CC5212-1

Procesamiento Masivo de Datos

Otoño 2018

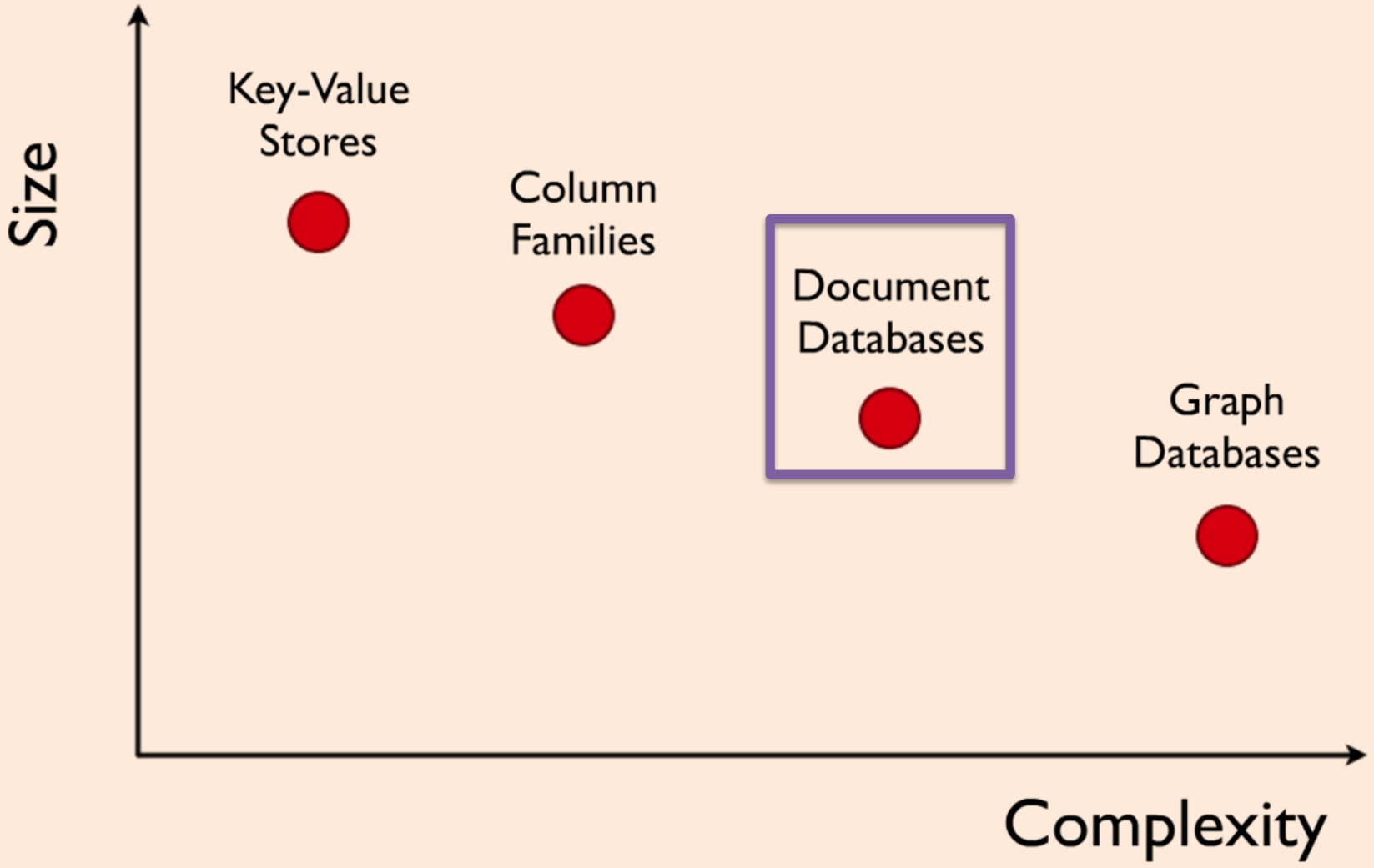
Lecture 9

NoSQL: MongoDB

Aidan Hogan

aidhog@gmail.com

NoSQL


Size
Key-Value Stores
Column Families
Document Databases
Graph Databases
Complexity

343 systems in ranking, July 2018

Rank Jul 2018	Rank Jun 2018	Rank Jul 2017	DBMS	Database Model	Score Jul 2018	Score Jun 2018	Score Jul 2017
1.	1.	1.	Oracle	Relational DBMS	1277.79	-33.47	-97.09
2.	2.	2.	MySQL	Relational DBMS	1196.07	-37.62	-153.04
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1053.41	-34.32	-172.59
4.	4.	4.	PostgreSQL	Relational DBMS	405.81	-4.86	+36.37
5.	5.	5.	MongoDB	Document store	350.33	+6.54	+17.56
6.	6.	6.	DB2	Relational DBMS	186.20	+0.56	-5.05
7.	7.	↑ 9.	Redis	Key-value store	139.91	+3.61	+18.40
8.	8.	↑ 10.	Elasticsearch	Search engine	136.22	+5.18	+20.25
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	132.58	+1.59	+6.45
10.	10.	↓ 8.	Cassandra	Wide column store	121.06	+1.84	-3.07
11.	11.	11.	SQLite	Relational DBMS	115.28	+1.02	+1.41
12.	12.	12.	Teradata	Relational DBMS	78.22	+2.45	-0.14
13.	↑ 14.	↑ 16.	Splunk	Search engine	69.24	+3.46	+8.94
14.	↓ 13.	↑ 18.	MariaDB	Relational DBMS	67.51	+1.67	+13.15
15.	↑ 16.	↓ 13.	SAP Adaptive Server	Relational DBMS	62.12	+0.64	-4.79
16.	↓ 15.	↓ 14.	Solr	Search engine	61.52	-0.55	-4.51
17.	17.	↓ 15.	HBase	Wide column store	60.77	+1.07	-2.85
18.	18.	↑ 20.	Hive	Relational DBMS	57.63	+0.30	+11.42
19.	19.	↓ 17.	FileMaker	Relational DBMS	56.39	+0.21	-2.26
20.	20.	↓ 19.	SAP HANA	Relational DBMS	51.60	+2.25	+3.65
21.	21.	↑ 22.	Amazon DynamoDB	Multi-model	49.63	+3.84	+13.17
22.	22.	↓ 21.	Neo4j	Graph DBMS	41.88	-0.09	+3.36
23.	23.	↑ 24.	Memcached	Key-value store	33.88	+0.07	+5.35
24.	24.	↓ 23.	Couchbase	Document store	33.07	+0.62	+0.06
25.	↑ 26.	↑ 26.	Microsoft Azure SQL Database	Relational DBMS	26.84	+0.55	+4.55
26.	↓ 25.	↓ 25.	Informix	Relational DBMS	26.59	+0.03	-1.08
27.	27.	↑ 28.	Vertica	Relational DBMS	20.82	-0.34	-0.97
28.	28.	↑ 30.	Firebird	Relational DBMS	20.66	+0.27	+1.67
29.	29.	↓ 27.	CouchDB	Document store	19.50	-0.70	-2.65
30.	30.	↑ 41.	Microsoft Azure Cosmos DB	Multi-model	19.45	+0.25	+11.74

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

Document Stores

Key–Value: a Distributed Map

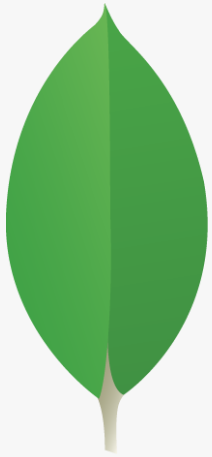
Countries	
Primary Key	**Value**
Afghanistan	capital:Kabul,continent:Asia,pop:31108077#2011
…	…

Tabular: Multi-dimensional Maps

Countries				
Primary Key	**capital**	**continent**	**pop-value**	**pop-year**
Afghanistan	Kabul	Asia	31108077	2011
…	…	…	…	…

Document: Value is a document

Countries	
Primary Key	**Value**
Afghanistan	{ cap: “Kabul”, con: “Asia”, pop: { val: 31108077, y: 2011 } }
…	…



mongoDB®

{JSON}

JavaScript Object Notation


mongoDB.

joepie91's Ramblings RSS

Why you should never, ever, ever use MongoDB

19 Jul 2015

MongoDB is evil. It...

- ... loses data (sources: 1, 2)
- ... in fact, for a long time, ignored errors by default and assumed every single write succeeded no matter what (which on 32-bits systems led to losing all data silently after some 3GB, due to MongoDB limitations)
- ... is slow, *even* at its advertised usecases, and claims to the contrary are completely lacking evidence (sources: 3, 4)
- ... forces the poor habit of implicit schemas in nearly all usecases (sources: 4)
- ... has locking issues (sources: 4)
- ... has an atrociously poor response time to security issues - it took them **two years** to patch an insecure default configuration that would expose *all of your data* to anybody who asked, without authentication (sources: 5)
- ... is not ACID-compliant (sources: 6)
- ... is a nightmare to scale and maintain
- ... isn't even exclusive in its offering of JSON-based storage; PostgreSQL does it too, and other (better) document stores like CouchDB have been around for a long time (sources: 7, 8)

http://cryto.net/~joepie91/blog/2015/07/19/why-you-should-never-ever-ever-use-mongodb/

MongoDB:
Data model

JavaScript Object Notation: JSON


```
{
  "id": 179,
  "name": "The Wire",
  "type": "Scripted",
  "language": "English",
  "genres": [ "Drama", "Crime", "Thriller" ],
  "status": "Ended",
  "runtime": 60,
  "premiered": "2002-06-02",
  "schedule": {
    "time": "21:00",
    "days": [
      "Sunday"
    ]
  },
  "rating": {
    "average": 9.4
  }
}
```


Binary JSON: BSON

```
{
  "_id": ObjectId(99a88b77c66d),
  "name": "The Wire",
  "type": "Scripted",
  "language": "English",
  "genres": [ "Drama", "Crime", "Thriller" ],
  "status": "Ended",
  "runtime": 60,
  "premiered": ISODate("2002-06-02"),
  "schedule": {
    "time": "21:00",
    "days": [
      "Sunday"
    ]
  },
  "rating": {
    "average": 9.4
  }
}
```

MongoDB: Datatypes

Boolean: `true`

String: `"sí"`

Array: `[]`

Object: {}

Double: `43.2`

{JSON}
JavaScript Object Notation

etc.

ObjectID: `ObjectId(0A0A0A0A0A0A)`

Date: `ISODate("2003-14-15T09:26:53.589Z")`

BSON{01010100 11101011 10101110 01010101}

etc.

MongoDB: Map from keys to BSON values

TVSeries	
Key	**BSON Value**

99a88b77c66d	<pre>{
	  "_id": ObjectId(99a88b77c66d),
	  "name": "The Wire",
	  "type": "Scripted",
	  "language": "English",
	  "genres": ["Drama", "Crime", "Thriller"],
	  "status": "Ended",
	  "runtime": 60,
	  "premiered": ISODate("2002-06-02"),
	  "schedule": {
	  "time": "21:00",
	  "days": [
	  "Sunday"
	  ]
	  },
	  "rating": {
	  "average": 9.4
	  }
	}</pre>

…	…

MongoDB Collection: Similar Documents

TVSeries	
Key	**BSON Value**
99a88b77c66d	`{ "_id": ObjectId(99a88b77c66d), "name": "The Wire", "type": "Scripted", ... }`
11f22e33d44c	`{ "_id": ObjectId(11f22e33d44c), "name": "Rick and Morty", "type": "Animation", ... }`

MongoDB Database: Related Collections

TVSeries

Key	BSON Value
…	…

TVEpisodes

Key	BSON Value
…	…

TVNetworks

Key	BSON Value
…	…

database: TV

Load database `tvdb` (and create if not exists):

```
> use tvdb

switched to db tvdb
```

See all (non-empty) databases (`tvdb` is empty):

```
> show dbs

local     0.00001 GB
test      0.00231 GB
```

See current database:

```
> db

tvdb
```

Drop current database:

```
> db.dropDatabase()

{ "dropped" : "tvdb", "ok" : 1 }
```

Create collection `series`:

```
> db.createCollection("series")

{ "ok" : 1 }
```

See all collections in current database:

```
> show collections

series
```

Drop collection `series`:

```
> db.series.drop()

true
```

Clear all documents from collection `series`:

```
> db.series.remove( {} )

WriteResult({ "nRemoved" : 0 })
```

Create capped collection (keeps only most recent):

```
> db.createCollection("last100episodes",
   { capped: true, size: 12285600, max: 100 } )

{ "ok" : 1 }
```

Create collection with default index on `_id`:

```
> db.createCollection("cast", { autoIndexId: true } )

{
    "note" : "the autoIndexId option is deprecated ...",
    "ok" : 1
}
```

MongoDB:
Inserting Data

Insert document: without _id

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "type": "Scripted",
    "runtime": 60,
    "genres": [
      "Science-Fiction",
      "Thriller"
    ]
  })

WriteResult({ "nInserted" : 1 })
```

Insert document: with _id

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "type": "Scripted",
    "runtime": 60,
    "genres": [
      "Science-Fiction",
      "Thriller"
    ]
  })

WriteResult({ "nInserted" : 1 })
```

… fails if _id already exists

… use update or save to update existing document(s)

Use save and _id to overwrite:

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "type": "Scripted",
    "runtime": 60
  })

WriteResult({ "nInserted" : 1 })
```

```
> db.series.save(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror (Overwritten)"
  })

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

... overwrites old document

MongoDB:

Queries with Selection

Query documents in a collection with `find`:

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "type": "Scripted",
    "runtime": 60
  })
```

```
> db.series.find()

{ "_id" : ObjectId("5951e0b265ad257d48f4a7d5"), "name" : "Black Mirror",
"type" : "Scripted", "runtime" : 60 }
```

… use `findOne()` to return one document

Pretty print results with `pretty`:

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "type": "Scripted",
    "runtime": 60
  })
```

```
> db.series.find().pretty()

{
    "_id" : ObjectId("5951e0b265ad257d48f4a7d5"),
    "name" : "Black Mirror",
    "type" : "Scripted",
    "runtime" : 60
}
```

Selection: find documents matching σ

```
> db.series.find(σ)
```

Selection σ: Equality

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "type": "Scripted",
    "runtime": 60
  })
```

Equality: `{ key: value }`

```
> db.series.find( { "type": "Scripted" } )

{ "name" : "Black Mirror", "type" : "Scripted", "runtime" : 60 }
```

Results would include `_id` but for brevity, we will omit this from examples where it is not important.

Selection σ: Nested key

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "rating": { "avg": 9.4 },
    "runtime": 60
  })
```

Key can access nested values:

```
> db.series.find( { "rating.avg": 9.4 } )

{ "name" : "Black Mirror", "rating": { "avg": 9.4 }, "runtime" : 60 }
```

Selection σ: Equality on null

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "rating": { "avg": null },
    "runtime": 60
  })
```

Equality on nested null value:

```
> db.series.find( { "rating.avg": null } )

{ "name" : "Black Mirror", "rating": { "avg": null }, "runtime" : 60 }
```

... matches when value is null or ...

Selection σ: Equality on null

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "rating": { "val": 9.4 },
    "runtime": 60
  })
```

Key can access nested values:

```
> db.series.find( { "rating.avg": null } )

{ "name" : "Black Mirror", "rating": { "val": 9.4 }, "runtime" : 60 }
```

... when field doesn't exist with key.

Selection σ: Equality on document

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "rating": { "avg": 9.4 },
    "runtime": 60
  })
```

Value can be an object/document:

```
> db.series.find( { "rating": { "avg": 9.4 } } )

{ "name" : "Black Mirror", "rating": { "avg": 9.4 }, "runtime" : 60 }
```

Selection σ: Equality on document

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "rating": { "avg": 9.4, "votes": 9001 },
    "runtime": 60
  })
```

Value can be an object/document:

```
> db.series.find(
  { "rating": { "votes": 9001 } }
)
```

… no results: needs to match full object.

Selection σ: Equality on document

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "rating": { "avg": 9.4, "votes": 9001 },
    "runtime": 60
  })
```

Value can be an object/document:

```
> db.series.find(
  { "rating": { "votes": 9001, "avg": 9.4 } }
)
```

… no results: order of attributes matters.

Selection σ: Equality on exact array

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Equality can match an exact array:

```
> db.series.find( { "genres": [ "Science-Fiction",
                                "Thriller" ] } )

{ "name" : "Black Mirror", "genres": [ "Science-Fiction", "Thriller" ],
"runtime" : 60 }
```

Selection σ: Equality on exact array

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Equality can match an exact array

```
> db.series.find( { "genres": [ "Science-Fiction" ] } )
```

… no results: needs to match full array.

Selection σ: Equality on exact array

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Equality can match an exact array

```
> db.series.find( { "genres": [ "Thriller",
                                "Science-Fiction" ] } )
```

… no results: order of elements matters.

Selection σ: Equality matches inside array

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Equality matches a value in an array:

```
> db.series.find( { "genres": "Thriller" } )

{ "name" : "Black Mirror", "genres": [ "Science-Fiction", "Thriller" ],
"runtime" : 60 }
```

Selection σ: Equality matches both

```
> db.series.insert( { "name": "A" , "val": [ 5, 6 ] } )
> db.series.insert( { "name": "B", "val": 5 } )
```

Equality matches a value inside and outside an array:

```
> db.series.find( { "val": 5 } )

{ "name": "A" , "val": [ 5, 6 ] }
{ "name": "B" , "val": 5 }
```

cough

Selection σ: Inequalities

Less than:	`{ key: { $lt: value } }`
Greater than:	`{ key: { $gt: value } }`
Less than or equal:	`{ key: { $lte: value } }`
Greater than or equal:	`{ key: { $gte: value } }`
Not equals:	`{ key: { $ne: value } }`

Selection σ: Less Than

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Less than: `{ key: { $lt: value } }`

```
> db.series.find({ "runtime": { $lt: 70 } })

{ "name" : "Black Mirror", "genres": [ "Science-Fiction", "Thriller" ],
"runtime" : 60 }
```

Selection σ: Match one of multiple values

Match any value: `{ key: { $in: [ v1, …, vn ] } }`

Match no value: `{ key: { $nin: [ v1, …, vn ] } }`

… also passes if key does not exist.

Selection σ: Match one of multiple values

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Match any value: `{ key: { $in: [ v1, …, vn ] } }`

```
> db.series.find({ "runtime": { $in: [30, 60] } })

{ "name" : "Black Mirror", "genres": [ "Science-Fiction", "Thriller" ],
"runtime" : 60 }
```

Selection σ: Match one of multiple values

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Match any value: `{ key: { $in: [ v1, …, vn ] } }`

```
> db.series.find({ "genres": { $in: ["Noir", "Thriller"] } })

{ "name" : "Black Mirror", "genres": [ "Science-Fiction", "Thriller" ],
"runtime" : 60 }
```

… if key references an array, any value of array should match any value of `$in`

Selection σ: Boolean connectives

And: `{ $and: [ σ , σ′ ] }`

Or: `{ $or: [ σ , σ′ ] }`

Not: `{ $not: [ σ ] }`

Nor: `{ $nor: [ σ , σ′ ] }`

… of course, can nest such conditions

Selection σ: And

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

And: `{ $and: [ σ , σ′ ] }`

```
> db.series.find({ $and: [
    { "runtime": { $in: [30, 60] } } ,
    { "name": { $ne: "Lost" } } ] }
)

{ "name" : "Black Mirror", "genres": [ "Science-Fiction", "Thriller" ],
"runtime" : 60 }
```

Selection σ: Attribute (not) exists

Exists: `{ key: { $exists : true } }`

Not Exists: `{ key: { $exists : false } }`

Selection σ: Attribute exists

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Exists: `{ key: { $exists : true } }`

```
> db.series.find({ "name": { $exists : true } })

{ "name" : "Black Mirror", "genres": [ "Science-Fiction", "Thriller" ],
"runtime" : 60 }
```

… checks that the field key exists
(even if value is NULL)

Selection σ: Attribute not exists

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Not exists: `{ key: { $exists : false } }`

```
> db.series.find({ "name": { $exists : false } })
```

… checks that the field key doesn't exist (empty results)

Selection σ: Arrays

All: `{ key: { $all : [v1, …, vn] } }`

Match one: `{ key: { $elemMatch : {σ1, …, σn } } }`

Size: `{ key: { $size : int } }`

Selection σ: Array contains (at least) all elements

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Sci-Fi", "Thriller", "Comedy" ],
    "runtime": 60
  })
```

All: `{ key: { $all : [v1, …, vn] } }`

```
> db.series.find(
  { "genres": { $all : [ "Comedy", "Sci-Fi" ] } })

{ "name" : "Black Mirror", "genres": [ "Sci-Fi", "Thriller", "Comedy" ],
"runtime" : 60 }
```

… all values are in the array

Selection σ: Array element matches (with AND)

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Match one: `{ key: { $elemMatch : {σ1, …, σn } } }`

```
> db.series.find(
  { "series": { $elemMatch : { $gt: 1, $lt: 3 } } })

{ "name" : "Black Mirror", "series": [ 1, 2, 3 ], "runtime" : 60 }
```

… one element matches all criteria (with AND)

Selection σ: Array with exact size

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Size: `{ key: { $size : int } }`

```
> db.series.find( { "series": { $size : 3 } })

{ "name" : "Black Mirror", "series": [ 1, 2, 3 ], "runtime" : 60 }
```

… only possible for exact size of array (not ranges)

Selection σ: Type of value

Type: `{ key: { $type: typename } }`

"timestamp"

"decimal"

"string"

"double"

"binData"

"date"

"object"

"int"

"array"

"long"

"bool"

"undefined"

"objectId"

"null"

"regex"

"dbPointer"

"array"

"javascript"

"number"

…

Selection σ: Type of value

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Type: `{ key: { $type: typename } }`

```
> db.series.find({ "runtime": { $type : "number" } })

{ "name" : "Black Mirror", "genres": [ "Science-Fiction", "Thriller" ],
"runtime" : 60 }
```

Selection σ: Matching an array by type?

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

Type: `{ key: { $type: typename } }`

```
> db.series.find({ "genres": { $type : "array" } })
```

… empty

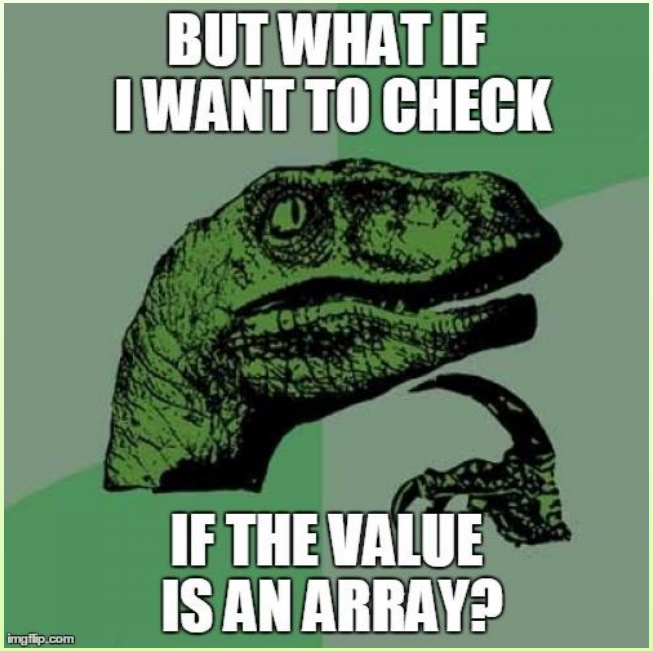
… passes if any value in the array has that type

Selection σ: Matching an array by type?

Arrays

When applied to arrays, `$type` matches any **inner** element that is of the specified BSON type. For example, when matching for `$type : 'array'`, the document will match if the field has a nested array. It will not return results where the field itself is an `array`.

https://docs.mongodb.com/manual/reference/operator/query/type/




```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [ "Science-Fiction", "Thriller" ],
    "runtime": 60
  })
```

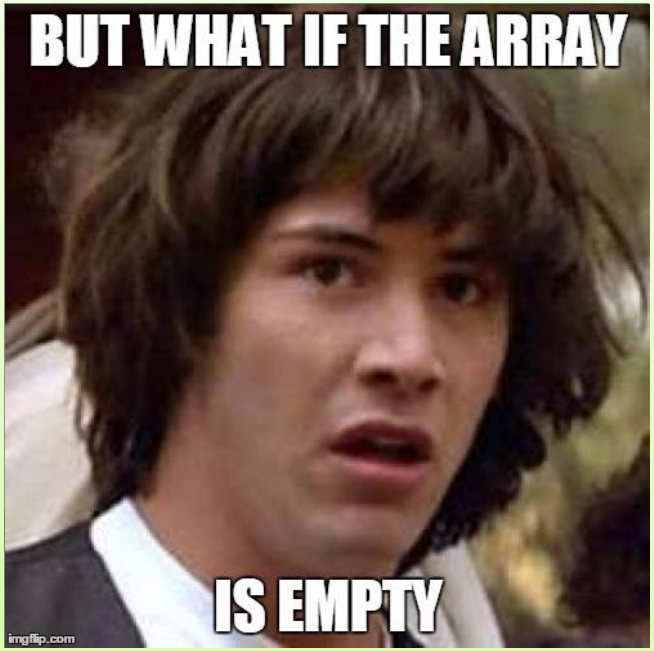
```
> db.series.find(
 { "genres": { $elemMatch: { $exists : true } } }
 )

{ "name" : "Black Mirror", "genres": [ "Science-Fiction", "Thriller"
], "runtime" : 60 }
```

Selection σ: Matching an array by type?

Arrays

When applied to arrays, `$type` matches any **inner** element that is of the specified BSON type. For example, when matching for `$type : 'array'`, the document will match if the field has a nested array. It will not return results where the field itself is an `array`.

https://docs.mongodb.com/manual/reference/operator/query/type/




```
> db.series.insert(
  {
    "name": "Black Mirror",
    "genres": [],
    "runtime": 60
  })
```

```
> db.series.find(
  { $or: [
   { "genres": { $elemMatch: { $exists: true } } },
   { "genres": [] }
  ] } )

{ "name" : "Black Mirror", "genres": [], "runtime" : 60 }
```


mongoDB
SEEMS LEGIT

Selection σ: Other operators

Mod: `{ key: { $mod [ div, rem ] } }`

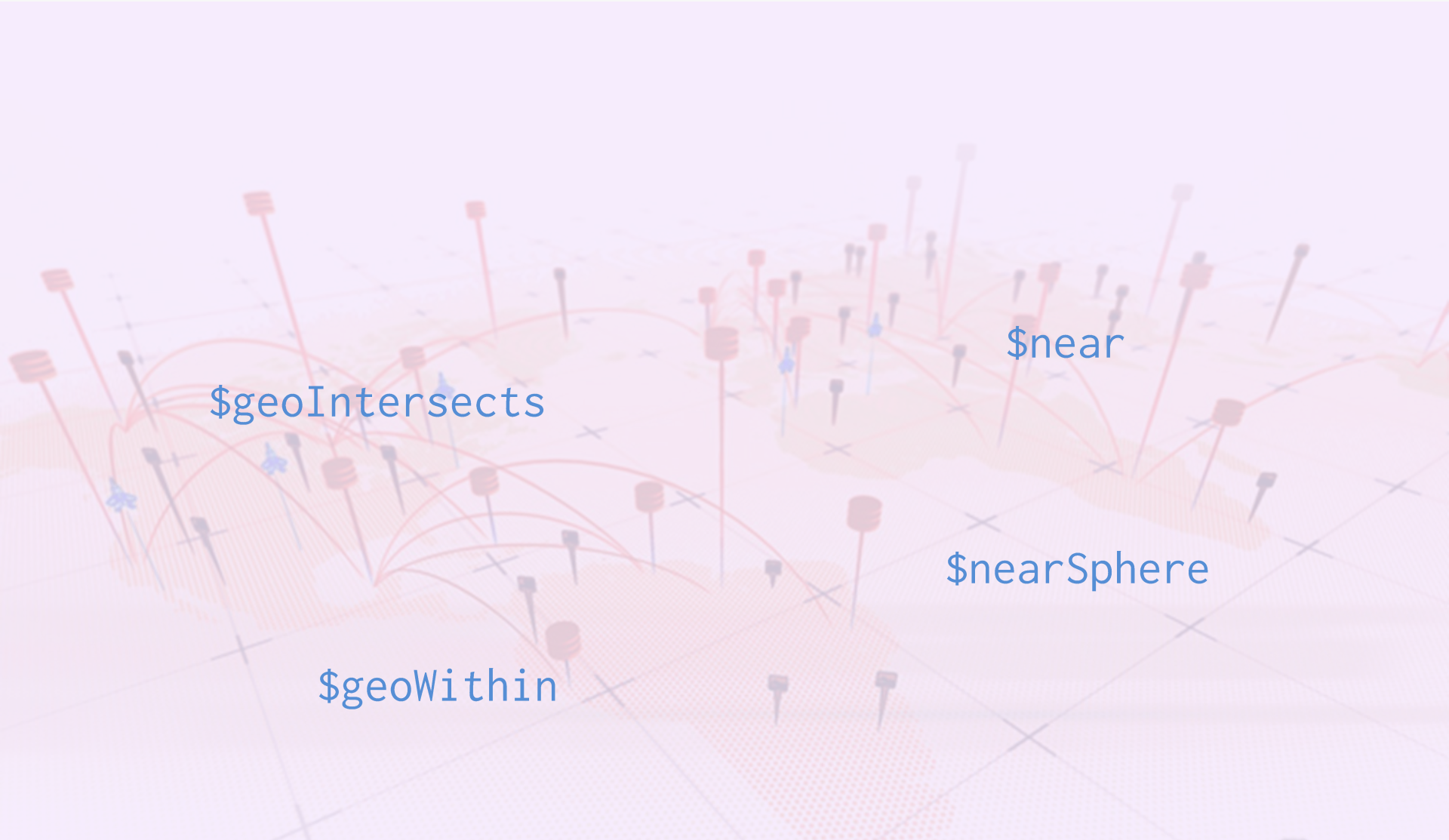
Regex: `{ key: { $regex: pattern } }`

Text search: `{ $text: { $search: terms } }`

Where (JS): `{ $where: javascript_code }`

… `where` is executed over all documents
(and should be avoided where possible)

Selection σ: Geographic features



https://docs.mongodb.com/manual/reference/operator/query-geospatial/

Selection σ: Bitwise features

$bitsAllClear

$bitsAllSet

$bitsAnyClear

$bitsAnySet

...

https://docs.mongodb.com/manual/reference/operator/query-bitwise/

MongoDB:

Projection of output values

Projection π: Choose output values

`key: 1`:	Output field(s) with `key`
`key: 0`:	Suppress field(s) with `key`
`array.$: 1`	Project first matching array element I
`$elemMatch`:	Project first matching array element II
`$slice`:	Output first or last `slice` array values

https://docs.mongodb.com/manual/reference/operator/projection/

Projection π: Output certain fields

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Project only certain fields: `{ k1: 1, ..., kn: 1 }`

```
> db.series.find(
  { "runtime": { $gt: 30 } },
  { "name": 1 , "runtime": 1, "network": 1 })

{ "_id" : ObjectId("5951e0b265ad257d48f4a7d5"), "name" : "Black Mirror",
"runtime" : 60 }
```

… outputs what is available; by default also outputs `_id` field

Projection π: Output embedded fields

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "rating": { "avg": 9.4 , "votes": 9001 },
    "runtime": 60
  })
```

Project (embedded) fields: `{ k1: 1, ..., kn: 1 }`

```
> db.series.find(
  { "runtime": { $gt: 30 } },
  { "name": 1 , "rating.avg": 1 })

{ "_id" : ObjectId("5951e0b265ad257d48f4a7d5"), "name" : "Black Mirror",
"rating" : { "avg": 9.4 } }
```

... field is still nested in output

Projection π: Output embedded fields in array

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "reviews": [
       { "user": "jack" , "score": 9.1 },
       { "user": "jill" , "score": 8.3 }
    ],
    "runtime": 60
  })
```

Project (embedded) fields: `{ k1: 1, ..., kn: 1 }`

```
> db.series.find(
  { "runtime": { $gt: 30 } },
  { "name": 1 , "reviews.score": 1 })

{ "_id" : ObjectId("5951e0b265ad257d48f4a7d5"), "name" : "Black Mirror",
"reviews" : [ { "score": 9.1 } , { "score": 8.3 } ] }
```

... projects from within the array.

Projection π: Suppress certain fields

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Return all but certain fields: `{ k1: 0, ..., kn: 0 }`

```
> db.series.find(
  { "runtime": { $gt: 30 } },
  { "series": 0 })

{ "_id" : ObjectId("5951e0b265ad257d48f4a7d5"), "name" : "Black Mirror",
"runtime" : 60 }
```

... cannot combine 0 and 1 except ...

Projection π: Suppress certain fields

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Suppress ID: `{ _id: 0, k1: 1, ..., kn: 1 }`

```
> db.series.find(
  { "runtime": { $gt: 30 } },
  { "_id": 0, "name": 1 , "series": 1 })

{ "name" : "Black Mirror", "series" : [ 1, 2, 3 ] }
```

... 0 suppresses _id when other fields are output

Projection π: Output first matching element

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Project first matching element: `array.$: 1`

```
> db.series.find(
  { "series": { $elemMatch: { $gt: 1 } } },
  { "series.$": 1 } )

{ _id: ..., "series": [ 2 ] }
```

Projection π: Output first matching element

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Project first matching element: $elemMatch

```
> db.series.find(
  { "series": { $elemMatch: { $gt: 1 } } },
  { "series": { $elemMatch: { $lt: 3 } } } )

{ "_id": ..., "series": [ 1 ] }
```

… allows to separate selection and projection criteria.

Projection π: Output first matching element

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Project first matching element: $elemMatch

```
> db.series.find(
  { "series": { $elemMatch: { $gt: 1 } } },
  { "series": { $elemMatch: { $gt: 3 } } } )

{ "_id": ... }
```

… drops array field entirely if no element is projected.

Projection π: Output first matching element

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "reviews": [
       { "user": "jack" , "score": 9.1 },
       { "user": "jill" , "score": 8.3 }
    ]
  })
```

Project first matching element: $elemMatch

```
> db.series.find(
  { "series": { $elemMatch: { $gt: 1 } } },
  { "reviews": { $elemMatch: { "score": { $gt: 8 } } } } )

{ "_id": ..., "reviews": [ { "user": "jack" , "score": 9.1 } ] }
```

... can match on array of documents.

Projection π: Output some matching elements

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Return first n elements: `$slice: n` (where n > 0)

```
> db.series.find(
  { "series": { $elemMatch: { $gt: 1 } } },
  { "series": { $slice: 2 } } )

{ "_id": ..., "name": "Black Mirror", "series": [ 1, 2 ], "runtime": 60 }
```

Projection π: Output some matching elements

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Return last n elements: `$slice: n` (where n < 0)

```
> db.series.find(
  { "series": { $elemMatch: { $gt: 1 } } },
  { "series": { $slice: -2 } } )

{ "_id": ..., "name": "Black Mirror", "series": [ 2, 3 ], "runtime": 60 }
```

Projection π: Output some matching elements

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

Skip n and return m: `$slice: [n,m]` (n, m > 0)

```
> db.series.find(
  { "series": { $elemMatch: { $gt: 1 } } },
  { "series": { $slice: [ 2, 1 ] } } )

{ "_id": ..., "name": "Black Mirror", "series": [ 3 ], "runtime": 60 }
```

Projection π: Output some matching elements

```
> db.series.insert(
  {
    "name": "Black Mirror",
    "series": [ 1, 2, 3 ],
    "runtime": 60
  })
```

From last n, return m: `$slice: [n,m]` (n < 0, m > 0)

```
> db.series.find(
  { "series": { $elemMatch: { $gt: 1 } } },
  { "series": { $slice: [ -2, 1 ] } } )

{ "_id": ..., "name": "Black Mirror", "series": [ 2 ], "runtime": 60 }
```

MongoDB:
Updates

Update **u**: Modify fields in documents

`$set`:	Set the value
`$unset`:	Remove the key and value
`$rename`:	Rename the field (change the key)
`$setOnInsert`:	You don't want to know
`$inc`:	Increment number by `inc`
`$mul`:	Multiply number by `mul`
`$min`:	Replace values less than `min` by `min`
`$max`:	Replace values greater than `max` by `max`
`$currentDate`:	Set to current date

https://docs.mongodb.com/manual/reference/operator/update-field/

Use update with query criteria and update criteria:

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "type": "Scripted",
    "language": "English",
  })

WriteResult({ "nInserted" : 1 })

> db.series.update(
    { "type": "Scripted" },
    { $set: { "type": "Fiction" } },
    { "multi": true }
  )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.series.find({ "name": "Black Mirror" })

{           "_id": ObjectId("5951e0b265ad257d48f4a7d5"),
            "name": "Black Mirror",
            "type": "Fiction",
            "language": "English"               }
```

Update **u**: Modify arrays in documents

`$addToSet`:	Adds value if not already present
`$pop`:	Deletes first or last value
`$push`:	Appends (an) item(s) to the array
`$pullAll`:	Removes values from a list
`$pull`:	Removes values that match a condition

(Sub-)operators used for pushing/adding values:

`$`:	Select first element matching query condition
`$each`:	Add or push multiple values
`$slice`:	After pushing, keep first or last `slice` values
`$sort`:	Sort the array after pushing
`$position`:	Push values to a specific array index

https://docs.mongodb.com/manual/reference/operator/update-array/

Update u: Modify arrays in documents

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "type": [ "Scripted", "Drama" ],
    "languages": [ "English", "Spanish" ],
  })

WriteResult({ "nInserted" : 1 })

> db.series.update(
    { "type": "Scripted" },
    { $pull: { "type": { $in: [ "Drama", "Sci-Fi" ] },
               "languages": "Spanish" } },
    { "multi": true } )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.series.find({ "name": "Black Mirror" })

{       "_id" : ObjectId("5951e0b265ad257d48f4a7d5"),
        "name" : "Black Mirror",
        "type" : [ "Scripted" ],
        "languages" : [ "English" ] }
```

Update u: Modify arrays in documents

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "ratings": [ 8, 11, 13, 9 ],
    "languages": [ "English", "Spanish" ],
  })

WriteResult({ "nInserted" : 1 })

> db.series.update(
    { "ratings": { $gt: 10 } },
    { $set: { "ratings.$": 10 } },
    { "multi": true } )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.series.find({ "name": "Black Mirror" })

{       "_id" : ObjectId("5951e0b265ad257d48f4a7d5"),
        "name" : "Black Mirror",
        "ratings" : [ 8, 10, 13, 9 ],
        "languages" : [ "English", "Spanish" ]  }
```

Update **u**: Modify arrays in documents

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "ratings": [ 8, 11, 13, 9 ],
    "languages": [ "English", "Spanish" ],
  })

WriteResult({ "nInserted" : 1 })

> db.series.update(
    { "ratings": { $gt: 10 } },
    { $push: { "ratings": { $each: [4, 9],
                            $sort: 1, $slice: 4 } } },
    { "multi": true } )

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.series.find({ "name": "Black Mirror" })

{       "_id" : ObjectId("5951e0b265ad257d48f4a7d5"),
        "name" : "Black Mirror",
        "ratings" : [ 4, 8, 9, 9 ],
        "languages" : [ "English", "Spanish" ]  }
```

MongoDB:

Aggregation and Pipelines

Aggregation: without grouping

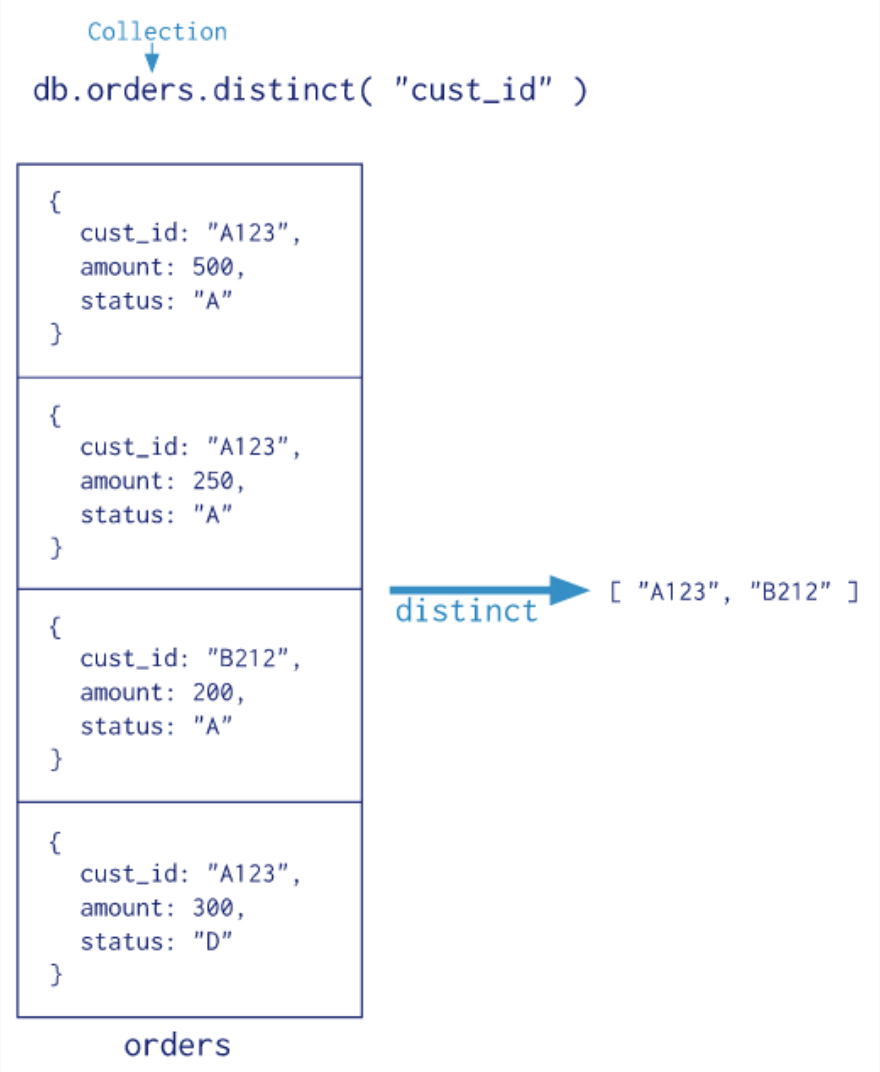
`db.coll.distinct(key)`: Array of unique values for that key

https://docs.mongodb.com/manual/reference/method/db.collection.distinct/

`db.coll.count()`: Count the documents

https://docs.mongodb.com/manual/reference/method/db.collection.count/

Aggregation: distinct



Pipelines: Transforming Collections

https://docs.mongodb.com/manual/reference/operator/aggregation/

Pipeline ρ: Stage Operators

`$match`: Filter by selection criteria σ

`$project`: Perform a projection π

`$group`: Group documents by a key/value

[used with *`$sum`*, *`$avg`*, *`$first`*, *`$last`*, *`$max`*, *`$min`*, etc.]

`$lookup`: Perform left-outer-join with another collection

`$unwind`: Copy each document for each array value

`$collStats`: Get statistics about collection

`$count`: Count the documents in the collection

`$sort`: Sort documents by a given key (`ASC|DESC`)

`$limit`: Return (up to) n first documents

`$sample`: Return (up to) n sampled documents

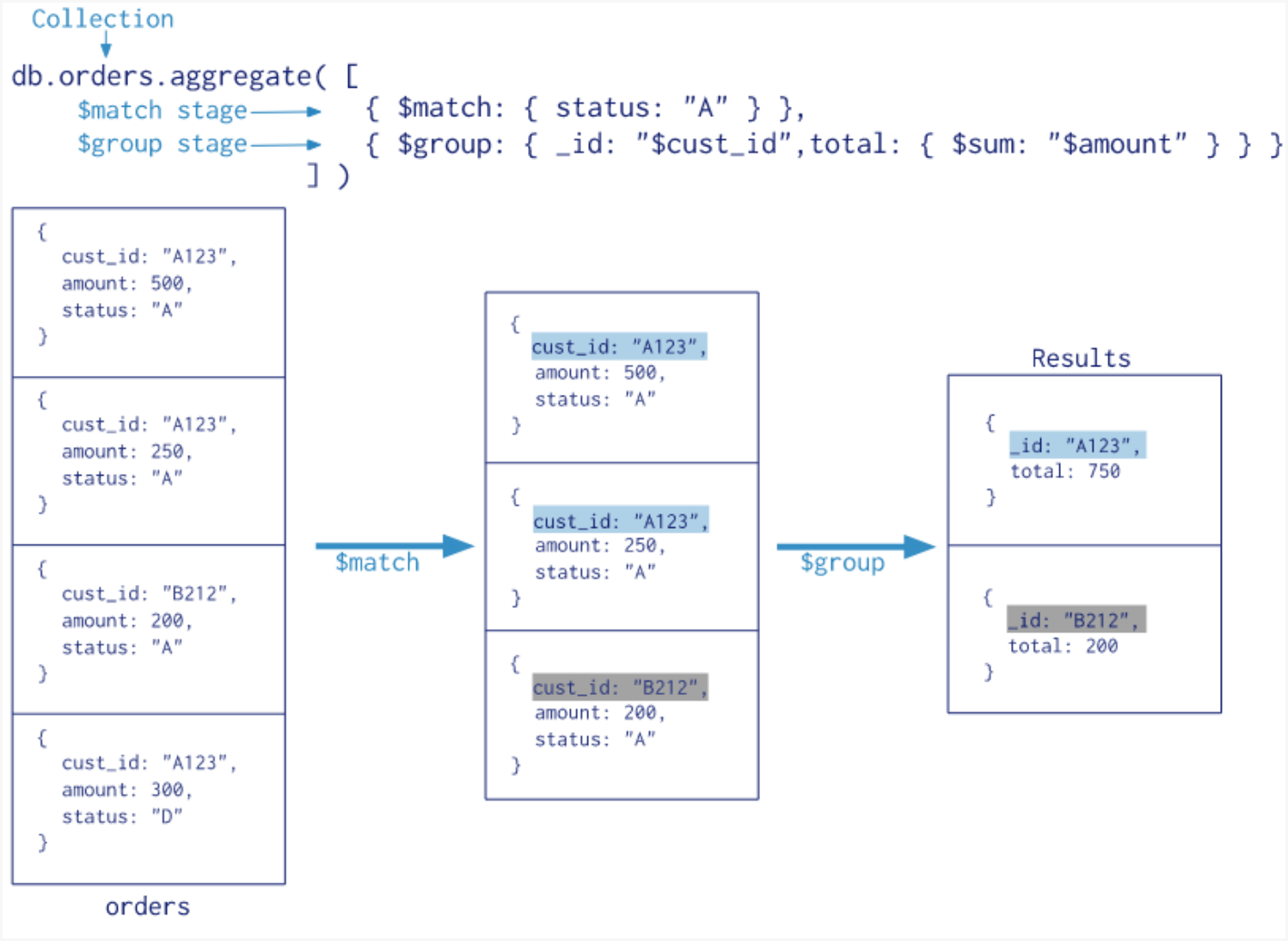
`$skip`: Skip n documents

`$out`: Save collection to MongoDB

(more besides)

https://docs.mongodb.com/manual/reference/operator/aggregation/

Pipeline Aggregation: $match and $group



Pipeline Aggregation: $lookup

```
{
  "name": "The Wire",
  "country": "US"
}
{
  "name": "Black Mirror",
  "country": "UK"
}
```

```
{
  "nation": "US",
  "network": "HBO"
}
{
  "nation": "US",
  "network": "FOX"
}
```

```
> db.series.aggregate( [
  { $lookup: { from: "networks", localField: "country",
     foreignField: "nation", as: "possibleNetworks" }
  } ] )
```

```
{
  "name": "The Wire",
  "country": "US",
  "possibleNetworks": [
    { "nation": "US", "network": "HBO" } ,
    { "nation": "US", "network": "FOX" }
  ]
}
{
  "name": "Black Mirror",
  "country": "UK"
  "possibleNetworks": []
}
```

Pipeline Aggregation: $unwind

```
{
  "name": "The Wire",
  "genres": [ "Drama", "Crime", "Thriller" ]
}
```

```
> db.series.aggregate( [ { $unwind : "$genres" } ] )
```

```
{
  "name": "The Wire",
  "genres": "Drama"
}
{
  "name": "The Wire",
  "genres": "Crime"
}
{
  "name": "The Wire",
  "genres": "Thriller"
}
```

MongoDB:

Indexing

MongoDB Indexing

Per collection:

- `_id` Index
- Single-field Index (sorted)
- Compound Index (sorted)
- Multikey Index (for arrays)
- Geospatial Index
- Full-text Index
- Hash-based indexing (hashed)

MongoDB: Text Indexing Example

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "summary": "A British sci-fi series with a dystopian setting",
    "languages": [ "English", "Spanish" ],
  })

WriteResult({ "nInserted" : 1 })

> db.series.createIndex( { summary: "text"})

> db.series.getIndexes()

[       {               "v" : 2,
                        "key" : { "_id" : 1 },
                        "name" : "_id_",
                        "ns" : "test.series"            },
        {               "v" : 2,
                        "key" : { "fts" : "text", "_ftsx" : 1 },
                        "name" : "summary_text",
                        "ns" : "test.series",
                        "weights" : { "summary" : 1 },
                        "default_language" : "english",
                        "language_override" : "language",
                        "textIndexVersion" : 3          }
]
```

MongoDB: Text Indexing Example

```
> db.series.insert(
  {
    _id: ObjectId("5951e0b265ad257d48f4a7d5"),
    "name": "Black Mirror",
    "summary": "A British sci-fi series with a dystopian setting",
    "languages": [ "English", "Spanish" ],
  })

WriteResult({ "nInserted" : 1 })

> db.series.createIndex( { summary: "text"})

> db.series.find(
     { $text: { $search: "dystopia sci-fi" } } )

{       "_id" : ObjectId("5951e0b265ad257d48f4a7d5"),
        "name" : "Black Mirror",
        "summary" : "A British sci-fi series with a dystopian setting",
        "languages" : [
                "English",
                "Spanish"
        ]               }
```

MongoDB:
Distribution

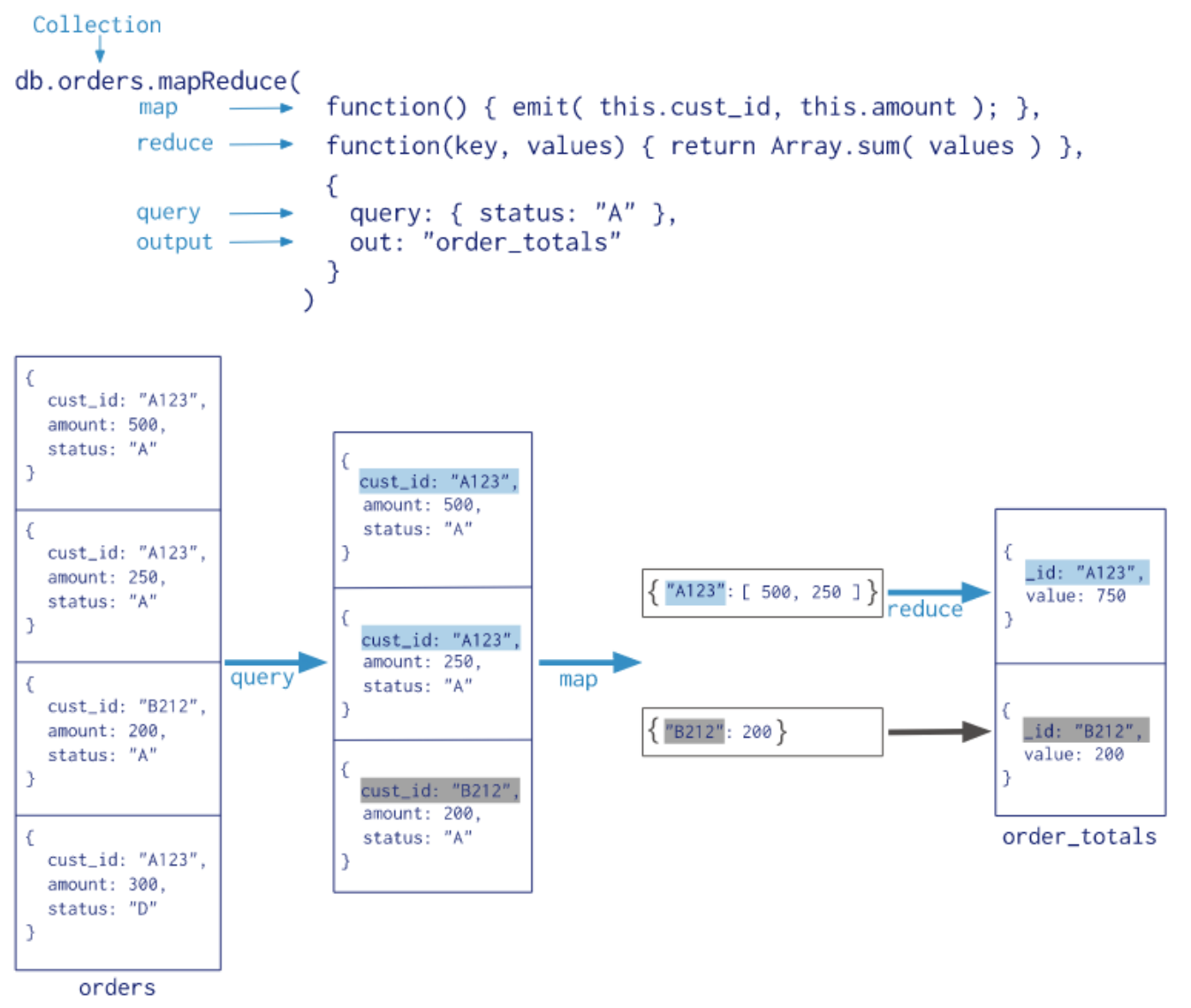
MongoDB: Distribution

- "Sharding":
  - Hash-based or Horizontal Ranged

    (Depends on indexes)

- Replication
  - Replica sets

mapreduce


Collection
db.orders.mapReduce(
map
function() { emit(this.cust_id, this.amount); },
reduce
function(key, values) { return Array.sum(values) },
{
query
query: { status: "A" },
output
out: "order_totals"
}
)
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
{ cust_id: "A123", amount: 300, status: "D" }
orders
query
{ cust_id: "A123", amount: 500, status: "A" }
{ cust_id: "A123", amount: 250, status: "A" }
{ cust_id: "B212", amount: 200, status: "A" }
map
{ "A123": [500, 250] }
{ "B212": 200 }
reduce
{ _id: "A123", value: 750 }
{ _id: "B212", value: 200 }
order_totals

MongoDB:

Why is it so popular?

330 systems in ranking, June 2017

Rank Jun 2017	Rank May 2017	Rank Jun 2016	DBMS	Database Model	Score Jun 2017	Score May 2017	Score Jun 2016
1.	1.	1.	Oracle	Relational DBMS	1351.76	-2.55	-97.49
2.	2.	2.	MySQL	Relational DBMS	1345.31	+5.28	-24.83
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1198.97	-14.84	+33.16
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	368.54	+2.63	+61.94
5.	5.	↓ 4.	MongoDB	Document store	335.00	+3.42	+20.38
6.	6.	6.	DB2	Relational DBMS	187.50	-1.34	-1.07
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	126.55	-3.33	+0.32
8.	8.	↓ 7.	Cassandra	Wide column store	124.12	+1.01	-7.00
9.	9.	↑ 10.	Redis	Key-value store	118.89	+1.44	+14.39
10.	10.	↓ 9.	SQLite	Relational DBMS	116.71	+0.64	+9.92
11.	11.	11.	Elasticsearch	Search engine	111.56	+2.74	+24.14
12.	12.	12.	Teradata	Relational DBMS	77.33	+1.00	+3.49
13.	13.	13.	SAP Adaptive Server	Relational DBMS	67.52	-0.23	-4.16
14.	14.	14.	Solr	Search engine	63.61	-0.16	-0.46
15.	15.	15.	HBase	Wide column store	61.87	+2.37	+8.88
16.	16.	↑ 18.	Splunk	Search engine	57.52	+0.82	+12.29
17.	17.	↓ 16.	FileMaker	Relational DBMS	57.08	+0.60	+8.39
18.	18.	↑ 20.	MariaDB	Relational DBMS	52.89	+1.91	+18.24
19.	19.	19.	SAP HANA	Relational DBMS	47.50	-1.55	+6.23
20.	20.	↓ 17.	Hive	Relational DBMS	44.38	+0.91	-2.62
21.	21.	21.	Neo4j	Graph DBMS	37.87	+1.73	+3.84
22.	22.	↑ 25.	Amazon DynamoDB	Document store	34.02	+0.82	+9.68
23.	23.	↑ 24.	Couchbase	Document store	31.92	-0.34	+6.61
24.	24.	↓ 23.	Memcached	Key-value store	28.74	-0.67	+1.32
25.	25.	↓ 22.	Informix	Relational DBMS	27.84	-0.40	-1.21
26.	26.	26.	CouchDB	Document store	22.15	-0.25	+0.44
27.	27.	27.	Microsoft Azure SQL Database	Relational DBMS	21.33	-0.22	+1.78
28.	28.	↑ 29.	Vertica	Relational DBMS	20.91	+0.23	+1.57
29.	29.	↓ 28.	Netezza	Relational DBMS	19.66	-0.13	+0.16

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

Questions?