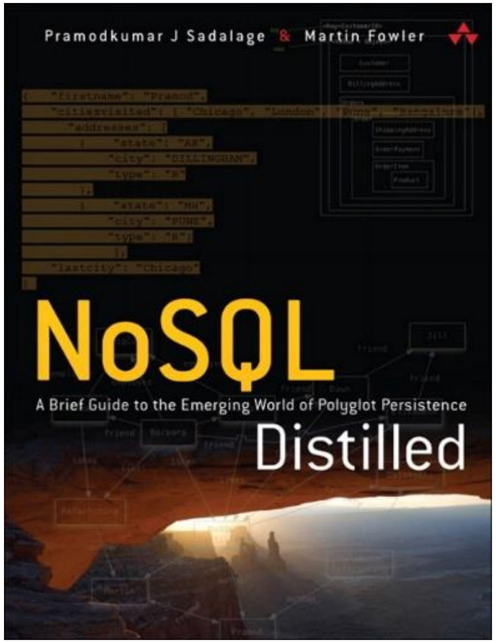
OPENLINK
VIRTUOSO

Wrapping-up NoSQL databases

- Common features
  - Distributed systems
  - Replication, fault tolerance
  - No JOINs,
  - No Adhoc queries
  - No common query language

- Rely on different data models
- Emerging technology, growing popularity
- If processing you need is different to how data is stored, you run into difficulties

Bibliography

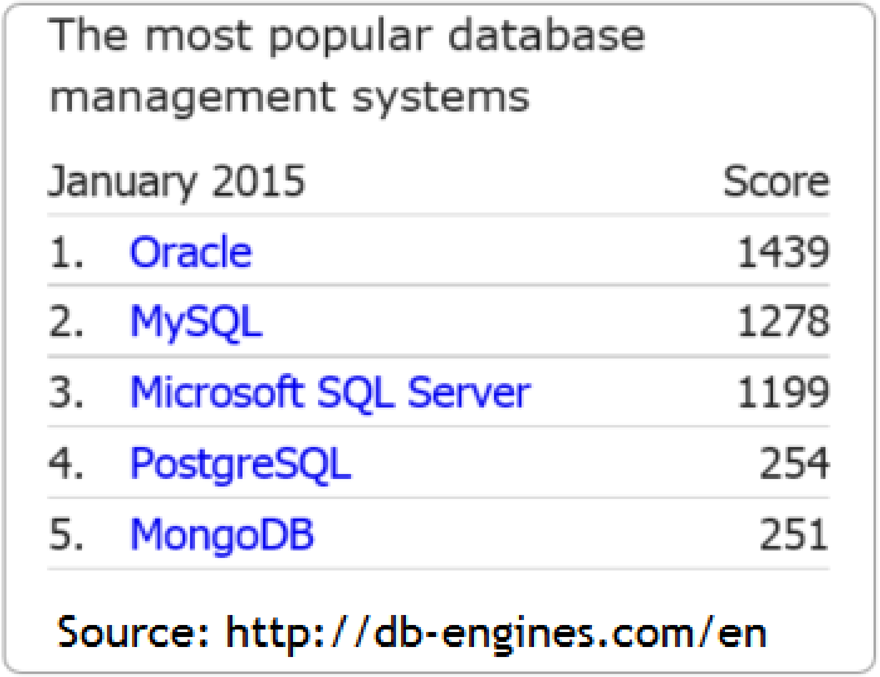

Pramodkumar J Sadalage & Martin Fowler
NoSQL
A Brief Guide to the Emerging World of Polyglot Persistence
Distilled

NoSQL Distilled by Pramod J. Sadalage and Martin Fowler, Addison Wesley, 2012

<u>Chapter 10, 11</u>

Wrapping-up and advices

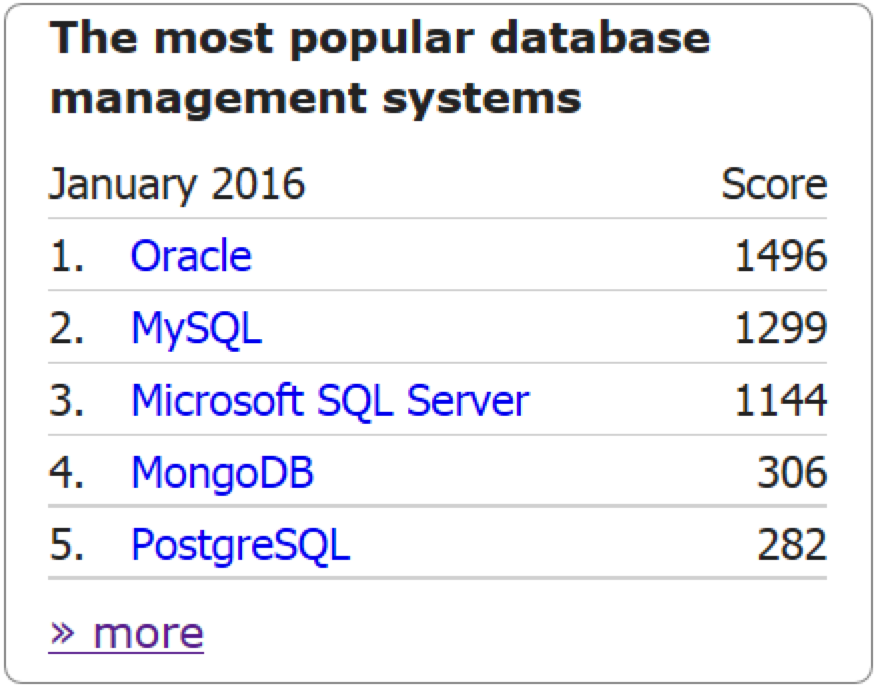
Wrapping-up databases



The most popular database management systems

January 2015		Score
1.	Oracle	1439
2.	MySQL	1278
3.	Microsoft SQL Server	1199
4.	PostgreSQL	254
5.	MongoDB	251

Source: http://db-engines.com/en



The most popular database management systems

January 2016		Score
1.	Oracle	1496
2.	MySQL	1299
3.	Microsoft SQL Server	1144
4.	MongoDB	306
5.	PostgreSQL	282

» more