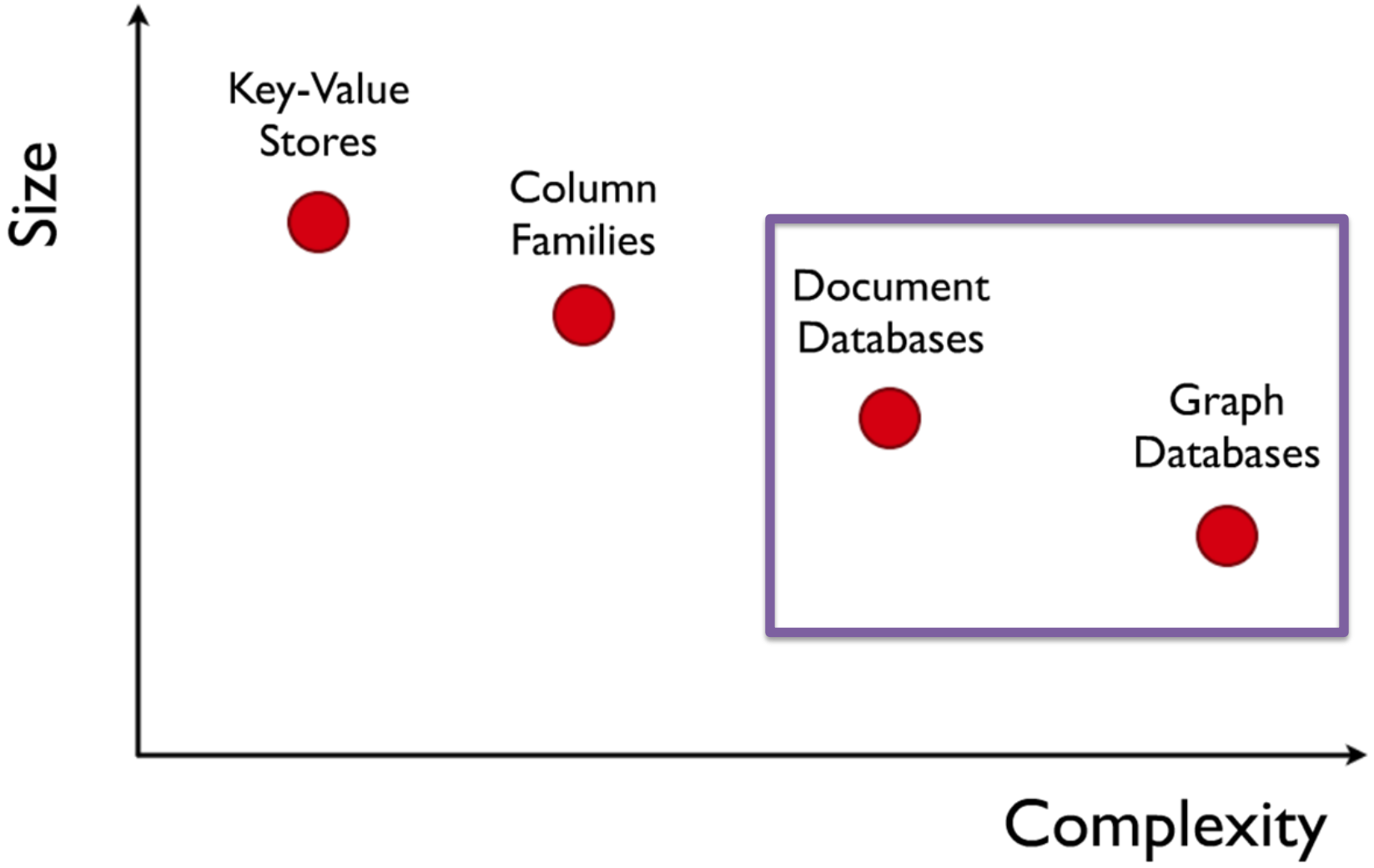
Big Data

Diplomado de Datos 2021

Clase 8: NoSQL (II)

Aidan Hogan
aidhog@gmail.com

NoSQL


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

373 systems in ranking, August 2021

Rank			DBMS	Database Model	Score		
Aug 2021	Jul 2021	Aug 2020			Aug 2021	Jul 2021	Aug 2020
1.	1.	1.	Oracle	Relational, Multi-model	1269.26	+6.59	-85.90
2.	2.	2.	MySQL	Relational, Multi-model	1238.22	+9.84	-23.36
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	973.35	-8.61	-102.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.05	-0.10	+40.28
5.	5.	5.	MongoDB	Document, Multi-model	496.54	+0.38	+52.98
6.	6.	↑ 7.	Redis	Key-value, Multi-model	169.88	+1.58	+17.01
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	165.46	+0.31	+3.01
8.	8.	8.	Elasticsearch	Search engine, Multi-model	157.08	+1.32	+4.76
9.	9.	9.	SQLite	Relational	129.81	-0.39	+3.00
10.	↑ 11.	10.	Microsoft Access	Relational	114.84	+1.39	-5.02
11.	↓ 10.	11.	Cassandra	Wide column	113.66	-0.35	-6.18
12.	12.	12.	MariaDB	Relational, Multi-model	98.98	+0.99	+8.06
13.	13.	13.	Splunk	Search engine	90.60	+0.55	+0.69
14.	14.	↑ 15.	Hive	Relational	83.93	+1.26	+8.64
15.	15.	↑ 17.	Microsoft Azure SQL Database	Relational, Multi-model	75.15	-0.06	+18.31
16.	16.	16.	Amazon DynamoDB	Multi-model	74.90	-0.30	+10.15
17.	17.	↓ 14.	Teradata	Relational, Multi-model	68.82	-0.13	-7.96
18.	18.	↑ 21.	Neo4j	Graph	56.95	-0.21	+6.77
19.	19.	19.	SAP HANA	Relational, Multi-model	55.57	+1.76	+2.46
20.	20.	20.	Solr	Search engine, Multi-model	51.06	-0.73	-0.63

NoSQL:
Almacenamiento de documentos

Llave–Valor: un mapa distribuido

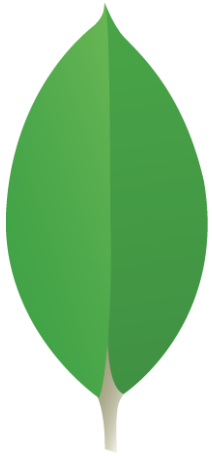
Countries	
Primary Key	**Value**
Afghanistan	capital:Kabul,continent:Asia,pop:31108077#2011
…	…

Tabular: un mapa distribuido multi-dimensional

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

Documentos: un mapa con valores de docs.

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



mongoDB®

{JSON}

JavaScript Object Notation

MongoDB:

Modelo de datos

"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
  }
}
```


JSON Binario: 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: Mapa de llaves a valores BSON

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>

…	…

Colección MongoDB:
Documentos relacionados

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

Base de datos MongoDB:
Colecciones relacionadas

TVSeries	
Key	**BSON Value**
...	...

TVEpisodes	
Key	**BSON Value**
...	...

TVNetworks	
Key	**BSON Value**
...	...

database: TV

Usar la database tvdb (la creará si no existe):

```
> use tvdb

switched to db tvdb
```

Ver todas las databases no vacías (tvdb es vacía aquí):

```
> show dbs

local     0.00001 GB
test      0.00231 GB
```

Ver la database actual:

```
> db

tvdb
```

Borrar la database actual:

```
> db.dropDatabase()

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

Crear la colección series:

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

{ "ok" : 1 }
```

Ver las colecciones en la database actual:

```
> show collections

series
```

Borrar la colección `series`:

```
> db.series.drop()

true
```

Borrar los documentos de la colección `series`:

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

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

Crear colección acotada ("capped": guarda los n más recientes):

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

{ "ok" : 1 }
```

Crear colección con índice (por defecto) sobre `_id`:

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

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

MongoDB:

Insertar datos

Insertar un documento: sin `_id`

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

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

Insertar documento: con _id

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

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

... falla si el valor de _id ya existe

... usar update or save para actualizar un documento

Usar save y _id para sobrescribir

```
> 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 })
```

... sobrescribe el documento previo

MongoDB:

Consultas con selección

`find()`:

Devolver todos los documentos de la colección

```
> 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 }
```

... usar `findOne()` para devolver un documento

pretty():

Imprimir con indentación

```
> 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
}
```

`find(σ)`:

Encontrar los documentos que satisfagan σ

```
> db.series.find(σ)
```

Selección σ: Igualdad

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

Igualdad: `{ key: value }`

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

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

Los resultados incluyen un valor de `_id` pero lo omitiremos aquí para mantener los ejemplos más concisos.

Selección σ: Llave anidada

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

La llave puede referenciar a un valor anidado

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

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

Selección σ: Igualdad con nulos

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

Igualdad con un nulo anidado

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

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

... coincide cuando el valor sea nulo o ...

Selección σ: Igualdad con nulos

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

Igualdad con un nulo anidado

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

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

... o cuando el key no exista.

Selección σ: Igualdad con un documento

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

Un valor puede ser un documento

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

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

Selección σ: Igualdad con un documento

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

Tiene que coincidir completamente:

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

... resultados vacíos: una coincidencia parcial.

Selección σ: Igualdad con un documento

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

El orden importa:

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

... resultados vacíos: el orden no coincide

Selección σ: Igualdad con un arreglo

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

Tiene que ser una coincidencia exacta:

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

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

Selección σ: Igualdad con un arreglo

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

Tiene que ser una coincidencia exacta:

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

... resultados vacíos: una coincidencia parcial.

Selección σ: Igualdad con un arreglo

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

El orden importa ...

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

... resultados vacíos: el orden no coincide

Selección σ: Pertenencia a un arreglo

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

Un valor coincidirá con un elemento de un arreglo

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

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

Selección σ: Pertenencia a un arreglo

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

… incluso adentro y afuera del arreglo al mismo tiempo

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

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

Selección σ: Desigualdades

Menor a: `{ key: { $lt: value } }`

Mayor a: `{ key: { $gt: value } }`

Menor a o igual a: `{ key: { $lte: value } }`

Mayor a o igual a: `{ key: { $gte: value } }`

No igual a: `{ key: { $ne: value } }`

Selección σ: Menor a

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

Menor a: `{ key: { $lt: value } }`

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

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

Selección σ: Múltiples valores

Algún valor: `{ key: { $in: [ v1, …, vn ] } }`

Ningún valor: `{ key: { $nin: [ v1, …, vn ] } }`

… o coincidirá si la llave no existe

Selección σ: Múltiples valores

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

Algún valor: `{ key: { $in: [ v1, …, vn ] } }`

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

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

Selección σ: Múltiples valores

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

Algún valor: `{ key: { $in: [ v1, …, vn ] } }`

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

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

… si la llave tiene un arreglo, cualquier valor del arreglo debería coincidir con cualquier valor de `$in`

Selección σ: Conectivos booleanos

Y: `{ $and: [ σ , σ′ ] }`

O: `{ $or: [ σ , σ′ ] }`

No: `{ $not: [ σ ] }`

No-O: `{ $nor: [ σ , σ′ ] }`

… se pueden anidar estas condiciones

Selección σ: Y

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

Y: { $and: [σ , σ′] }

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

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

Selección σ: Atributo (no) existe

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

No Existe: `{ key: { $exists : false } }`

Selección σ: Atributo existe

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

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

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

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

... verifica que la llave `key` exista
(incluso si el valor es `NULL`)

Selección σ: Atributo no existe

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

No existe: `{ key: { $exists : false } }`

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

... verifica que la llave key no exista (resultados vacíos)

Selección σ: Arreglos

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

Cualquiera: `{ key: { $elemMatch : { σ1, …, σn } } }`

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

Selección σ: Arreglo tiene (al menos) cada elemento

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

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

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

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

… cada valor está en el arreglo

Selección σ: Cada condición está satisfecha

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

Cualquiera: { key: { $elemMatch : {σ1, …, σn } } }

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

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

… para cada criterio, existe un elemento que lo satisfaga

Selección σ: Arreglo con largo exacto

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

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

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

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

... no hay rangos de largo (solo un valor exacto)

Selección σ: Tipo de un valor

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

"timestamp"

"decimal"

"string"

"double"

"binData"

"date"

"object"

"int"

"array"

"long"

"bool"

"undefined"

"objectId"

"null"

"regex"

"dbPointer"

"array"

"javascript"

"number"

…

Selección σ: Tipo de un valor

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

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

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

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

Selección σ: Encontrar valores que son arreglos

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

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

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

... resultados vacíos:
coincidirá si algún valor del arreglo es de ese tipo

Selección σ: Encontrar valores que son arreglos

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
  })
```

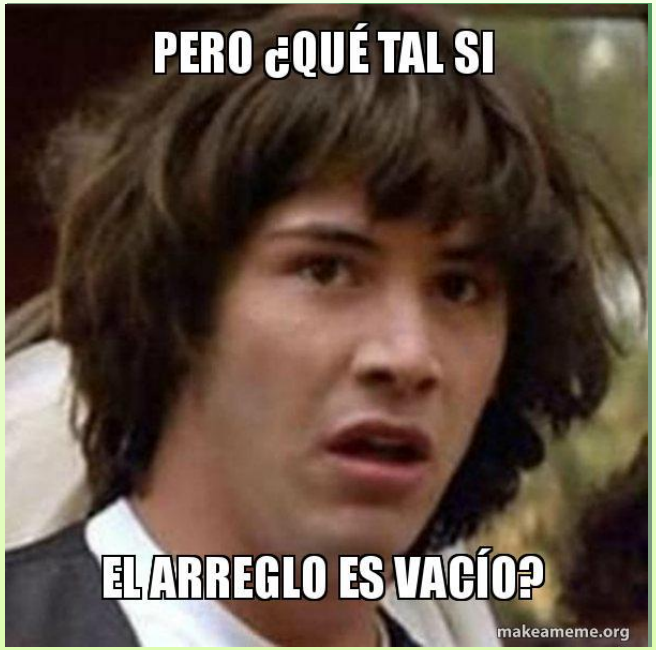
```
> db.series.find(
 { "genres": { $elemMatch: { $exists : true } } }
 )

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

Selección σ: Encontrar valores que son arreglos

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 }
```

Selección σ: Otros operadores

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

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

Palabras claves: `{ $text: { $search: terms } }`

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

... se ejecuta `where` para todos los documentos
(y debería ser evitado cuando sea posible)

Selección σ: Características geográficas

...

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

Selección σ: Operadores al nivel de bits

`$bitsAllClear`

`$bitsAllSet`

`$bitsAnyClear`

`$bitsAnySet`

...

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

MongoDB:

Proyección de valores de salida

Proyección π: Elegir valores de salida

`key: 1`:	Emitir el campo con `key`
`key: 0`:	Suprimir el campo con `key`
`array.$: 1`	Proyectar la 1.ª coincidencia del arreglo
`$elemMatch`:	Proyectar la 1.ª coincidencia del arreglo
`$slice`:	Emitir un sub-arreglo de valores

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

Proyección π: Emitir algunos campos

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

Emitir algunos campos: { k1: 1, ..., kn: 1 }

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

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

... emite lo que está disponible (incluso `_id` por defecto)

Proyección π: Emitir campos anidados

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

Emitir campos (anidados): `{ k.k': 1 }`

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

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

... el campo se mantiene anidado en la salida

Proyección π: Emitir valores de un arreglo

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

Emitir valores de un arreglo: `{ k.k': 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 } ] }
```

... proyecta valores del arreglo

Proyección π: Suprimir algunos campos

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

Emitir todos menos ...: `{ k1: 0, ..., kn: 0 }`

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

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

... no se puede combinar 0 y 1 salvo ...

Proyección π: Suprimir algunos campos

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

Suprimir 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 ] }
```

... se puede suprimir `_id` cuando se emiten otros valores

Proyección π: Emitir primera coincidencia

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

Proyectar primera coincidencia del arreglo: `array.$: 1`

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

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

Proyección π: Emitir primera coincidencia

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

Proyectar primera coincidencia del arreglo: `$elemMatch`

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

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

... separa las condiciones de selección y proyección

Proyección π: Emitir primera coincidencia

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

Proyectar primera coincidencia del arreglo: $elemMatch

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

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

... suprime el arreglo si no existe ninguna coincidencia

Proyección π: Emitir primera coincidencia

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

Proyectar primera coincidencia del arreglo: $elemMatch

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

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

... funciona con arreglos de documentos

Proyección π: Emitir un sub-arreglo

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

Devolver los primeros n elementos: $slice: n (n > 0)

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

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

Proyección π: Emitir un sub-arreglo

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

Devolver los últimos n elementos: $slice: n (n < 0)

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

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

Proyección π: Emitir un sub-arreglo

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

Saltar n y devolver 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 }
```

Proyección π: Emitir un sub-arreglo

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

De los últimos n, devolver 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:

Actualizaciones

Actualizar `u`: Modificar campos

`$set`:	Establecer el valor
`$unset`:	Borrar la llave y su valor
`$rename`:	Renombrar la llave de un campo
`$setOnInsert`:	Establecer el valor si es un doc. nuevo
`$inc`:	Incrementar un número por `inc`
`$mul`:	Multiplicar un número por `mul`
`$min`:	Reemplazar valores < `min` por `min`
`$max`:	Reemplazar valores > `max` por `max`
`$currentDate`:	Establecer el valor con la fecha actual

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

update con criterios de query y update:

```
> 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"              }
```

Actualizar **u**: Modificar arreglos

`$addToSet`: Agregar valor si no existía antes
`$pop`: Borrar el primer o último elemento
`$push`: (Empujar) Agregar un valor al fin del arreglo
`$pullAll`: Borrar todos los valores del arreglo
`$pull`: Borrar valores que satisfagan una condición

(Sub-)operadores para agregar o empujar valores:

`$`: Actualizar la primera coincidencia (`$push`)
`$each`: Agregar o empujar múltiples valores (`$push`, `$addToSet`)
`$slice`: Después de haber empujado, mantener los primeros o últimos `slice` valores (`$push`)
`$sort`: Ordenar el arreglo (`$push`)
`$position`: Empujar a una posición en particular (`$push`)

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

Actualizar u: Modificar arreglos

```
> 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" ] }
```

Actualizar u: Modificar arreglos

```
> 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:
Agregación y "pipelines"

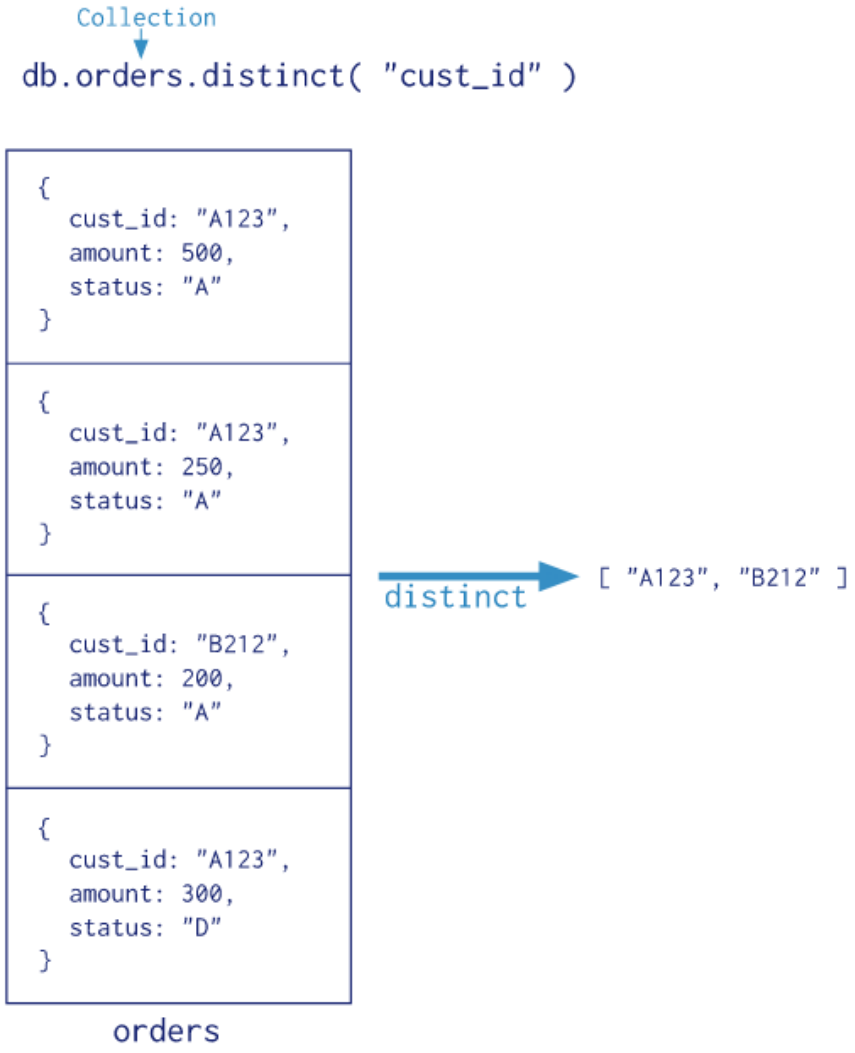
Agregación: Sin agrupación

`db.coll.distinct(key)`: Arreglo de valores únicos de `key`

`db.coll.count()`: Contar los documentos

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

Agregación: distinct



"Pipelines": Transformando colecciones

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

Pipeline ρ: Operadores de etapa

`$match`: Filtrar por el criterio de selección σ

`$project`: Proyectar π

`$group`: Agrupar documentos por llave o valor

[usado con *`$sum`*, *`$avg`*, *`$first`*, *`$last`*, *`$max`*, *`$min`*, etc.]

`$lookup`: Hacer un left-outer-join con otra colección

`$unwind`: Copiar cada documento para cada valor de un arreglo

`$collStats`: Ver estadísticas sobre la colección

`$count`: Contar los documentos en la colección

`$sort`: Ordenar docs por los valores de una llave (`ASC`|`DESC`)

`$limit`: Devolver (hasta) los `n` primeros documentos

`$sample`: Devolver (hasta) `n` documentos muestreados

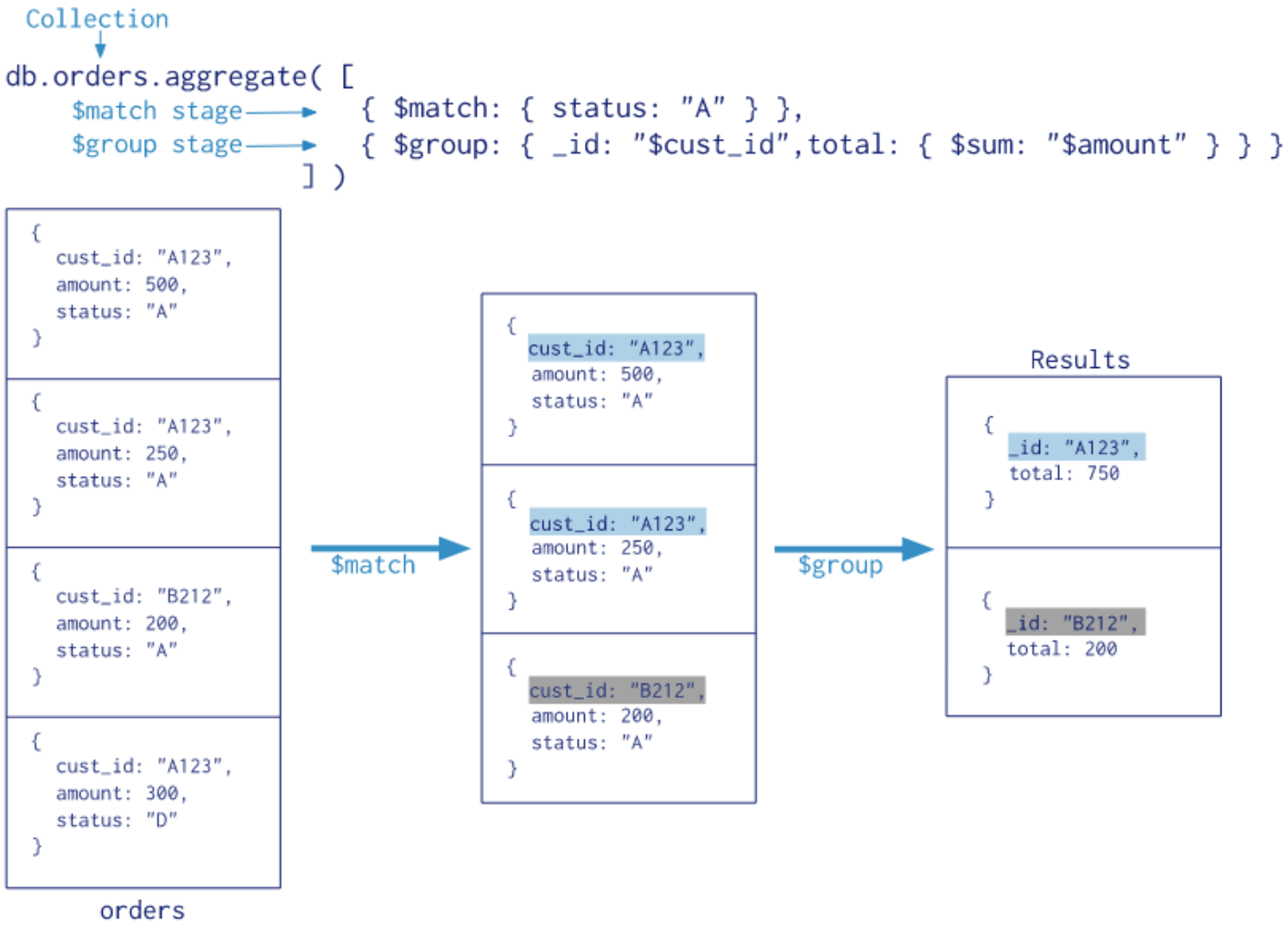
`$skip`: Saltar `n` documentos

`$out`: Guardar la colección a MongoDB

(… y aún más)

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

Agregación de Pipeline: $match y $group



Agregación de Pipeline: $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": []
}
```

Agregación de Pipeline: $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:

Indexación

MongoDB: Indexación

Por colección:

- "`_id` Index"
- "Single-field Index" (ordenado)
- "Compound Index" (ordenado por varios campos)
- "Multikey Index" (para arreglos)
- "Geospatial Index" (búsqueda de vecino)
- "Full-text Index" (búsqueda de texto)
- "Hash-based indexing" (hasheado)

https://docs.mongodb.com/v3.4/indexes/

MongoDB: Indexar texto

```
> 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: Indexar texto

```
> 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: Indexar texto

```
> 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:

Distribución

MongoDB: Distribución

- "Sharding" (Partición):
  - Hashing o Horizontal por rango

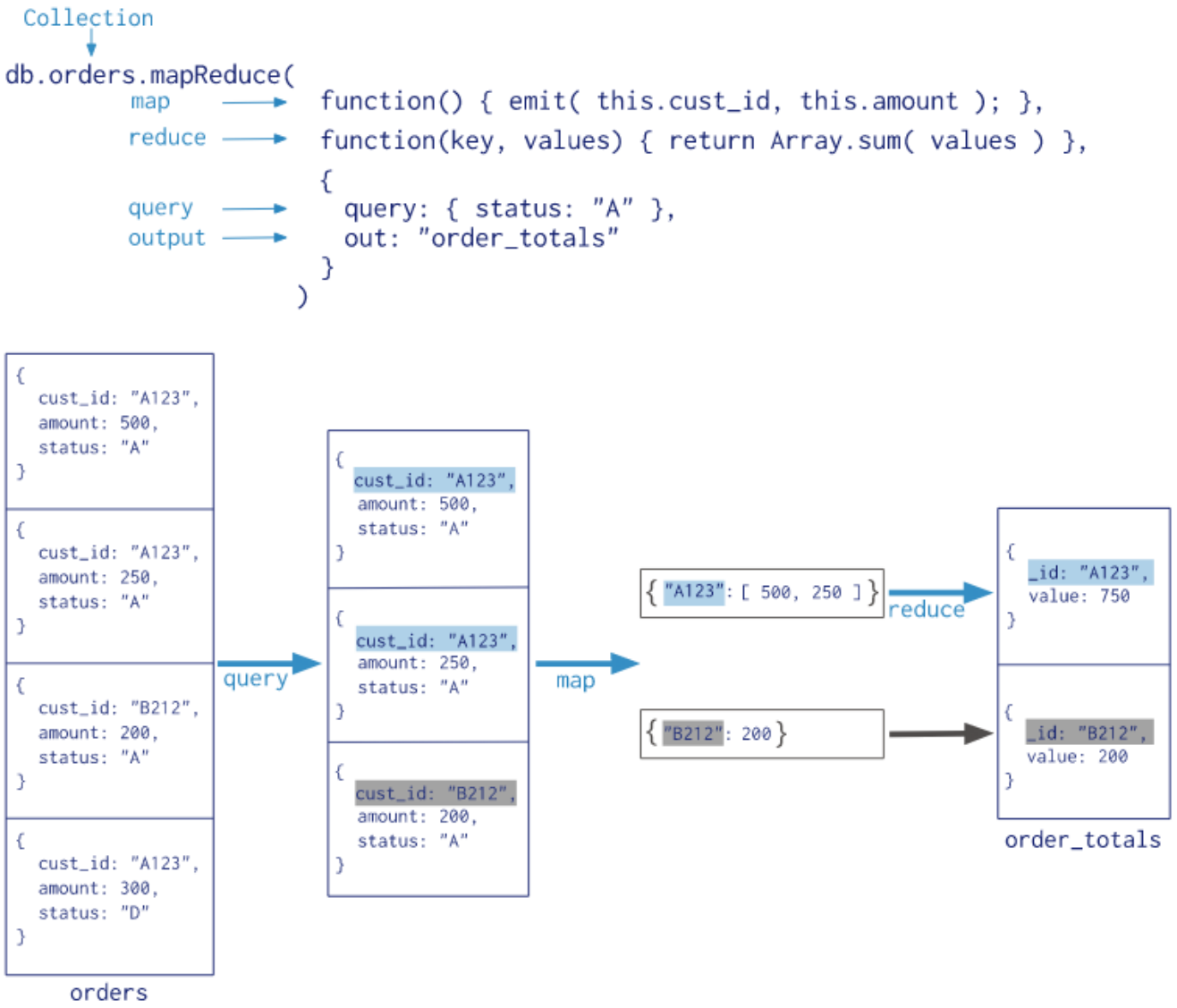
    (Depende de los índices)

    https://docs.mongodb.com/v3.4/sharding/

- Replicación
  - Replica sets con replicas primarias/secundarias

    https://docs.mongodb.com/v3.4/replication/

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:

Una advertencia

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/

Ransomware groups have deleted over 10,000 MongoDB databases

Five groups of attackers are competing to delete as many publicly accessible MongoDB databases as possible

By Lucian Constantin

CSO Senior Writer, IDG News Service | JAN 6, 2017 10:02 AM PST

Gerd Altmann (CC0)

Groups of attackers have adopted a new tactic that involves deleting publicly exposed MongoDB databases and asking for money to restore them. In a matter of days, the number of affected databases has risen from hundreds to more than 10,000.

The issue of misconfigured MongoDB installations, allowing anyone on the internet to access sensitive data, is not new. Researchers have been finding such open databases for years, and the latest estimate puts their number at more than 99,000.

https://www.computerworld.com/article/3155260/ransomware-groups-have-deleted-over-10000-mongodb-databases.html

MongoDB "open-source" Server Side Public License rejected

Red Hat won't use MongoDB in Red Hat Enterprise Linux or Fedora thanks to MongoDB's new Server Side Public License.

By Steven J. Vaughan-Nichols for Linux and Open Source |
January 16, 2019 -- 19:33 GMT (11:33 PST) | Topic: Cloud

MongoDB is an open-source document NoSQL database with a problem. While very popular, cloud companies, such as Amazon Web Services (AWS), IBM Cloud, Scalegrid, and ObjectRocket has profited from it by offering it as a service while MongoDB Inc. hasn't been able to monetize it to the same degree. MongoDB's answer? Relicense the program under its new Server Side Public License (SSPL). Open-source powerhouse Red Hat's reaction? Drop MongoDB from Red Hat Enterprise Linux (RHEL) 8.

Red Hat's Technical and Community Outreach Program Manager Tom Callaway explained, in a note stating MongoDB is being removed from Fedora Linux, that "It is the belief of Fedora that the SSPL is intentionally crafted to be aggressively discriminatory towards a specific class of users." Debian Linux had already dropped MongoDB from its distribution.

AI, ML & DATA ENGINEERING

Jepsen Disputes MongoDB's Data Consistency Claims

LIKE 5

MAY 22, 2020 • 4 MIN READ

by
Jonathan Allen
FOLLOW

In an article titled MongoDB and Jepsen, MongoDB claimed that their database passed "the industry's toughest data safety, correctness, and consistency Tests". In response, Jepsen published an article stating that MongoDB 3.6.4 had in fact failed their tests; the newer MongoDB 4.2.6 has more problems including "retrocausal transactions" where a transaction reverses order so that a read can see the result of a future write.

Jepsen LLC's response begins with this reply to Maxime Beugnet on their official Twitter feed:

> "I have to admit raising an eyebrow when I saw that web page. In that report, MongoDB lost data and violated causal by default. Somehow that became "among the strongest data consistency, correctness, and safety guarantees of any database available today"!

https://www.infoq.com/news/2020/05/Jepsen-MongoDB-4-2-6/

MongoDB:

¿Por qué es tan popular?

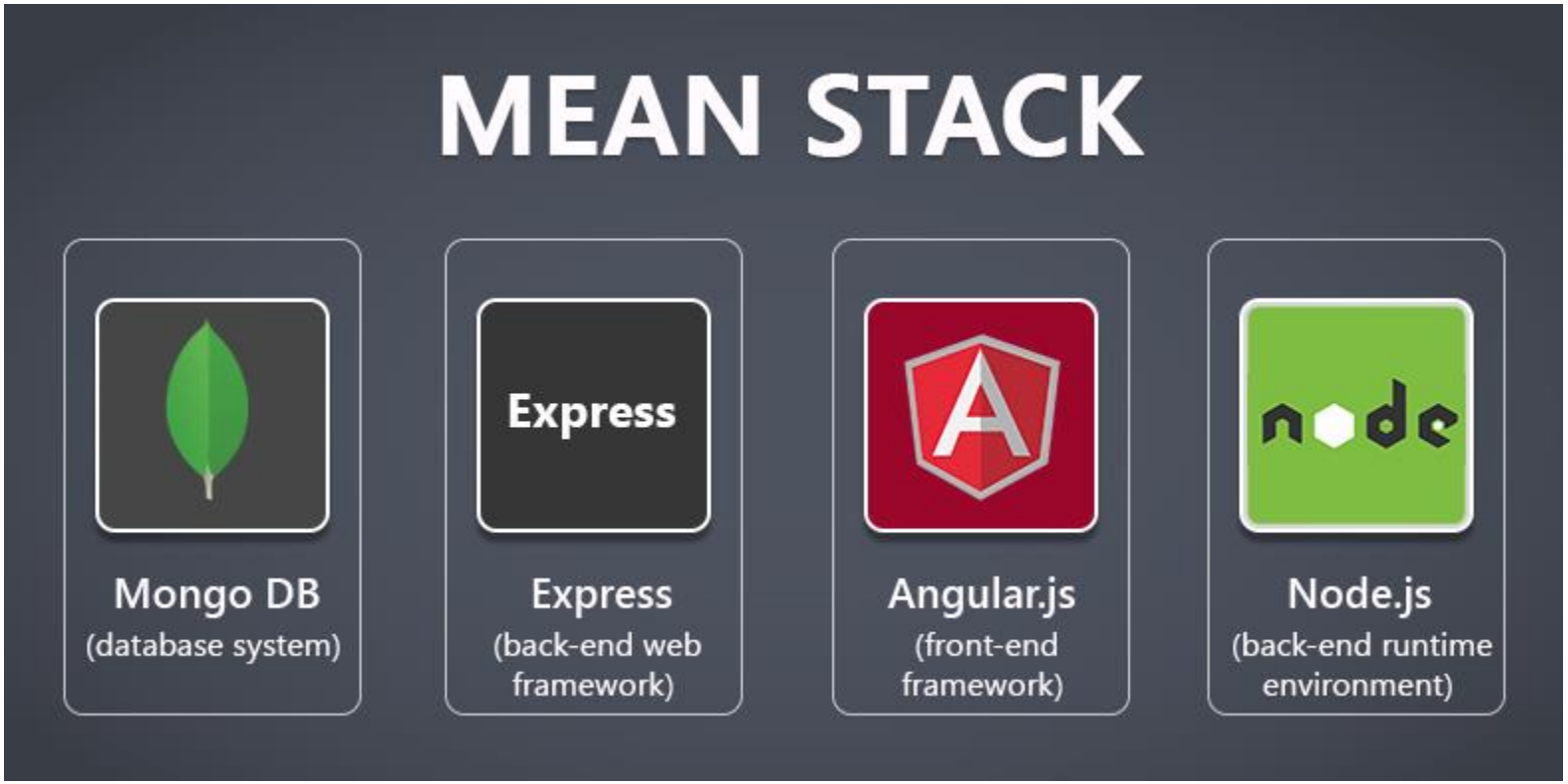
373 systems in ranking, August 2021

Rank			DBMS	Database Model	Score		
Aug 2021	Jul 2021	Aug 2020			Aug 2021	Jul 2021	Aug 2020
1.	1.	1.	Oracle	Relational, Multi-model	1269.26	+6.59	-85.90
2.	2.	2.	MySQL	Relational, Multi-model	1238.22	+9.84	-23.36
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	973.35	-8.61	-102.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.05	-0.10	+40.28
5.	5.	5.	MongoDB	Document, Multi-model	496.54	+0.38	+52.98
6.	6.	↑ 7.	Redis	Key-value, Multi-model	169.88	+1.58	+17.01
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	165.46	+0.31	+3.01
8.	8.	8.	Elasticsearch	Search engine, Multi-model	157.08	+1.32	+4.76
9.	9.	9.	SQLite	Relational	129.81	-0.39	+3.00
10.	↑ 11.	10.	Microsoft Access	Relational	114.84	+1.39	-5.02
11.	↓ 10.	11.	Cassandra	Wide column	113.66	-0.35	-6.18
12.	12.	12.	MariaDB	Relational, Multi-model	98.98	+0.99	+8.06
13.	13.	13.	Splunk	Search engine	90.60	+0.55	+0.69
14.	14.	↑ 15.	Hive	Relational	83.93	+1.26	+8.64
15.	15.	↑ 17.	Microsoft Azure SQL Database	Relational, Multi-model	75.15	-0.06	+18.31
16.	16.	16.	Amazon DynamoDB	Multi-model	74.90	-0.30	+10.15
17.	17.	↓ 14.	Teradata	Relational, Multi-model	68.82	-0.13	-7.96
18.	18.	↑ 21.	Neo4j	Graph	56.95	-0.21	+6.77
19.	19.	19.	SAP HANA	Relational, Multi-model	55.57	+1.76	+2.46
20.	20.	20.	Solr	Search engine, Multi-model	51.06	-0.73	-0.63

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

MEAN Stack

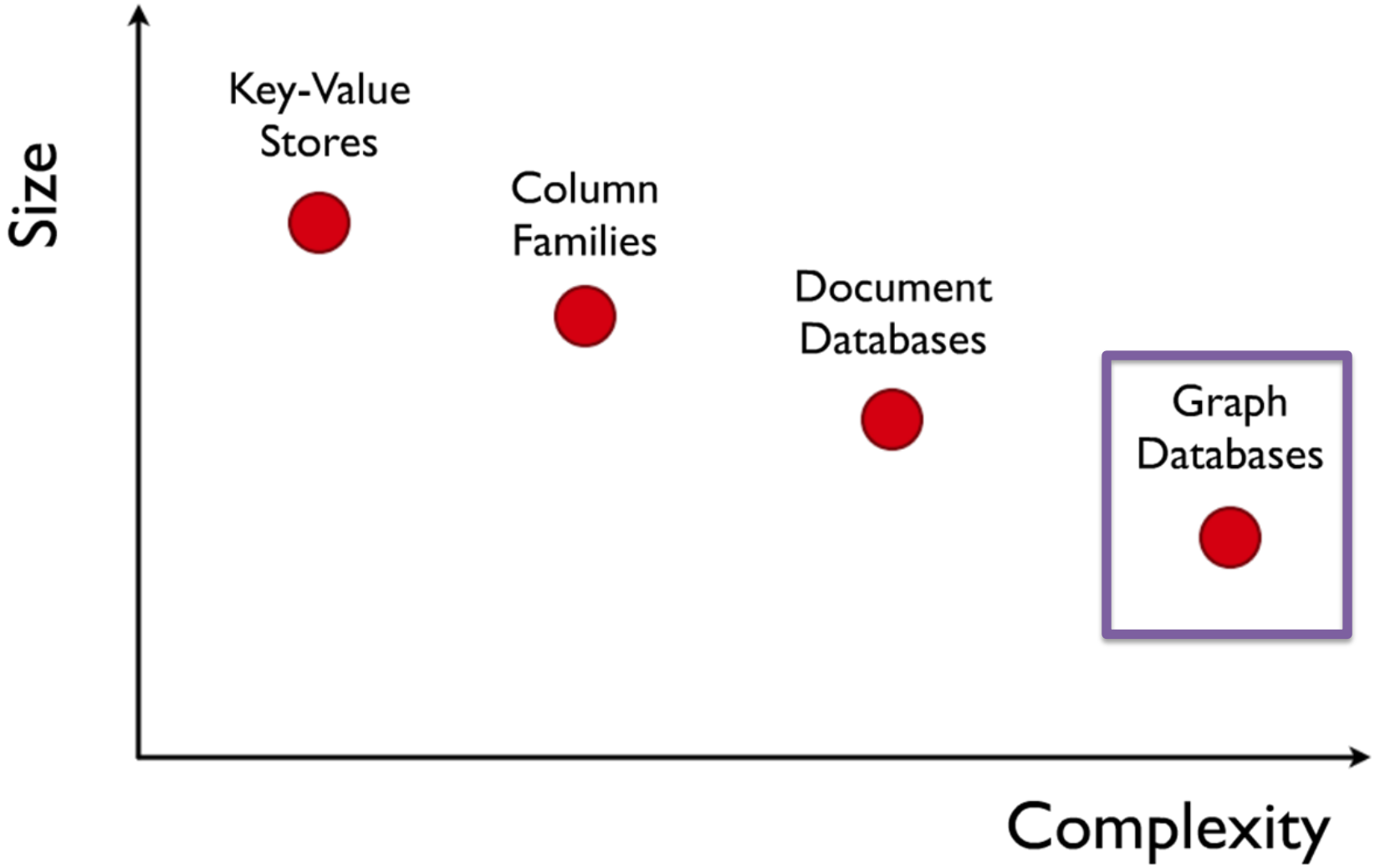
- MongoDB, Express.js, Angular(JS), Node.js



https://medium.com/@jeremyvsjeremy/what-is-the-mean-stack-9d11ae2cd384

Bases de datos de grafos

NoSQL


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

373 systems in ranking, August 2021

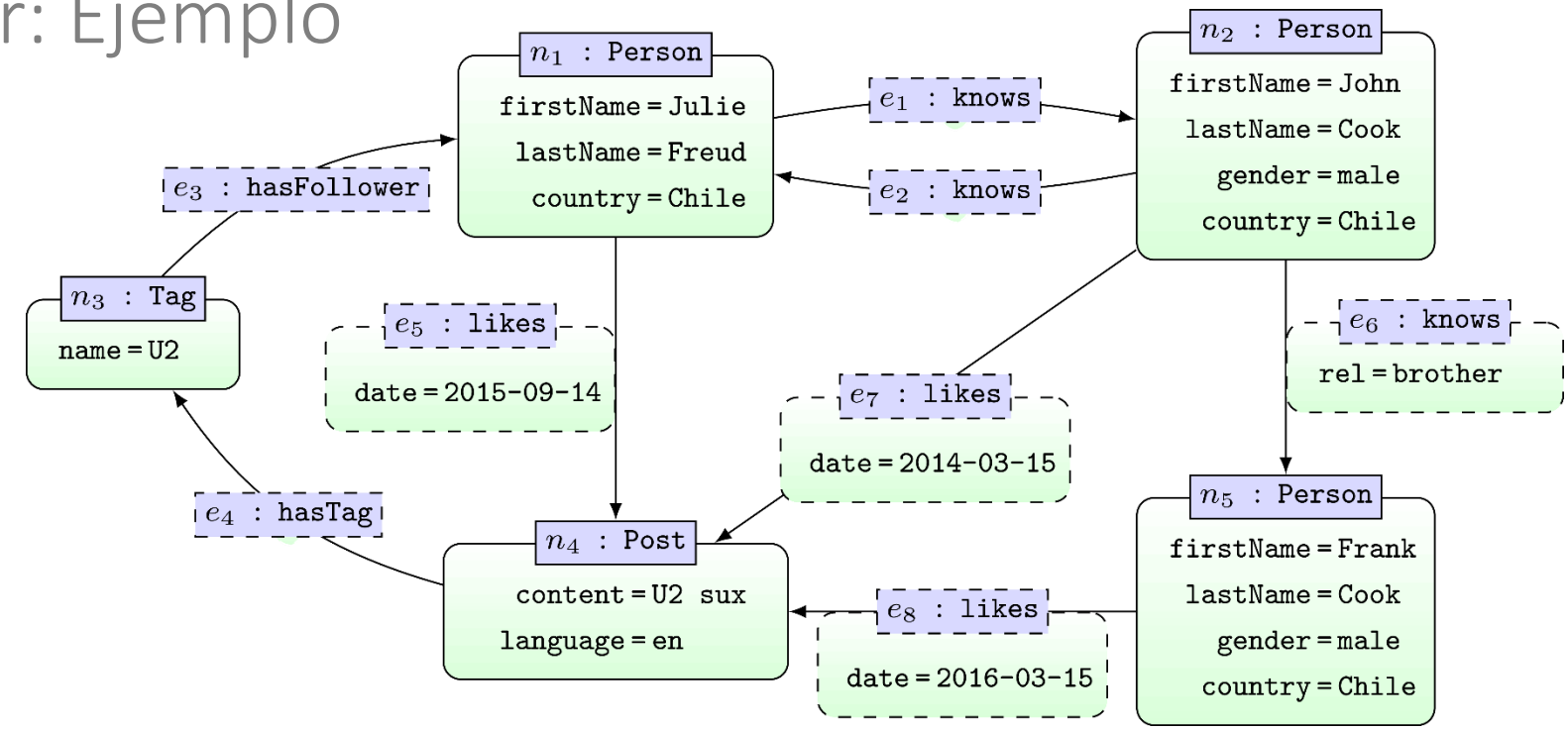
Rank			DBMS	Database Model	Score		
Aug 2021	Jul 2021	Aug 2020			Aug 2021	Jul 2021	Aug 2020
1.	1.	1.	Oracle	Relational, Multi-model	1269.26	+6.59	-85.90
2.	2.	2.	MySQL	Relational, Multi-model	1238.22	+9.84	-23.36
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	973.35	-8.61	-102.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.05	-0.10	+40.28
5.	5.	5.	MongoDB	Document, Multi-model	496.54	+0.38	+52.98
6.	6.	↑ 7.	Redis	Key-value, Multi-model	169.88	+1.58	+17.01
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	165.46	+0.31	+3.01
8.	8.	8.	Elasticsearch	Search engine, Multi-model	157.08	+1.32	+4.76
9.	9.	9.	SQLite	Relational	129.81	-0.39	+3.00
10.	↑ 11.	10.	Microsoft Access	Relational	114.84	+1.39	-5.02
11.	↓ 10.	11.	Cassandra	Wide column	113.66	-0.35	-6.18
12.	12.	12.	MariaDB	Relational, Multi-model	98.98	+0.99	+8.06
13.	13.	13.	Splunk	Search engine	90.60	+0.55	+0.69
14.	14.	↑ 15.	Hive	Relational	83.93	+1.26	+8.64
15.	15.	↑ 17.	Microsoft Azure SQL Database	Relational, Multi-model	75.15	-0.06	+18.31
16.	16.	16.	Amazon DynamoDB	Multi-model	74.90	-0.30	+10.15
17.	17.	↓ 14.	Teradata	Relational, Multi-model	68.82	-0.13	-7.96
18.	18.	↑ 21.	Neo4j	Graph	56.95	-0.21	+6.77
19.	19.	19.	SAP HANA	Relational, Multi-model	55.57	+1.76	+2.46
20.	20.	20.	Solr	Search engine, Multi-model	51.06	-0.73	-0.63

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

Neo4j: Sistema de base de datos de grafo

- Modelo de datos: Property Graphs
- Lenguaje de consulta: Cypher
- Lenguaje de *scripting*: Gremlin
- Licencia: *Open Source* (Una máquina)
  Comercial (Con distribución)

Cypher: Ejemplo

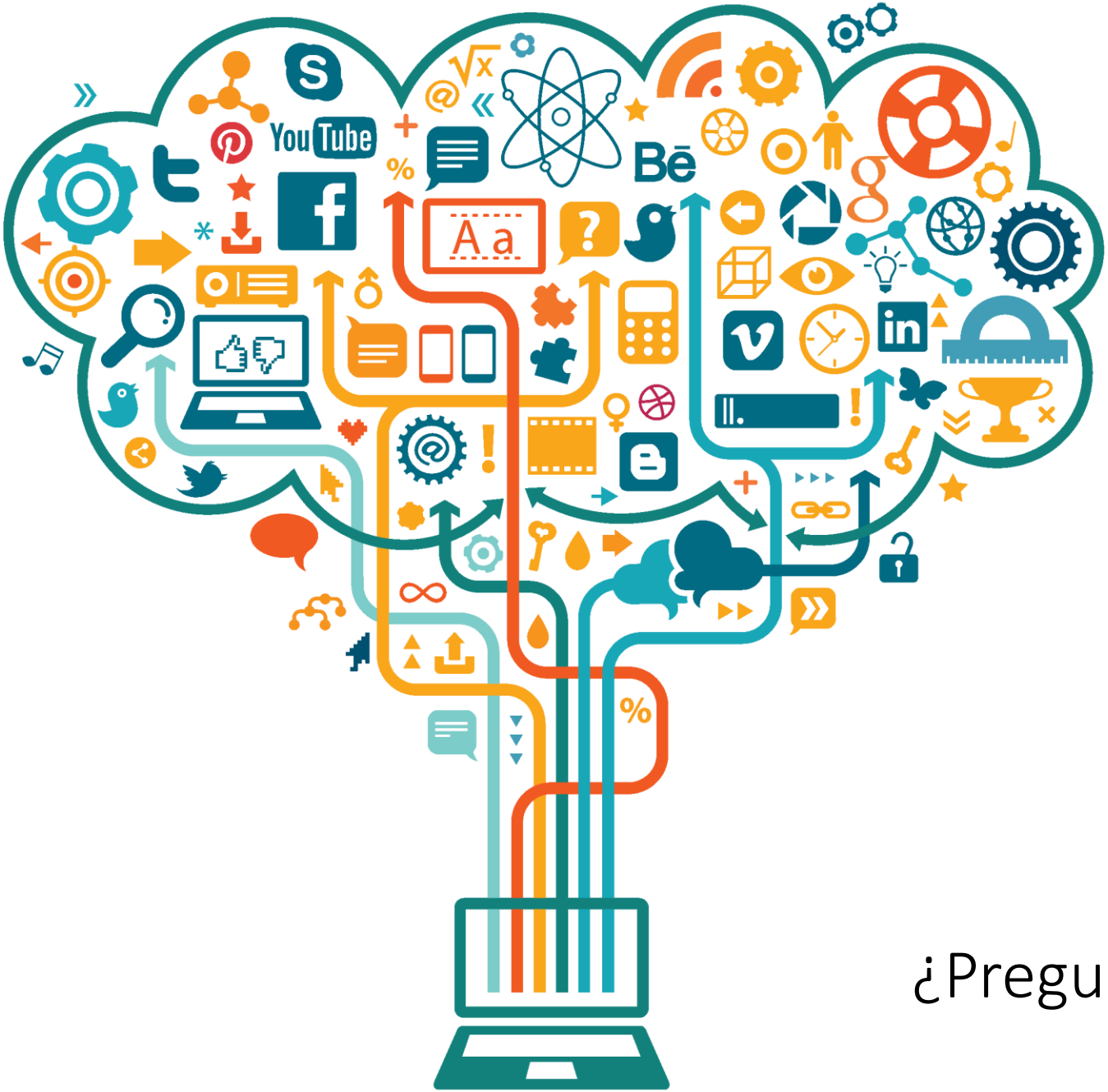



```
MATCH (x1)-[:knows*]->(x2)
RETURN x1.firstName, x2.firstName
```

¿Qué devolverá la consulta?

x1.firstName	x2.firstName
Julie	John
Julie	Frank
Julie	Julie
John	Julie
John	John
John	Frank
John	Frank

... los caminos visitan a cada nodo como máximo una vez



¿Preguntas?