NOSQL Systems and MongoDB

History, Features, Data Model

讚 · 追蹤 · 分享 · ···

瞭解詳情

Bill Gates
@BillGates

首頁
關於
貼文
相片
影片
網誌
活動
社群

建立粉絲專頁

貼文

Bill Gates
12月15日 23:20 ·

I wait for new Vaclav Smil books the way some people wait for the next Star Wars movie.

In his latest book, he lays out how our quest for energy has shaped human history.

翻譯年糕

How energy makes life possible
One of my favorite authors explains how energy drives human history.
WWW.GATESNOTES.COM

讚 · 留言 · 分享

5,631 · 最相關留言

409次分享

留言……
請按 Enter 鍵發表。

Evelyn Laureano-Osorio Dear Bill : I would select Smil book as my second book to read in 2018. Seems to be clear that energy consumption & economic growth are correlated strongly. I will like to explore that:, "to become rich requires a substantial increase in energy use." I love You..nice time.
翻譯年糕
讚 · 回覆 · 6 · 昨天 9:19 · 已編輯
5則回覆 · 8分鐘

Md Raihan Kabir Prince I've collected this new book by Vaclav Smil, based on your reco. I'll read this with full attention and full heart. Thanks for the tip.
翻譯年糕
讚 · 回覆 · 5 · 12月15日 23:46
1 則回覆

公眾人物

社群 查看全部
邀請朋友 對這個粉絲專頁按讚
19,516,273 人說這讚
19,344,952 個人正在追蹤

關於 查看全部
www.gatesnotes.com
公眾人物

此專頁按讚的粉絲專頁

United Nations Fou... 讚
Chan Zuckerberg I... 讚
Bill & Melinda Gate... 讚

中文(台灣) · English (US) · Español · Português (Brasil) · Français (France) +

隱私政策 · 使用條款 · 廣告 · Ad Choices · Cookie · 更多 ·
Facebook © 2017

- How to manage huge amount of unstructured data?
  - Text, video, audio
  - Various no. of responses
- Is relational DBMS a good solution?

Relational DB

- good for large amount of structured data
- offer too many services (powerful query language, concurrency control, etc), which the application may not need.

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

Overview

In one day:
24 million transactions processed by Walmart
100 TB of data uploaded to Facebook
175 million tweets on Twitter
……

How to **store**, **query** and **process** these data efficiently?

Overview

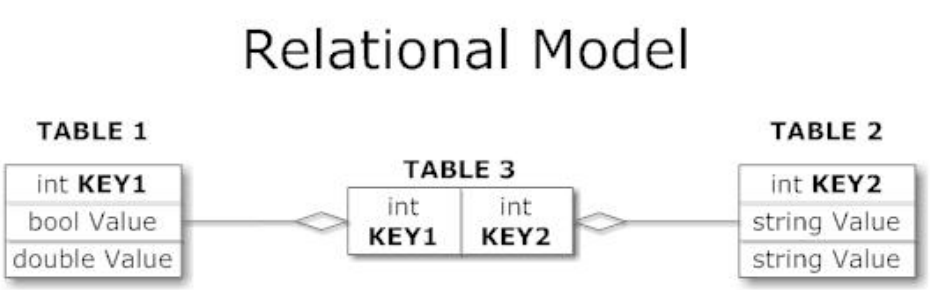
- The problems with Relational Database:
  - Overhead for complex select, update, delete operations
    - Select: Joining too many tables to create a huge size table.
    - Update: Each update affects many other tables.
    - Delete: Must guarantee the consistency of data.
  - Not well-supported the mix of unstructured data.
  - Not well-scaling with very large size of data.

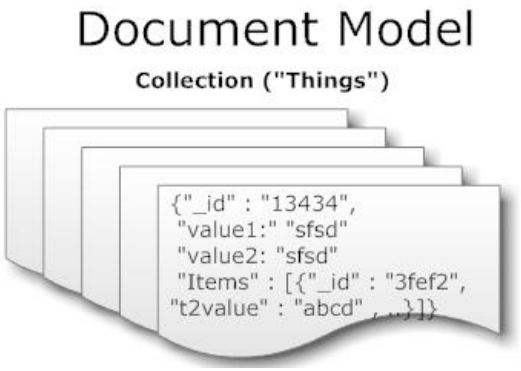
NoSQL is a good solution to deal with these problems.

Overview

- What is NoSQL:
  - NoSQL = Not only SQL
  - Wikipedia's definition:

A **NoSQL** database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.





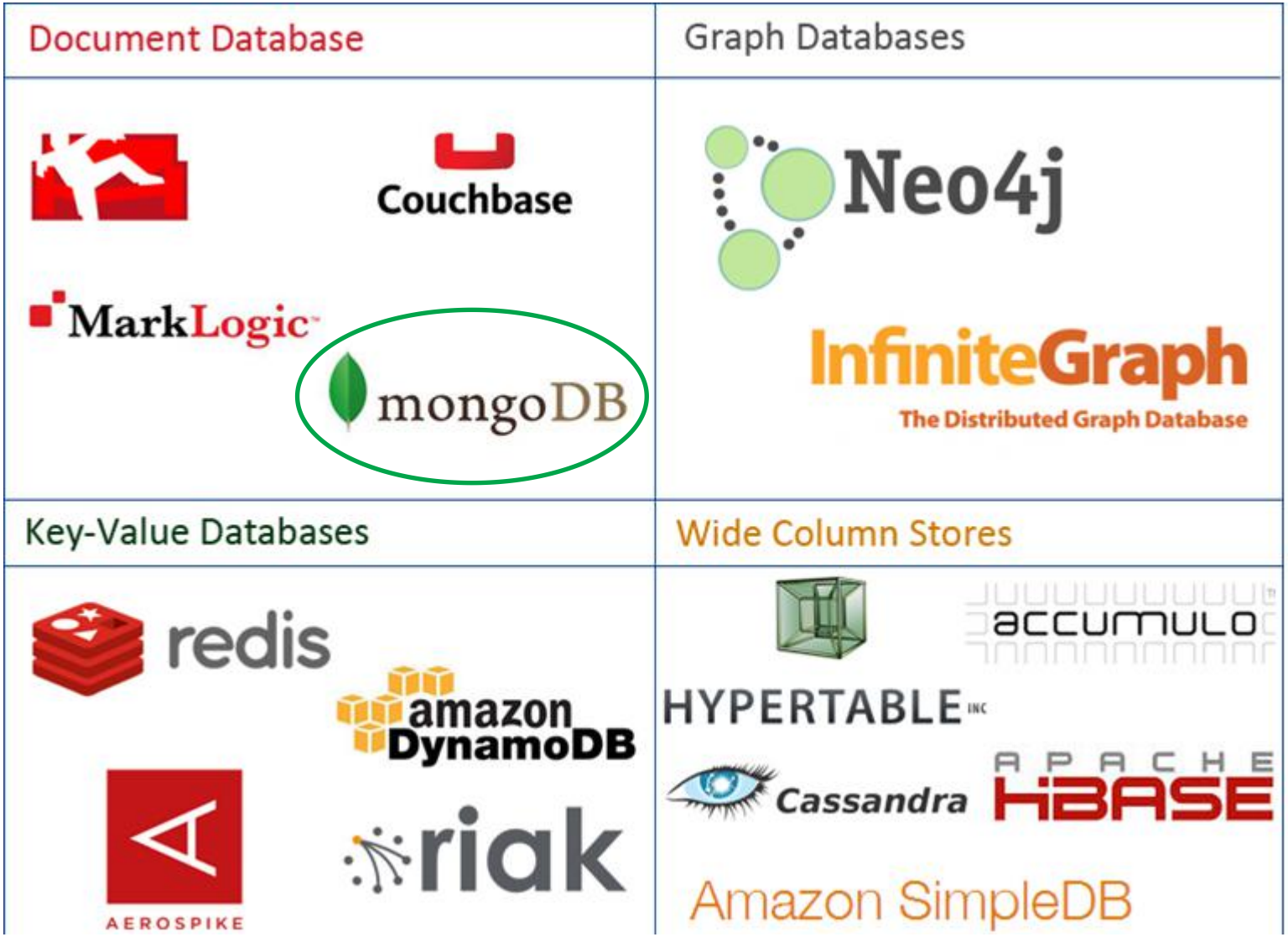
NOSQL Systems

- Most NOSQL systems are distributed databases
  - with a focus on
    - semi-structured data storage
    - high performance
    - availability
    - data replication
    - scalability
  - as opposed to an emphasis on
    - immediate data consistency
    - powerful query languages
    - structured data storage

Characteristics of NOSQL Systems

- Related to distributed database systems
  - Scalability: horizontal and vertical
  - Availability, replication and eventual consistency
  - Replication models
  - Sharding of files
  - High-performance data access
- Related to data models and query languages
  - Not requiring a schema
  - Less powerful query language
  - Versioning

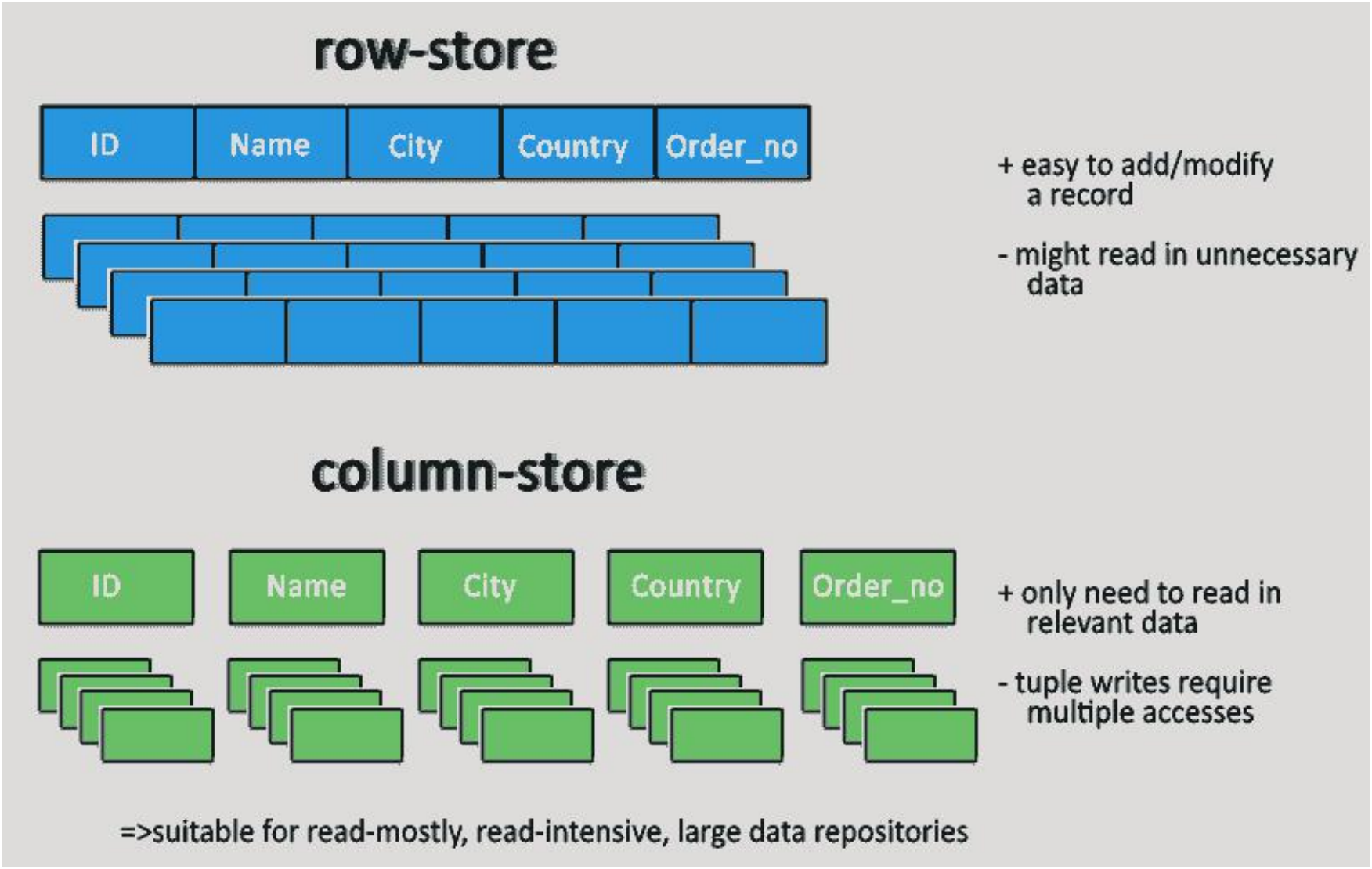
Overview – NoSQL Family



Major Categories of NOSQL Systems

- Document-based
  - Store data in the form of documents, such as JSON
- Key-value stores
  - (key value), the value can be a record, an object, a document, or even have a more complex data structure
- Column-based or wide column
  - Partition a table by column into column families (vertical partitioning)
- Graph-based
  - Data is represented as graphs

Row-Store and Column-Store



Column Families and Inverted Indexes

Column Store

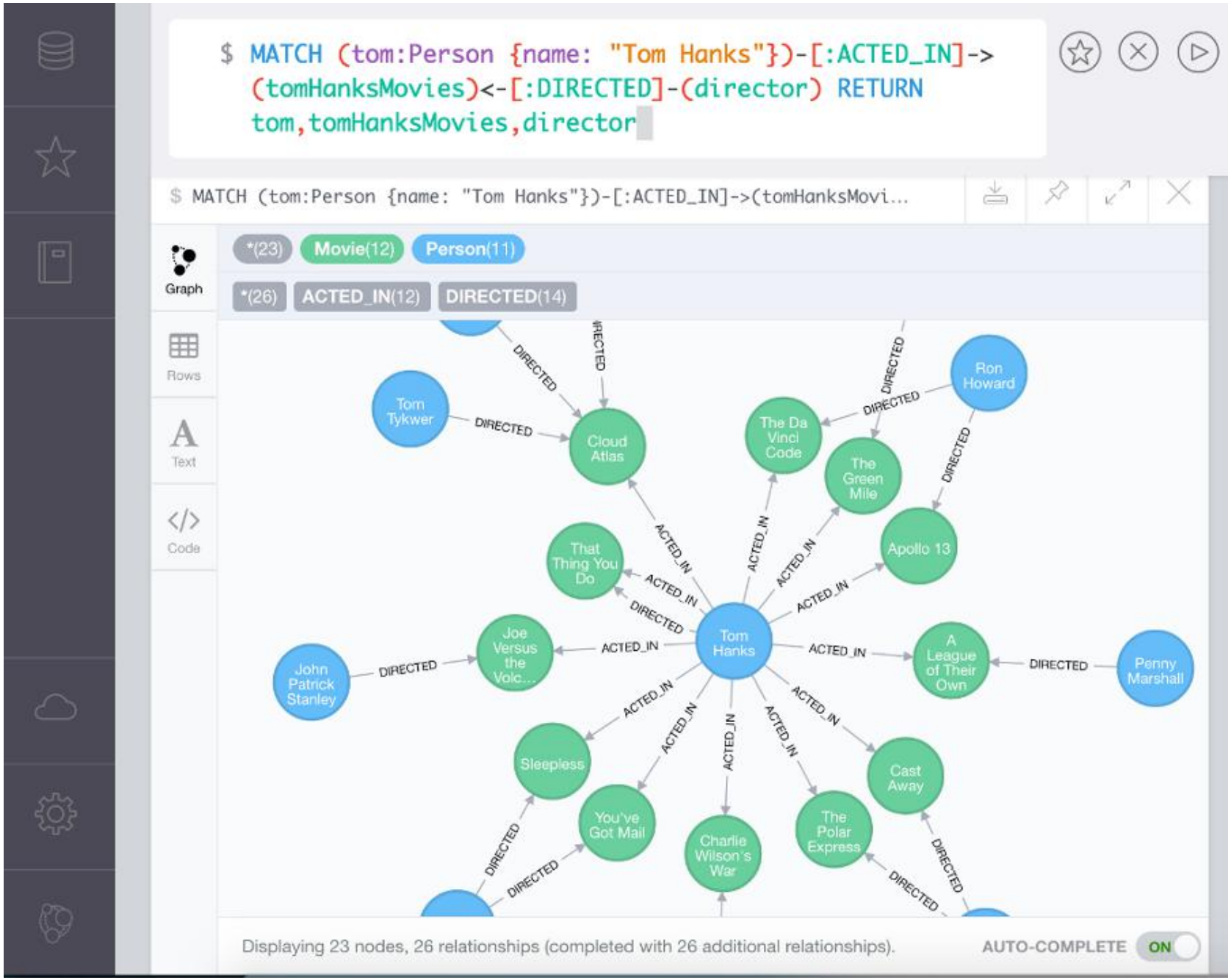
		Dimensions		Measures	
(Row)	Timestamp	State	Brand	Total Revenue	Total Cost
1	2007-01-01 00:00:00	CA	Ebony	377.01	150.1706
2	2007-01-01 00:00:00	CA	Hermanos	401.41	155.6475
3	2007-01-01 00:00:00	CA	Tell Tale	438.12	175.4235
4	2007-01-01 00:00:00	CA	Tri-State	368.71	142.9442
5	2007-01-01 00:00:00	OR	Best Choice	390.19	150.2856
6	2007-01-01 00:00:00	OR	Hermanos	571.66	230.8476
7	2007-01-01 00:00:00	OR	High Quality	391.53	154.0914
8	2007-01-01 00:00:00	OR	High Top	414.33	159.5064
9	2007-01-01 00:00:00	OR	Sunset	412.2	165.6198
10	2007-01-01 00:00:00	OR	Tri-State	407.85	166.7503
11	2007-01-01 00:00:00	WA	BBB Best	416.4	162.1617
12	2007-01-01 00:00:00	WA	Best Choice	444.66	180.1409

Inverted Indexes

Brand	Rows
Best Choice	{ 5, 12 }
BBB Best	{ 11 }

State	Rows
CA	{ 1 - 4 }
OR	{ 5 - 10 }
WA	{ 11 - 12 }

Graph-based DBMS: Neo4J



What is MongoDB?

- First developed by **10gen** (later MongoDB, Inc.) in 2007
- Name comes from "**humongous**"
- Became **open source** in 2009
- It is a **NoSQL** database
- A **document-oriented** database
- Designed with both **scalability** and developer **agility**

Example of a mongoDB Document

```
{
_id: 1234,
author: { name: “Bob Jones”, email: “b@b.com” },
post: “In these troubled times I like to ...“,
date: { $date: “2014-03-12 13:23UTC” },
location: [ -121.2322, 48.1223222 ],
rating: 2.2,
comments: [
        { user: “lalal@hotmail.com”,
          upVotes: 22,
          downVotes: 14,
          text: “Great point! I agree” },
        { user: “pedro@gmail.com”,
          upVotes: 421,
          downVotes: 22,
          text: “You are a...” }
        ],
        tags: [ “databases”, “mongo” ]
}
```

Motivations

- Problems with SQL
  - Rigid schema
  - Not easily scalable (designed for 90's technology or worse)
  - Requires unintuitive joins
- Perks of mongoDB
  - Easy interface with common languages (Java, Javascript, PHP, etc.)
  - DB tech should run anywhere (VM's, cloud, etc.)
  - Keeps essential features of RDBMS's while learning from key-value noSQL systems

http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13

Company Using mongoDB

"MongoDB powers Under Armour's online store, and was chosen for its dynamic schema, ability to scale horizontally and perform multi-data center replication."

http://www.mongodb.org/about/production-deployments/

In Good Company

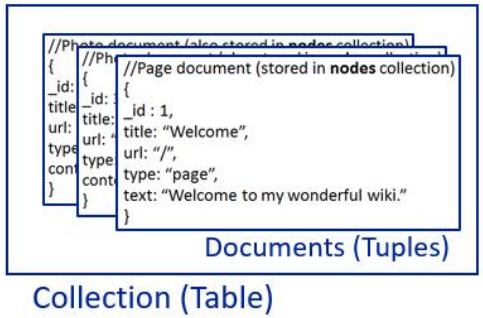
mongoDB

SQL vs MongoDB

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document
column	field
index	index
table joins (e.g. select queries)	embedded documents and linking
foreign Key	reference
primary keys	**_id** field is always the primary key
aggregation (e.g. group by)	aggregation pipeline
relational schema	schema-less

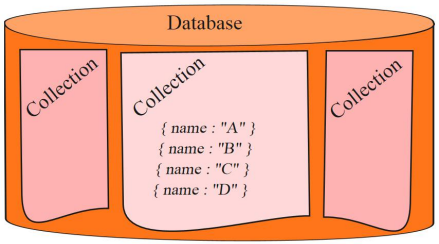
PROJECT

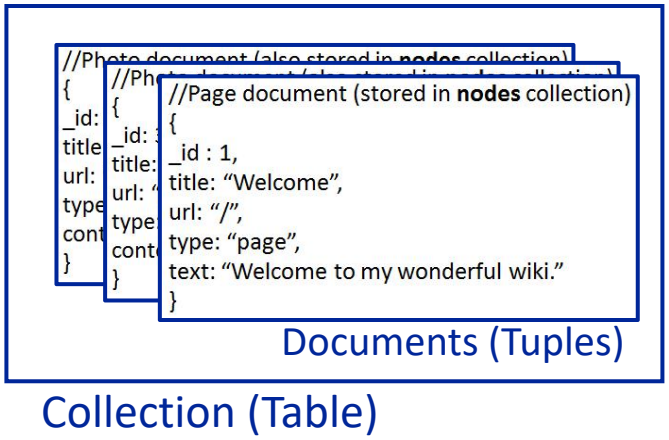
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4



Mongo is basically schema-free

- Why can't we have the database operating on the same data structure as in program?
  - That is where mongoDB comes in
- Every "*document*" in a database "*collection*" is a data structure, much like a "struct" in C, or a "class" in Java.
- A *collection* is then an array (or list) of such data structures





C/Java/
JavaScript
Program

Basic Ideas

```
{
_id: 1234,
author: { name: “Bob Jones”, email:
  “b@b.com” },
post: “In these troubled times I like to ...“,
date: { $date: “2014-03-12 13:23UTC” },
location: [ -121.2322, 48.1223222 ],
rating: 2.2,
comments: [
        { user: “lalal@hotmail.com”,
          upVotes: 22,
          downVotes: 14,
          text: “Great point! I agree” },
        { user: “pedro@gmail.com”,
          upVotes: 421,
          downVotes: 22,
          text: “You are a...” }
        ],
        tags: [ “databases”, “mongo” ]
}
```

- Collections of JSON objects
- Data stored as BSON (Binary JSON)
- Embed objects within a single document
- References/Linking
- No schema
- No join
- Max document size of 16MB, larger documents handled with GridFS.
- Runs on most common OS
  - ✓ Windows, Linux, Mac, Solaris.

Another Example


```
{
"business_id": "rncjoVoEFUJGCUoC1JgnUA",
"full_address": "8466 W Peoria Ave\nSte 6\nPeoria, AZ 85345",
"open": true,
"categories": ["Accountants", "Professional Services", "Tax Services",],
"city": "Peoria",
"review_count": 3,
"name": "Peoria Income Tax Service",
"neighborhoods": [],
"longitude": -112.241596,
"state": "AZ",
"stars": 5.0,
"latitude": 33.581867000000003,
"type": "business"
}
```

Installation

1. Download and install suitable package for each platform [Windows, Linux, Mac OSX, Solaris]
2. Create a folder e.g. C:\mongodb
3. Go to **bin** of installation folder.
4. Type following command: mongod --dbpath=C:/mongodb
5. Run another command: mongo.exe
6. The mongodb server is running.

MongoDB

Data Model,
Create, Retrieve, Update, Delete
(CRUD)

JSON

- Easy for humans to write/read, easy for computers to parse/generate
- Objects can be nested
- Built on
  - Field-value pairs
  - Ordered list of values

```
{
"_id" :     "37010"
"city" :    "ADAMS",
"pop" :     2660,
"state" :   "TN",
“councilmen” : [“John Smith”, “Jim Curry”, “Mary Lee”]
}
```

Another JSON Example

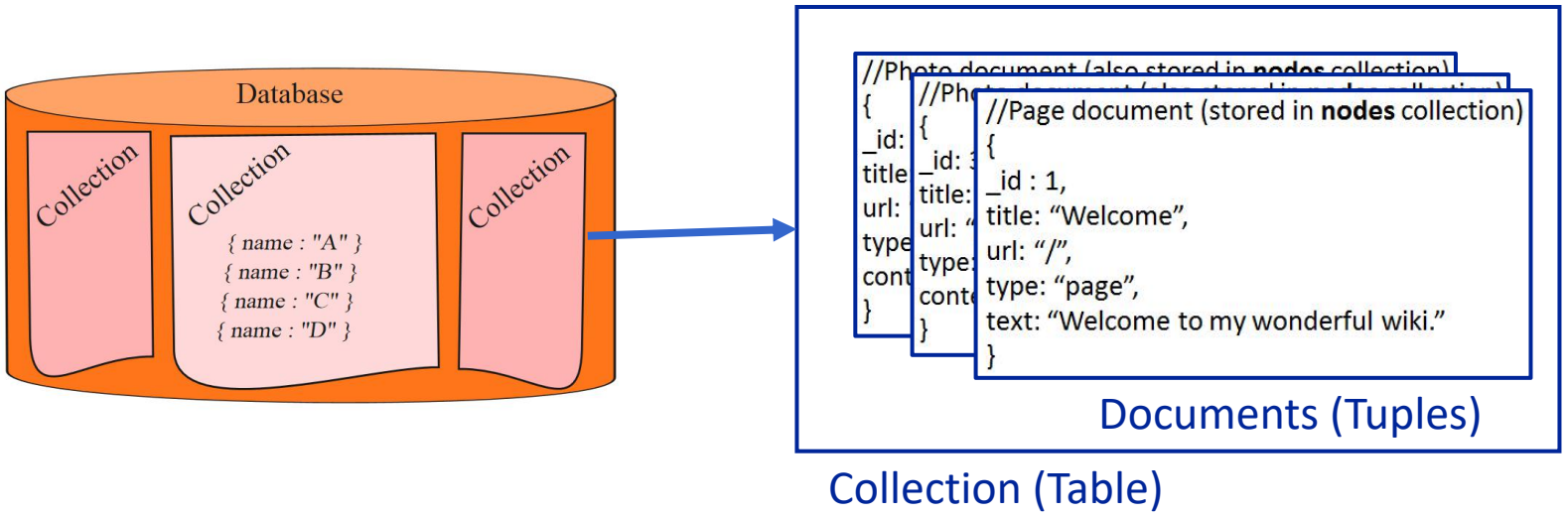
```
{
    '_id' : 1,
    'name' : { 'first' : 'John', 'last' : 'Backus' },
    'contribs' : [ 'Fortran', 'ALGOL', 'Backus-Naur Form', 'FP' ],
    'awards' : [
        {
            'award' : 'W.W. McDowell Award',
            'year' : 1967,
            'by' : 'IEEE Computer Society'
        }, {
            'award' : 'Draper Prize',
            'year' : 1993,
            'by' : 'National Academy of Engineering'
        }
    ]
}
```

JSON and BSON

- JSON (JavaScript Object Notation)
  - A JSON database returns query results that can be easily parsed, with little or no transformation, directly by JavaScript and most popular programming languages – reducing the amount of logic you need to build into your application layer.
- BSON (Binary JSON)
  - MongoDB represents JSON documents in binary-encoded format called BSON behind the scenes.
  - BSON extends the JSON model to provide additional data types and to be efficient for encoding and decoding within different languages.

MongoDB Data Model$_{-1}$

- A mongo database is a set of collections.
- Collections
  - Like tables of relational db's.
  - A collection include documents, having index set in common
  - A document is like a row of a relational table.



MongoDB Data Model$_{-2}$

- Documents in a collection
  - must have an _id.
  - BSON format, consisting of field-value pairs
  - max 16 MB per document
  - do not have to have uniform structure
    - Enables simpler schema migration.
    - Better mapping of object-oriented inheritance and polymorphism.

```
//Page document (stored in nodes collection)
{
_id : 1,
title: "Welcome",
url: "/",
type: "page",
text: "Welcome to my wonderful wiki."
}
```

```
//Photo document (also stored in nodes collection)
{
_id: 3,
title: "Cool Photo",
url: "/photo.jpg",
type: "photo",
date: "2015-1-1",
content: Binary(...)
}
```

The _id Field

- By default, each document contains an _id field. This field has a number of special characteristics:
  - Value serves as primary key for collection.
  - Value is unique, immutable, and may be any non-array type.
  - Default data type is ObjectId, which is "small, likely unique, fast to generate, and ordered."
  - Sorting on an ObjectId value is roughly equivalent to sorting on creation time.

```
//Page document (stored in nodes collection)
{
_id : 1,
title: "Welcome",
url: "/",
type: "page",
text: "Welcome to my wonderful wiki."
}
```

The Value of Field

- Native data types
- Arrays
- Other documents

```
{
    '_id' : 1,
    'name' : { 'first' : 'John', 'last' : 'Backus' },
    'contribs' : [ 'Fortran', 'ALGOL', 'Backus-Naur Form', 'FP' ],
    'awards' : [
        {
            'award' : 'W.W. McDowell Award',
            'year' : 1967,
            'by' : 'IEEE Computer Society'
        }, {
            'award' : 'Draper Prize',
            'year' : 1993,
            'by' : 'National Academy of Engineering'
        }
    ]
}
```

BSON Types

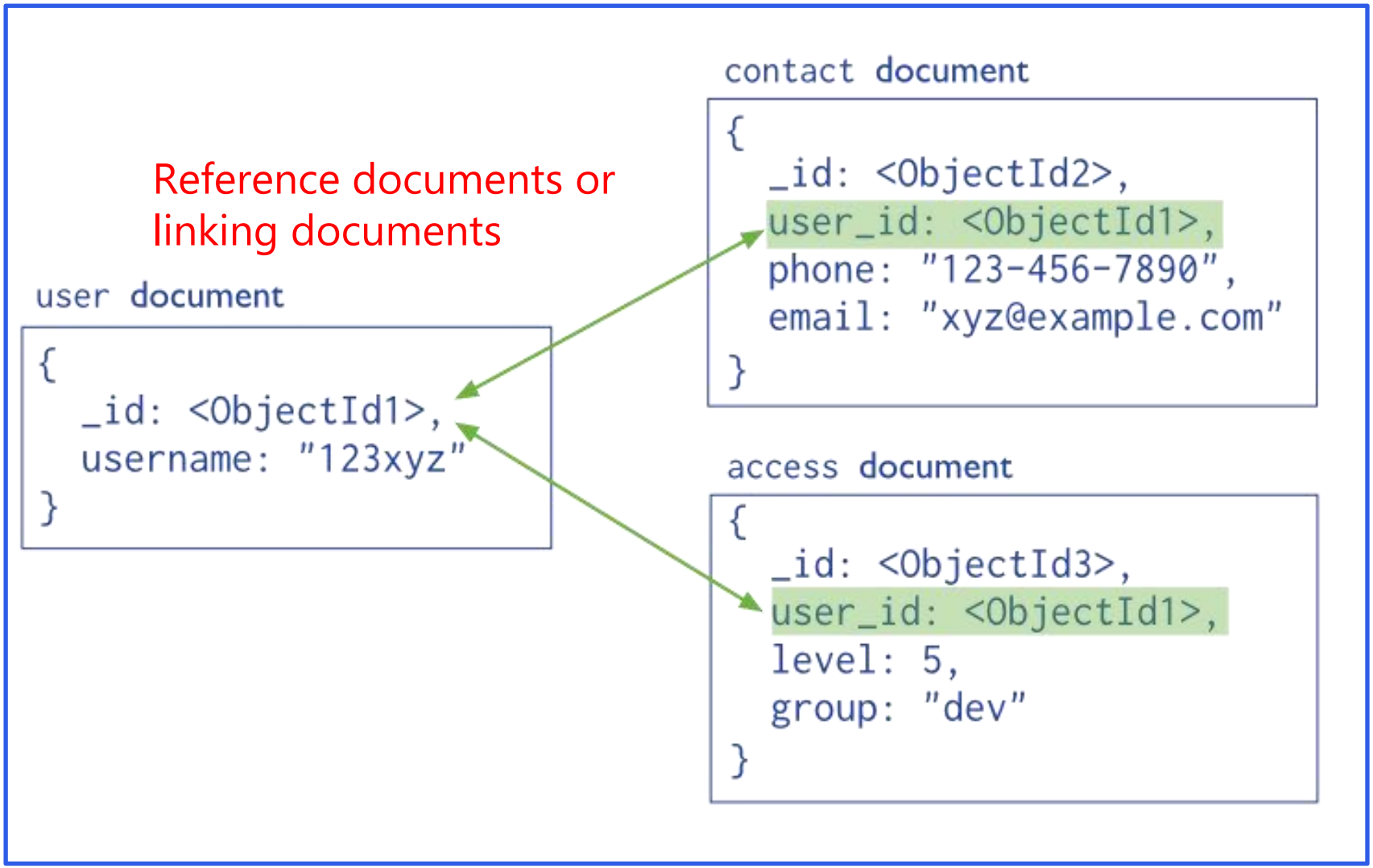
Type	Number
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

The number can be used with the $type operator to query by type!

http://docs.mongodb.org/manual/reference/bson-types/

Embedded Sub-Document

Reference/Linking Documents



A (Denormalized) Embedded Structure
An Array of Values


```
{
"business_id": "rncjoVoEFUJGCUoC1JgnUA",
 "full_address": "8466 W Peoria Ave\nSte 6\nPeoria, AZ 85345",
"open": true,
"categories": ["Accountants", "Professional Services", "Tax Services"],
 "city": "Peoria",
"review_count": 3,
"name": "Peoria Income Tax Service",
 "neighborhoods": [],
 "longitude": -112.241596,
 "state": "AZ",
"stars": 5.0,
"latitude": 33.581867000000003,
"type": "business“
}
```

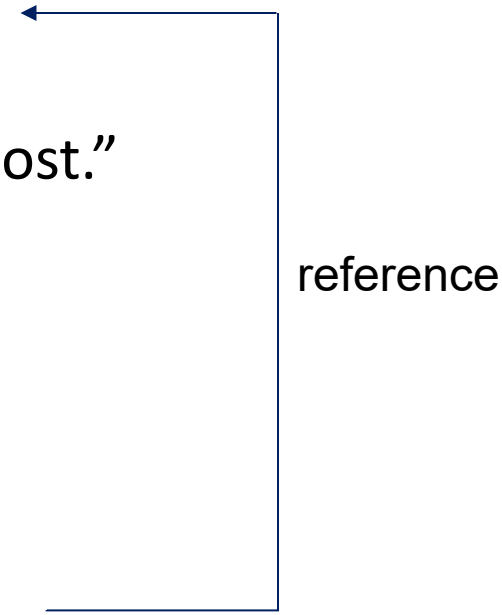
A (Denormalized) Embedded Structure
An Array of Sub Documents

```
{
“_id” : “First Post”,
“comments” : [
        {“author” : “Bob”, “text” : “Nice Post!”},
        {“author” : “Tom”, “text” : “Dislike!”}
],
“comment_count” : 2
}
```

A Normalized Structure (Reference/Linking)

```
//db.post schema
{
"_id" : "First Post",
"author" : "Rick",
"text" : "This is my first post."
}

//db.comments schema
{
"_id" : ObjectID(...),
"post_id" : "First Post",
"author" : "Bob",
"text" : "Nice Post!"
}
```



MongoDB

Query, Index, Sharding, Replication

CRUD: Using the Shell

To check which db you're using	db
Show all databases	show dbs
Switch db's/make a new one	use <db_name>
See what collections exist	show collections

```
C:\mongodb\bin>mongo.exe
MongoDB shell version: 2.4.9
connecting to: test
> show dbs
blog    0.203125GB
local   0.078125GB
test    0.203125GB
> use blog
switched to db blog
```

MongoDB Create and Query

- Create a collection:
  db.createCollection(<name>, <options>)
  - ✓ options: specify the number of documents in a collection etc.

  db.createCollection("students", {max : 5000})

- Insert a document:
  db.<collection>.insert(<document>)

  db.students.insert({"name": "nguyen", "age": 24, "gender": "male"})

  <=>

  INSERT INTO students(name, age, gender)
  VALUES ('nguyen', 24, 'male');

SQL Statement	MongoDB commands
SELECT * FROM table	db.collection.find()
SELECT * FROM table WHERE artist = ‘Nirvana’	db.collection.find({Artist:”Nirvana”})
SELECT* FROM table ORDER BY Title	db.collection.find().sort(Title:1)
DISTINCT	.distinct()
GROUP BY	.group()
>=, <	$gte, $lt

CRUD: Querying

- Get all docs: db.<collection>.find()
  - Returns a cursor, which is iterated over shell to display up to first 20 results.
  - Add .limit(<number>) to limit results
  - SELECT * FROM <table>;
- Get one doc: db.<collection>.findOne()
- Query with conditions:

  db.<collection>.find({<field>:<value>})

  db.students.find({ “gender”: “female”}).limit(5)

CRUD: Querying

“AND”

db.<collection>.find({<field1>:<value1>, <field2>:<value2>})

db.students.find({ “gender”: “female”, “age”: {$lte:20}})

SELECT *
FROM students
WHERE gender = “female” AND age <= 20;

CRUD: Querying

“OR”

db.<collection>.find({ $or: [<field>:<value1>, <field>:<value2>]})

db.students.find({ $or: [“age”: {$lte:20}, “age”: {$gte:40}]})

```
SELECT *
FROM students
WHERE age <= 20 OR age >= 40;
```

Checking for multiple values of same field

db.<collection>.find({<field>: {$in [<value>, <value>]}})

CRUD: Querying

Including/excluding document fields

db.<collection>.find({<field1>:<value>}, {<field2>: 0})

SELECT field1
FROM <table>;

db.<collection>.find({<field>:<value>}, {<field2>: 1})

Excluding <field2>
Including <field2>

Find documents with or w/o field

db.<collection>.find({<field>: { $exists: true}})

Query Example

```
db.posts.find({ author.name: “mike” })
db.posts.find({ rating: { $gt: 2 }})
db.posts.find({ tags: “software” }).pretty()
db.posts.find().sort({date: -1}).limit(10)

// select * from posts where ‘economy’ in tags order by ts DESC
db.posts.find({tags :‘economy’}).sort({ts :-1 }).limit(10);
```

Note: pretty() displays the results in a formatted way.

http://try.mongodb.org/

CRUD: Updating

```
db.<collection>.update(
{<field1>:<value1>},                  //all docs in which field = value
{$set: {<field2>:<value2>}},          //set field to value
{multi:true} )                        //update multiple docs
```

```
db.students.update({'name':'nguyen'}, { $set:{'age': 20 } }  )
```

upsert: if true, creates a new doc when none matches search criteria.

```
UPDATE students
SET age = 20
WHERE name = “nguyen”;
```

CRUD: Updating

- **To replace the existing document with new one:**

  save method

  db.students.save({_id:ObjectId('string_id'), "name": "ben", "age": 23, "gender": "male"}

- **To remove a field**

  db.<collection>.update({<field>:<value>}, { $unset: { <field>: 1}})

  To remove the field quantity from the first document in the products collection where the field sku has a value of unknown.

  db.products.update({sku: "unknown"}, {$unset: {quantity: 1}})

CRUD: Delete

- Drop a database
  - Show database: show dbs
  - Use a database: use <db_name>
  - Drop it: db.dropDatabase()
- Drop a collection:
  - db.<collection>.drop()
- Delete all documents where field = value
  - db.<collection>.remove({<field>:<value>})

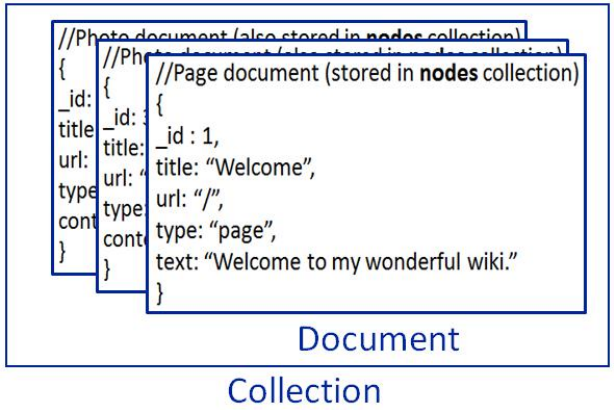
```
DELETE FROM <table>
WHERE <field> = <value>;
```

- As above, but **only delete first document**
  - db.<collection>.remove({<field>:<value>}, true)

CRUD: Isolation

- By default, all writes are atomic **only** on the level of a single document.
- This means that, by default, all writes can be interleaved with other operations.
- You can isolate writes on an **unsharded** collection by adding $isolated:1 in the query area:

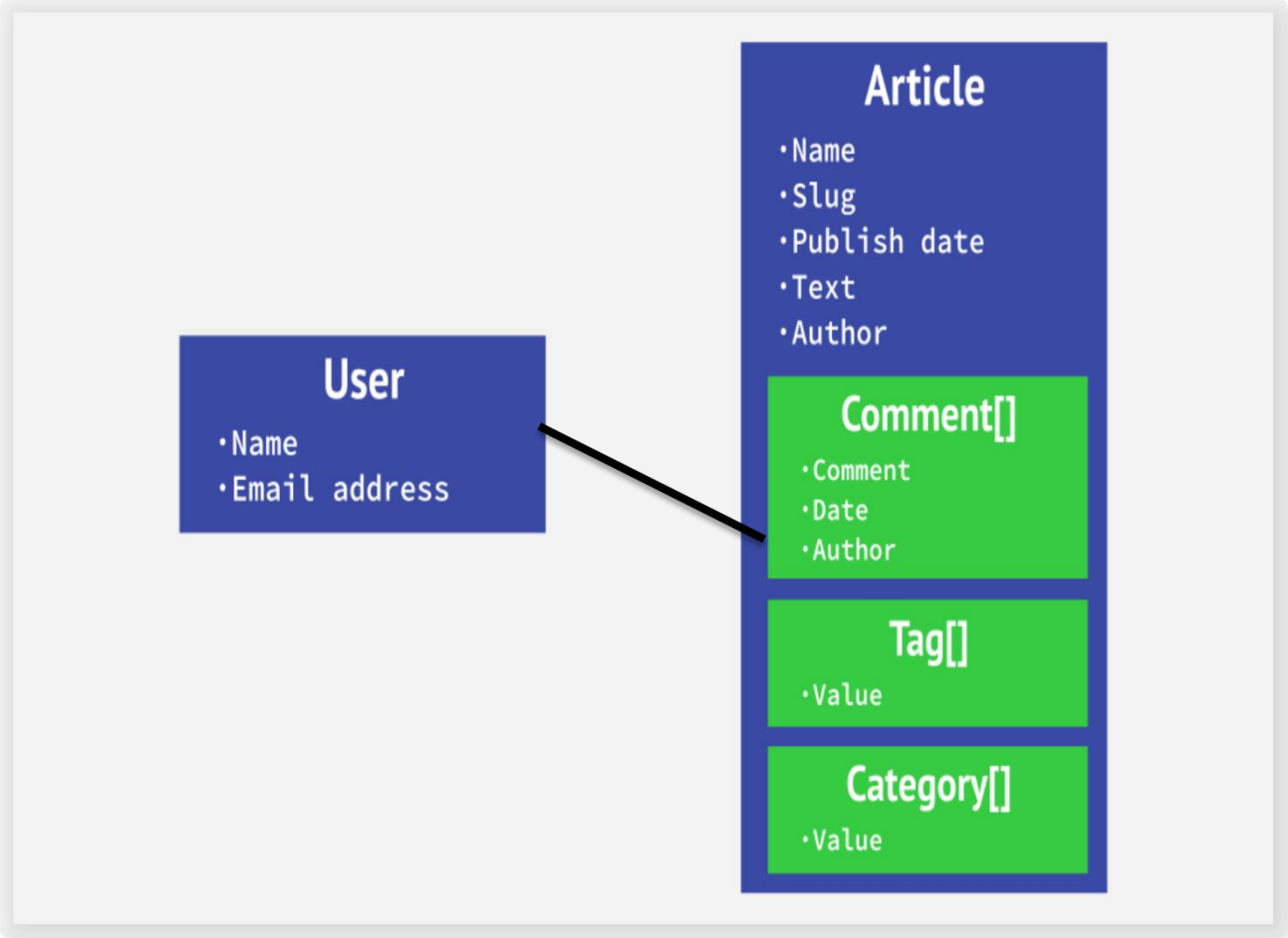
db.<collection>.remove({<field>:<value>, $isolated: 1})



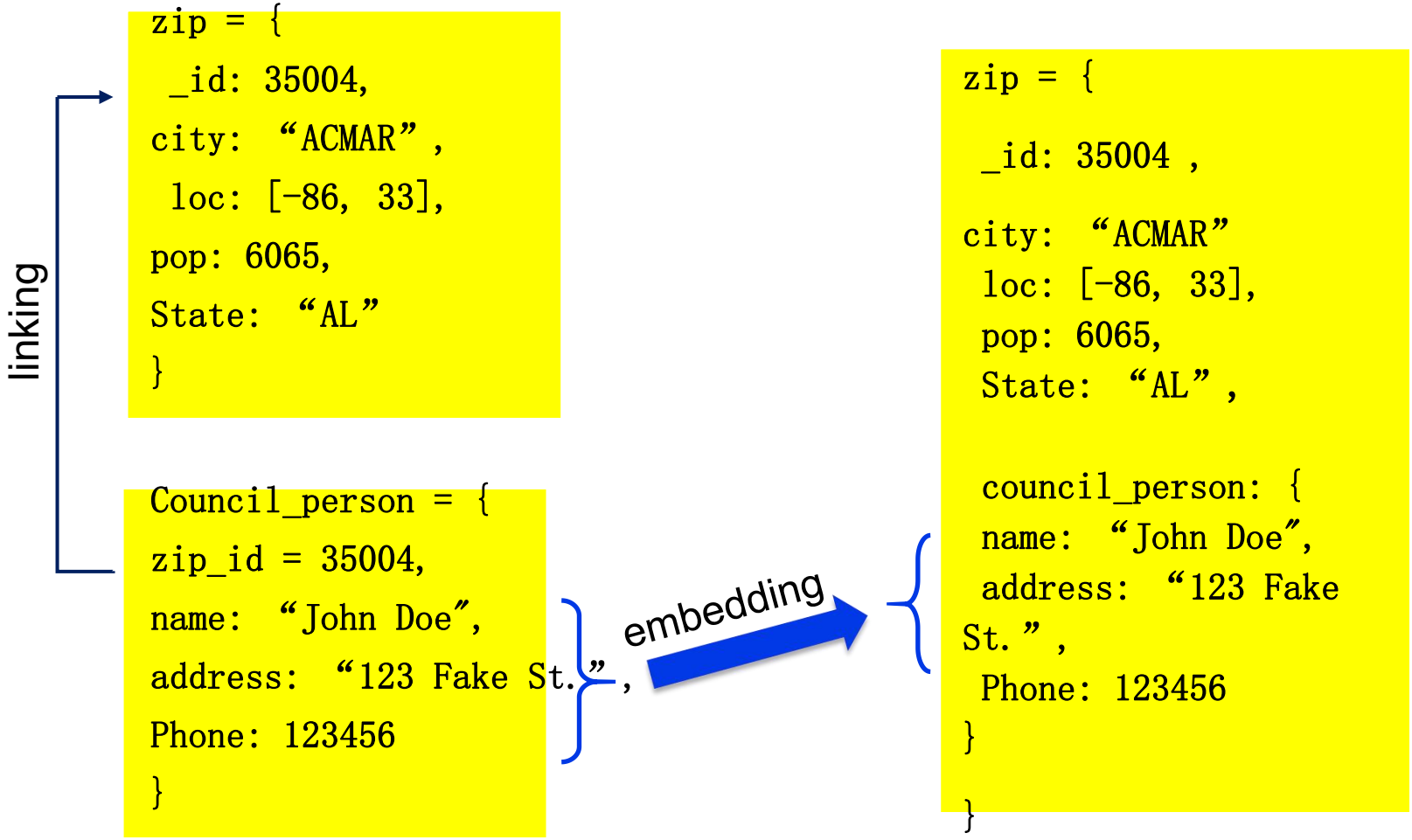
No operations will interleave with this removal of the matched documents before completion.

Relationship between Documents

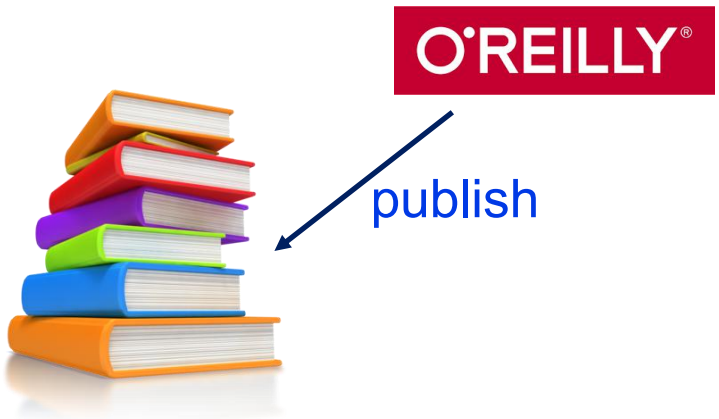
Two ways to establish the connection:
embedding and linking



One to One relationship



One to many relationship - Embedding




```
book = {
title: "MongoDB: The Definitive Guide",
authors: [ "Kristina Chodorow", "Mike Dirolf" ]
published_date: ISODate("2010-09-24"),
pages: 216,
language: "English",
publisher: {
name: "O' Reilly Media",
founded: "1980",
location: "CA"
}
}
```

One to many relationship – Linking


```
publisher = {
    _id: "oreilly",
    name: "O' Reilly Media",
    founded: "1980",
    location: "CA"
}
```

```
book = {
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "oreilly"
}
```

Many to many relationship

- Can put relation in either one of the documents (embedding in one of the documents)
- Focus how data is queried

```
book = {
    title: "MongoDB: The Definitive Guide",
    authors : [
        { _id: "kchodorow", name: "Kristina Chodorow" },
        { _id: "mdirolf", name: "Mike Dirolf" }
    ]
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}
```

```
author = {
    _id: "kchodorow",
    name: "Kristina Chodorow",
    hometown: "New York"
}
```

```
db.book.find( { authors.name : "Kristina
Chodorow" } )
```

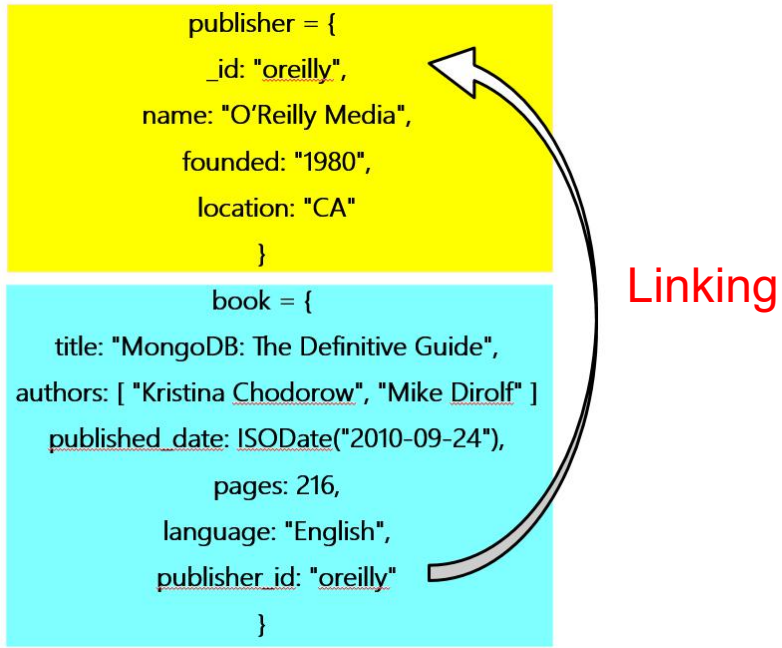
Linking vs. Embedding

- To embed or not to embed. That is the question!
  - ✓ Rule of thumb is to embed whenever possible.
- Embed when the "many" objects always appear with (viewed in the context of) their parents.
- Embedding (de-normalization) is a bit like pre-joining data which provides **data locality** and improves **speed.**
- Linking when you need more flexibility

Embedding

```
book = {
title: "MongoDB: The Definitive Guide",
authors: [ "Kristina Chodorow", "Mike Dirolf" ]
published_date: ISODate("2010-09-24"),
pages: 216,
language: "English",
publisher: {
name: "O'Reilly Media",
founded: "1980",
location: "CA"
}
}
```

Linking



Example-Book Checkout

- Book can be checked out by one student at a time
- Student can check out many books

```
student = {
    _id: "joe"
    name: "Joe Bookreader",
    join_date: ISODate("2011-10-15"),
    address: { ... }
}
```

```
book = {
    _id: "123456789"
    title: "MongoDB: The Definitive Guide",
    authors: [ "Kristina Chodorow", "Mike Dirolf" ],
    ...
}
```

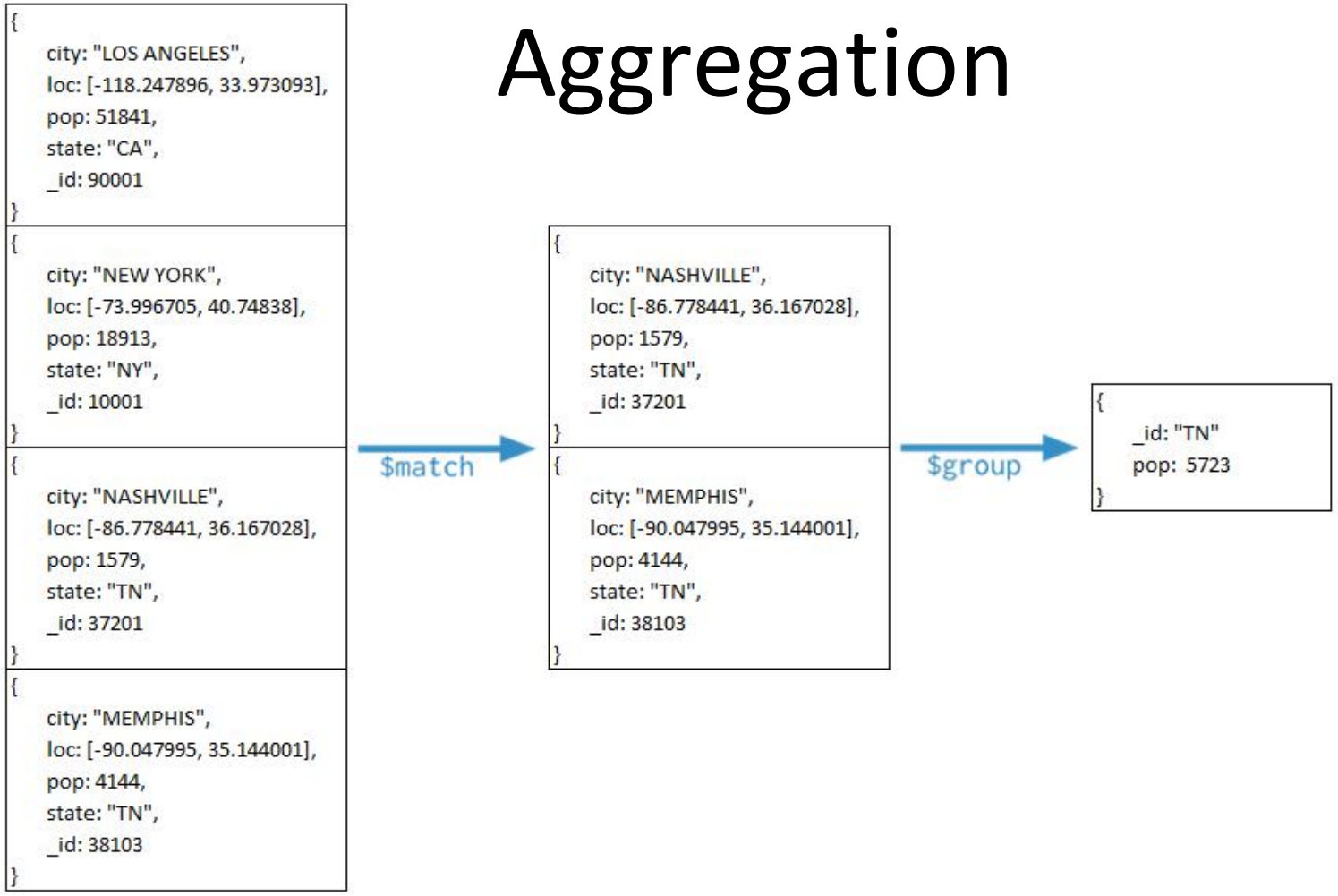
Modeling Checkouts

```
student = {
    _id: "joe"
    name: "Joe Bookreader",
    join_date: ISODate("2011-10-15"),
    address: { ... },
    checked_out: [
        { book_id: "123456789", checked_out: "2012-10-15" },
        { book_id: "987654321", checked_out: "2012-09-12" },
        ...
    ]
}
```

Joe checks out many books.

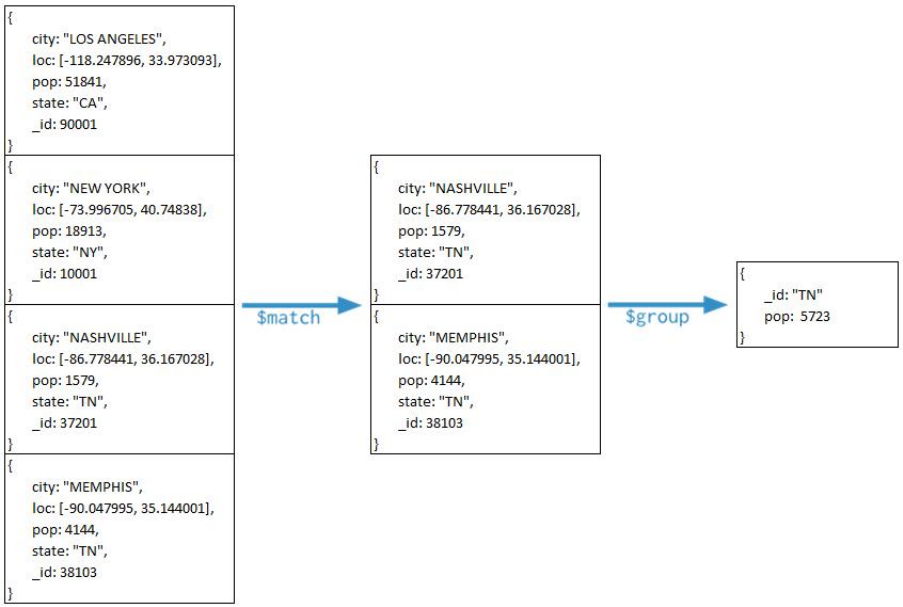
MongoDB

Aggregation



Aggregation

- Operations that process data records and return computed results.
- MongoDB provides aggregation operations.
  - Aggregation piplelines
  - MapReduce
  - Single purpose aggregation operations
- Running data aggregation on the mongod instance simplifies application code and limits resource requirements.

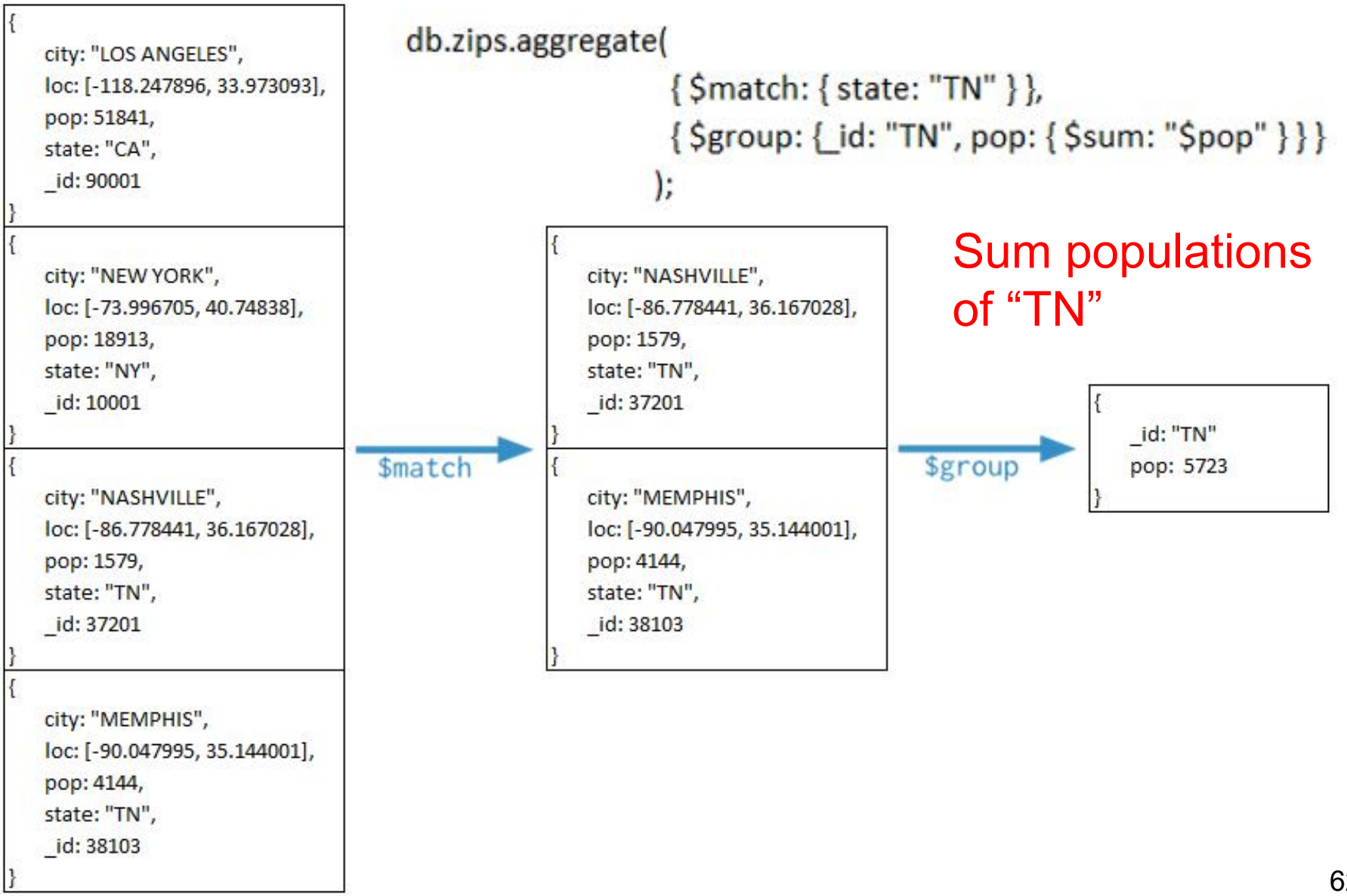


Aggregation Pipelines

- Modeled on the concept of data processing pipelines.
- Provides:
  - *filters* that operate like queries
  - *document transformations* that modify the form of the output document.
- Provides tools for:
  - grouping and sorting by field
  - aggregating the contents of arrays, including arrays of documents
- Can use operators for tasks such as calculating the average or concatenating a string.

Aggregation Pipelines

zips



Pipelines

- $limit
  - Passes the first *n* documents unmodified to the pipeline where *n* is the specified limit.
- $skip
  - Skips the first *n* documents where *n* is the specified skip number and passes the remaining documents unmodified to the pipeline.
- $sort
  - Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified.

```
db.article.aggregate(
    { $limit : 5 }
);
```

```
db.article.aggregate(
    { $skip : 5 }
);
```

```
db.users.aggregate(
  [
    { $sort : { age : -1, posts: 1 } }
  ]
)
```

descending

```
db.article.aggregate({“$project” : {“author” : 1}},
   {“$group” : {“_id” : “$author”, “count” : {“$sum” : 1}}},
   {“$sort” : {“count” : -1}},
   { $limit : 5 })
```

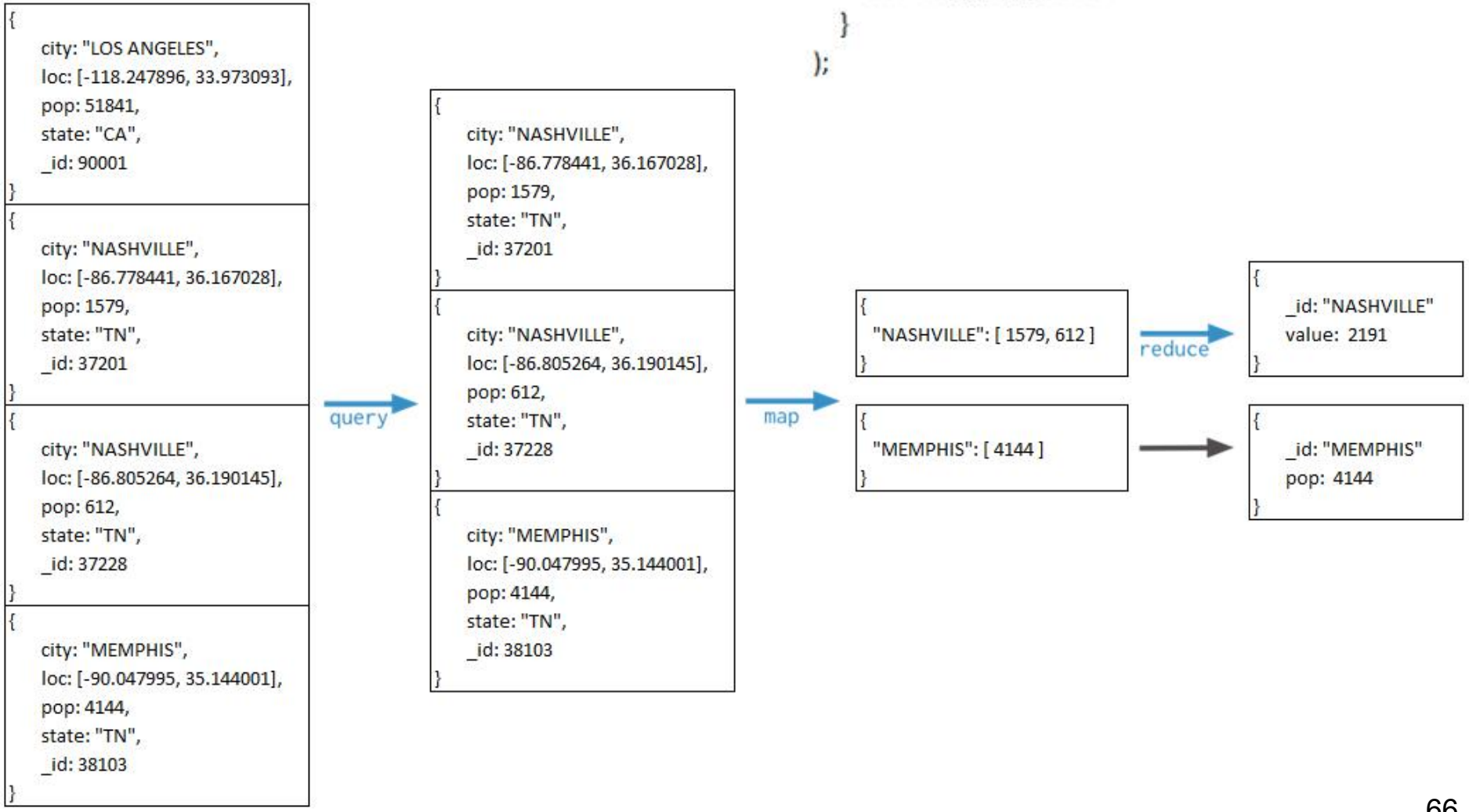
```
{
    “result” : [
        {
            “_id” : “R. L. Stine”,
            “count” : 430
        },
        {
            “_id” : “Edgar Wallace”,
            “count” : 175
        },
        {
            “_id” : “Nora Roberts”,
            “count” : 170
        },
        {
            “_id” : “Agatha Christie”,
            “count” : 140
        },
        {
            “_id” : “Stanle Gardner”,
            “count” : 80
        },
    ],
    “ok” : 1
}
```

equivalent to operator count

Map-Reduce

- Has two phases:
  - A *map* stage that processes each document and *emits* one or more objects for each input document.
  - A *reduce* phase that combines the output of the map operation.
  - An optional *finalize* stage for final modifications to the result.
- Uses Custom JavaScript functions
  - Provides greater flexibility but is less efficient and more complex than the aggregation pipeline
- Can have output sets that exceed the **16 megabyte output** limitation of the aggregation pipeline.

```
db.zips.mapReduce(
                    function() { emit( this.city, this.pop ); },
                    function(key, values) { return Array.sum( values ) },
                    {
                      query: { state: "TN" },
                      out: "city_pop_totals"
                    }
                  );
```



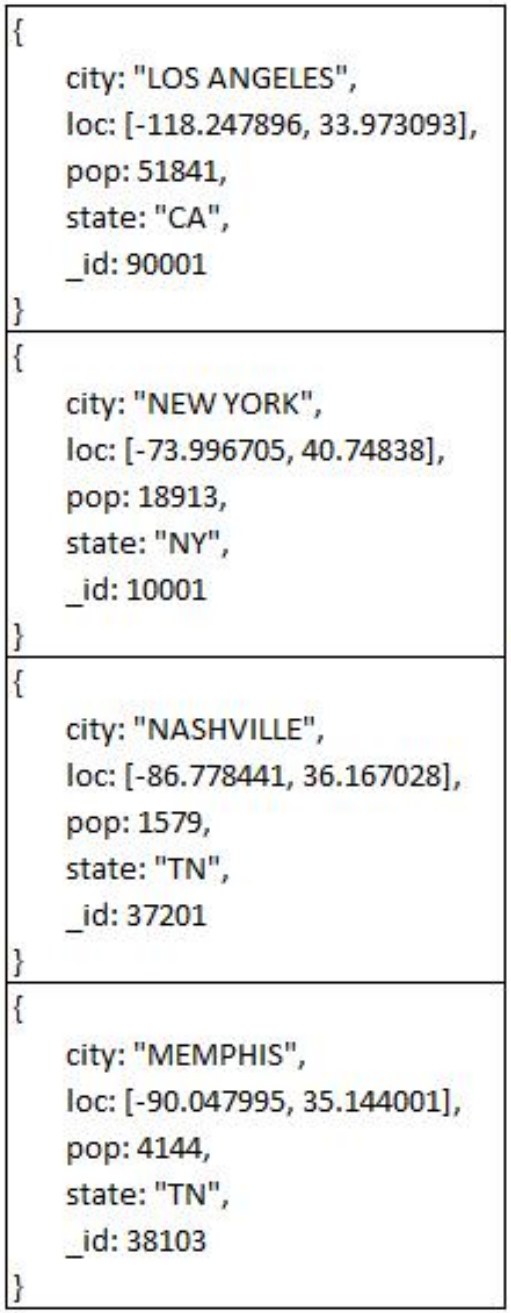
Single Purpose Aggregation Operations

- Special purpose database commands:
  - count: returning a count of matching documents

    db.people.count()
  - distinct: returning the distinct values for a field

    db.people.distinct("age")

    Return distinct values of "age" in collection "people".
  - group: grouping data based on the values of a field.
- Aggregate documents from a single collection.
- Lack the flexibility and capabilities of the aggregation pipeline and map-reduce.



In collection “zips”,
find distinct values in “state” field.

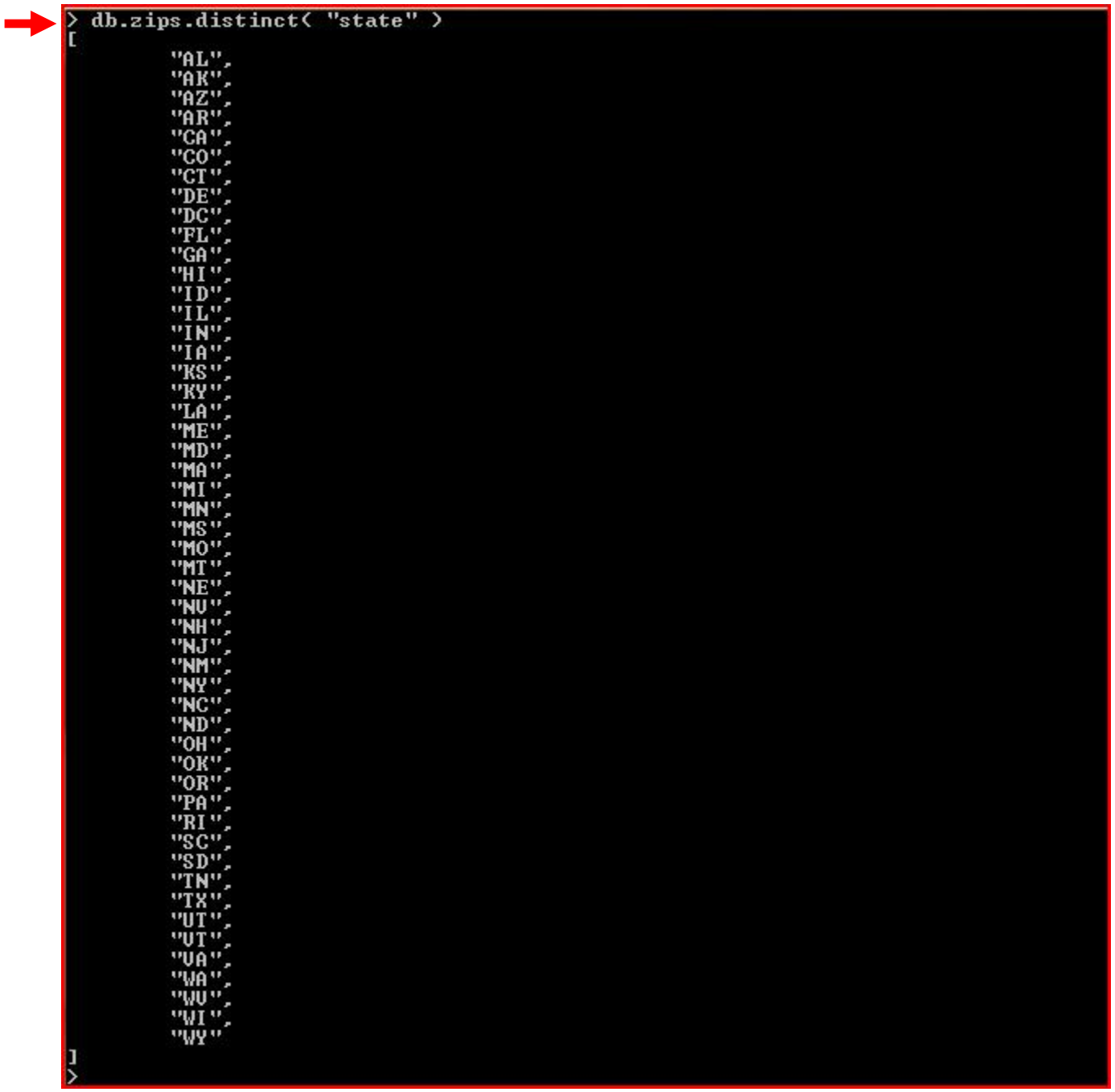
```
db.zips.distinct( "state" );
```

distinct → ["CA", "NY", "TN"]

Aggregation Pipelines and Map-Reduce

```
C:\mongodb\bin>mongo.exe
MongoDB shell version: 2.4.9
connecting to: test
> show dbs
blog    0.203125GB
local   0.078125GB
test    0.203125GB
> use blog
switched to db blog
> db.zips.aggregate( {$match: {state: "TN"}}, {$group: {_id: "TN", pop: {$sum: "
$pop"}}})
{ "result" : [ { "_id" : "TN", "pop" : 4876457 } ], "ok" : 1 }
> db.zips.mapReduce(
... function() { emit( this.city, this.pop ); },
... function(key, values) { return Array.sum( values ) },
... {
...    query: { state: "TN" },
...    out: "city_pop_totals"
... }
... );
{
        "result" : "city_pop_totals",
        "timeMillis" : 198,
        "counts" : {
                "input" : 582,
                "emit" : 582,
                "reduce" : 13,
                "output" : 505
        },
        "ok" : 1,
}
> db.city_pop_totals.find({_id: "NASHVILLE"})
{ "_id" : "NASHVILLE", "value" : 349822 }
>
```

Single Purpose Aggregation Operation



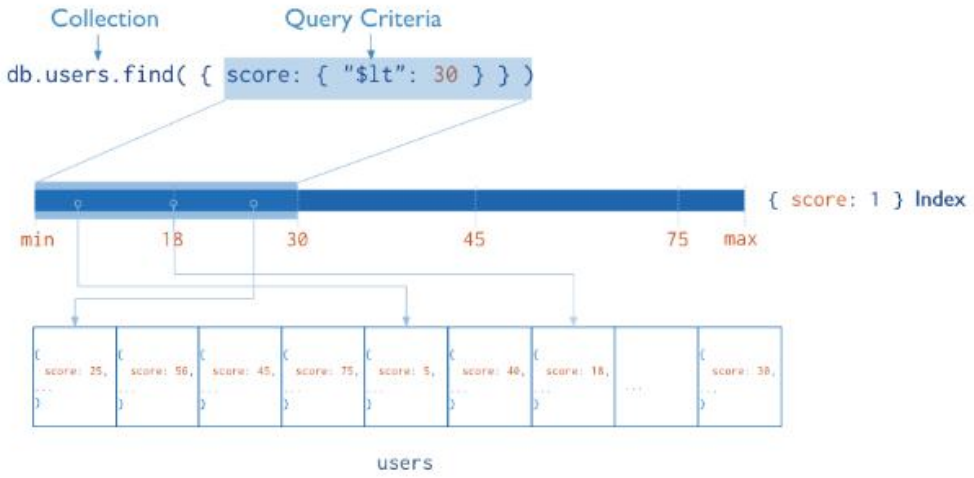
	aggregate	mapReduce	group
Description	*New in version 2.2.* Designed with specific goals of improving performance and usability for aggregation tasks. Uses a "pipeline" approach where objects are transformed as they pass through a series of pipeline operators such as `$group`, `$match`, and `$sort`. See Aggregation Reference for more information on the pipeline operators.	Implements the Map-Reduce aggregation for processing large data sets.	Provides grouping functionality. Is slower than the `aggregate` command and has less functionality than the `mapReduce` command.
Key Features	Pipeline operators can be repeated as needed. Pipeline operators need not produce one output document for every input document. Can also generate new documents or filter out documents.	In addition to grouping operations, can perform complex aggregation tasks as well as perform incremental aggregation on continuously growing datasets. See Map-Reduce Examples and Perform Incremental Map-Reduce.	Can either group by existing fields or with a custom `keyf` JavaScript function, can group by calculated fields. See `group` for information and example using the `keyf` function.

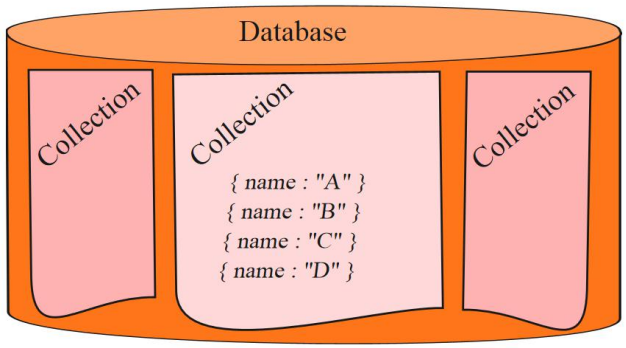
Flexibility	Limited to the operators and expressions supported by the aggregation pipeline. However, can add computed fields, create new virtual sub-objects, and extract sub-fields into the top-level of results by using the `$project` pipeline operator. See `$project` for more information as well as Aggregation Reference for more information on all the available pipeline operators.	Custom `map`, `reduce` and `finalize` JavaScript functions offer flexibility to aggregation logic. See `mapReduce` for details and restrictions on the functions.	Custom `reduce` and `finalize` JavaScript functions offer flexibility to grouping logic. See `group` for details and restrictions on these functions.
Output Results	Returns results inline. The result is subject to the BSON Document size limit.	Returns results in various options (inline, new collection, merge, replace, reduce). See `mapReduce` for details on the output options. *Changed in version 2.2:* Provides much better support for sharded map-reduce output than previous versions.	Returns results inline as an array of grouped items. The result set must fit within the maximum BSON document size limit. *Changed in version 2.2:* The returned array can contain at most 20,000 elements; i.e. at most 20,000 unique groupings. Previous versions had a limit of 10,000 elements.
Sharding	Supports non-sharded and sharded input collections.	Supports non-sharded and sharded input collections.	Does **not** support sharded collection.

MongoDB

Index

db.users.find({ score: { “$lt” : 30} })

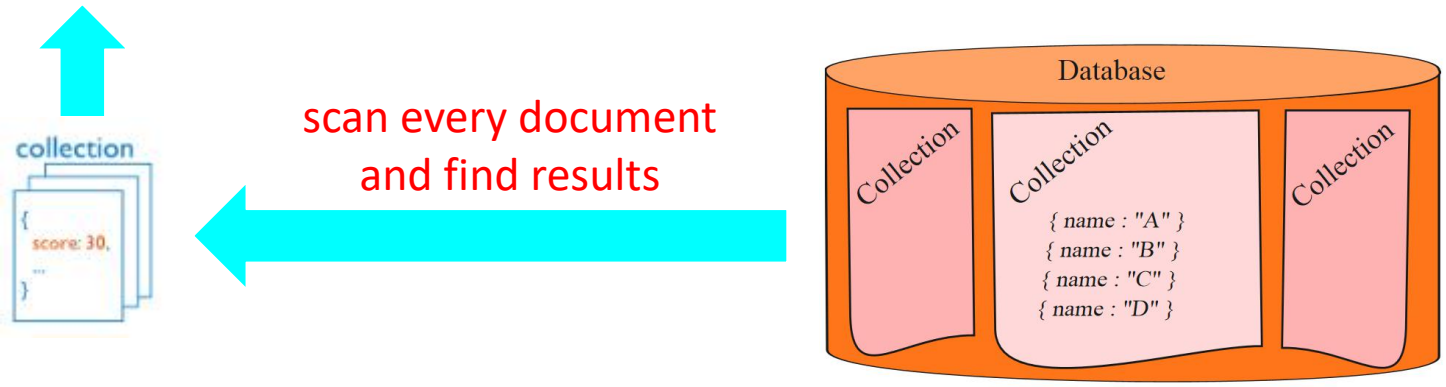




Before Index

- What does database normally do when we query?
  - MongoDB must scan every document.
  - Inefficient because process large volume of data

db.users.find({ score: { “$lt” : 30} })



Definition of Index

- Definition
  - Indexes are special data structures that store a small portion of the collection's data set in an easy to traverse form.

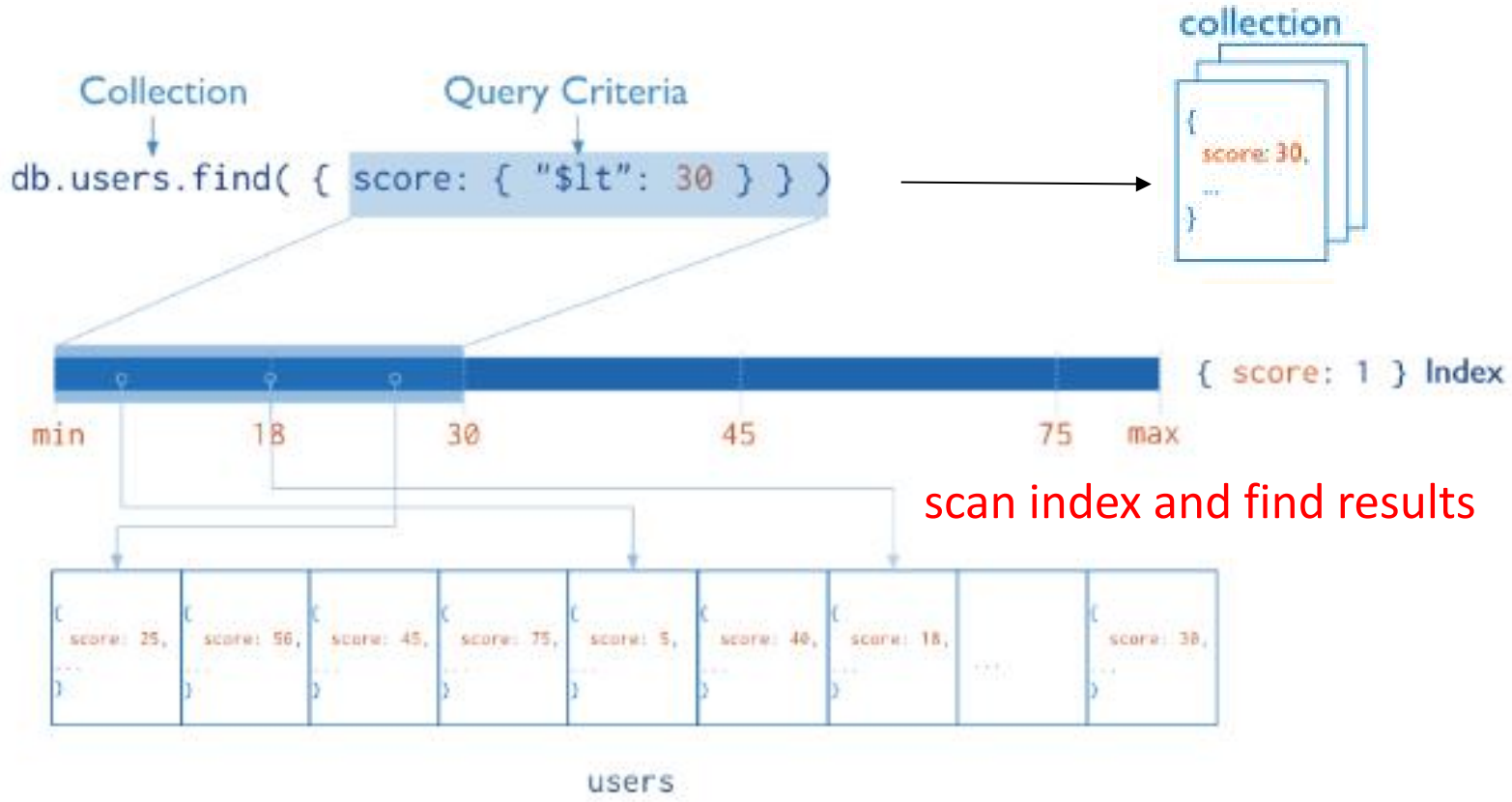


Diagram of a query that uses an index to select

Create Index in MongoDB

Types

- Single Field Indexes
- Compound Field Indexes
- Multikey Indexes

Single Field Indexes

db.users.ensureIndex({ score: 1 })

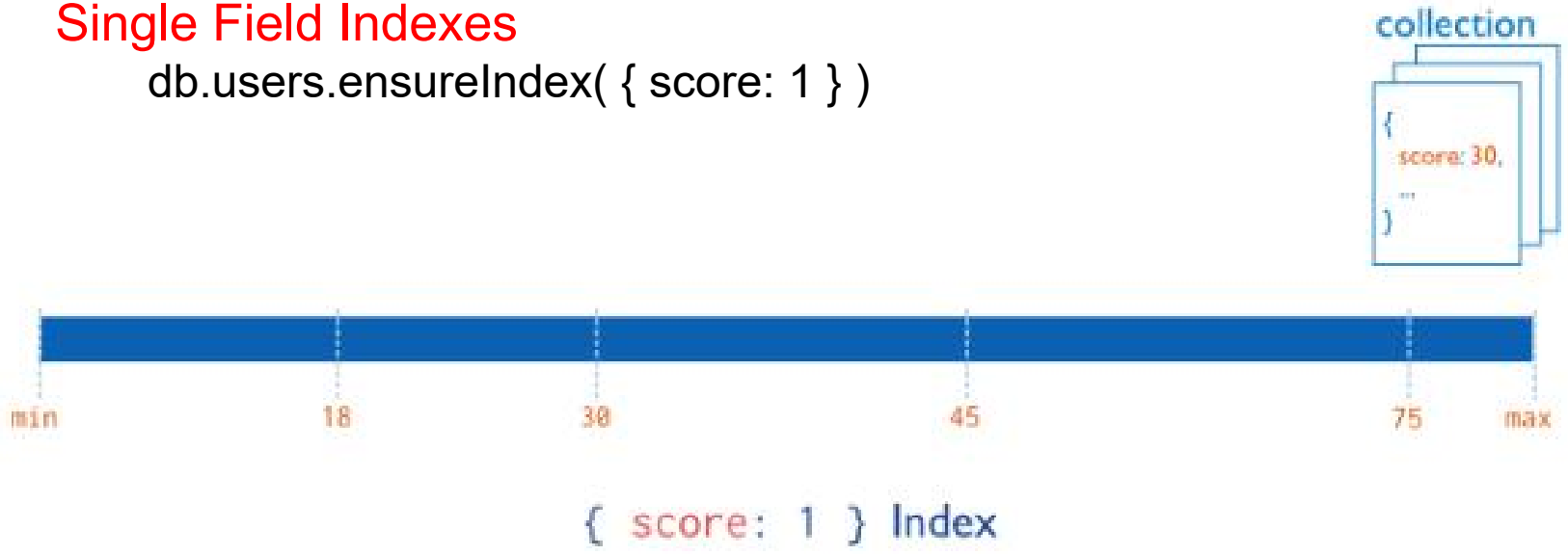


Diagram of an index on the score field (ascending).

Compound Field Index

Types

- Single Field Indexes
- Compound Field Indexes
- Multikey Indexes

Compound Field Indexes

db.users.ensureIndex({ userid:1, score: -1 })

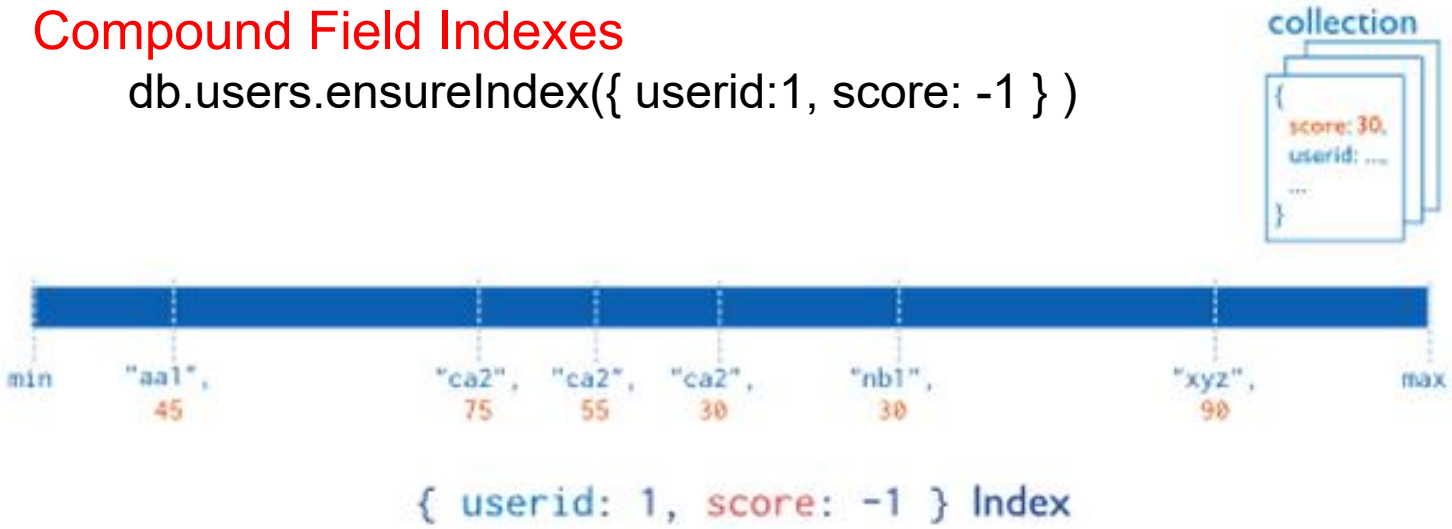


Diagram of a compound index on the `userid` field (ascending) and the `score` field (descending). The index sorts first by the `userid` field and then by the `score` field.

Multikey Indexes

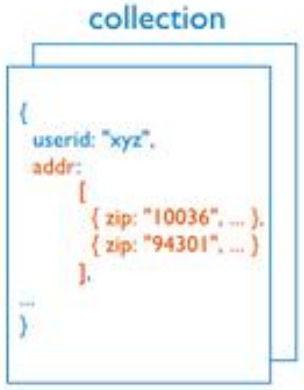
Types

- Single Field Indexes
- Compound Field Indexes
- Multikey Indexes

In single field index, there is only one index key in each document

Multikey Indexes

db.users.ensureIndex({addr.zip:1})



MongoDB creates an index key for each element in the array 'addr'.

Two index keys, 10036 and 94301, in this document

{ "addr.zip": 1 } Index

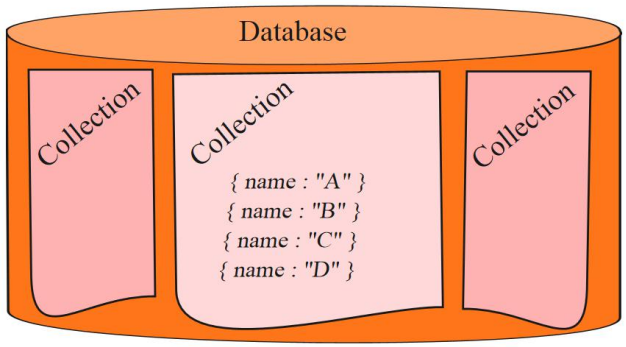
Diagram of a multikey index on the addr.zip field. The addr field contains an array of address documents. The address documents contain the zip field.

Index in MongoDB

- Creation index
  - db.users.ensureIndex({ score: 1 })
- Show existing indexes
  - db.users.getIndexes()
- Drop index
  - db.users.dropIndex({score: 1})
- Explain—Explain
  - db.users.find().explain()
  - Returns a document that describes the process and indexes
- Hint
  - db.users.find().hint({score: 1})
  - Overide MongoDB's default index selection

Demo of indexes in MongoDB

- Import Data
- Create Index
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Show Existing Index
- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes
- Explain
- Compare with data without indexes



Import Data: zips Collection

```
> db.zips.find().limit(20)
{ "city" : "ACMAR", "loc" : [  -86.51557,  33.584132 ], "pop" : 6055, "state" : "AL", "_id" : "35004" }
{ "city" : "ADAMSVILLE", "loc" : [  -86.959727,  33.588437 ], "pop" : 10616, "state" : "AL", "_id" : "35005" }
{ "city" : "ADGER", "loc" : [  -87.167455,  33.434277 ], "pop" : 3205, "state" : "AL", "_id" : "35006" }
{ "city" : "KEYSTONE", "loc" : [  -86.812861,  33.236868 ], "pop" : 14218, "state" : "AL", "_id" : "35007" }
{ "city" : "NEW SITE", "loc" : [  -85.951086,  32.941445 ], "pop" : 19942, "state" : "AL", "_id" : "35010" }
{ "city" : "ALPINE", "loc" : [  -86.208934,  33.331165 ], "pop" : 3062, "state" : "AL", "_id" : "35014" }
{ "city" : "ARAB", "loc" : [  -86.489638,  34.328339 ], "pop" : 13650, "state" : "AL", "_id" : "35016" }
{ "city" : "BAILEYTON", "loc" : [  -86.621299,  34.268298 ], "pop" : 1781, "state" : "AL", "_id" : "35019" }
{ "city" : "BESSEMER", "loc" : [  -86.947547,  33.409002 ], "pop" : 40549, "state" : "AL", "_id" : "35020" }
{ "city" : "HUEYTOWN", "loc" : [  -86.999607,  33.414625 ], "pop" : 39677, "state" : "AL", "_id" : "35023" }
{ "city" : "BLOUNTSVILLE", "loc" : [  -86.568628,  34.092937 ], "pop" : 9058, "state" : "AL", "_id" : "35031" }
{ "city" : "BREMEN", "loc" : [  -87.004281,  33.973664 ], "pop" : 3448, "state" : "AL", "_id" : "35033" }
{ "city" : "BRENT", "loc" : [  -87.211387,  32.93567 ], "pop" : 3791, "state" : "AL", "_id" : "35034" }
{ "city" : "BRIERFIELD", "loc" : [  -86.951672,  33.042747 ], "pop" : 1282, "state" : "AL", "_id" : "35035" }
{ "city" : "CALERA", "loc" : [  -86.755987,  33.1098 ], "pop" : 4675, "state" : "AL", "_id" : "35040" }
{ "city" : "CENTREVILLE", "loc" : [  -87.11924,  32.950324 ], "pop" : 4902, "state" : "AL", "_id" : "35042" }
{ "city" : "CHELSEA", "loc" : [  -86.614132,  33.371582 ], "pop" : 4781, "state" : "AL", "_id" : "35043" }
{ "city" : "COOSA PINES", "loc" : [  -86.337622,  33.266928 ], "pop" : 7985, "state" : "AL", "_id" : "35044" }
{ "city" : "CLANTON", "loc" : [  -86.642472,  32.835532 ], "pop" : 13990, "state" : "AL", "_id" : "35045" }
{ "city" : "CLEVELAND", "loc" : [  -86.559355,  33.992106 ], "pop" : 2369, "state" : "AL", "_id" : "35049" }
> db.zips.find().count()
29467
```

Create Index

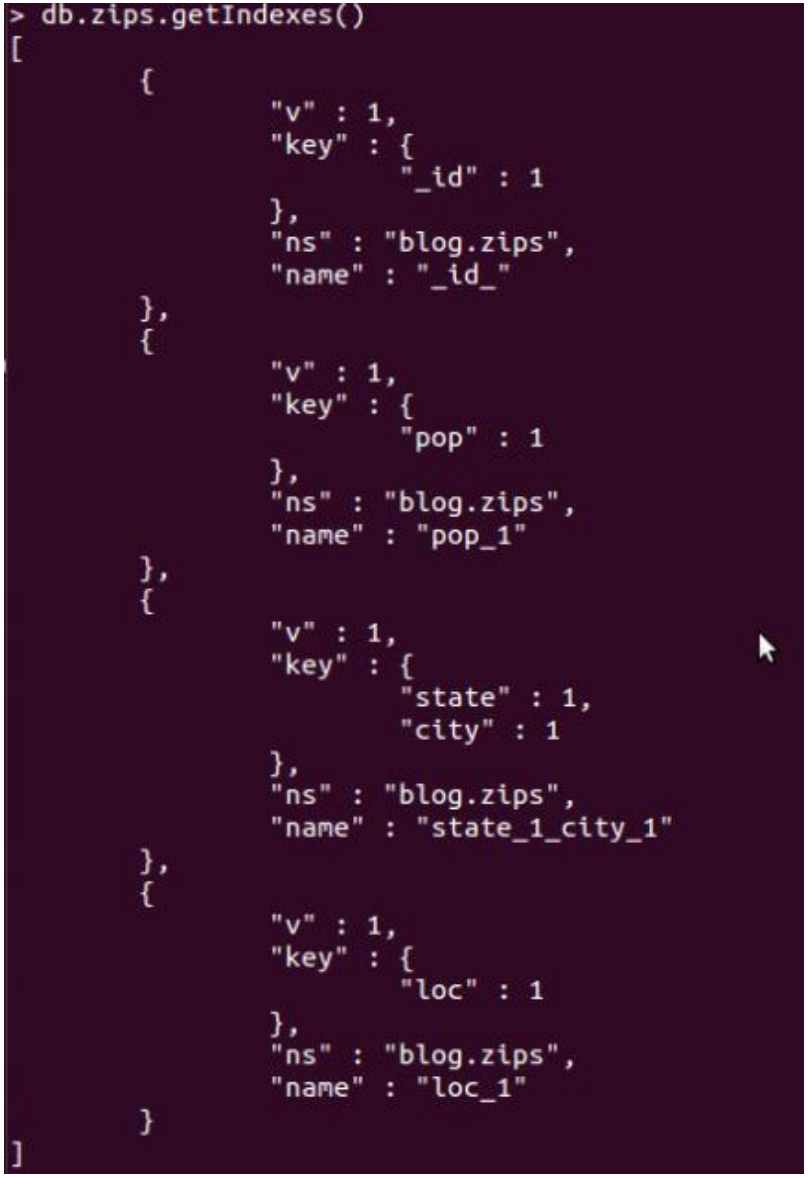
- Single Field Index
- Compound Field Indexes
- Multikey Indexes

```
db.zips.ensureIndex({pop: -1})
db.zips.ensureIndex({state: 1, city: 1})
db.zips.ensureIndex({loc: -1})
```

Show Existing Index

DB: blog
Collection: zips

- One identifier index
- Three user defined indices.



Hint

- Hint: force MongoDB to use the specified index
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes

Pop in decreasing order

```
> db.zips.find().limit(20).hint({pop: -1})
{ "city" : "CHICAGO", "loc" : [  -87.7157,  41.849015 ], "pop" : 112047, "state" : "IL", "_id" : "60623" }
{ "city" : "BROOKLYN", "loc" : [  -73.956985,  40.646694 ], "pop" : 111396, "state" : "NY", "_id" : "11226" }
{ "city" : "NEW YORK", "loc" : [  -73.958805,  40.768476 ], "pop" : 106564, "state" : "NY", "_id" : "10021" }
{ "city" : "NEW YORK", "loc" : [  -73.968312,  40.797466 ], "pop" : 100027, "state" : "NY", "_id" : "10025" }
{ "city" : "BELL GARDENS", "loc" : [  -118.17205,  33.969177 ], "pop" : 99568, "state" : "CA", "_id" : "90201" }
{ "city" : "CHICAGO", "loc" : [  -87.556012,  41.725743 ], "pop" : 98612, "state" : "IL", "_id" : "60617" }
{ "city" : "LOS ANGELES", "loc" : [  -118.258189,  34.007856 ], "pop" : 96074, "state" : "CA", "_id" : "90011" }
{ "city" : "CHICAGO", "loc" : [  -87.704322,  41.920903 ], "pop" : 95971, "state" : "IL", "_id" : "60647" }
{ "city" : "CHICAGO", "loc" : [  -87.624277,  41.693443 ], "pop" : 94317, "state" : "IL", "_id" : "60628" }
{ "city" : "NORWALK", "loc" : [  -118.081767,  33.90564 ], "pop" : 94188, "state" : "CA", "_id" : "90650" }
{ "city" : "CHICAGO", "loc" : [  -87.654251,  41.741119 ], "pop" : 92005, "state" : "IL", "_id" : "60620" }
{ "city" : "CHICAGO", "loc" : [  -87.706936,  41.778149 ], "pop" : 91814, "state" : "IL", "_id" : "60629" }
{ "city" : "CHICAGO", "loc" : [  -87.653279,  41.809721 ], "pop" : 89762, "state" : "IL", "_id" : "60609" }
{ "city" : "CHICAGO", "loc" : [  -87.704214,  41.946401 ], "pop" : 88377, "state" : "IL", "_id" : "60618" }
{ "city" : "JACKSON HEIGHTS", "loc" : [  -73.878551,  40.740388 ], "pop" : 88241, "state" : "NY", "_id" : "11373" }
{ "city" : "ARLETA", "loc" : [  -118.420692,  34.258081 ], "pop" : 88114, "state" : "CA", "_id" : "91331" }
{ "city" : "BROOKLYN", "loc" : [  -73.914483,  40.662474 ], "pop" : 87079, "state" : "NY", "_id" : "11212" }
{ "city" : "SOUTH GATE", "loc" : [  -118.201349,  33.94617 ], "pop" : 87026, "state" : "CA", "_id" : "90280" }
{ "city" : "RIDGEWOOD", "loc" : [  -73.896122,  40.703613 ], "pop" : 85732, "state" : "NY", "_id" : "11385" }
{ "city" : "BRONX", "loc" : [  -73.871242,  40.873671 ], "pop" : 85710, "state" : "NY", "_id" : "10467" }
```

Hint

- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes

State and city in increasing order

```
> db.zips.find().limit(20).hint({state: 1, city: 1})
{ "city" : "98791", "loc" : [  -176.310048,  51.938901 ], "pop" : 5345, "state" : "AK", "_id" : "98791" }
{ "city" : "AKHIOK", "loc" : [  -152.500169,  57.781967 ], "pop" : 13309, "state" : "AK", "_id" : "99615" }
{ "city" : "AKIACHAK", "loc" : [  -161.39233,  60.891854 ], "pop" : 481, "state" : "AK", "_id" : "99551" }
{ "city" : "AKIAK", "loc" : [  -161.199325,  60.890632 ], "pop" : 285, "state" : "AK", "_id" : "99552" }
{ "city" : "AKUTAN", "loc" : [  -165.785368,  54.143012 ], "pop" : 589, "state" : "AK", "_id" : "99553" }
{ "city" : "ALAKANUK", "loc" : [  -164.60228,  62.746967 ], "pop" : 1186, "state" : "AK", "_id" : "99554" }
{ "city" : "ALEKNAGIK", "loc" : [  -158.619882,  59.269688 ], "pop" : 185, "state" : "AK", "_id" : "99555" }
{ "city" : "ALLAKAKET", "loc" : [  -152.712155,  66.543197 ], "pop" : 170, "state" : "AK", "_id" : "99720" }
{ "city" : "AMBLER", "loc" : [  -156.455652,  67.46951 ], "pop" : 8, "state" : "AK", "_id" : "99786" }
{ "city" : "ANAKTUVUK PASS", "loc" : [  -151.679005,  68.11878 ], "pop" : 260, "state" : "AK", "_id" : "99721" }
{ "city" : "ANCHORAGE", "loc" : [  -149.876077,  61.211571 ], "pop" : 14436, "state" : "AK", "_id" : "99501" }
{ "city" : "ANCHORAGE", "loc" : [  -150.093943,  61.096163 ], "pop" : 15891, "state" : "AK", "_id" : "99502" }
{ "city" : "ANCHORAGE", "loc" : [  -149.893844,  61.189953 ], "pop" : 12534, "state" : "AK", "_id" : "99503" }
{ "city" : "ANCHORAGE", "loc" : [  -149.74467,  61.203696 ], "pop" : 32383, "state" : "AK", "_id" : "99504" }
{ "city" : "ANCHORAGE", "loc" : [  -149.828912,  61.153543 ], "pop" : 20128, "state" : "AK", "_id" : "99507" }
{ "city" : "ANCHORAGE", "loc" : [  -149.810085,  61.205959 ], "pop" : 29857, "state" : "AK", "_id" : "99508" }
{ "city" : "ANCHORAGE", "loc" : [  -149.897401,  61.119381 ], "pop" : 17094, "state" : "AK", "_id" : "99515" }
{ "city" : "ANCHORAGE", "loc" : [  -149.779998,  61.10541 ], "pop" : 18356, "state" : "AK", "_id" : "99516" }
{ "city" : "ANCHORAGE", "loc" : [  -149.936111,  61.190136 ], "pop" : 15192, "state" : "AK", "_id" : "99517" }
{ "city" : "ANCHORAGE", "loc" : [  -149.886571,  61.154862 ], "pop" : 8116, "state" : "AK", "_id" : "99518" }
```

Hint

- Hint
  - Single Field Index
  - Compound Field Indexes
  - Multikey Indexes

loc in decreasing order

```
> db.zips.find().limit(20).hint({loc: -1})
{ "city" : "BARROW", "loc" : [  -156.817409,  71.234637 ], "pop" : 3696, "state" : "AK", "_id" : "99723" }
{ "city" : "WAINWRIGHT", "loc" : [  -160.012532,  70.620064 ], "pop" : 492, "state" : "AK", "_id" : "99782" }
{ "city" : "NUIQSUT", "loc" : [  -150.997119,  70.192737 ], "pop" : 354, "state" : "AK", "_id" : "99789" }
{ "city" : "PRUDHOE BAY", "loc" : [  -148.559636,  70.070057 ], "pop" : 153, "state" : "AK", "_id" : "99734" }
{ "city" : "KAKTOVIK", "loc" : [  -143.631329,  70.042889 ], "pop" : 245, "state" : "AK", "_id" : "99747" }
{ "city" : "POINT LAY", "loc" : [  -162.906148,  69.705626 ], "pop" : 139, "state" : "AK", "_id" : "99759" }
{ "city" : "POINT HOPE", "loc" : [  -166.72618,  68.312058 ], "pop" : 640, "state" : "AK", "_id" : "99766" }
{ "city" : "ANAKTUVUK PASS", "loc" : [  -151.679005,  68.11878 ], "pop" : 260, "state" : "AK", "_id" : "99721" }
{ "city" : "ARCTIC VILLAGE", "loc" : [  -145.423115,  68.077395 ], "pop" : 107, "state" : "AK", "_id" : "99722" }
{ "city" : "KIVALINA", "loc" : [  -163.733617,  67.665859 ], "pop" : 689, "state" : "AK", "_id" : "99750" }
{ "city" : "AMBLER", "loc" : [  -156.455652,  67.46951 ], "pop" : 8, "state" : "AK", "_id" : "99786" }
{ "city" : "KIANA", "loc" : [  -158.152204,  67.18026 ], "pop" : 349, "state" : "AK", "_id" : "99749" }
{ "city" : "BETTLES FIELD", "loc" : [  -151.062414,  67.100495 ], "pop" : 156, "state" : "AK", "_id" : "99726" }
{ "city" : "VENETIE", "loc" : [  -146.413723,  67.010446 ], "pop" : 184, "state" : "AK", "_id" : "99781" }
{ "city" : "NOATAK", "loc" : [  -160.509453,  66.97553 ], "pop" : 395, "state" : "AK", "_id" : "99761" }
{ "city" : "SHUNGNAK", "loc" : [  -157.613496,  66.958141 ], "pop" : 0, "state" : "AK", "_id" : "99773" }
{ "city" : "KOBUK", "loc" : [  -157.066864,  66.912253 ], "pop" : 306, "state" : "AK", "_id" : "99751" }
{ "city" : "KOTZEBUE", "loc" : [  -162.126493,  66.846459 ], "pop" : 3347, "state" : "AK", "_id" : "99752" }
{ "city" : "NOORVIK", "loc" : [  -161.044132,  66.836353 ], "pop" : 534, "state" : "AK", "_id" : "99763" }
{ "city" : "CHALKYITSIK", "loc" : [  -143.638121,  66.719 ], "pop" : 99, "state" : "AK", "_id" : "99788" }
```

Explain

- to provide information on the query plan

```
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
        "cursor" : "BtreeCursor state_1_city_1",
        "isMultiKey" : false,
        "n" : 19,
        "nscannedObjects" : 19,
        "nscanned" : 19,
        "nscannedObjectsAllPlans" : 19,
        "nscannedAllPlans" : 19,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 0,
        "indexBounds" : {
                "state" : [
                        [
                                "TN",
                                "TN"
                        ]
                ],
                "city" : [
                        [
                                "NASHVILLE",
                                "NASHVILLE"
                        ]
                ]
        },
        "server" : "g:27017"
}
```

with index

```
> db.zips.dropIndexes()
{
        "nIndexesWas" : 4,
        "msg" : "non-_id indexes dropped for collection",
        "ok" : 1
}
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
        "cursor" : "BasicCursor",
        "isMultiKey" : false,
        "n" : 19,
        "nscannedObjects" : 29467,
        "nscanned" : 29467,
        "nscannedObjectsAllPlans" : 29467,
        "nscannedAllPlans" : 29467,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 33,
        "indexBounds" : {

        },
        "server" : "g:27017"
}
```

without index

Replication & Sharding

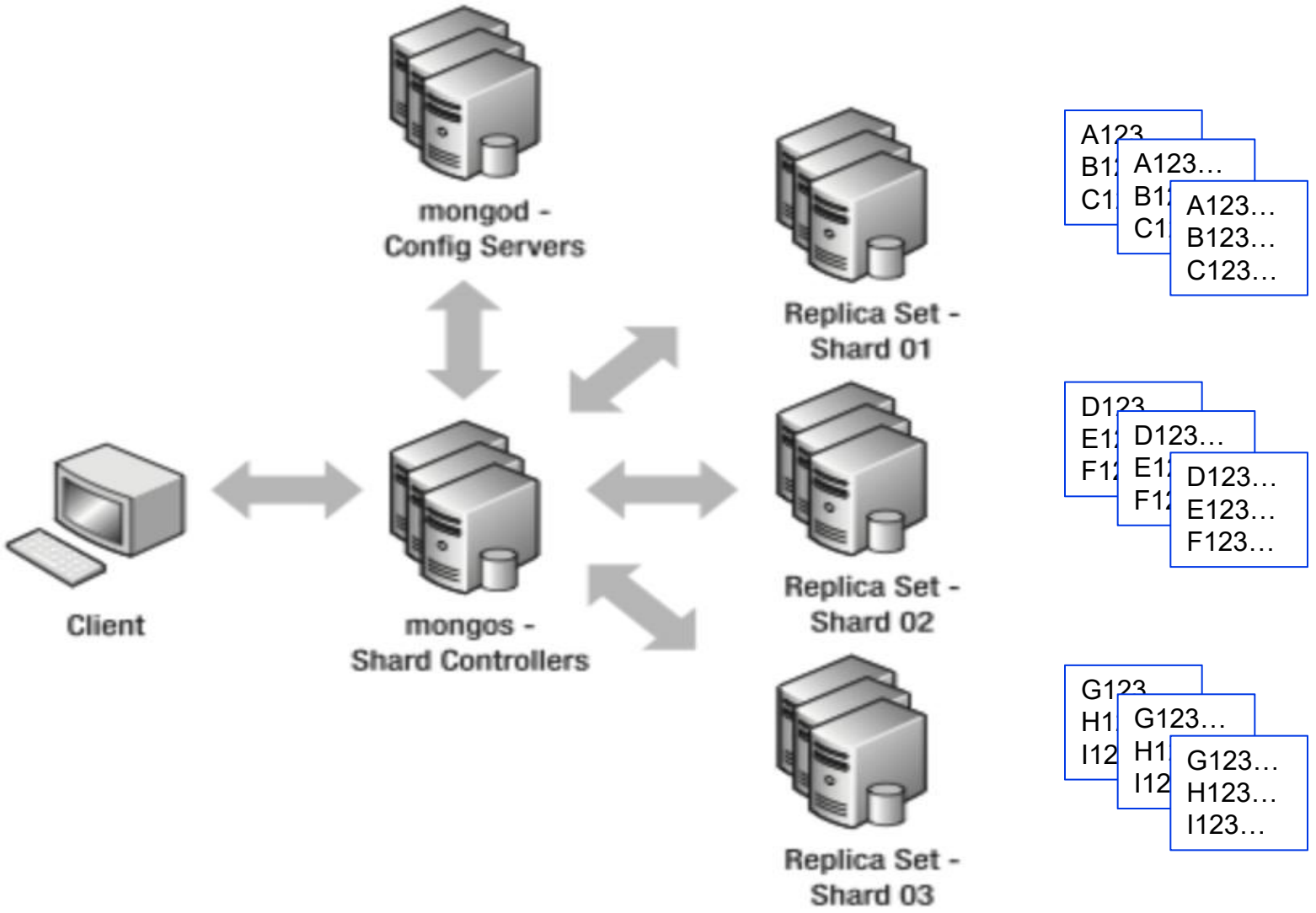


Image source: http://mongodb.in.th

Replication

- What is replication?
- Purpose of replication/redundancy
  - Fault tolerance
  - Availability
    - ✓Increase read capacity

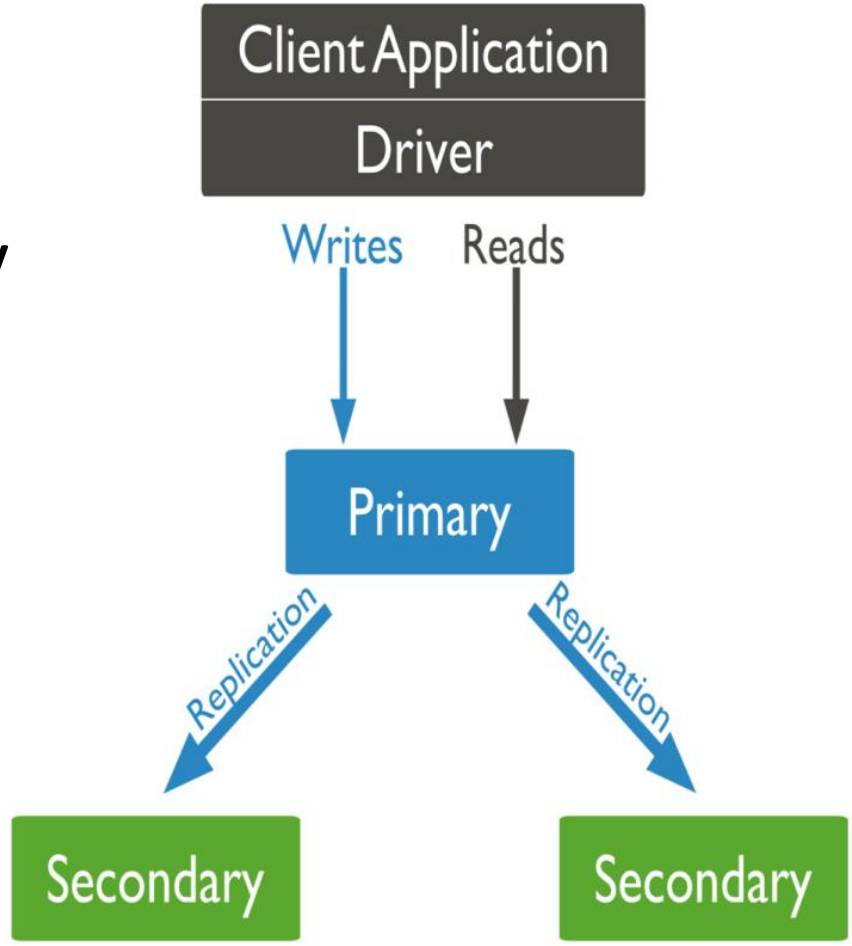


Figure 1: Diagram of default routing of reads and writes to the primary.

Replication in MongoDB

- Replica Set Members
  - Primary
    - ✓ Read, Write operations
  - Secondary
    - ✓ Asynchronous Replication
    - ✓ Can be primary
  - Arbiter
    - ✓ Voting
    - ✓ Can't be primary
  - Delayed Secondary
    - ✓ Can't be primary

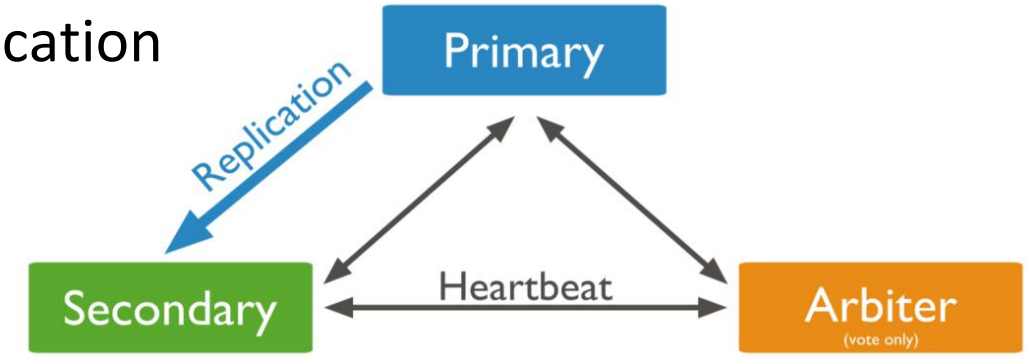


Figure 3: Diagram of a replica set that consists of a primary, a secondary, and an arbiter.

Replication in MongoDB

- Automatic Failover
  - Heartbeats (every 2 seconds)
  - Elections
- The Standard Replica Set Deployment
- Deploy an Odd Number of Members
- Rollback
- Security
  - SSL/TLS

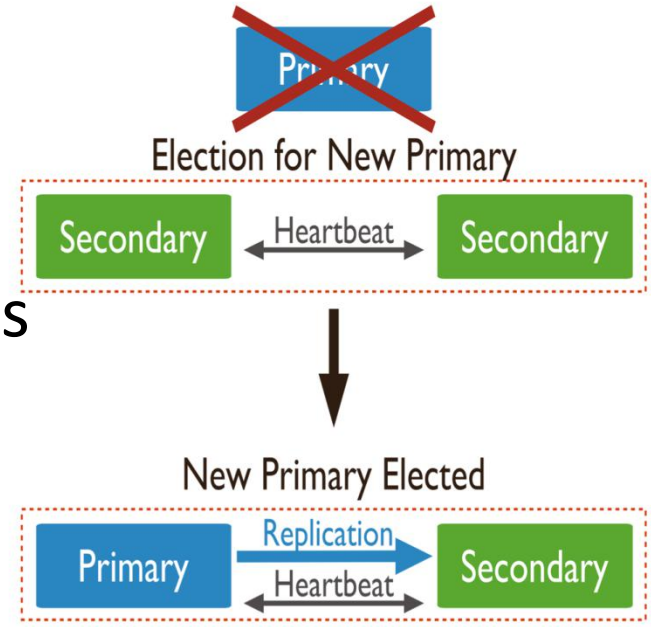
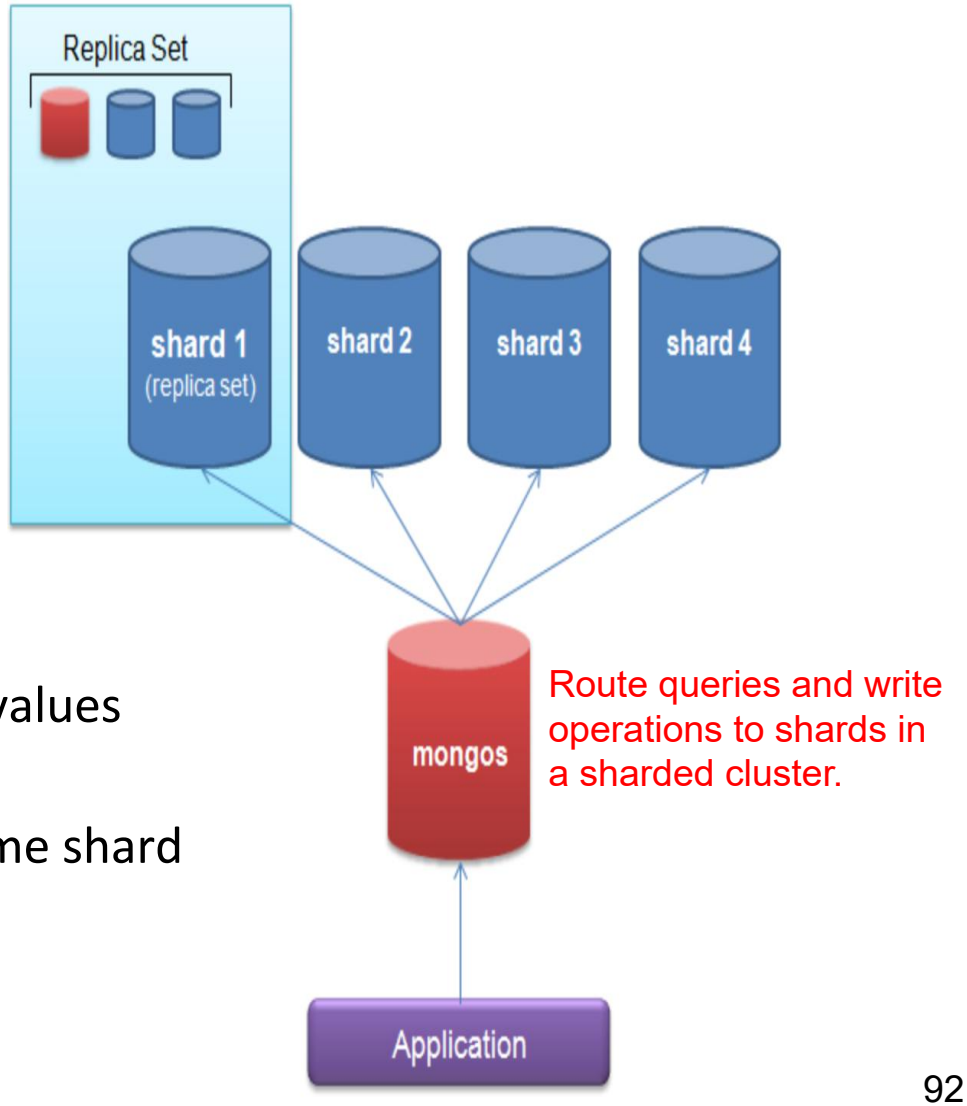


Figure 21: Diagram of an election of a new primary. In a three member replica set with two secondaries, the primary becomes unreachable. The loss of a primary triggers an election where one of the secondaries becomes the new primary

Sharding

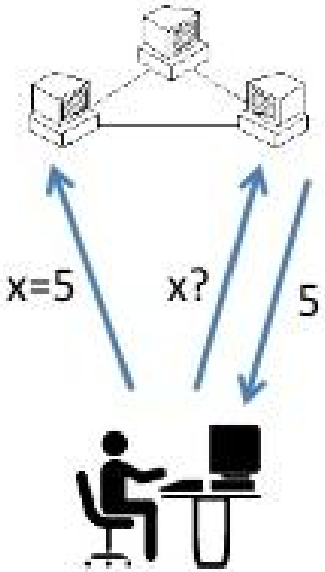
- What is sharding?
- Purpose of sharding
  - Horizontal scaling out
- Query Routers
  - mongos
- Shard keys
  - Range based sharding
  - Sufficient cardinality
    - Enough number of distinct values
  - Avoid hotspotting
    - All writes landing on the same shard



CAP Theorem

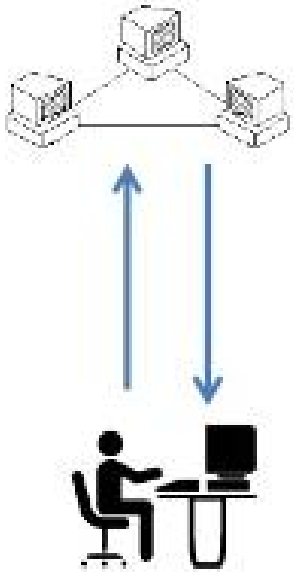
- It is not possible to guarantee all three of the desirable properties at the same time in a distributed system with data replication.
- In NOSQL systems, a weaker consistency level is often acceptable, and guaranteeing the other two is important.

Consistency



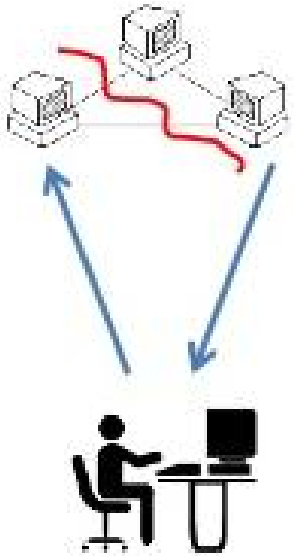
All clients always have the same view of the data.

Availability



Each client can always read and write.

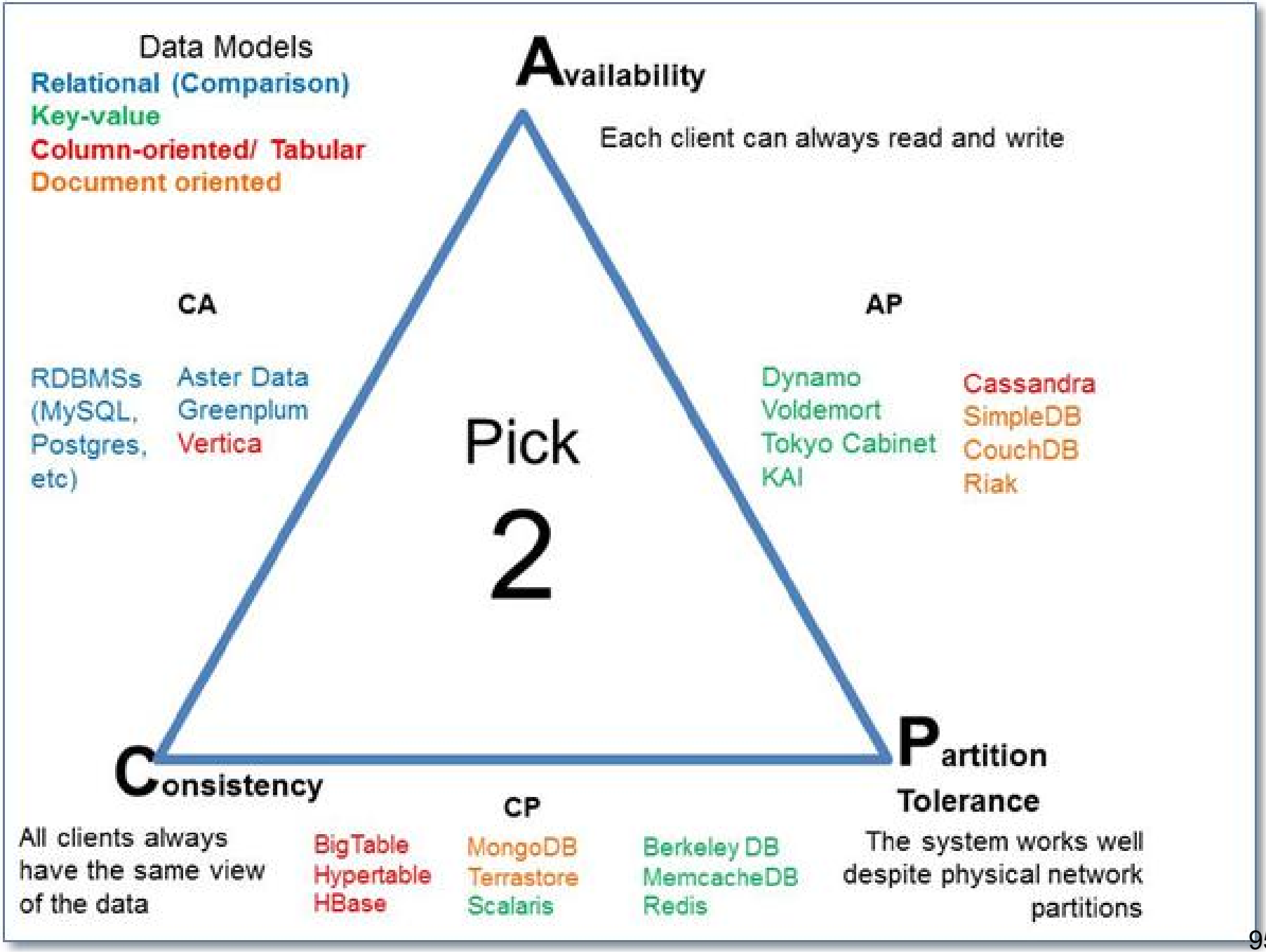
Partition tolerance



The system works well despite physical network partitions.

CAP Theorem

- No distributed system is safe from network failures, thus network partitioning generally has to be tolerated.
- In the presence of a partition, one is then left with two options: consistency or availability.
- When choosing consistency over availability, the system will return an error or a time out if particular information cannot be guaranteed to be up to date due to network partitioning.
- When choosing availability over consistency, the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up to date due to network partitioning.
- In the absence of network failure -- that is, when the distributed system is running normally -- both availability and consistency can be satisfied.


Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
Pick 2
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
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