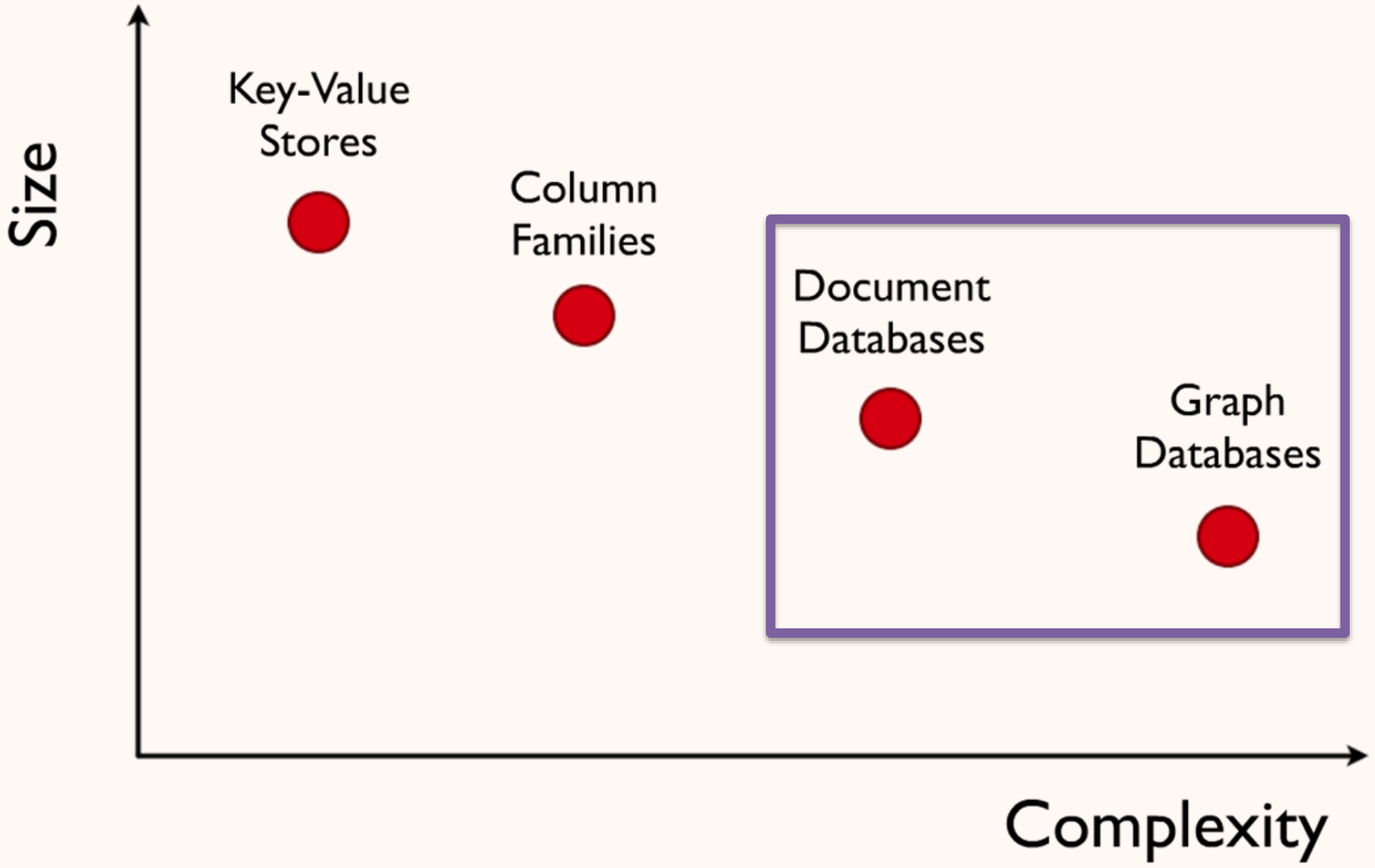
CC5212-1
Procesamiento Masivo de Datos
Otoño 2017

Lecture 10: NoSQL II

Aidan Hogan
aidhog@gmail.com

NoSQL


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

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

DOCUMENT STORES

Key–Value: a Distributed Map

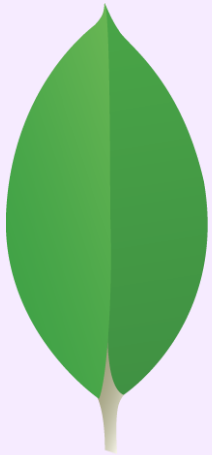
Countries	
Primary Key	**Value**
Afghanistan	capital:Kabul,continent:Asia,pop:31108077#2011
…	…

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

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 all 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 }
```

… 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
  })
```

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

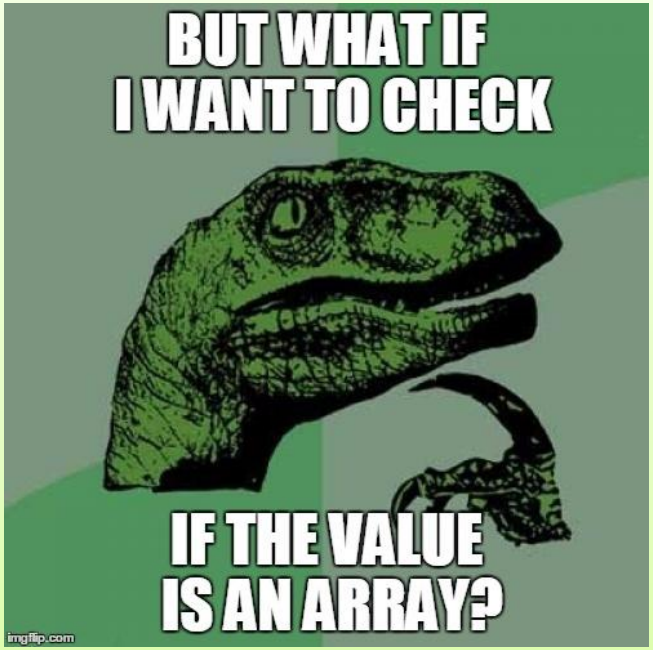
… passes if any value in the array has that type

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
  })
```

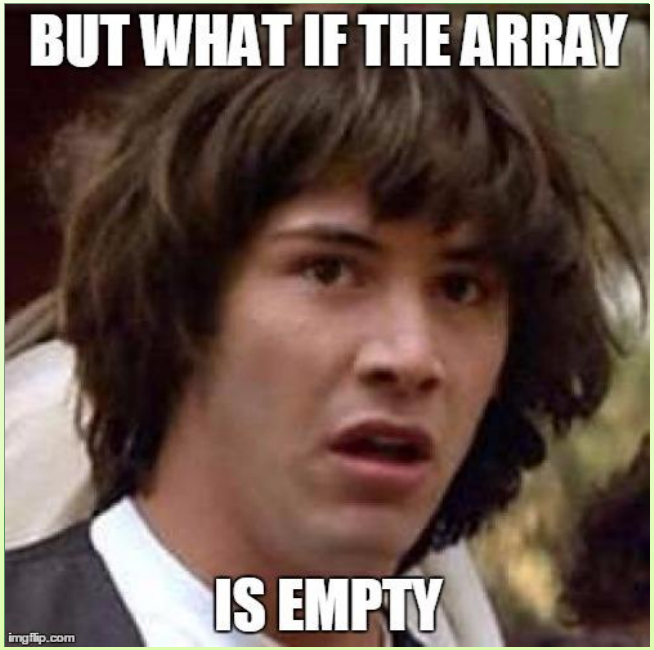
```
> db.series.find(
 { "genres": { $elemMatch: { $exists : true } } }
 )

{ "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 ] } }`

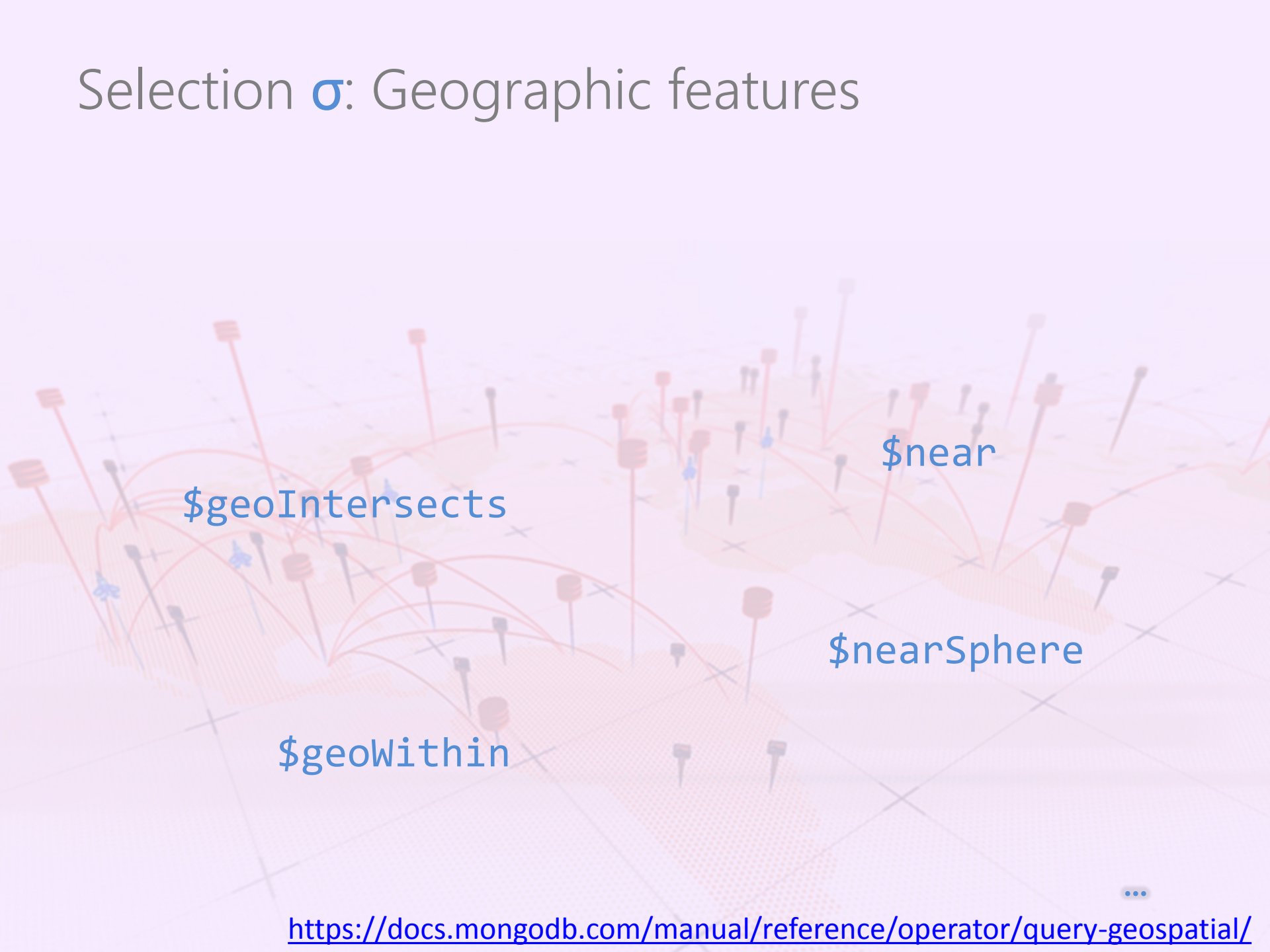
Regex: `{ key: { $regex: pattern } }`

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 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 first matching element

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

Select first 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 first matching element

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

Select last 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 first matching element

```
> 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 first matching element

```
> 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

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 })
```

… overwrites `"Scripted"` with `"Fiction"`
in all (`"multi": true`) matching documents

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/

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:

`$`: Select first element matching condition
`$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/

MONGODB:
AGGREGATION AND PIPELINES

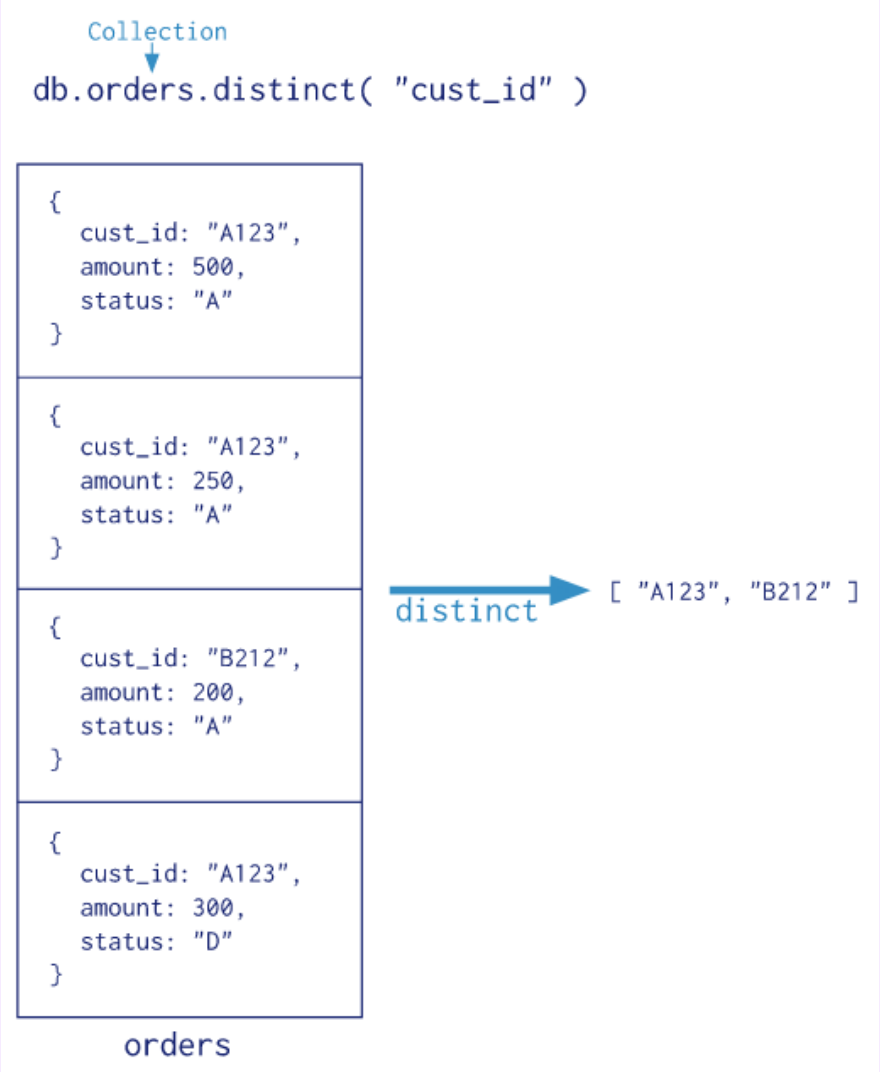
Aggregation: without grouping

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

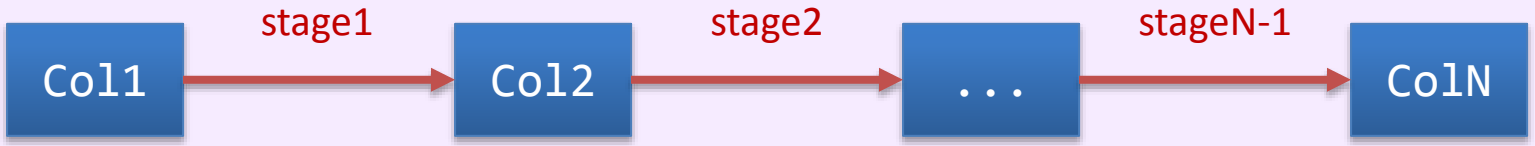
`db.coll.count()`: Count the documents

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

Aggregation: `distinct`



Pipelines: Transforming Collections



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

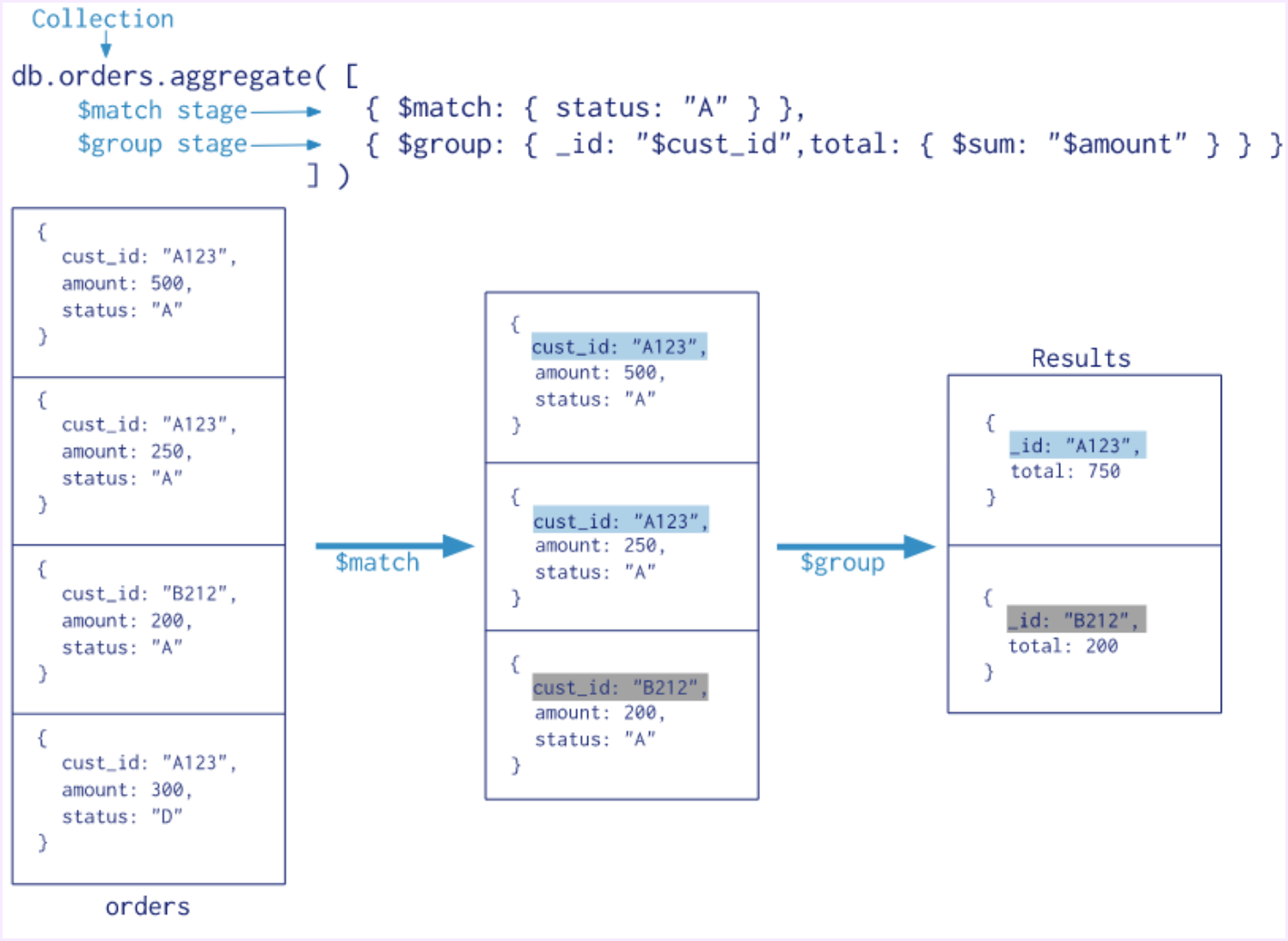
Pipeline ρ: Stage Operators

`$match`:	Filter by selection criteria σ
`$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 coll.
`$unwind`:	Copy each document for each array value
`$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)

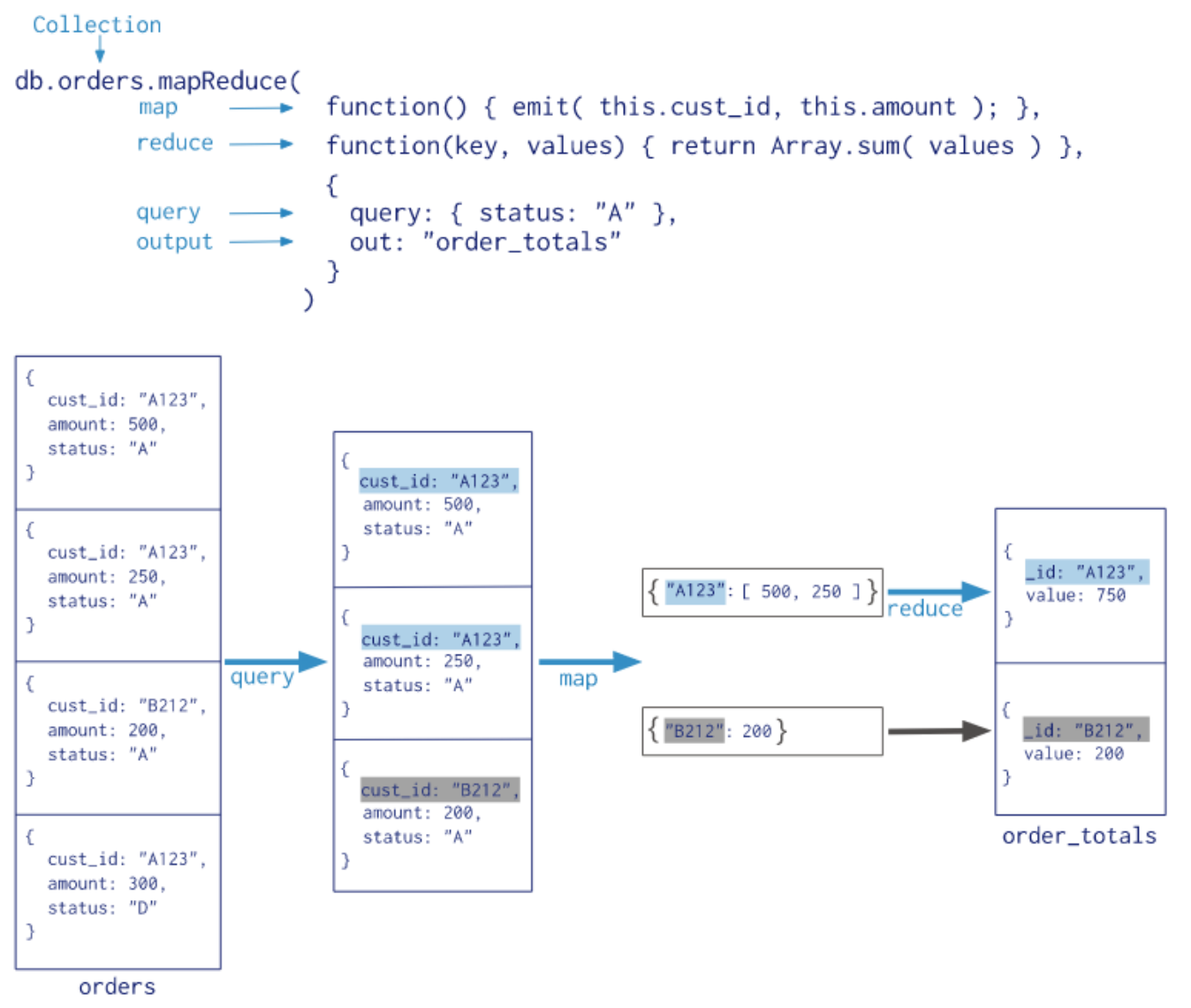
MONGODB:
DISTRIBUTION

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?

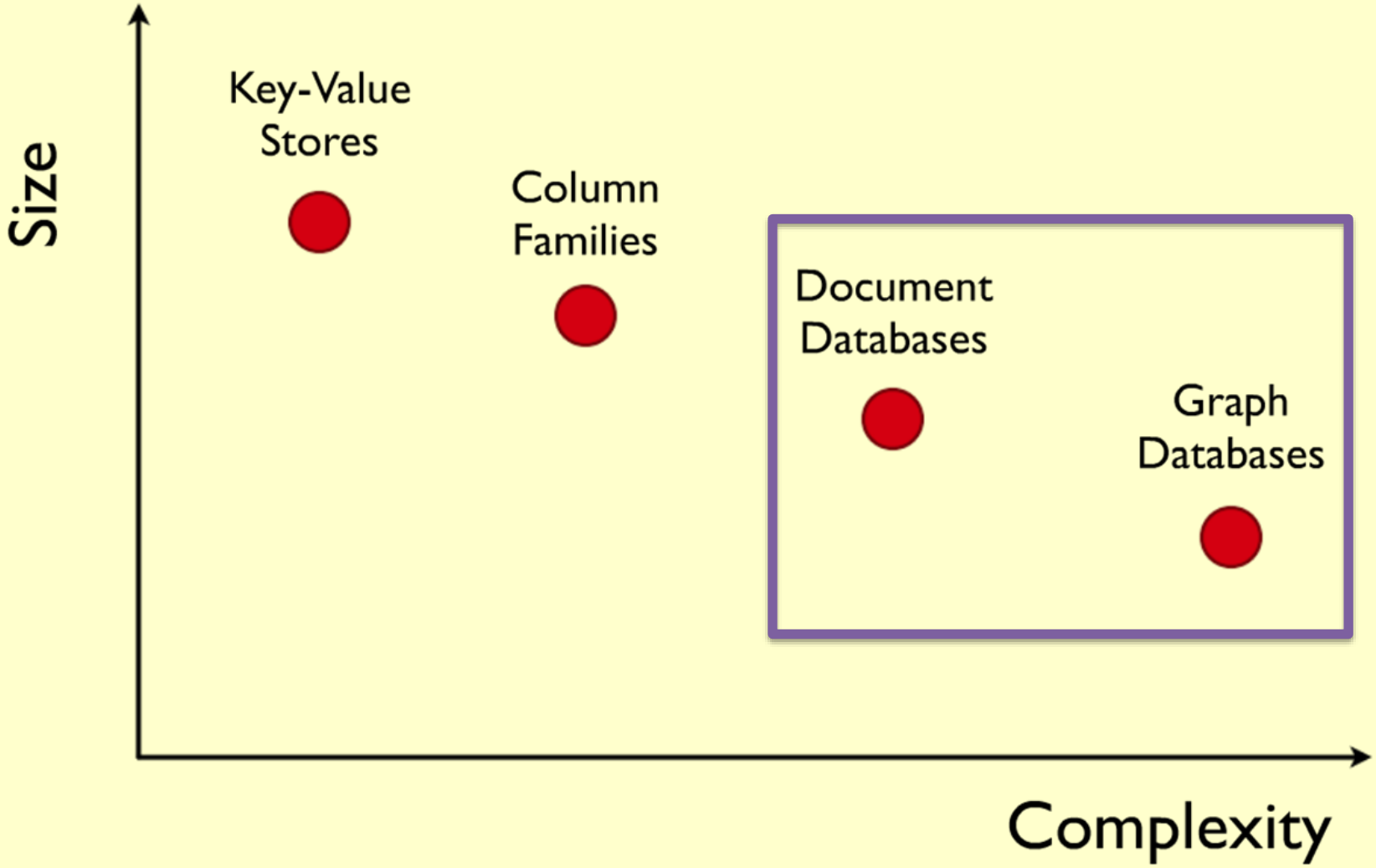
330 systems in ranking, June 2017

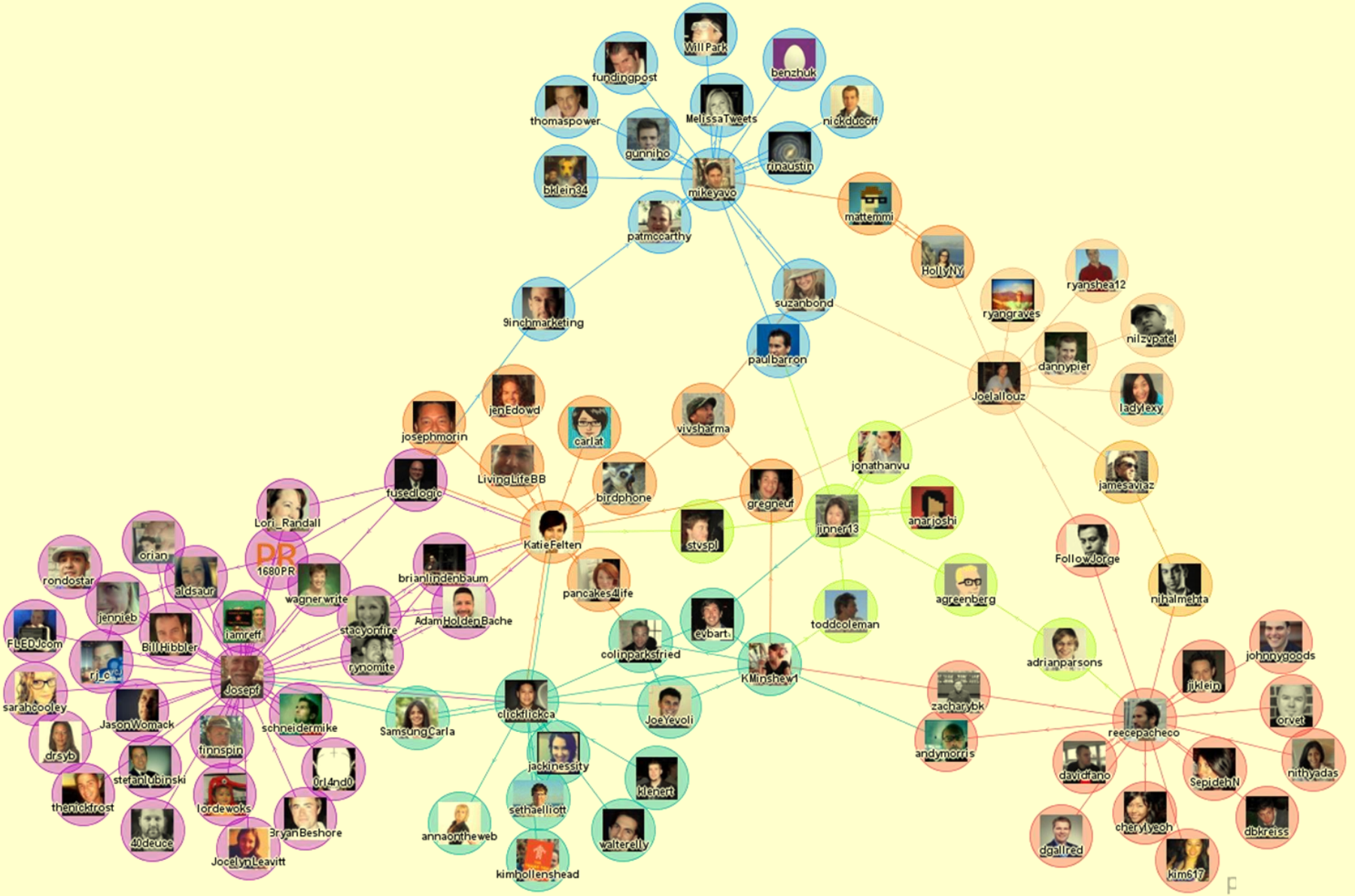
Rank Jun 2017	Rank May 2017	Rank Jun 2016	DBMS	Database Model	Score Jun 2017	Score May 2017	Score Jun 2016
1.	1.	1.	Oracle	Relational DBMS	1351.76	-2.55	-97.49
2.	2.	2.	MySQL	Relational DBMS	1345.31	+5.28	-24.83
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1198.97	-14.84	+33.16
4.	4.	↑ 5.	PostgreSQL	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

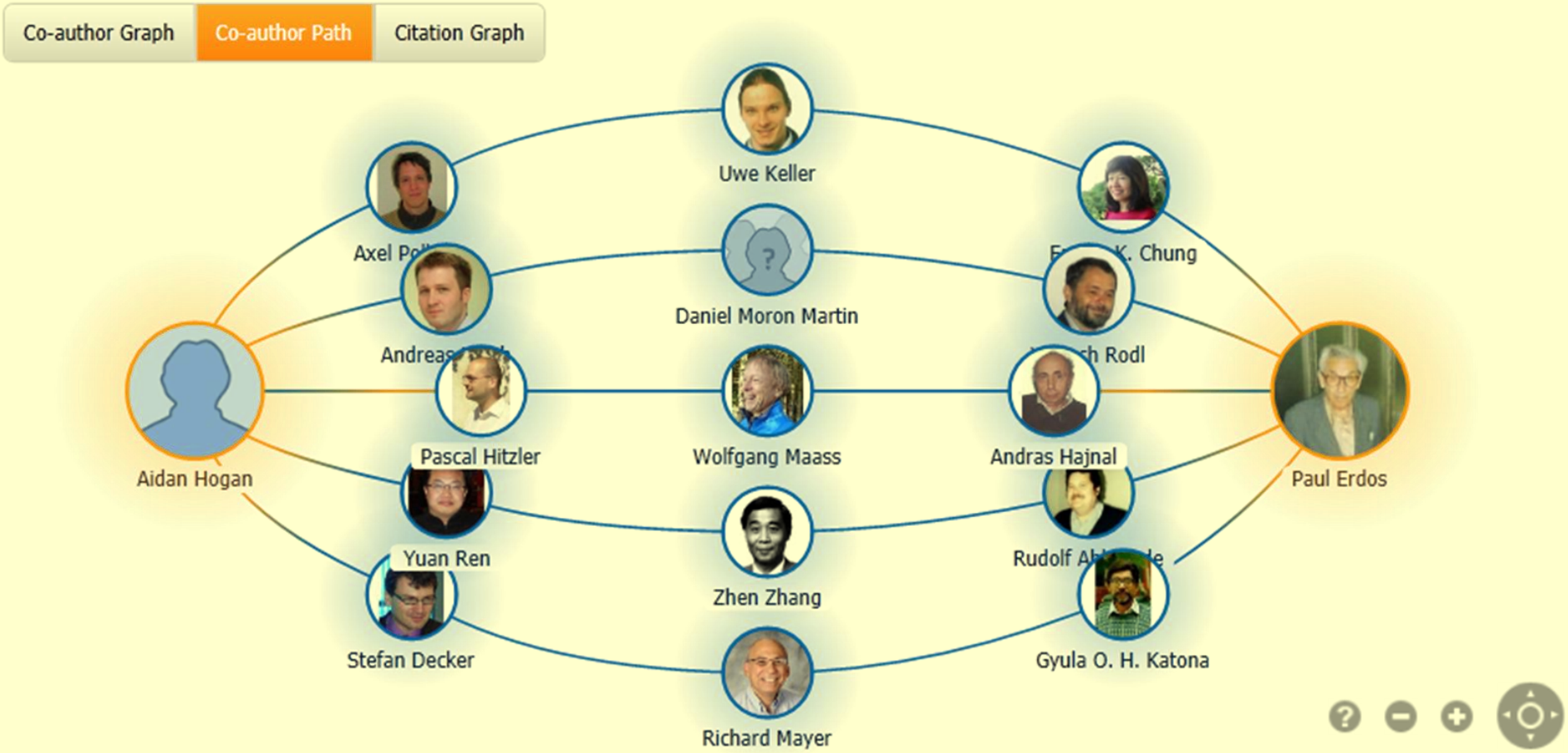
http://db-engines.com/en/ranking

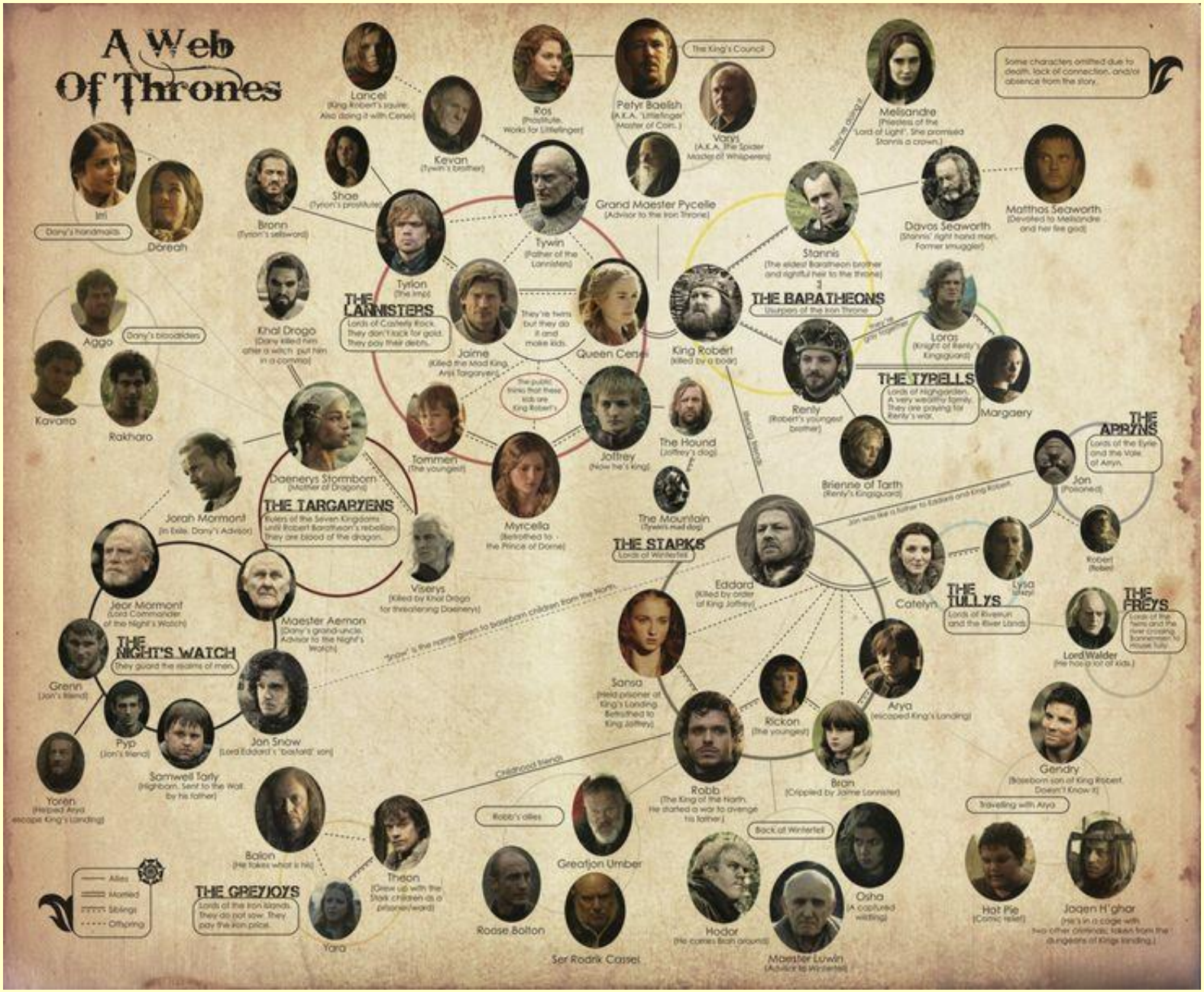
GRAPH STORES

NoSQL

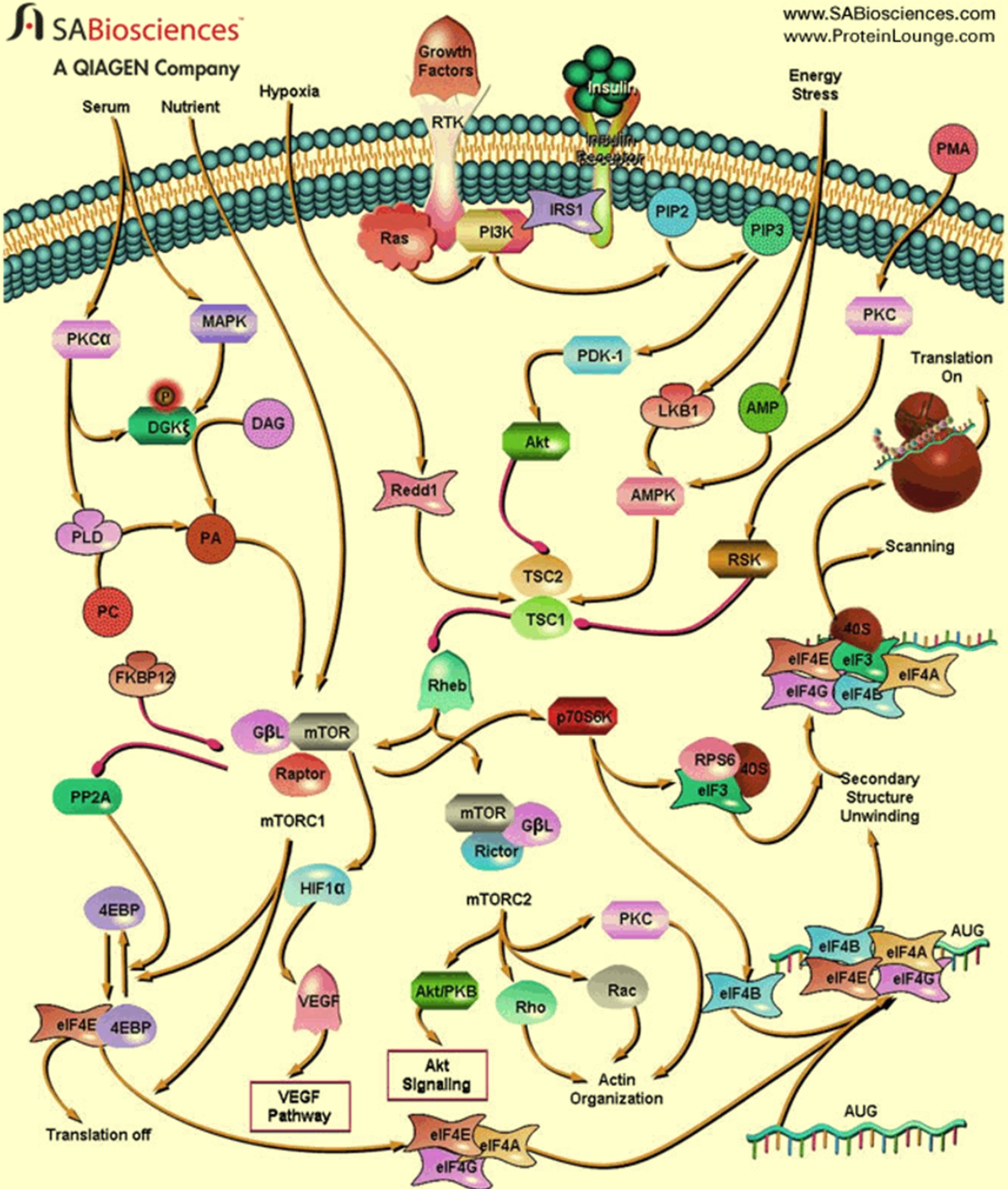

Size
Key-Value Stores
Column Families
Document Databases
Graph Databases
Complexity


WillPark
fundingpost
benzhuk
thomaspower
MelissaTweets
nickducoff
gunniho
rinaustin
bklein34
mikeyavo
mattemmi
patmccarthy
HollyNY
ryanshea12
suzanbond
ryangraves
9inchmarketing
nilzvpatel
paulbarron
dannypier
Joelallouz
ladylexy
jenEdowd
josephmorin
carlat
vivsharma
jonathanvu
jamesaviaz
LivingLifeBB
birdphone
fusedlogic
gregneuf
jinner13
anarjoshi
Lori_Randall
KatieFelten
stvspl
FollowJorge
orian
PR
1680PR
rondostar
aldsaur
brianlindenbaum
pancakes4life
agreenberg
nihalmehta
wagnerwrite
jennieb
AdamHoldenBache
toddcoleman
FLEDJcom
BillHibbler
iamreff
stacyonfire
evbart
colinparksfried
adrianparsons
johnnygoods
rj_c
rynomite
KMinshew1
jklein
Josepf
sarahcooley
zacharybk
orvet
JasonWomack
clickfilickca
JoeYevoli
schneidermike
SamsungCarla
reecepacheco
drsyb
finnspin
andymorris
jackinessity
nithyadas
stefanlubinski
klenert
davidfano
SepidehN
0rl4nd0
thenickfrost
sethaelliott
cherlyeoh
dbkreiss
lordewoks
annaontheweb
dgallred
3ryanBeshore
walterelly
kim617
40deuce
kimhollenshead
JocelynLeavitt


Co-author Graph
Co-author Path
Citation Graph
Uwe Keller
Axel Pol
K. Chung
Daniel Moron Martin
Andreas
ch Rodl
Aidan Hogan
Pascal Hitzler
Wolfgang Maass
Andras Hajnal
Paul Erdos
Yuan Ren
Rudolf Ah
Zhen Zhang
Stefan Decker
Gyula O. H. Katona
Richard Mayer


A Web Of Thrones
Some characters omitted due to death, lack of connection, and/or absence from the story.
Lancel
(King Robert's squire. Also doing it with Cersei)
Kevan
(Tywin's brother)
Ros
(Prostitute. Works for Littlefinger)
Petyr Baelish
(A.K.A. "Littlefinger" Master of Coin...)
The King's Council
Varys
(A.K.A. The Spider Master of Whisperers)
Melisandre
(Priestess of the "Lord of Light". She promised Stannis a crown.)
Shae
(Tyrion's prostitute)
Grand Maester Pycelle
(Advisor to the Iron Throne)
Tywin
(Father of the Lannisters)
Stannis
(The eldest Baratheon brother and rightful heir to the throne)
Davos Seaworth
(Stannis' right hand man. Former smuggler)
Matthos Seaworth
(Devoted to Melisandre and her fire god)
Iri
(Dany's handmaids)
Doreah
Bronn
(Tyrion's sellsword)
Tyrion
(The imp)
THE LANNISTERS
Lords of Casterly Rock. They don't look for gold. They pay their debts.
They're twins but they do it and make kids.
Jaime
(Killed the Mad King, Aryi Targaryen)
Queen Cersei
King Robert
(killed by a boar)
THE BARATHEONS
Usurpers of the Iron Throne
Loras
(Knight of Renly's Kingsguard)
Renly
(Robert's youngest brother)
THE TYRELLS
Lords of Highgarden. A very wealthy family. They are paying for Renly's war.
Margaery
Aggo
(Dany's bloodriders)
Kovarro
Rakharo
Khal Drogo
(Dany killed him after a witch put him in a coma)
The public thinks that these kids are King Robert's
Tommen
(The youngest)
Joffrey
(Now he's king)
The Hound
(Joffrey's dog)
The Mountain
(Tyrion's mad dog)
Daenerys Stormborn
(Mother of Dragons)
THE TARGARYENS
Rulers of the Seven Kingdoms until Robert Baratheon's rebellion. They are blood of the dragon.
Myrcella
(Betrothed to the Prince of Dorne)
Brienne of Tarth
(Renly's Kingsguard)
Jon
(Poisoned)
THE ARRYNS
Lords of the Eyrie and the Vale of Arryn.
Robert
(Robert)
Jorah Mormont
(In Exile. Dany's Advisor)
Viserys
(Killed by Khal Drogo for threatening Daenerys)
Jeor Mormont
(Lord Commander of the Night's Watch)
Maester Aemon
(Dany's grand-uncle. Advisor to the Night's Watch)
THE NIGHT'S WATCH
They guard the realms of men.
THE STARKS
Lords of Winterfell
Eddard
(Killed by order of King Joffrey)
Catelyn
Lysa
(crazy)
THE TULLYS
Lords of Riverrun and the River Lands
THE FREYS
Lords of the twins and the river crossing. Bannermen to House Tully.
Lord Walder
(He has a lot of kids.)
"Snow" is the name given to bastard children from the North.
Grenn
(Jon's friend)
Pyp
(Jon's friend)
Samwell Tarly
(Highborn. Sent to the Wall by his father)
Jon Snow
(Lord Eddard's "bastard" son)
Sansa
(Held prisoner at King's Landing. Betrothed to King Joffrey)
Rickon
(The youngest)
Arya
(escaped King's Landing)
Yoren
(Helped Arya escape King's Landing)
Robb
(The King of the North. He started a war to avenge his father.)
Bran
(Crippled by Jaime Lannister)
Gendry
(Baseborn son of King Robert. Doesn't know it)
Childhood friends
Traveling with Arya
Robb's allies
Back at Winterfell
Balon
(He takes what is his)
Theon
(Grew up with the Stark children as a prisoner/ward)
Greatjon Umber
Osha
(A captured wildling)
Hot Pie
(Comic relief)
Jaqen H'ghar
(He's in a cage with two other criminals, taken from the dungeons of Kings Landing.)
Roose Bolton
Ser Rodrik Cassel
Hodor
(He comes from around)
Maester Luwin
(Advisor to Winterfell)
Yara
THE GREYJOYS
Lords of the Iron Islands. They do not sow. They pay the iron price.
Allies
Married
Siblings
Offspring


N
DISTANCIAS GEOGRÁFICAS DISTORSIONADAS
CIUDAD EMPRESARIAL
Aeropuerto Internacional AMB
Av. Américo Vespucio
2
VESPUCIO NORTE
Zapadores
Dorsal
Einstein
Cementerios
Cerro Blanco
Patronato
Puente Cal y Canto
Av. Zapadores
Av. Recoleta
Av. El Salto
Ruta 5 Norte
Av. Independencia
Costanera Norte
Río Mapocho
Carrascal
Mapocho
Av. Perú
CERRO SAN CRISTÓBAL
Av. Santa María
Costanera Norte
Av. Marqueshue Norte
Av. Vitacura
Av. Kennedy
Padre Hurtado
Av. Teniente Cruz
1
SAN PABLO
Av. San Pablo
Lo Prado
Blanqueado
Gruta de Lourdes
Quinta Normal
Cumming
SANTA ANA
Catedral
Plaza de Armas
Bellas Artes
BAQUEDANO
Salvador
M. Montt
P. de Valdivia
Los Leones
El Golf
Alcántara
Escuela Militar
Manquehue
H. de Magallanes
Pudahuel
Barrancas
Ruta 68
Neptuno
Alameda
LOS HÉROES
CERRO SANTA LUCÍA
Av. Providencia
TOBALABA
Av. Apoquindo
LOS DOMINICOS
C. Colón
Francisco Bilbao
Príncipe de Gales
Simón Bolívar
Plaza Egaña
Los Orientales
Grecia
Los Presidentes
Quilín
Las Torres
Macul
Laguna Sur
Las Parcelas
Monte Tabor
Pajaritos
Las Rejas
Ecuador
San Alberto Hurtado
U. de Santiago
Estación Central
U.L.A.
República
La Moneda
U. de Chile
Santa Lucía
U. Católica
Toesca
Parque O`Higgins
Rondizzoni
P. Bustamante
Santa Isabel
Irarrázaval
Ñuble
Av. Francisco Bilbao
Av. Matta
Av. Las Rejas
Av. Gral. Velásquez
Av. Grecia
Av. José Pedro Alessandri
Av. Tobalaba
Autopista del Sol
Del Sol
Franklin
El Llano
San Miguel
Lo Vial
Departamental
Ciudad del Niño
Lo Ovalle
El Parrón
Gran Avenida
Av. Santa Rosa
Rodrigo de Araya
Carlos Valdovinos
Camino Agrícola
San Joaquín
Pedrero
Mirador
B. de La Florida
Av. Vicuña Mackenna
Av. Departamental
VICUÑA MACKENNA
4A
Av. Pajaritos
Santiago Bueras
Av. 5 de Abril
Cerrillos
5
PLAZA DE MAIPÚ
VICENTE VALDÉS
5
Santa Julia
La Granja
Rojas Magallanes
Trinidad
San José de la Estrella
Los Quillayes
Elisa Correa
Hospital Sótero del Río
Protectora de la Infancia
Las Mercedes
Av. Lo Ovalle
Av. Américo Vespucio
San Ramón
Santa Rosa
LA CISTERNA
2
4A
Av. Gabriela
Domingo Tocornal
PLAZA DE PUENTE ALTO
4
Av. Presidente Jorge Alessandri - Ruta 5 Sur
SIMBOLOGÍA - SYMBOLS
Líneas de Metro - Metro Lines
Transbordo Buses Interurbanos - Tranship Interurban Busses
Transbordo Tren - Train Tranship
Oficina de Atención al Cliente - Customer Service Office
BibloMetro
BiciMetro
Combinación Líneas de Metro - Metro Lines Combination
Recorrido Corto - L1 - Short Trip L1


SABiosciences
A QIAGEN Company
www.SABiosciences.com
www.ProteinLounge.com
Growth Factors
Insulin
Energy Stress
Serum
Nutrient
Hypoxia
RTK
Insulin Receptor
PMA
Ras
PI3K
IRS1
PIP2
PIP3
PKCα
MAPK
PKC
PDK-1
Translation On
LKB1
AMP
DGKξ
DAG
Akt
Redd1
AMPK
PLD
PA
RSK
Scanning
TSC2
TSC1
PC
40S
eIF4E
eIF3
eIF4A
eIF4G
eIF4B
FKBP12
Rheb
p70S6K
GβL
mTOR
Raptor
RPS6
PP2A
mTORC1
Secondary Structure Unwinding
mTOR
GβL
Rictor
HIF1α
4EBP
mTORC2
PKC
AUG
VEGF
Akt/PKB
Rho
Rac
Akt Signaling
Actin Organization
VEGF Pathway
Translation off


Google
Knowledge Graph

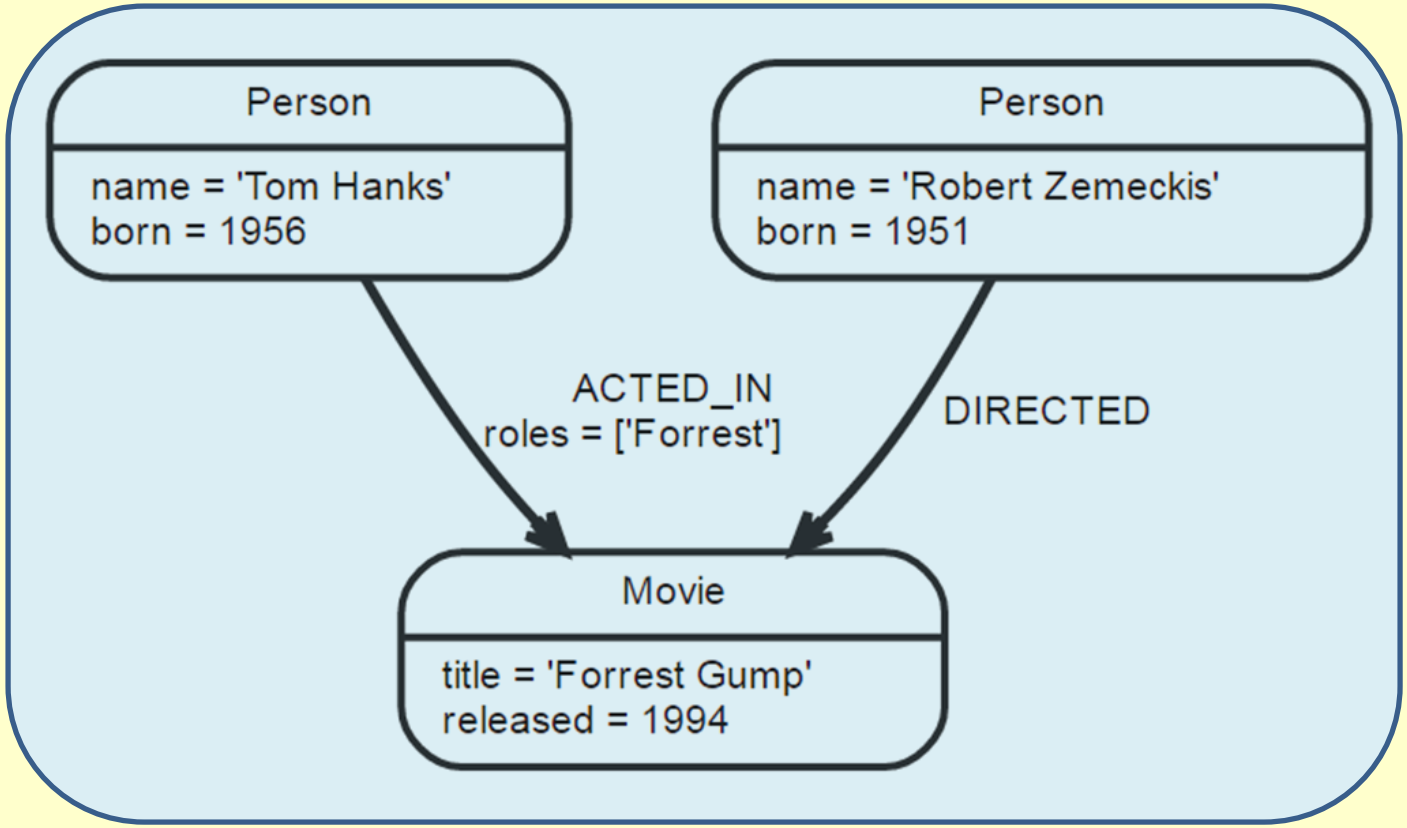
330 systems in ranking, June 2017

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

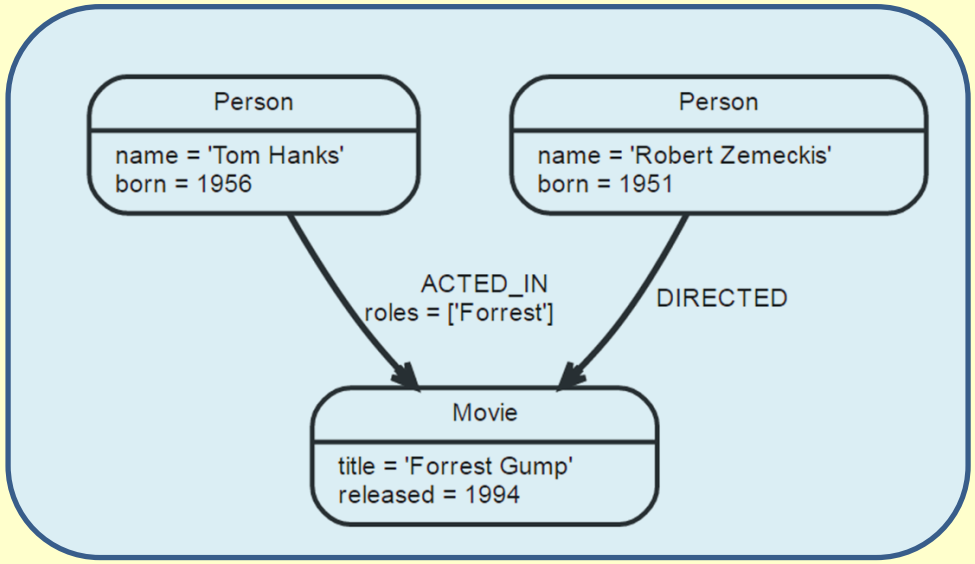
http://db-engines.com/en/ranking


neo4j

Neo4j Graph Data Model: “Property Graph”



Graph Query Language: Cypher

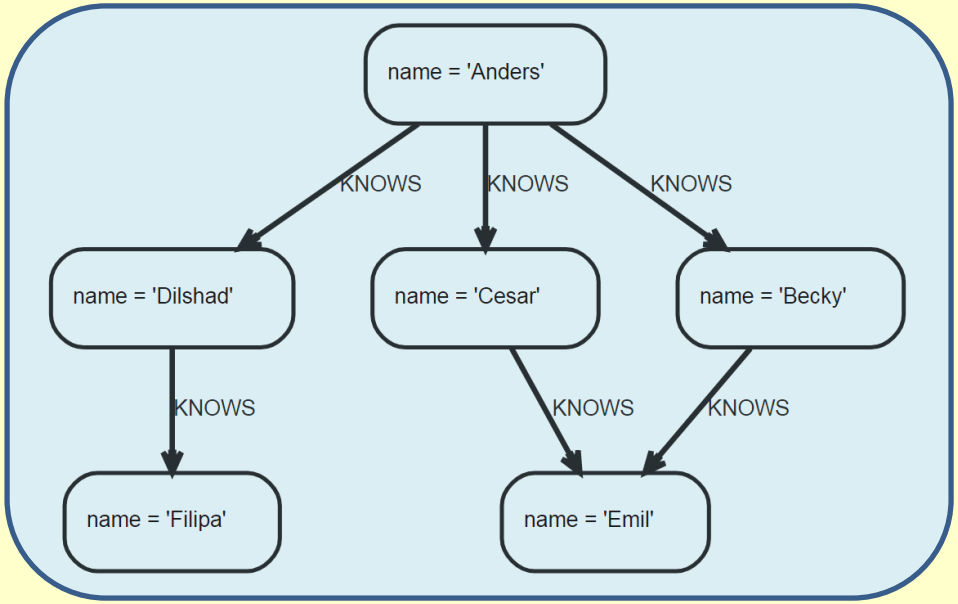



```
MATCH (p:Person { name:"Tom Hanks" })-[r:ACTED_IN]->(m:Movie)
RETURN m.title, r.roles
```

What will this query return?

m.title	r.roles
'Forrest Gump'	['Forrest']

Graph Query Language: Cypher




```
MATCH (me)-[:KNOWS*1..2]-(remote_friend)
WHERE me.name = 'Filipa'
RETURN remote_friend.name
```

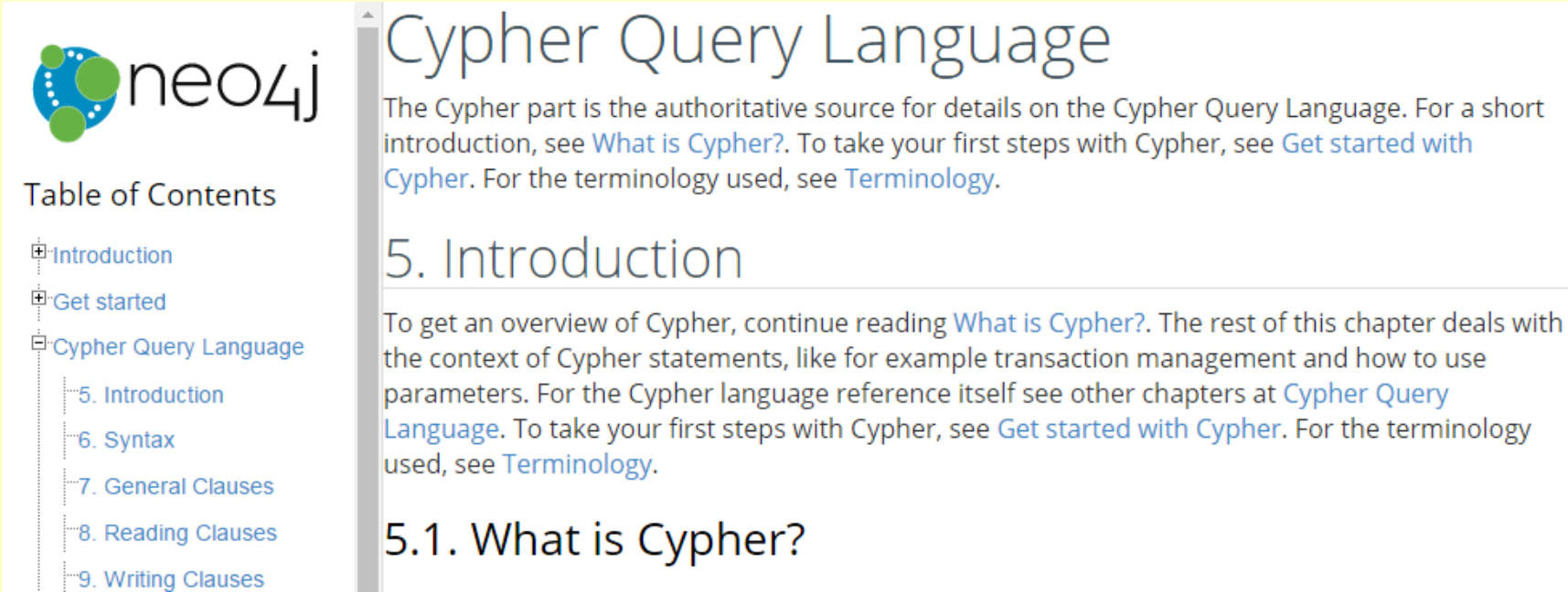
What will this query return?

remote_friend.name
'Dilshad'
'Anders'

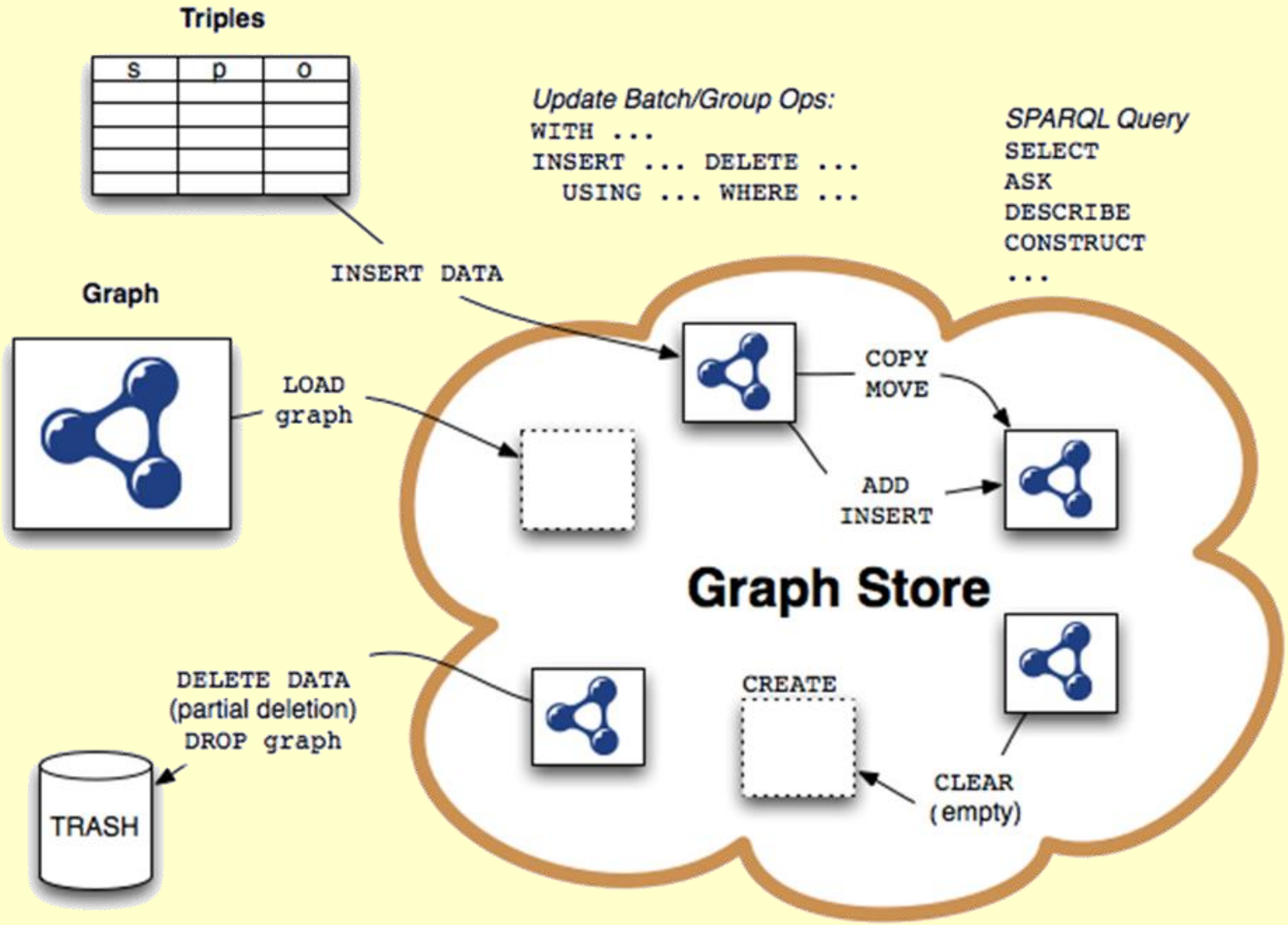
Graph Query Language: Cypher

- Relational operators:
  - UNION, FILTER, ORDER BY, GROUP BY, COUNT, etc.
- Paths:
  - Recursive (:EDGE*), Fixed length (:EDGE*1..5)

http://neo4j.com/docs/developer-manual/current/#cypher-query-lang



Graph Query Language: SPARQL



Read more on querying graphs

http://aidanhogan.com/docs/graph_database_query_survey.pdf

A

Foundations of Modern Query Languages for Graph Databases[1]

RENZO ANGLES, Universidad de Talca & Center for Semantic Web Research
MARCELO ARENAS, Pontificia Universidad Católica de Chile & Center for Semantic Web Research
PABLO BARCELÓ, DCC, Universidad de Chile & Center for Semantic Web Research
AIDAN HOGAN, DCC, Universidad de Chile & Center for Semantic Web Research
JUAN REUTTER, Pontificia Universidad Católica de Chile & Center for Semantic Web Research
DOMAGOJ VRGOČ, Pontificia Universidad Católica de Chile & Center for Semantic Web Research

We survey foundational features underlying modern graph query languages. We first discuss two popular graph data models: *edge-labelled graphs*, where nodes are connected by directed, labelled edges; and *property graphs*, where nodes and edges can further have attributes. Next we discuss the two most fundamental graph querying functionalities: *graph patterns* and *navigational expressions*. We start with graph patterns, in which a graph-structured query is matched against the data. Thereafter we discuss navigational expressions, in which patterns can be matched recursively against the graph to navigate paths of arbitrary length; we give an overview of what kinds of expressions have been proposed, and how they can be combined with graph patterns. We also discuss several semantics under which queries using the previous features can be evaluated, what effects the selection of features and semantics has on complexity, and offer examples of such features in three modern languages that are used to query graphs: SPARQL, Cypher and Gremlin. We conclude by discussing the importance of formalisation for graph query languages; a summary of what is known about SPARQL, Cypher and Gremlin in terms of expressivity and complexity; and an outline of possible future directions for the area.

CCS Concepts: •**Information systems** → **Query languages;** •**Theory of computation** → **Database query languages (principles);**

Additional Key Words and Phrases: Property graphs, graph databases, query languages, graph patterns, navigation, aggregation

ACM Reference Format:
ACM V, N, Article A (January YYYY), 42 pages.
DOI: 0000001.0000001

CC5212-1
Procesamiento Masivo de Datos
Otoño 2017

Lecture 10.1: Conclusion

Aidan Hogan
aidhog@gmail.com

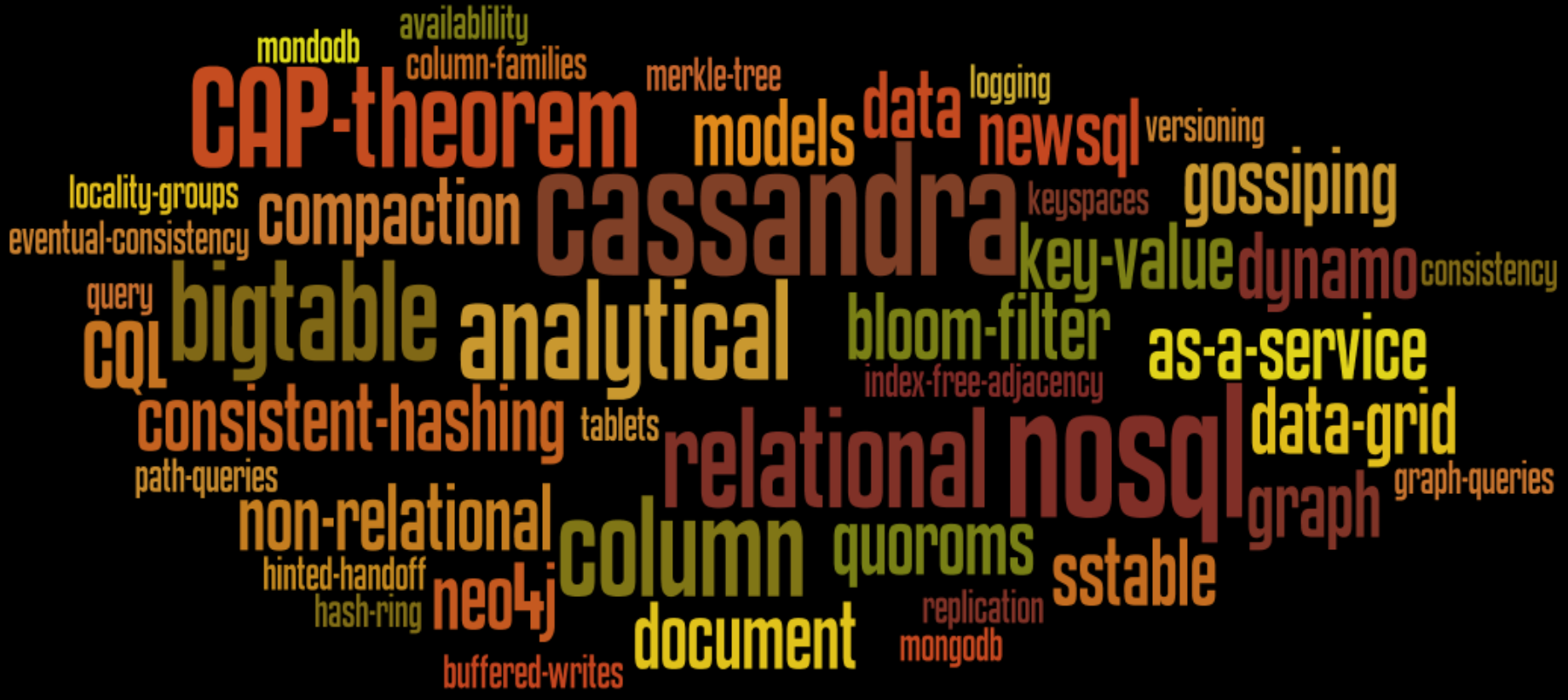
The value of data …

Distributed Systems

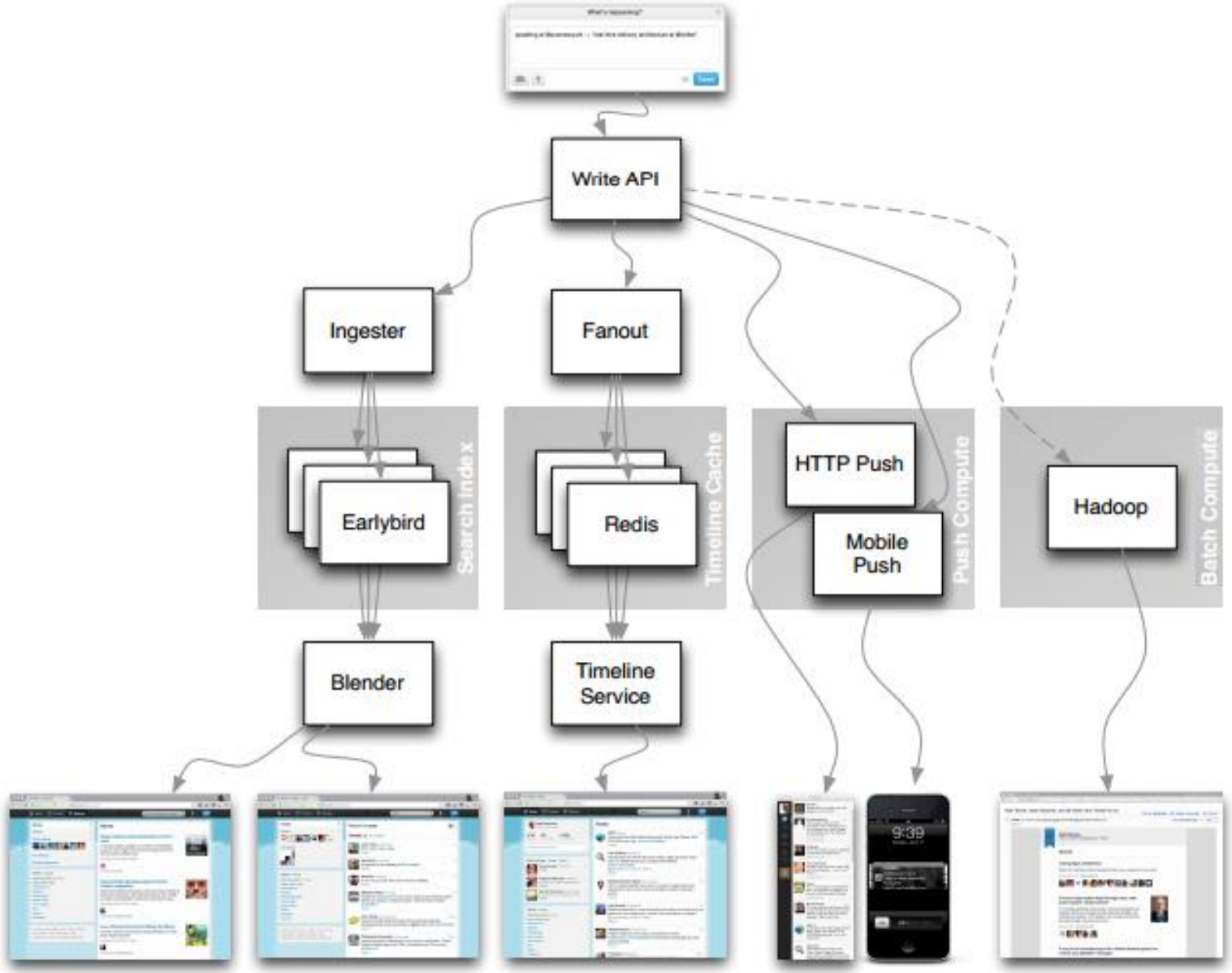
Hadoop/MapReduce/Pig/Spark:
Processing Un/Structured Information

Information Retrieval:
Storing Unstructured Information

NoSQL:
Storing (Semi-)Structured Information

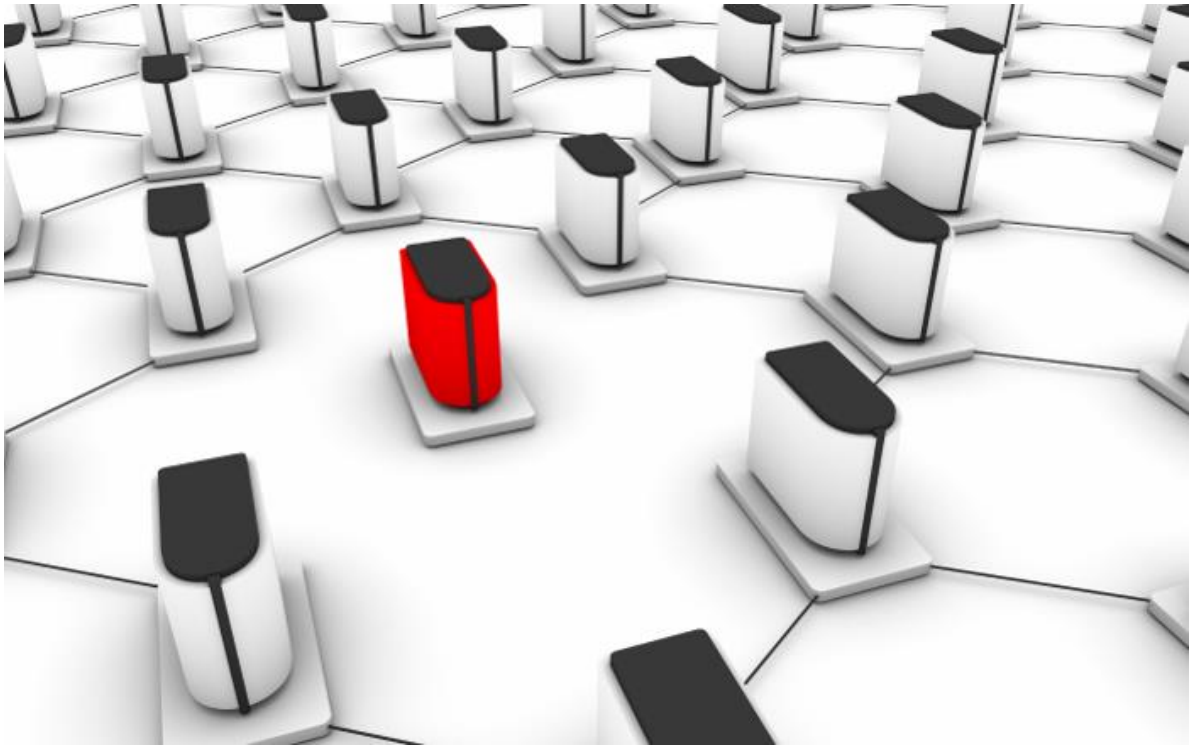


Twitter architecture

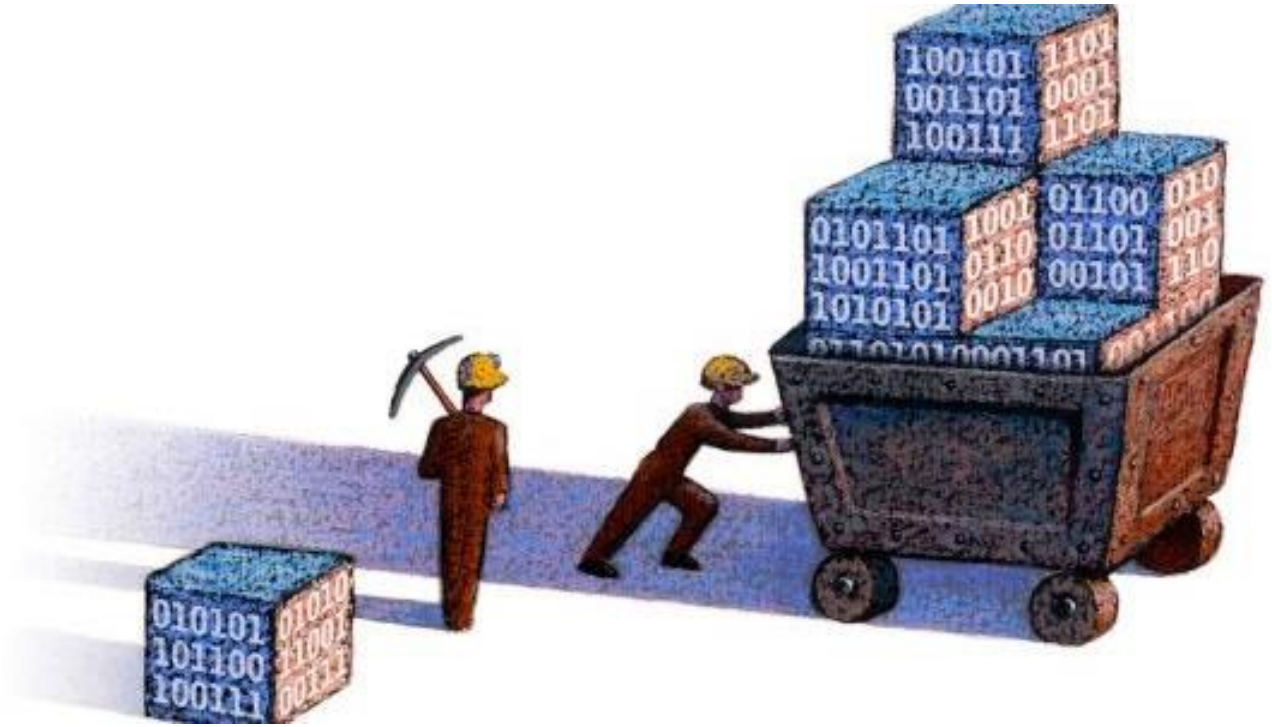


Generalise concepts to …

Distribution



Working with large datasets



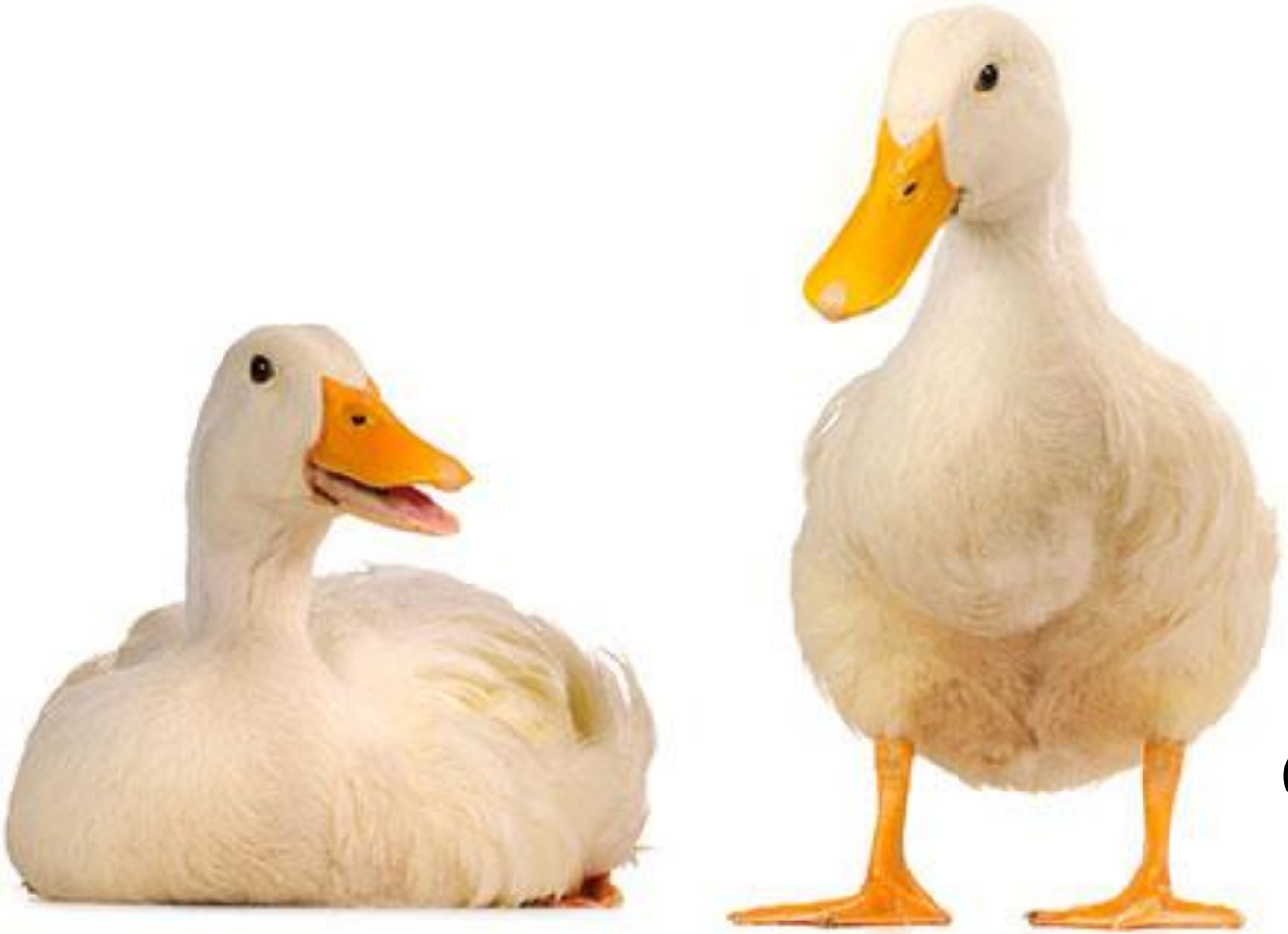
The Big Data Buzz-word

“Data Scientist” Job Postings (2016)

Here are the top 10 in-demand skills for data scientists:

Skills	Job skill appears in	% of jobs with skill
SQL	1987	56%
Hadoop	1713	49%
Python	1367	39%
Java	1287	36%
R	1120	32%
Hive	1099	31%
Mapreduce	768	22%
NoSQL	657	18%
Pig	561	16%
SAS	560	16%

http://visit.crowdflower.com/rs/416-ZBE-142/images/CrowdFlower_DataScienceReport_2016.pdf



Questions?