POLI [TECH> NIKA Czestochowa University of Technology

[@> Faculty of Computer Science and Artificial Intelligence

SCRIPTING LANGUAGES IN WEB APPLICATIONS

L00 – Introduction

Shared Nothing Architectures for Data Analysis

Shared-nothing architecture

The most general and scalable

Set-oriented

Each dataset D is represented by a table of rows

Key-value

Each row is represented by a <key, value> pair, where
Key uniquely identifies the value in D
Value is a list of (attribute name : attribute value)

Can represent structured (relational) data or NoSQL data

But graph is another story (see Pregel or DEX)

Examples

- *<row-id#5, (part-id:5, part-name:iphone5, supplier:Apple)>*
- *<doc-id#10, (content:<html> html text ... </html>)>*
- *<akeyword,(doc-id:id1,doc-id:id2,doc-id:id10)*

Shared Nothing Architectures for Data Analysis

Big datasets

- Data partitioning and indexing
  - Problem with skewed data distributions
- Parallel algorithms for algebraic operators
  - Select is easy, Join is difficult
- Disk is very slow (10K times slower than RAM)
  - Exploit main memory data structures and compression

Query parallelization and optimization

- Automatic if the query language is declarative (e.g. SQL)
- Programmer assisted otherwise (e.g. MapReduce)

Transaction support

- Hard: need for distributed transactions (distributed locks and 2PC)
  - NoSQL systems don't provide transactions

Fault-tolerance and availability

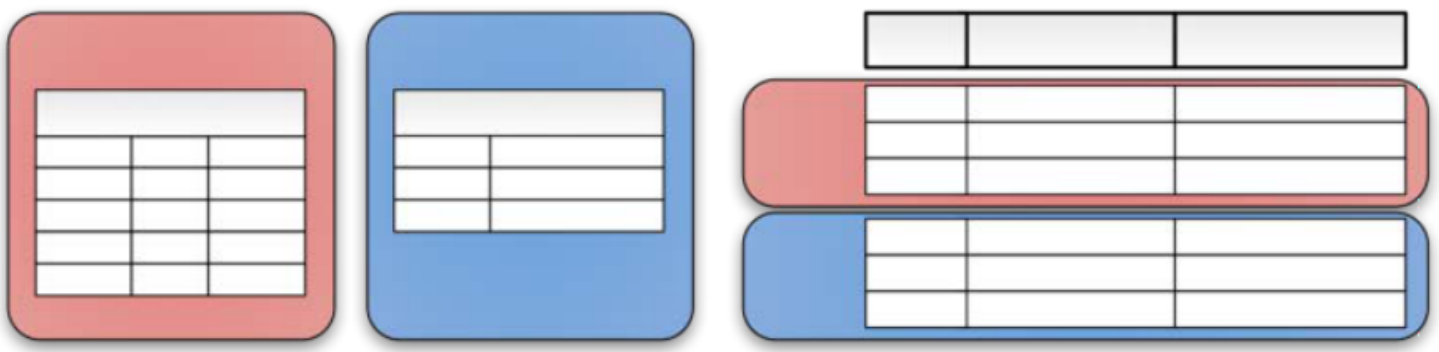
- With many nodes (e.g. several thousand), node failure is the norm

Data Partitioning

Horizontal Partitioning (sharding)

Vertical Partitioning

Key-Value partitioning



Vertical Horizontal

Horizontal Partitioning (sharding)

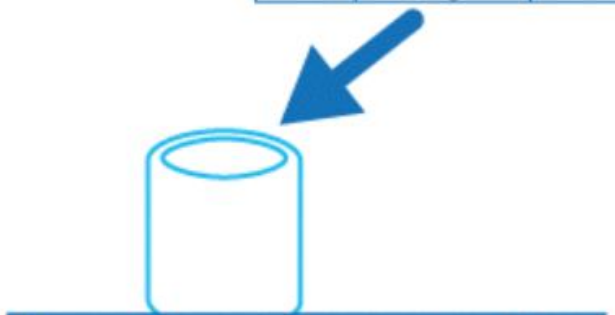
Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013

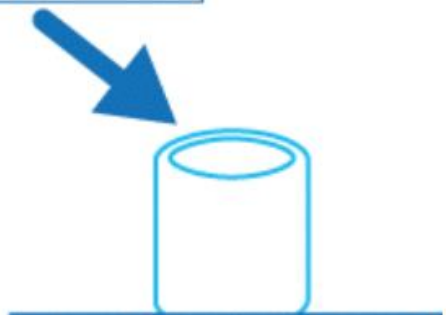
Key	Name	Description	Stock	Price	LastOrdered
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Vertical Partitioning

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

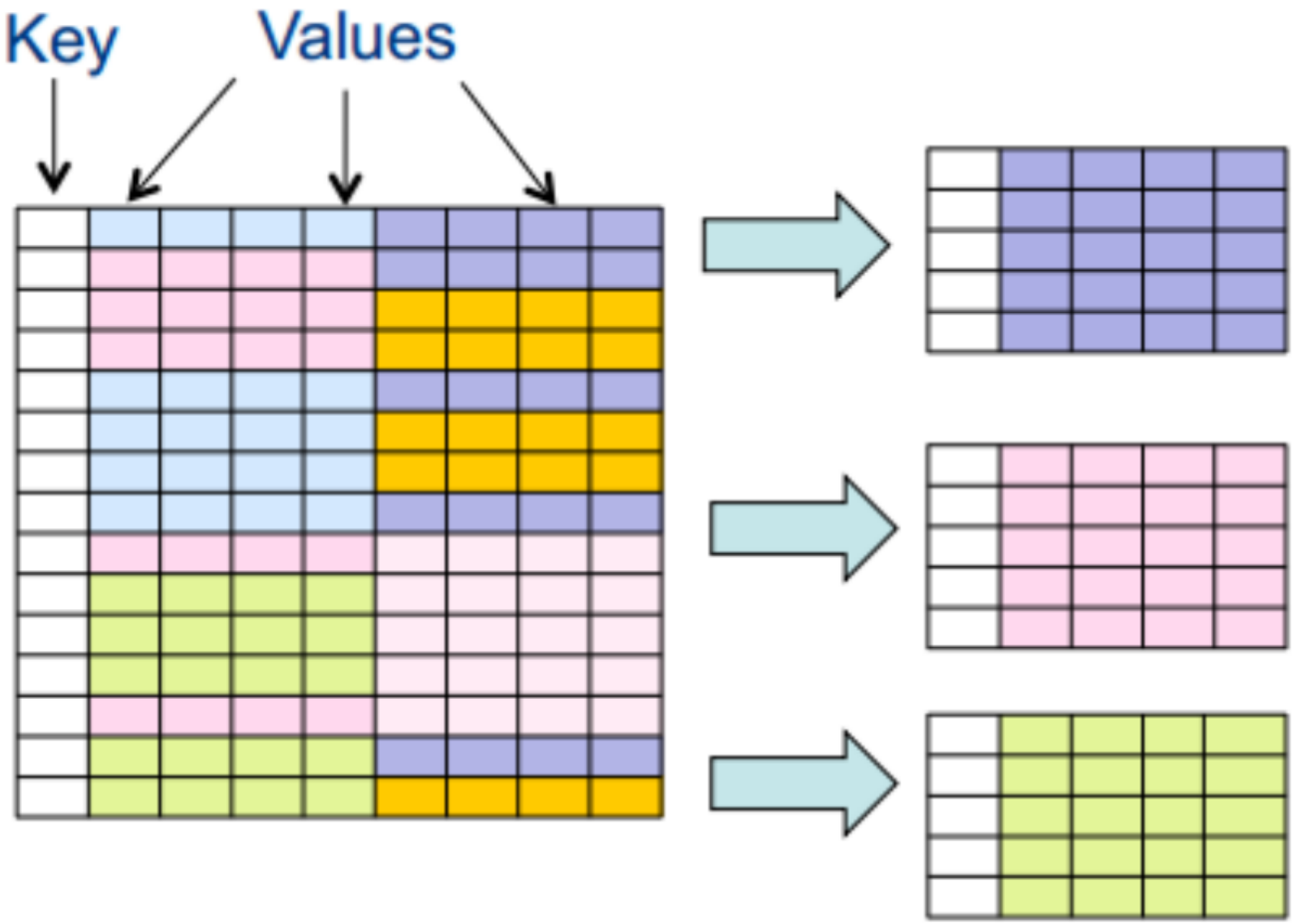


Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99



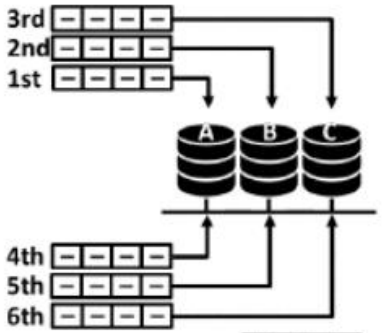
Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

Key-Value Partitioning

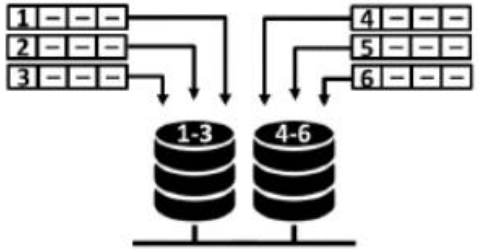


Partition distribution schemes

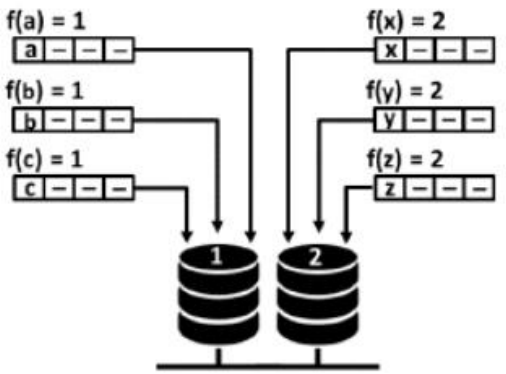
Round Robin



Range driven

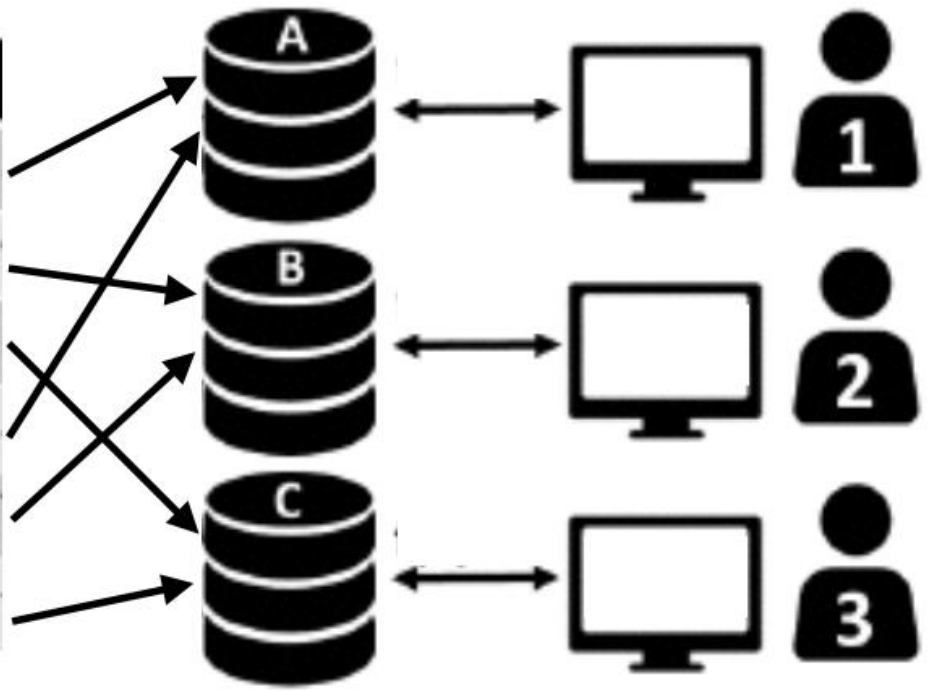


Hash driven



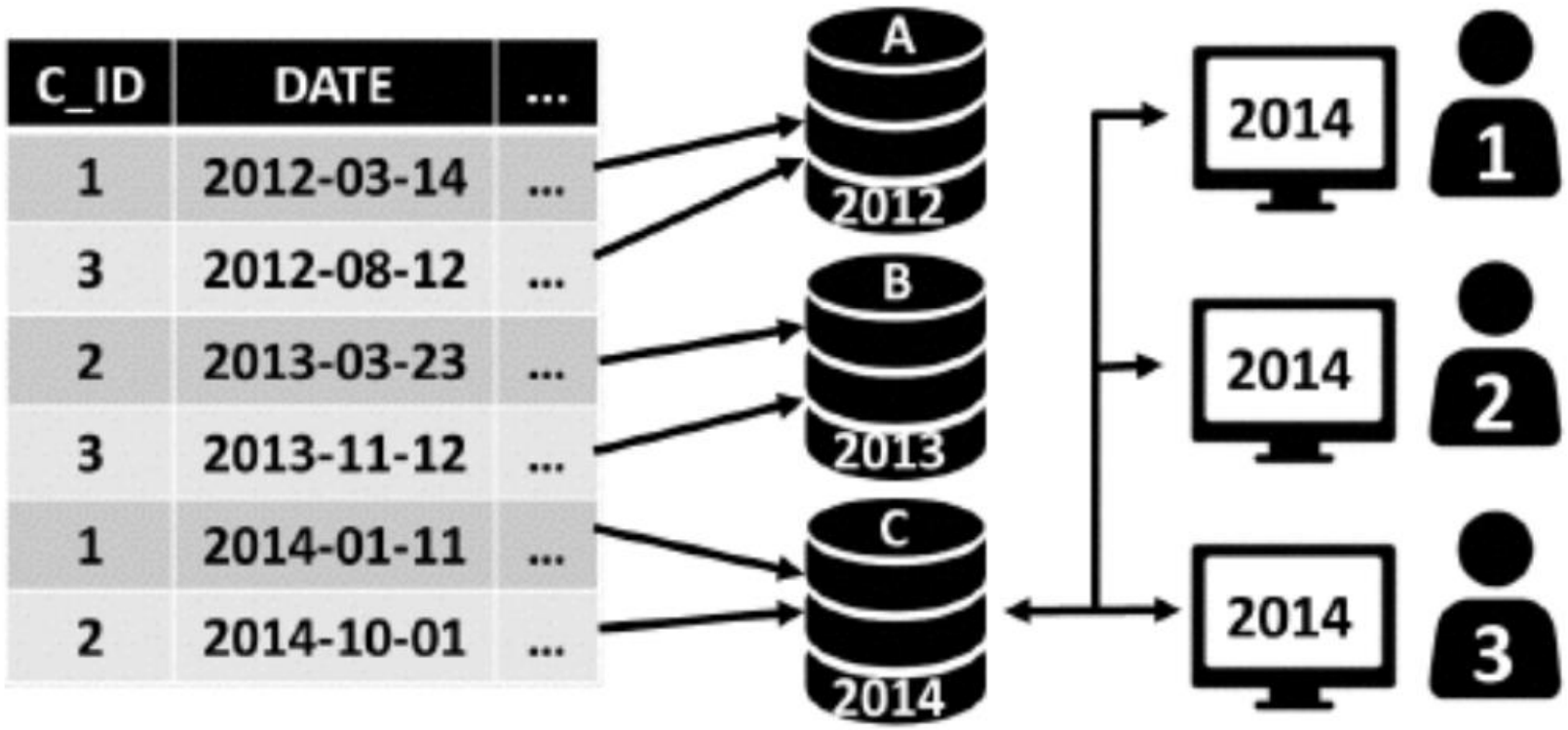
Partition distribution schemes

	C_ID	DATE	...
f(1) = A	1	2012-03-14	...
f(3) = C	3	2012-08-12	...
f(2) = B	2	2013-03-23	...
f(3) = C	3	2013-11-12	...
f(1) = A	1	2014-01-11	...
f(2) = B	2	2014-10-01	...



Round Robin

Partition distribution schemes



Range Driven

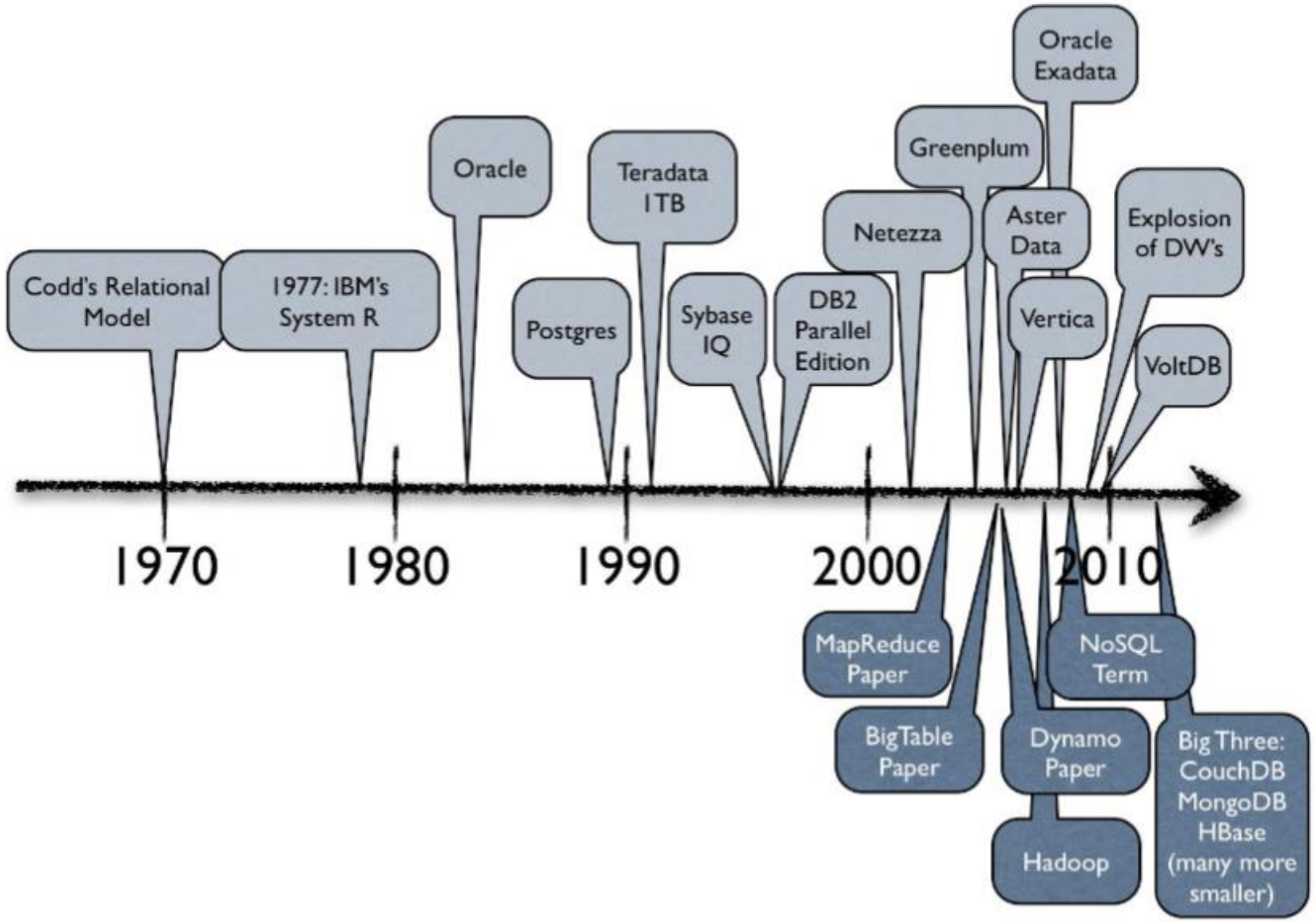
Partition distribution schemes

Hash Driven

From SQL to NoSQL

- Database - Organized collection of data
- DBMS - Database Management System: a software package with computer programs that controls the creation, maintenance and use of a database
- Databases are created to operate large quantities of information by inputting, storing, retrieving, and managing that information.

From SQL to NoSQL

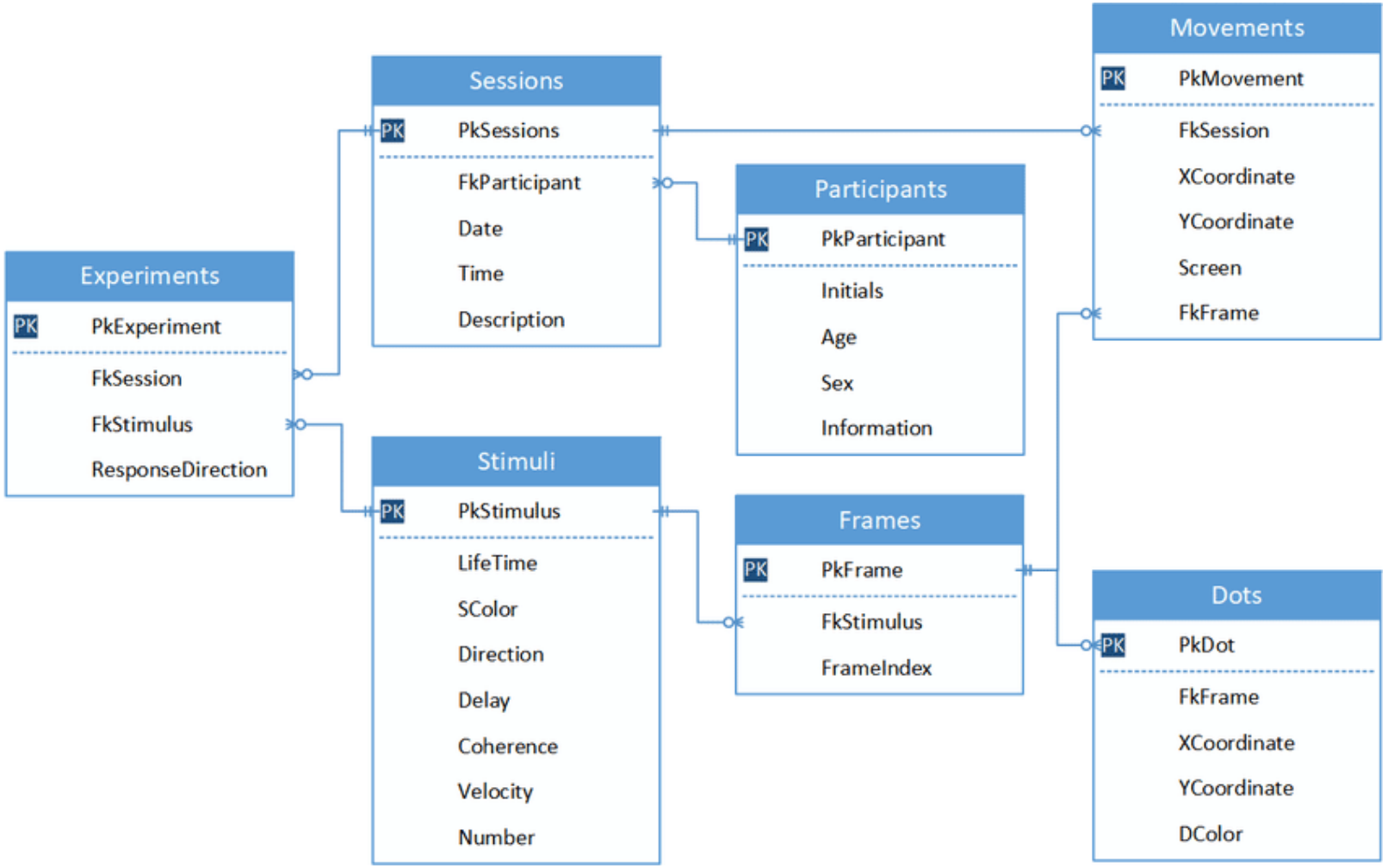


Relational databases

- Benefits of Relational databases:

- Designed for all purposes
- ACID
- Strong consistancy, concurrency, recovery
- Mathematical background
- Standard Query language (SQL)
- Lots of tools to use with i.e: Reporting services, entity frameworks, ...

Relational databases



Relational databases

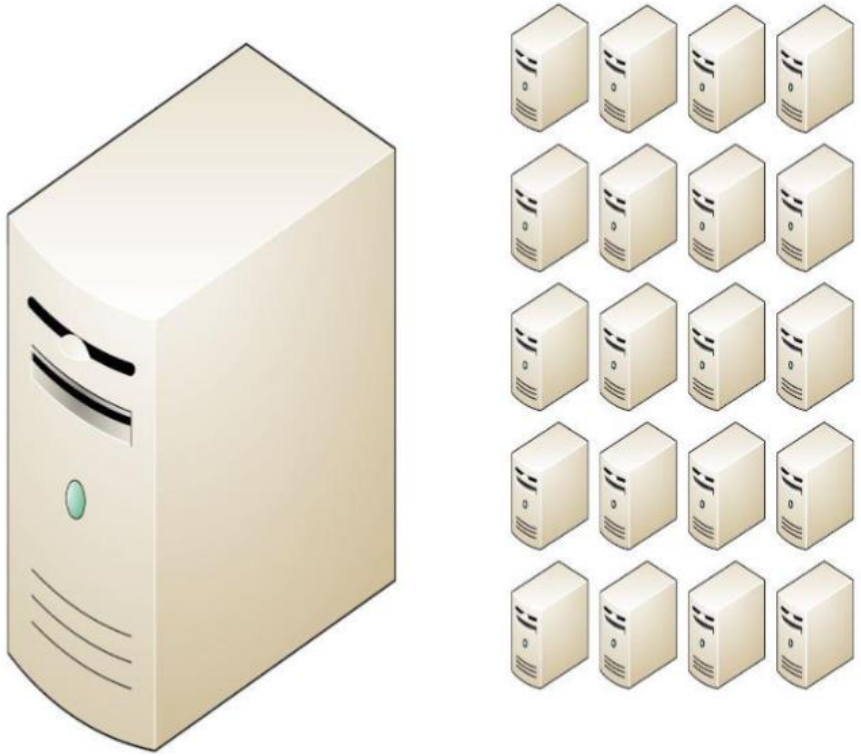
But...

- Relational databases were not built for **distributed applications.**

Because...

- Joins are expensive
- Hard to scale horizontally
- Impedance mismatch occurs
- Expensive (product cost, hardware, Maintenance)

Era of Distributed Computing



Relational databases

But...

- Relational databases were not built for **distributed applications.**

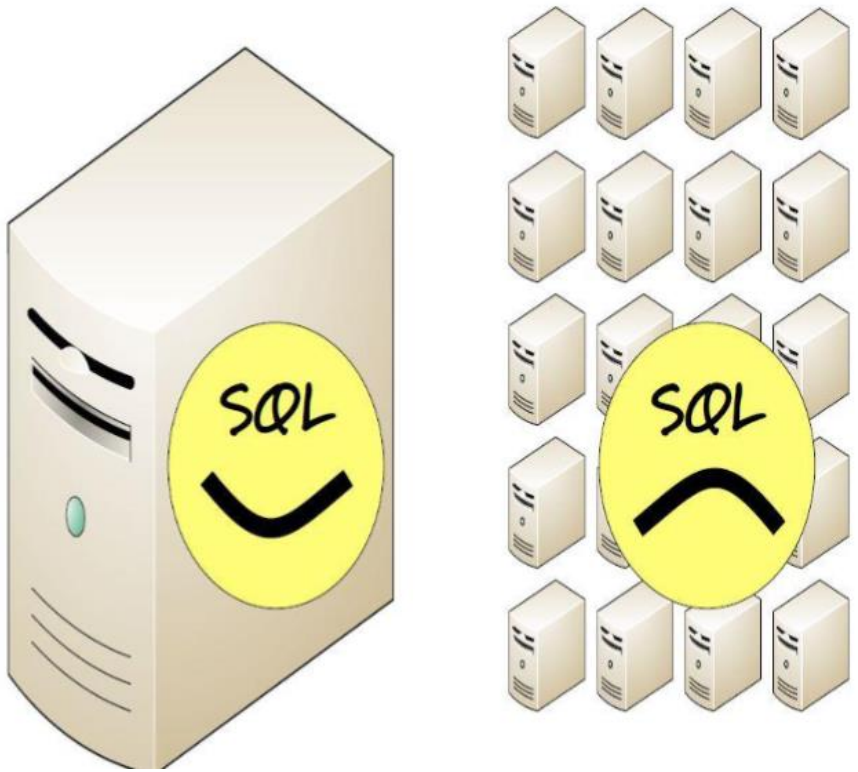
Because...

- Joins are expensive
- Hard to scale horizontally
- Impedance mismatch occurs
- Expensive (product cost, hardware, Maintenance)

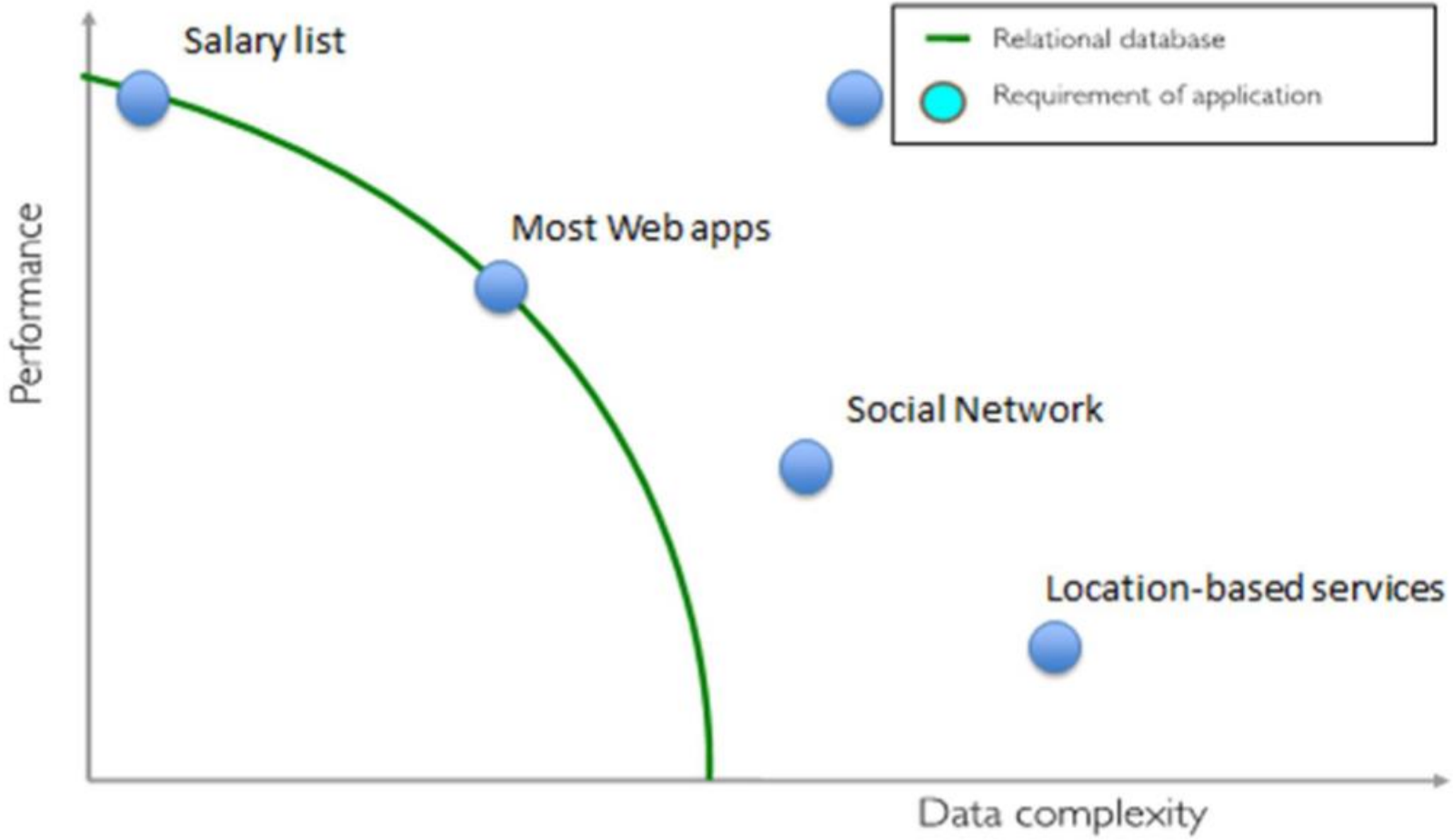
And....

It's weak in:

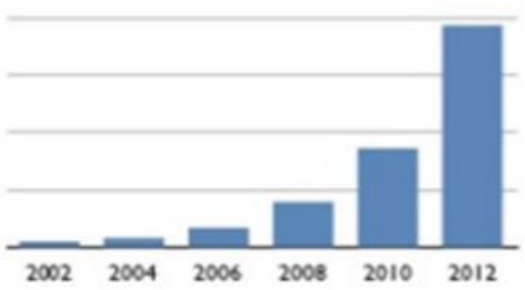
- Speed (performance)
- High availability
- Partition tolerance



Relational databases



New trends



Big data

Connectivity

P2P Knowledge

Concurrency

Diversity

Cloud-Grid

Not Only SQL

- A No SQL database provides a mechanism for storage and retrieval of data that employs less constrained consistency models than traditional relational database

- No SQL systems are also referred to as "NotonlySQL“ to emphasize that they do in fact allow SQL-like query languages to be used.

Not Only SQL

NoSQL avoids:

- Overhead of ACID transactions
- Complexity of SQL query
- Burden of up-front schema design
- DBA presence
- Transactions (It should be handled application layer)

Provides:

- Easy and frequent changes to DB
- Fast development
- Large data volumes(eg.Google)
- Schema less

Not Only SQL

When and when not to use it?

WHEN / WHY ?

- When traditional RDBMS model is too restrictive (flexible schema)
- When ACID support is not "really" needed
- Object-to-Relational (O/R) impedance
- Because RDBMS is neither distributed nor scalable by nature
- Logging data from distributed sources
- Storing Events / temporal data
- Temporary Data (Shopping Carts / Wish lists / Session Data)
- Data which requires flexible schema
- **Polyglot Persistence** i.e. best data store depending on nature of data.

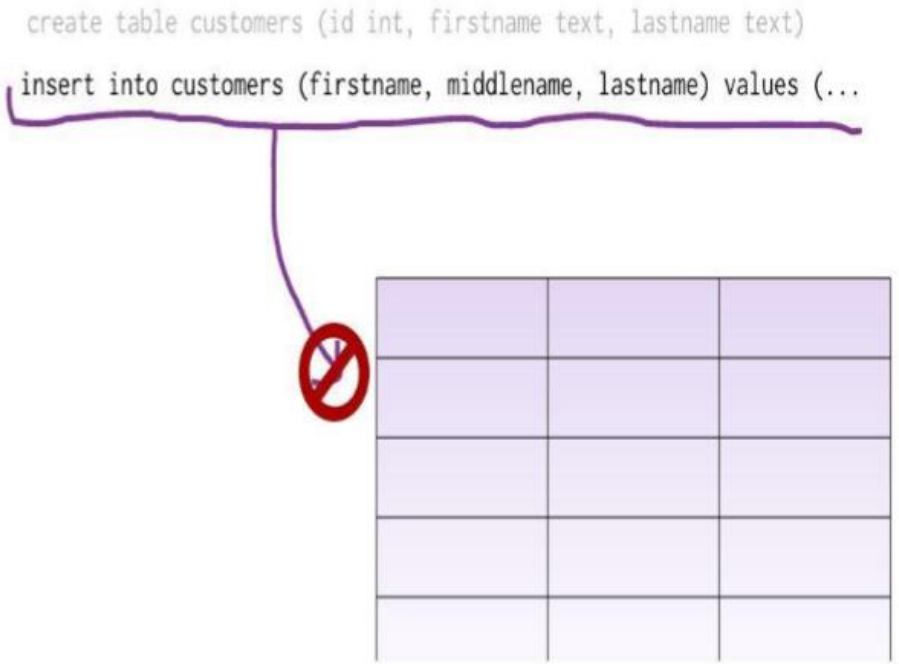
WHEN NOT ?

- Financial Data
- Data requiring strict ACID compliance
- Business Critical Data

Schema-less models

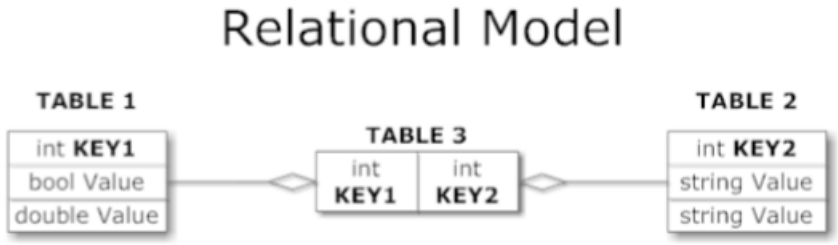
In relational Databases:

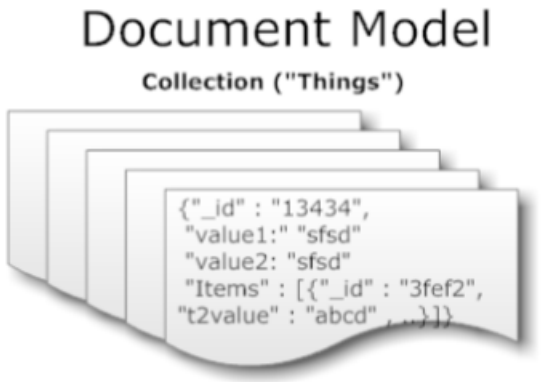
- You can't add a record which does not fit the schema
- You need to add NULLs to unused items in a row
- We should consider the datatypes. i.e : you can't add a stirng to an interger field
- You can't add multiple items in a field (You should create another table: primary-key, foreign key, joins, normalization, ... !!!)



Schema-less models

In NoSQL Databases:





- There is no schema to consider
- There is no unused cell
- There is no datatype (implicit)
- Most of considerations are done in application layer
- We gather all items in an aggregate (document)

NoSQL data models

NoSQL databases are classified in four major datamodels:

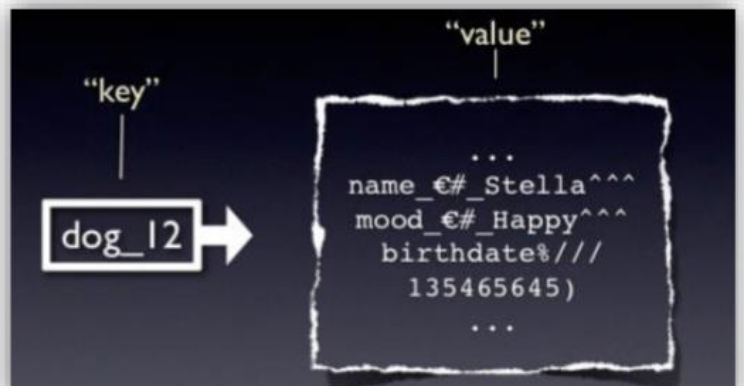
- Key-value
- Document
- Column family
- Graph

Each DB has its own query language

NoSQL data models: key-value

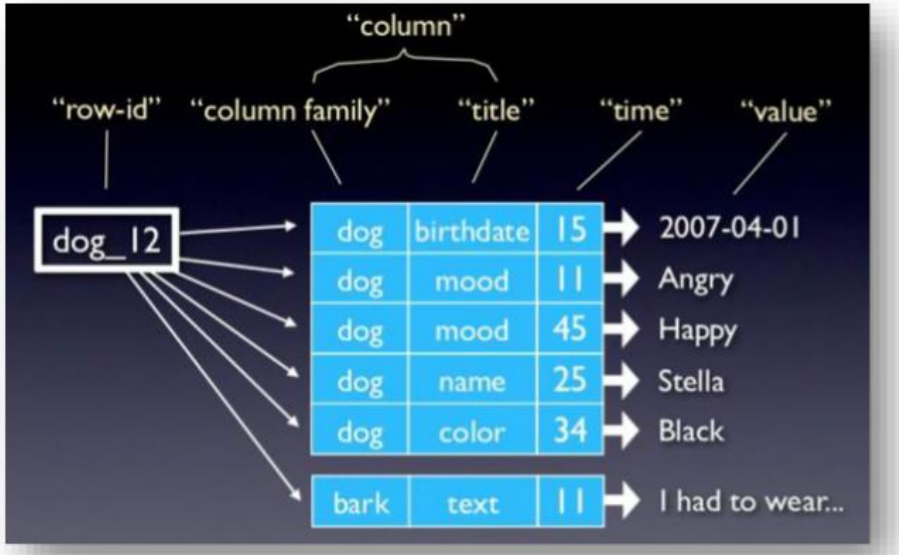
- Simplest NOSQL databases
- The main idea is the use of a hash table
- Access data (values) by strings called keys
- Data has no required format data may have any format
- Data model: (key, value) pairs
- Basic Operations:
  Insert(key,value),
  Fetch(key),

Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year:2005 Transmission: Auto



NoSQL data models: column family

- The column is lowest/smallest instance of data.
- It is a tuple that contains a name, a value and a timestamp



ColumnFamily: Authors

Key	Value
"Eric Long"	Columns: Name / Value — "email": "eric (at) long.com"; "country": "United Kingdom"; "registeredSince": "01/01/2002"
"John Steward"	Columns: Name / Value — "email": "john.steward (at) somedomain.com"; "country": "Australia"; "registeredSince": "01/01/2009"
"Ronald Mathies"	Columns: Name / Value — "email": "ronald (at) sodeso.nl"; "country": "Netherlands, The"; "registeredSince": "01/01/2010"

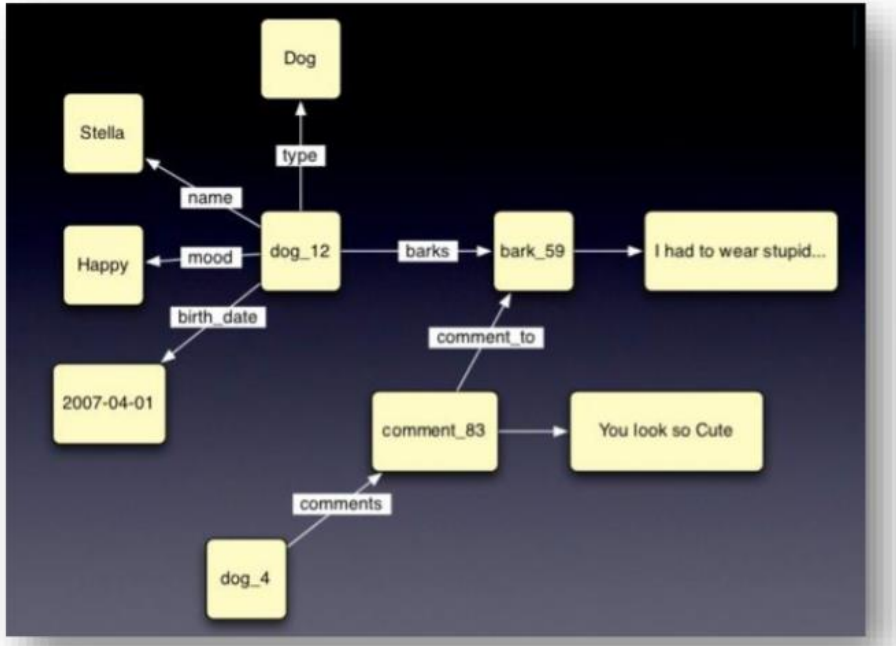
NoSQL data models: column family

Some statistics about Facebook Search (using **Cassandra**)

- MySQL > 50 GB Data
  - Writes Average : ~300 ms
  - Reads Average : ~350 ms
- Rewritten with Cassandra > 50 GB Data
  - Writes Average : 0.12 ms
  - Reads Average : 15 ms

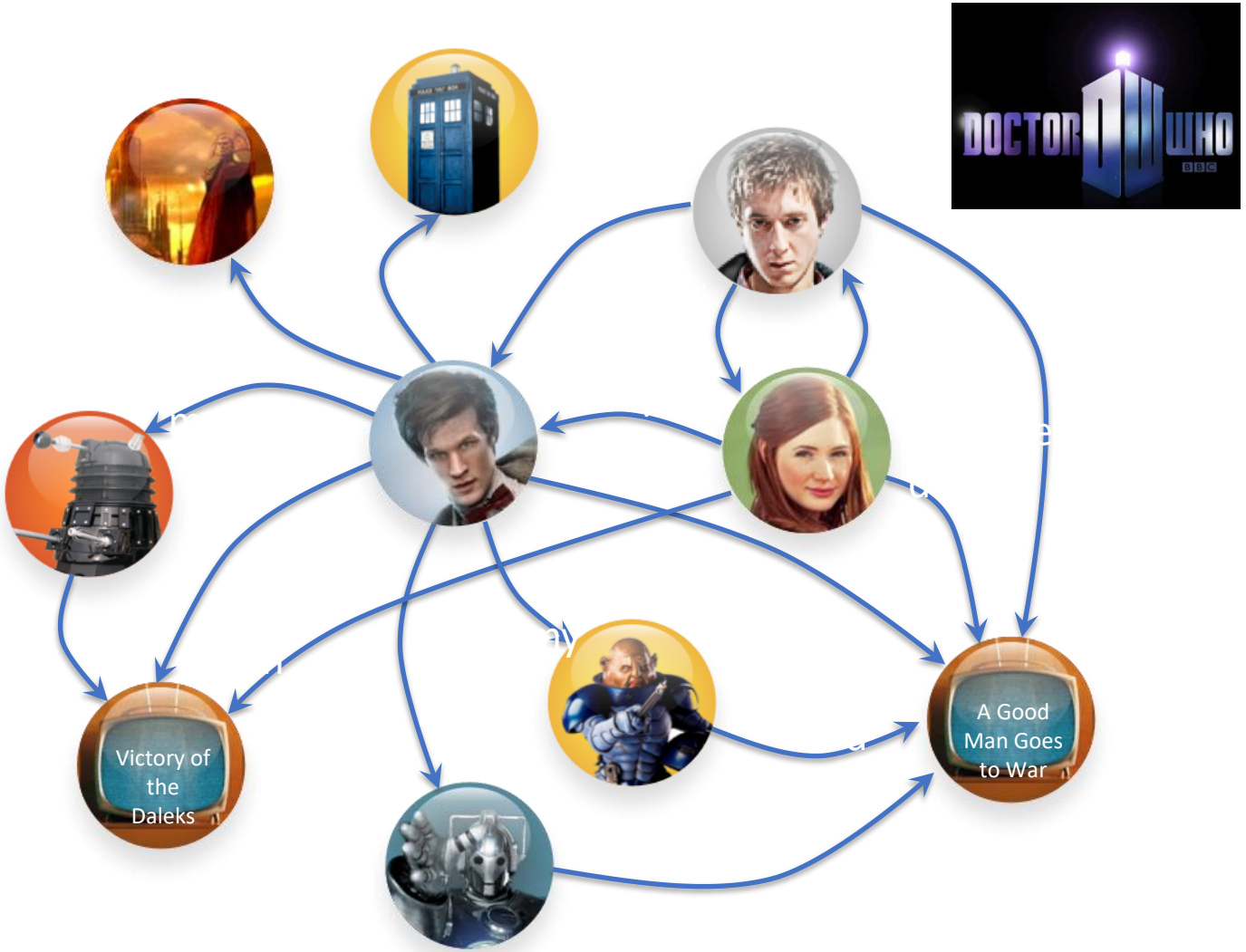
NoSQL data models: graph based

- Based on Graph Theory.
- Scale vertically, no clustering.
- You can use graph algorithms easily
- Transactions
- ACID

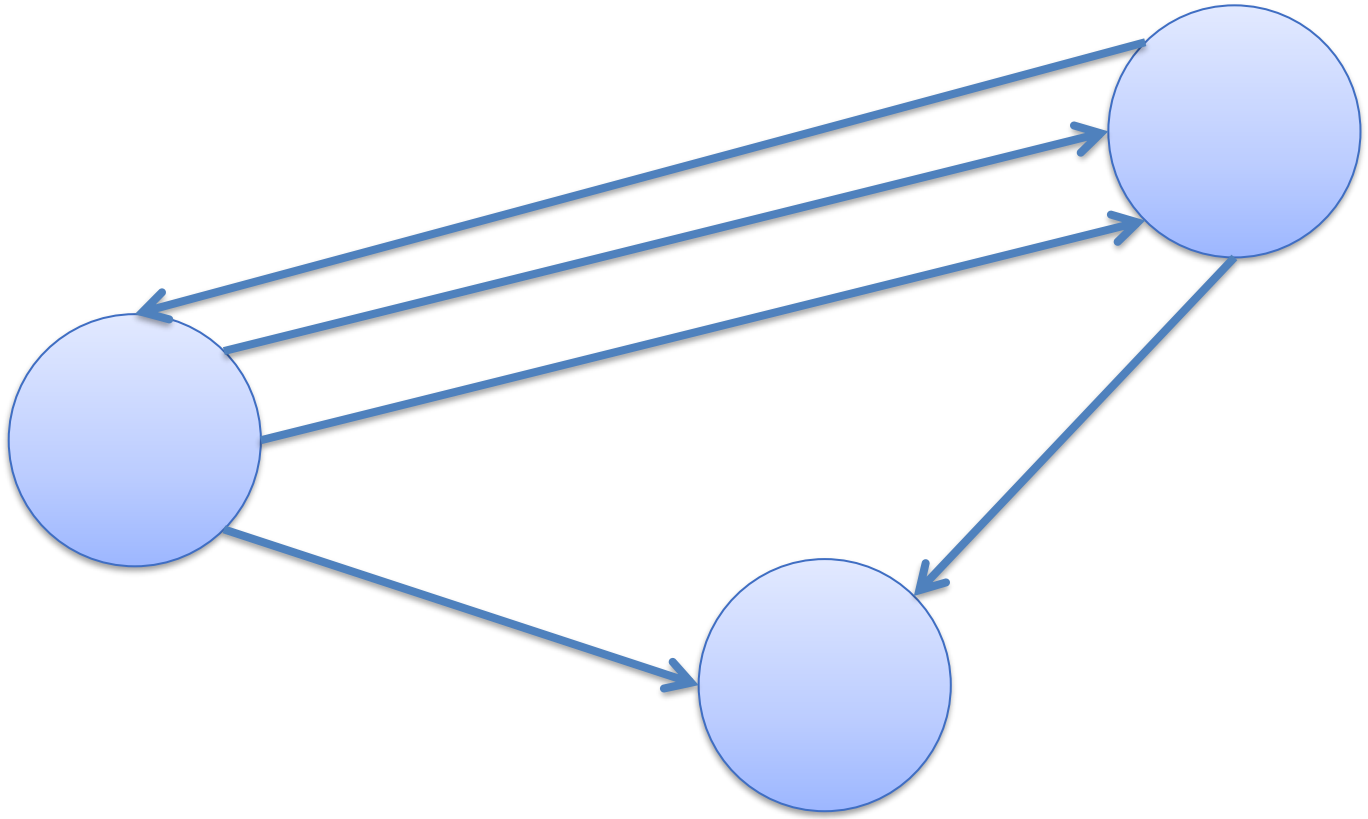


neo4j

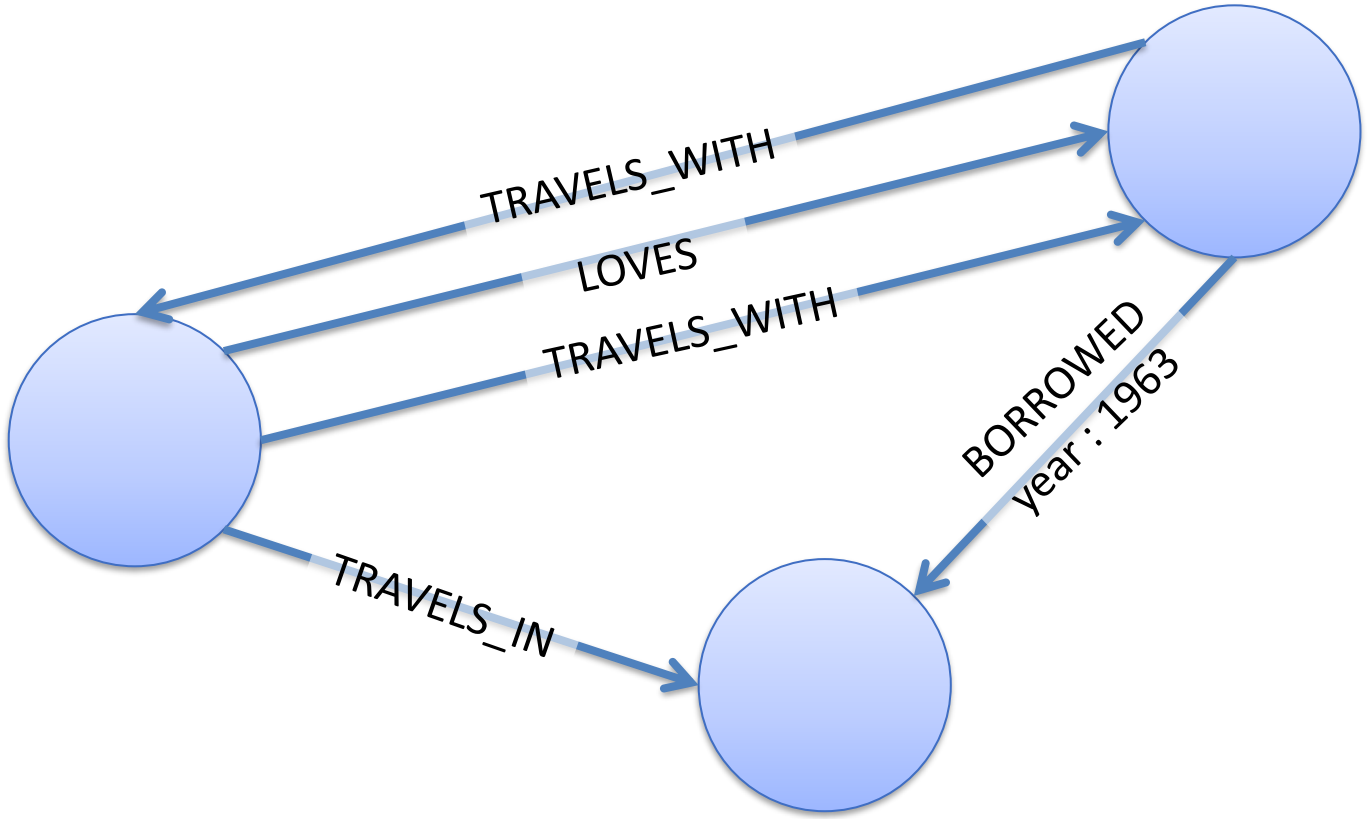
From NoSQL to graphs



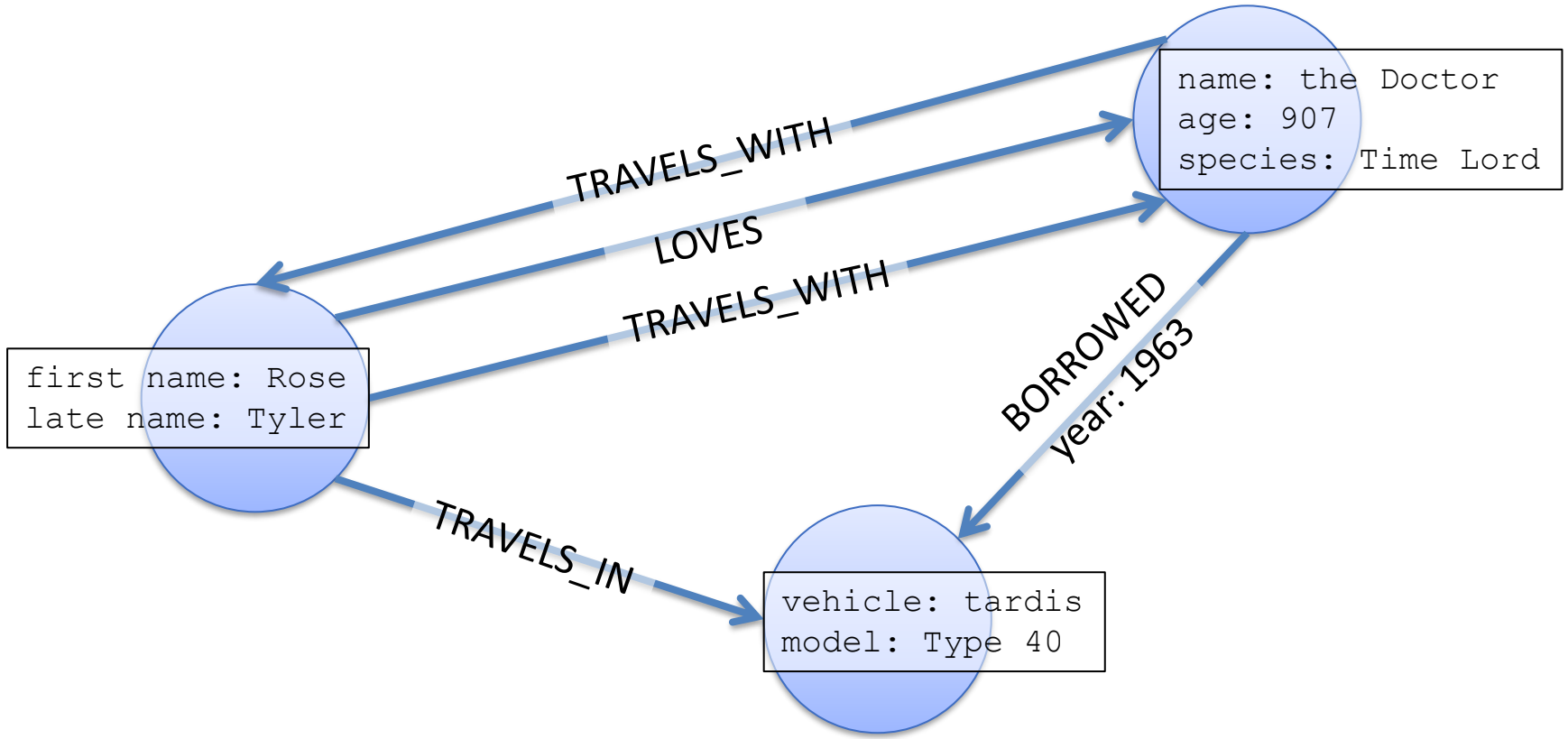
From NoSQL to graphs



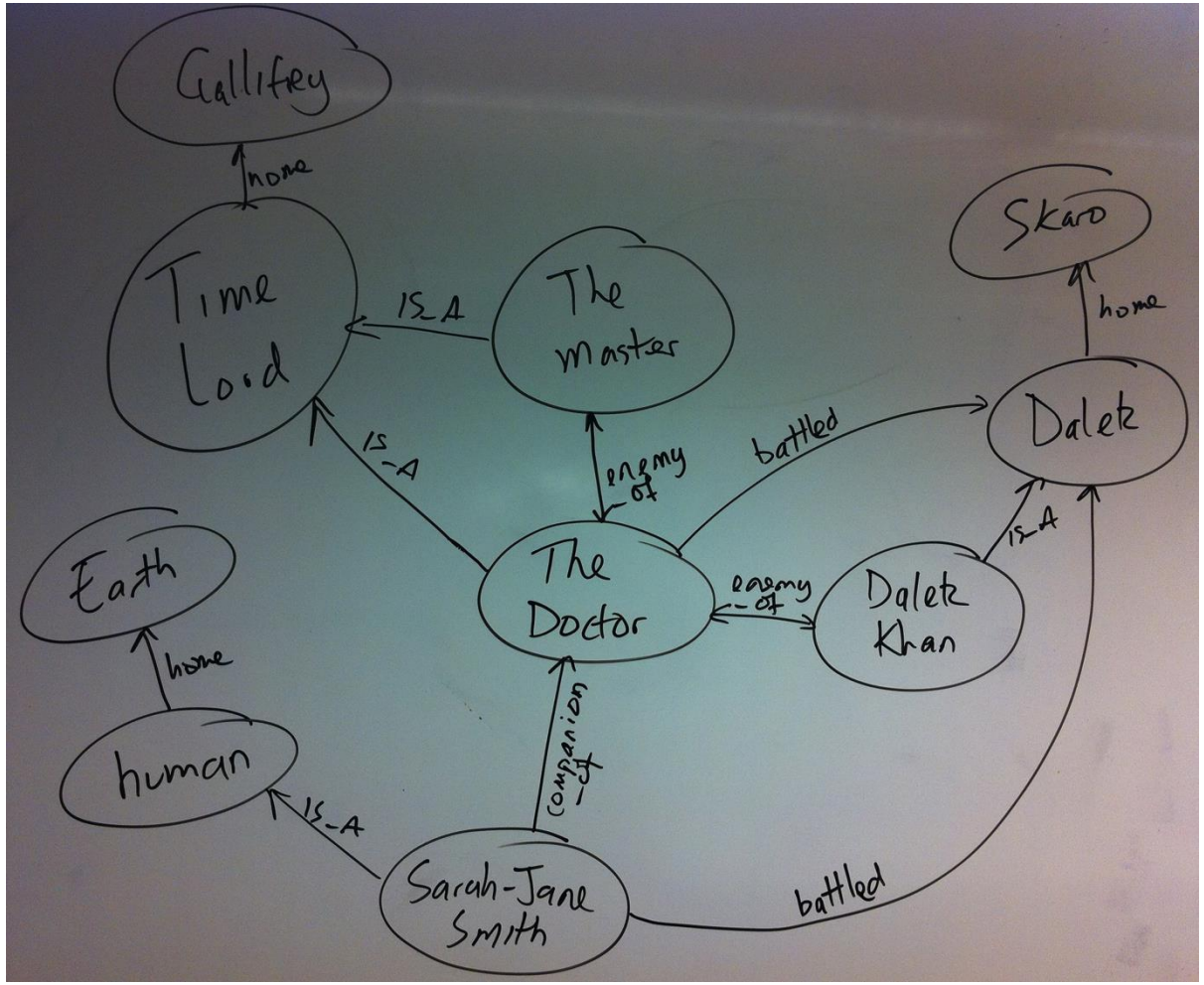
From NoSQL to graphs



From NoSQL to graphs



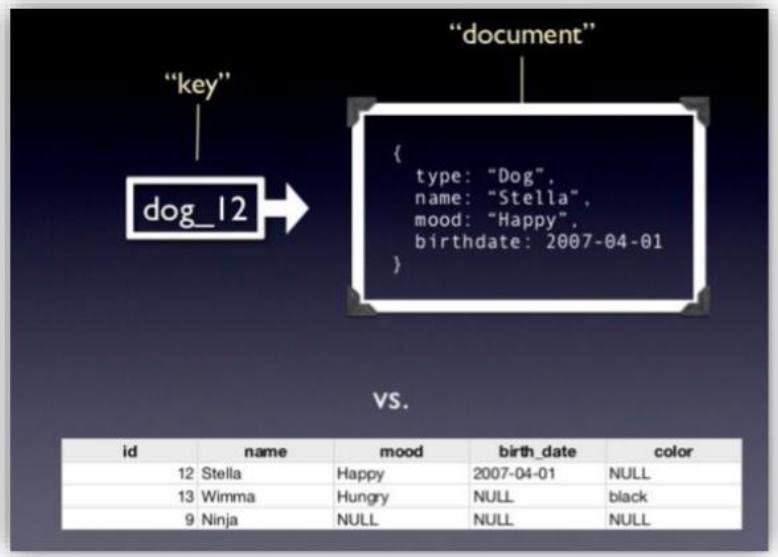
From NoSQL to graphs

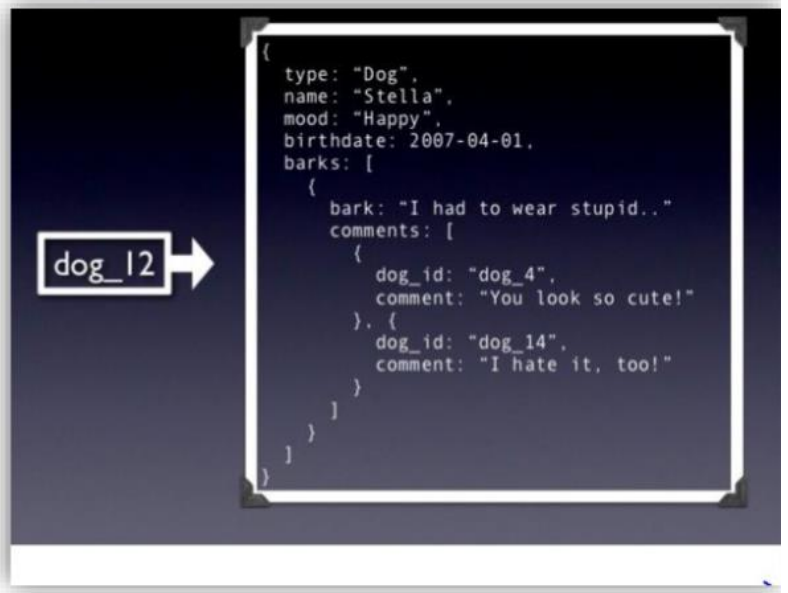


NoSQL data models: document based

- Pair each key with complex data structure known as data structure.
- Indexes are done via B-Trees.
- Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

```
{
  person: {
    first_name: "Peter",
    last_name: "Peterson",
    addresses: [
      {street: "123 Peter St"},
      {street: "504 Not Peter St"}
    ],
  }
}
```

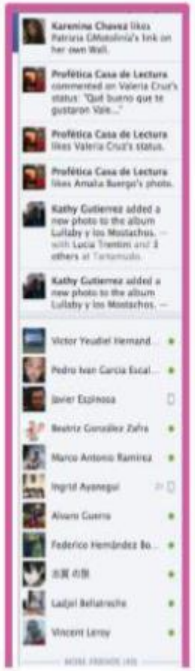




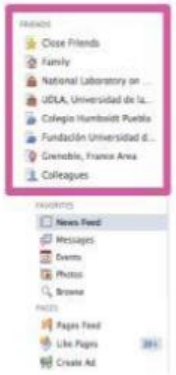
NoSQL data models: document based

```
SELECT  name, pic, profile_url
FROM    user
WHERE   uid = me()
```


```
SELECT  message, attachment
FROM    stream
WHERE   source_id = me() AND type = 80
```



```
SELECT  name
FROM    friendlist
WHERE   owner = me()
```



```
SELECT  name
FROM    group
WHERE   gid IN ( SELECT  gid
                 FROM    group_member
                 WHERE   uid = me() )
```

```
SELECT  name, pic
FROM    user
WHERE   online_presence = "active"
        AND
        uid IN ( SELECT  uid2
                 FROM    friend
                 WHERE   uid1 = me() )
```

SQL and NoSQL databases and models

	SQL Databases	No SQL Database
Example	Oracle , mysql	Mondo DB, CouchDB, Neo4J
Storage Model	Rows and tables	Key-value. Data stored as single document in JSON, XML
Schemas	Static	Dynamic
Scaling	Vertical & Horizontal	Horizontal
Transactions	Yes	Certain levels
Data Manipulation	Select, Insert , Update	Through Object Oriented API's

The Benefits of NoSQL

When compared to relational databases, NoSQL databases are **more scalable and provide superior performance**, and their data model addresses several issues that the relational model is not designed to address:

- Geographically distributed architecture instead of expensive, monolithic architecture
- Large volumes of rapidly changing structured, semi-structured, and unstructured data
- Agile sprints, quick schema iteration, and frequent code pushes
- Object-oriented programming that is easy to use and flexible

Database Scaling

- RDBMS are "scaled up" by adding hardware processing power
- NoSQL is "scaled out" by spreading the load
  - Partitioning (sharding) / replication

NoSQL Database Types

- **Graph stores** are used to store information about networks of data, such as social connections. Graph stores include Neo4J and triple stores like Fuseki.
- **Document databases** pair each key with a complex data structure known as a document.
- **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value. Examples of key-value stores are Riak and Berkeley DB.
- **Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

Document Store

- The central concept is the notion of a "document“ which corresponds to a row in RDBMS.
- A document comes in some standard formats like JSON (BSON).
- Documents are addressed in the database via a unique *key* that represents that document.
- The database offers an API or query language that retrieves documents based on their contents.
- Documents are schema free, i.e., different documents can have structures and schema that differ from one another. (An RDBMS requires that each row contain the same columns.)

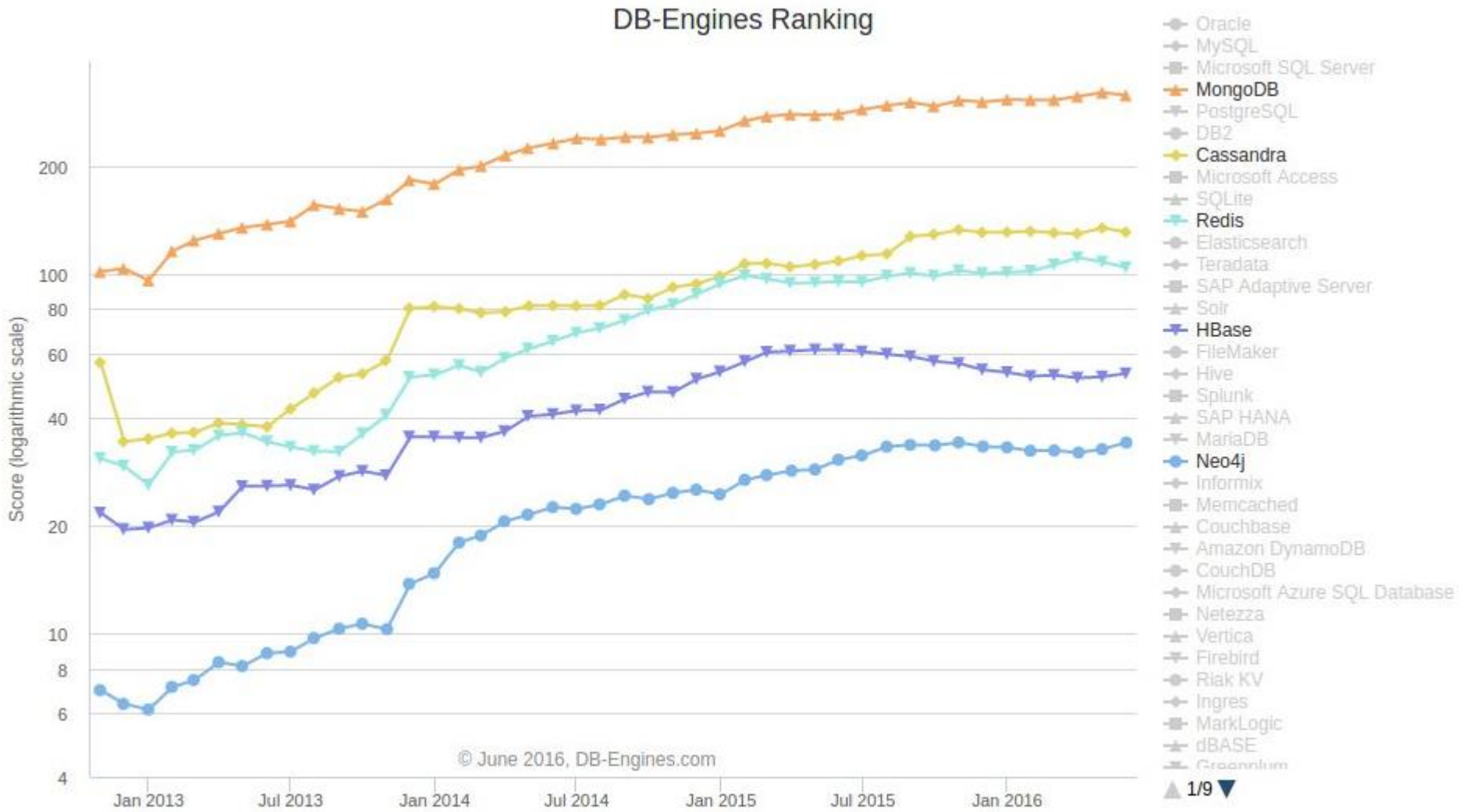
MongoDB to documents (JSON):

```
{
        _id: ObjectId("51156a1e056d6f966f268f81"),
        type: "Article",
        author: "Derick Rethans",
        title: "Introduction to Document Databases with MongoDB",
        date: ISODate("2013-04-24T16:26:31.911Z"),
        body: "This arti…"
},
{
        _id: ObjectId("51156a1e056d6f966f268f82"),
        type: "Book",
        author: "Derick Rethans",
        title: "php|architect's Guide to Date and Time Programming with PHP",
        isbn: "978-0-9738621-5-7"
}
```

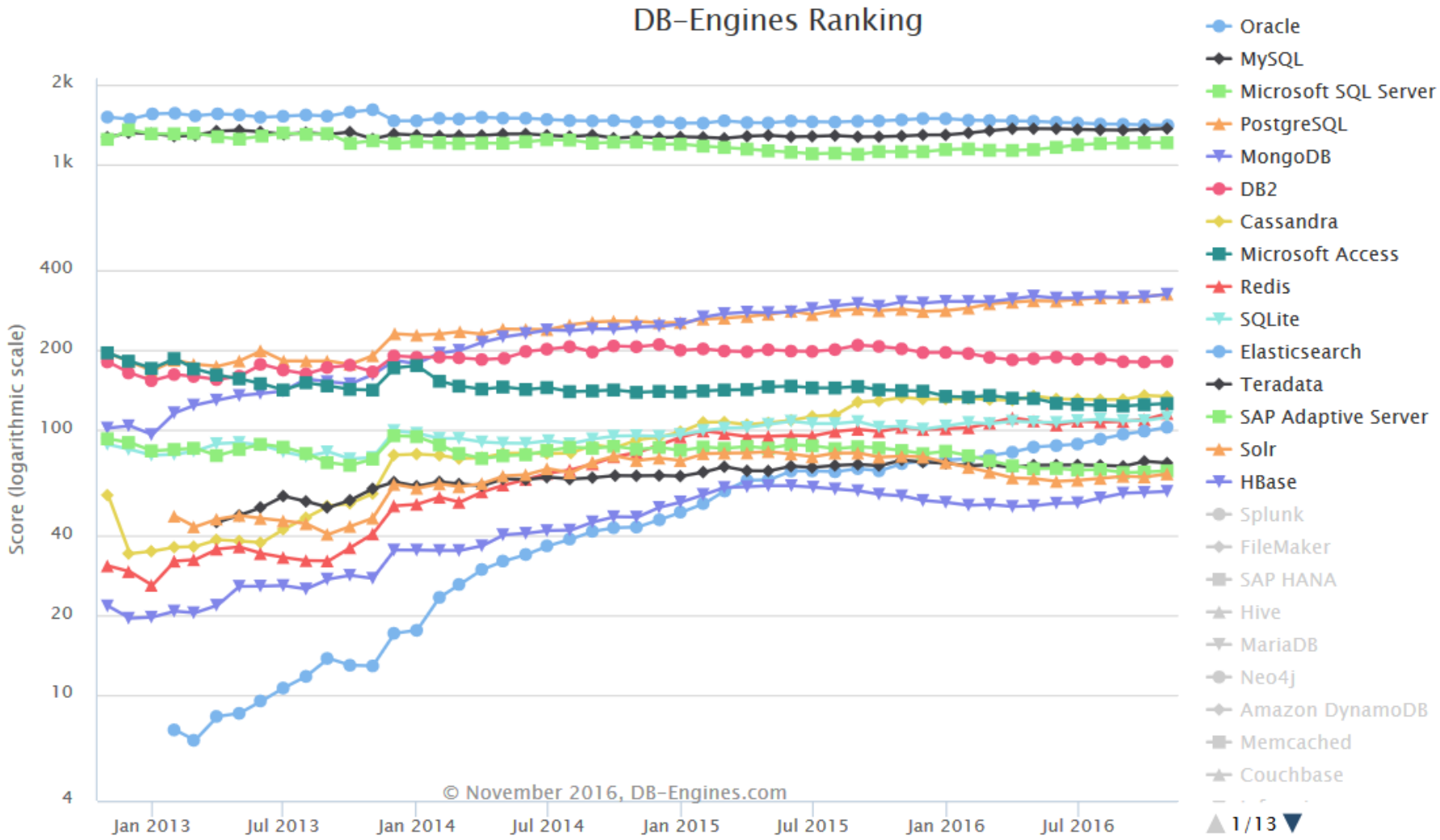
Most popular NoSQL database?

Asking “what NoSQL database is the most popular” is a bit incorrect since different problems require different types of NoSQL solutions. …focus on solving very specific problems. While this allows to achieve the best possible results in those specific cases, it comes at a cost of some other functionalities.

Most popular NoSQL database?



Most popular NoSQL database?



MongoDB

- A document-oriented database
  - documents encapsulate and encode data in some standard formats
- NoSQL database
  - non-adherence to the widely used relational database systems
  - highly optimized for retrieve and append operations
- It uses BSON format
- It is schema-less
  - No more configuring database columns with types
  - Documents are self-describing
  - Documents are analogous to structures in programming languages that associate keys with values
  - The values of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents
- No transactions
- No joins

```
{
  name: "sue",                          <-- field: value
  age: 26,                              <-- field: value
  status: "A",                          <-- field: value
  groups: [ "news", "sports" ]          <-- field: value
}
```

MongoDB

- MongoDB is an open source document store written in C++
- provides indexes on collections
- lockless
- provides a document query mechanism
- supports automatic sharding
- Replication is mostly used for failover
- does not provide the global consistency of a traditional DBMS
  - but you can get local consistency on the up-to-date primary copy of a document
- supports dynamic queries with automatic use of indices, like RDBMSs
- also supports map-reduce – helps complex aggregations across docs
- provides atomic operations on fields

MongoDB: Atomic Ops on Fields

- The update command supports “modifiers” that facilitate atomic changes to individual values
  - $set sets a value
  - $inc increments a value
  - $push appends a value to an array
  - $pushAll appends several values to an array
  - $pull removes a value from an array, and $pullAll removes several values from an array
- Since these updates normally occur “in place”, they avoid the overhead of a return trip to the server
- There is an “update if current” convention for changing a document only if field values match a given previous value
- MongoDB supports a findAndModify command to perform an atomic update and immediately return the updated document
  - useful for implementing queues and other data structures requiring atomicity

MongoDB: Index

- MongoDB indices are explicitly defined using an ensureIndex call
  - any existing indices are automatically used for query processing
- To find all products released last year (2015) or later costing under $100 you could write:
- db.products.find(
  {released: {$gte: new Date(2015, 1, 1,)}, price
  {'$lte': 100},})

MongoDB: Replication

- MongoDB supports master-slave replication with automatic failover and recovery
  - Replication (and recovery) is done at the level of shards
  - Replication is asynchronous for higher performance, so some updates may be lost on a crash

MongoDB: Data

- MongoDB stores data in a binary JSON-like format called BSON
  - BSON supports boolean, integer, float, date, string and binary types
  - MongoDB can also support large binary objects, eg. images and videos
  - These are stored in chunks that can be streamed back to the client for efficient delivery

JSON and BSON

JSON

- Text-based open standard for data interchange
  - Serializing and transmitting structured data
- JSON = JavaScript Object Notation
  - Derived from JavaScript scripting language
  - Uses conventions of the C-family of languages
- Filename: *.json
- Language independent

(see www.json.org)

BSON

It is binary-encoded serialization of JSON documents

JSON representation of record describing a person

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Why Document

- XML and JSON are popular for data exchange
  - Recently mainly JSON
- Data stored in document DB can be used directly

- Databases often store objects from memory
  - Using RDBMS, we must do Object Relational Mapping (ORM)
    - ORM is relatively demanding
  - JSON is much closer to structure of memory objects
    - It was originally for JavaScript objects
    - Object Document Mapping (ODM) is faster

JSON (JavaScript Object Notation)

- Very lightweight data exchange format
  - Much less verbose and easier to parse than XML
  - Increasingly used for data exchange over Web: many Web APIs use JSON to return responses/results
- Based on JavaScript
  - Conforms to JavaScript object/array syntax—you can directly manipulate JSON representations in JavaScript
- But it has gained widespread support by all programming languages

JSON vs. XML

```
[
{ "ISBN": "ISBN-10",
"price": 80.00,
"title": "Foundations of Databases",
"authors": [ “Abiteboul", "Hull", "Vianu" ],
"publisher": "Addison Wesley",
"year": 1995,
"sections":
        [{ "title": "Section 1", "sections":
                [ { "title": "Section 1.1” },
                { "title": "Section 1.2” } ] },
                { "title": "Section 2" } ] },
                …
        …
…
```

```
<bibliography>
<book ISBN="ISBN-10" price="80.00">
<title>Foundations of Databases</title>
<author>Abiteboul</author>
<author>Hull</author>
<author>Vianu</author>
<publisher>Addison Wesley</publisher>
<year>1995</year>
<section>
<title>Section 1</title>
<section><title>Section 1.1</title></section>
<section><title>Section 1.2</title></section>
</section>
<section>
<title>Section 2</title>
</section>
</book>
</bibliography>
```

JSON data model

- Two basic constructs
  - Array: comma-separated list of “things” enclosed by brackets
    - Order is important
  - Object: comma-separated set of pairs enclosed by braces; each pair consists of an attribute name (string) and a value (any “thing”)
    - Order is unimportant
    - Attribute names “should” be unique within an object
- Simple types: numbers, strings (in double quotes), and special values “true”, “false”, and “null”
- Thing = a simple value or an array or an object

JSON Schema

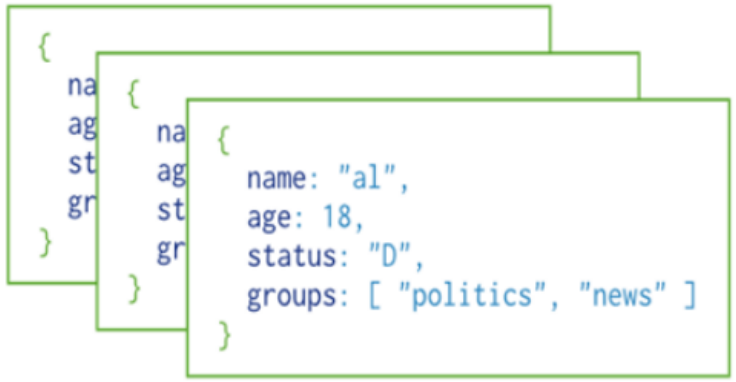
- Recall the advantages of having a schema
  - Defines a structure, helps catch errors, facilitates exchange/automation, informs optimization…
- Just like relational data and XML, JSON is getting a schema standard too!
  - Up and coming, but still a draft at this stage

```
{
 "definitions": {
  "sections": {
    "type": "array",
    "description": "Sections.",
    "sections": {"$ref":"#definitions/sections"},
    "minItems": 0
  }
 },
 "title": "Book",
 "type": "object",
 "properties": {
  "ISBN": {
    "type": "string",
    "description": "The book's ISBN number."
  },
  "price": {
    "type": "number",
    "description": "The book's price.",
    "exclusiveMinimum": 0
  },
  ... ...
  "sections": {"$ref":"#definitions/sections"},
  }
 }
 ... ...
}
```

Terminology

RDBMS	MongoDB
database instance	MongoDB instance
schema	database
table	collection
row	document
rowid	_id

- each JSON document:
  - belongs to a collection
  - has a field _id
    - unique within the collection
- each collection:
  - belongs to a “database”



Collection

Documents

- A document is a set of keys (also called field names) with associated values (no duplicate keys!!)
- Represented as JSON objects: *{"greeting" : "Hello, world!"}*
- Serialized as a BSON object
- Documents contain multiple key/value pairs: *{"greeting" : "Hello, world!", "foo" : 3}*
- Restrictions on keys:
  - The key cannot start with the $ character
    - Reserved for operators
  - The key cannot contain the . character
    - Reserved for accessing sub-fields
- Every document must have field _id
  - Used as a primary key
  - Unique within the collection
  - Immutable
  - Any type other than an array
  - Can be generated automatically
- The value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents

NOTE: the double quotes around the key can be omitted

Data Types

- *null:* can be used to represent both a null value and a nonexistent field:

  *{"x" : **null**}*

- *Boolean:* fields can be set to *true* and *false*:

  *{"x" : **true**}*

- *number:*

  If not specified the shell uses 64-bit floating point numbers

  {"x" : 3.14} or {"x" : 3}

  4-byte or 8-byte signed integers require the classes *NumberInt* or *NumberLong*

  {"x" : NumberInt("3")} or *{"x" : NumberLong("3")}*

- *string:* any sequence of UTF-8 characters

  {"x" : "foobar"}

Data Types

- *date:* stored as milliseconds :

  *{"x" : **new** Date()}*
- *regular expression:* using JavaScript's regular expression syntax:

  {"x" : /foobar/i}
- *Array:* lists of values can be represented as arrays:

  {"x" : ["a", "b", "c"]}
- *embedded document:* Documents can contain entire documents embedded as values in a field

  {"x" : {"foo" : "bar"}}

Data Types

- *binary data:* strings of arbitrary bytes
- *code:* queries and documents can include arbitrary JavaScript code: {"x" : *function*() { */* ... */* }}
- *object id:* 12-byte ID for documents.
  {"x" : ObjectId()} this method creates and returns an ObjectID
- An example: **507f1f77bcf86cd799439011**
- MongoDB documents must have an *"_id"* key.
- *"_id"* can be of any type, but it defaults to an *ObjectId*.
- In a collection, every document must have a unique value for *"_id"*

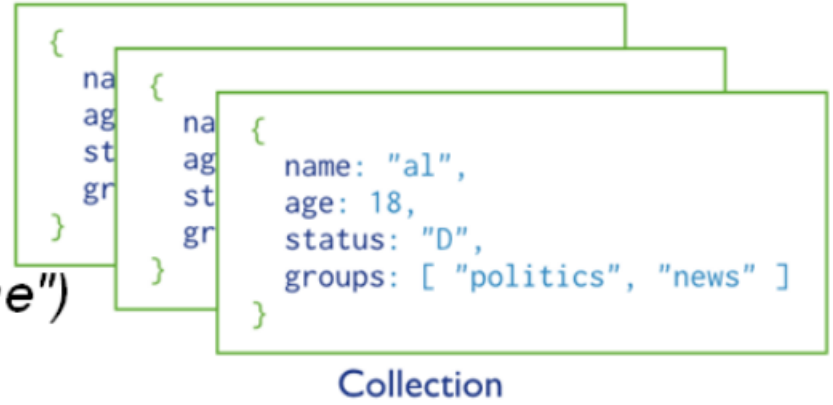
Collections

- A *collection* is a group of related documents
- Collections can be thought of as the analog to a tables in relational databases
- Collections have *dynamic schemas:*

  The documents within a single collection can have different structures

  {"greeting" : "Hello, world!"}

  {"foo" : 5}



- Collections are created using
  - the method *db.createCollection("name")*

    db.createCollection("myCollection")
  - implicitly by using it; if a collection that does not exist is referenced in a command, the system automatically creates it

    db.myCollection2.insert({("name" : "Max"})

Databases

- MongoDB groups collections into *databases*
- A single instance of MongoDB can host several databases
- Each database groups together zero or more collections
- A database has its own permissions, users and roles and each database is stored in separate files on disk
- A good rule is storing all data for a single application in the same database
- Reserved database names:
  - *admin:* the “root” database, in terms of authentication. If a user is added to the *admin* database, the user automatically inherits permissions for all databases.
  - *Config:* used for storing information of sharded setup

POLI
[TECH]
NIKA

Czestochowa
University
of Technology

[@]

Faculty of Computer Science
and Artificial Intelligence