mongoDB

un nuevo modelo de consultas basado en grafos
y un potente motor de búsquedas geoespaciales

Fernando Ortega

Marzo 2017

Contexto

¿Dónde se situa MongoDB?

Big Data

INFORMACIÓN

SISTEMAS DE ALMACENAMIENTO

ALGORITMOS INTELIGENTES

REPRESENTACIÓN DE DATOS

Fernando Ortega (fernando.ortega@upm.es)

Tipos de datos

ESTRUCTURADOS

SEMI ESTRUCTURADOS

NO ESTRUCTURADOS

Fernando Ortega (fernando.ortega@upm.es)

Sistemas de almacenamiento

- Datos estructurados
  - Hojas de cálculo
  - Bases de datos relacionales
- Datos semi o no estructurados
  - Se necesita rediseñar los sistemas de almacenamiento

Fernando Ortega (fernando.ortega@upm.es)

Introducción

Conceptos básicos sobre MongoDB

MongoDB

- MongoDB ("humongous") es una base de datos orientada a documentos
- Líder de las bases de datos NoSQL
- Es gratis y open-source
- Usa UTF-8 como codificación
- http://www.mongodb.org/

JSON

- Se basa en el formato JSON para su sintaxis:

```
{
    “nombre”: “Juan”,
    “apellidos”: “García Alonso”,
    “edad”: 42,
    “hijos”: true,
    “telefonos”: [“625741265”,”915478522”],
    “direccion”: {
        “calle”: “C/ Arcoiris”,
        “numero”: 89,
        “poblacion”: ”Madrid”,
        “codigo_postal”: 28123
    }
}
```

MongoDB vs SQL

MONGO DB		SQL
Base de datos	⟷	Base de datos
Colección	⟷	Tabla
Documento	⟷	Fila

Fernando Ortega (fernando.ortega@upm.es)

Documentos

- MongoDB almacena la información en forma de documentos
  - … que son pares clave-valor en formato JSON

Fernando Ortega (fernando.ortega@upm.es)

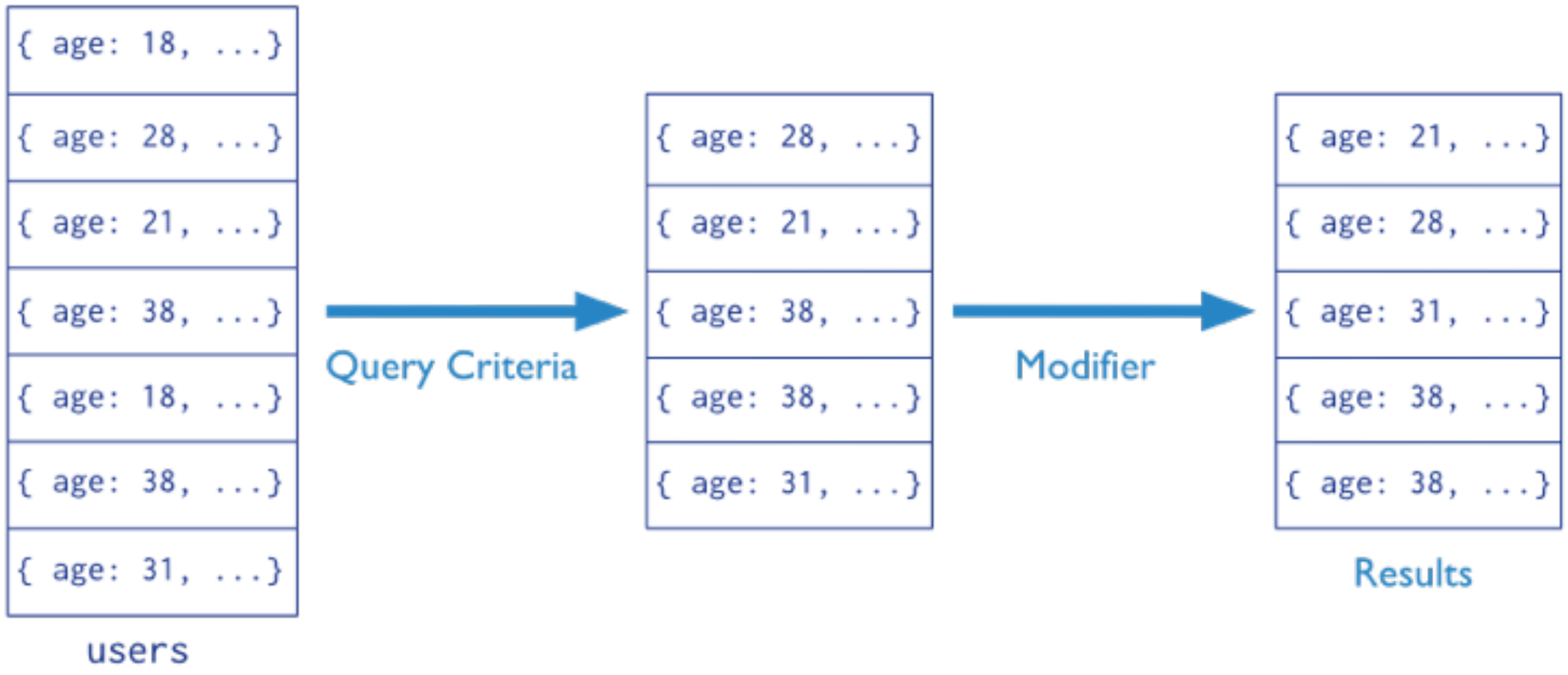
Colecciones

- MongoDB almacena todos los documentos en colecciones
- Una colección es un grupo de documentos relacionados semánticamente

Queries

- En MongoDB las consultas se hacen sobre una colección de documentos
  - Se especifican los criterios de los documentos a recuperar

```
Collection                Query Criteria                Modifier
db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 } )
```



Conceptos básicos

- Los documentos en MongoDB tienen un esquema flexible
  - Las colecciones de MongoDB no obligan a que sus documentos tengan un formato único
- Una colección puede tener varios documentos con una estructura diferente
  - En la práctica los documentos de una colección comparten una estructura similar
  - Todos los documentos tendrán un campo _id

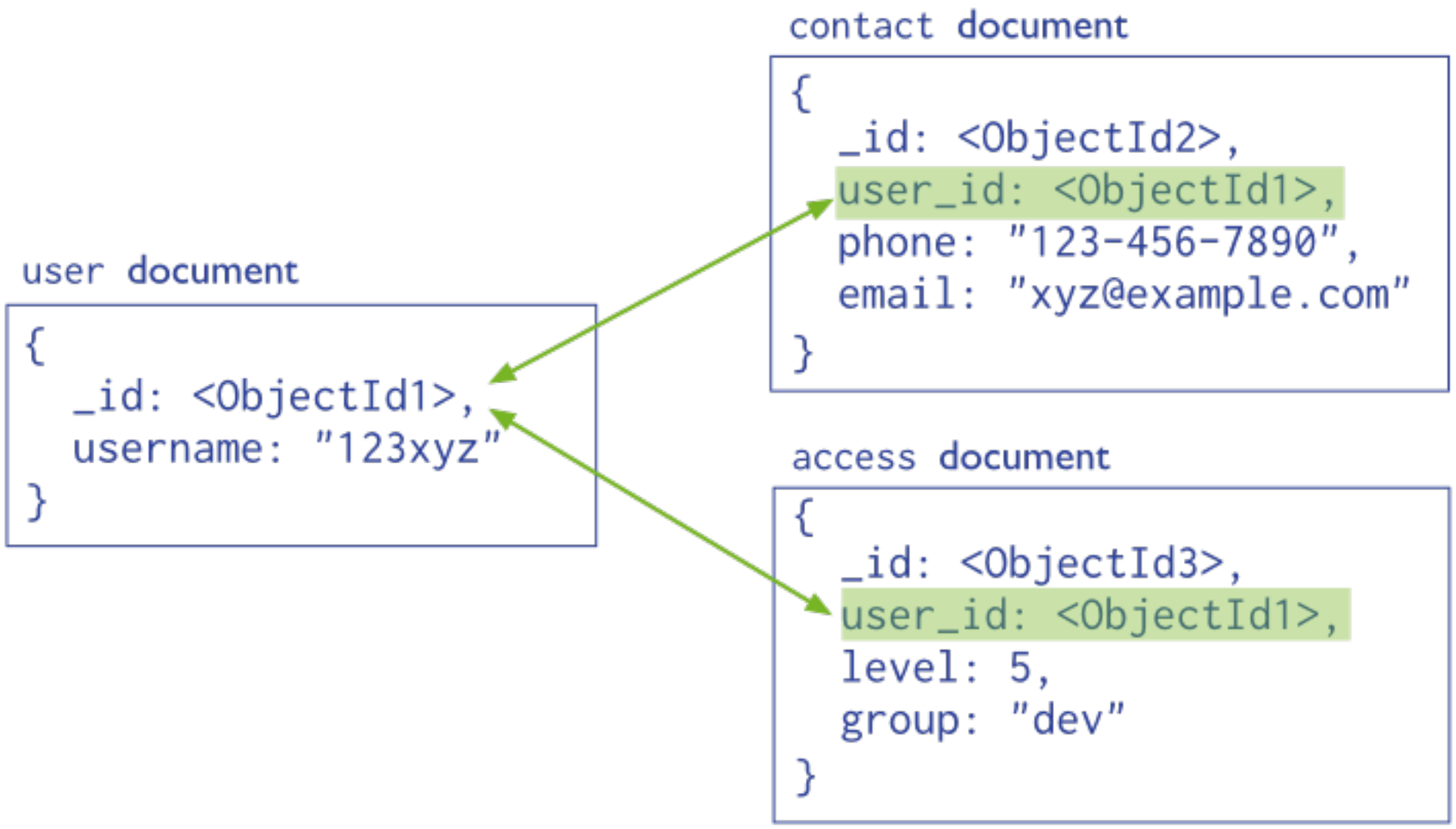
Fernando Ortega (fernando.ortega@upm.es)

Relaciones entre documentos

- ¿Cómo se representan las relaciones entre los datos?
- Existen dos formas de hacerlo:
  - Referencias a otros documentos
  - Subdocumentos
- Se permite (y aconseja) duplicar información

Fernando Ortega (fernando.ortega@upm.es)

Modelo normalizado



Modelo con subdocumentos

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
             phone: "123-456-7890",
             email: "xyz@example.com"
           },
  access: {
             level: 5,
             group: "dev"
           }
}
```

Embedded sub-document

Embedded sub-document

Relaciones 1:1

Normalized data model

```
{
    _id: "joe",
    name: "Joe Bookreader"
}
```

```
{
    patron_id: "joe",
    street: "123 Fake Street",
    city: "Faketon",
    state: "MA",
    zip: "12345"
}
```

Embedded data model

```
{
    _id: "joe",
    name: "Joe Bookreader",
    address: {
        street: "123 Fake Street",
        city: "Faketon",
        state: "MA",
        zip: "12345"
    }
}
```

- Con el modelo basado en subdocumentos la aplicación puede recuperar toda la información del documento con una sola query

Relaciones 1:N (subdocumentos)

Normalized data model

```
{
    _id: "joe",
    name: "Joe Bookreader"
}
```

```
{
    patron_id: "joe",
    street: "123 Fake Street",
    city: "Faketon",
    state: "MA",
    zip: "12345"
}

{
    patron_id: "joe",
    street: "1 Some Other Street",
    city: "Boston",
    state: "MA",
    zip: "12345"
}
```

Embedded data model

```
{
    _id: "joe",
    name: "Joe Bookreader",
    addresses: [
        {
            street: "123 Fake Street",
            city: "Faketon",
            state: "MA",
            zip: "12345"
        },
        {
            street: "1 Some Other Street",
            city: "Boston",
            state: "MA",
            zip: "12345"
        }
    ]
}
```

Relaciones 1:N (referencias)

Embedded data model

```
{
    title: "MongoDB: The Definitive Guide",
    author: [ “K. Chodorow", “M. Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher: {
                name: "O'Reilly Media",
                founded: 1980,
                location: "CA"
              }
}

{
    title: "50 Tips and Tricks for MongoDB",
    author: “K. Chodorow",
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English",
    publisher: {
                name: "O'Reilly Media",
                founded: 1980,
                location: "CA"
              }
}
```

Referenced data model

```
{
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA",
    books: [12346789, 234567890, ...]
}

{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ “K. Chodorow", “M. Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English"
}

{
    _id: 234567890,
    title: "50 Tips and Tricks for MongoDB",
    author: “K. Chodorow",
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English"
}
```

¿Qué solución es mejor?

- La clave cuando modelamos es balancear:
  - Las necesidades de la aplicación
  - El rendimiento
  - Las consultas que realizamos a los datos
- El modelo de datos está altamente relacionado con el uso que hacemos de los datos

Fernando Ortega (fernando.ortega@upm.es)

Ejemplo

- Sistema de votación de películas
- Disponemos de:
  - Usuarios
  - Películas
- Cada usuario puede votar tantas películas como desee

Ejemplo

- Modelo normalizado:

Movies

```
{
    _id: 111111,
    title: “Star Wars”,
    director: “George Lucas”,
    year: 1977
}

{
    _id: 222222,
    title: “La Vida de Brian”,
    director: “Terry Jones”,
    year: 1979
}
```

Users

```
{
    _id: 999999,
    nick: “juan007”,
    email: “juan007@gmail.com”
}

{
    _id: 888888,
    nick: “aruiz”,
    email: “aruiz@mtr.com”
}
```

Ratings

```
{
    user: 999999,
    movie: 111111,
    rating: 10
}

{
    user: 999999,
    movie: 222222,
    rating: 7
}

{
    user: 888888,
    movie: 111111,
    rating: 8
}
```

Ejemplo

- Pros:
  - Modelo normalizado
  - Sin información duplicada
    - Un cambio en una votación se actualiza al instante
- Contras:
  - Lento
  - No sigue la filosofía de MongoDB
  - Recuperar todos los votos de una película implica varias consultas

Ejemplo

- Modelo orientado a películas:

Movies

```
{
    _id: 111111,
    title: “Star Wars”,
    director: “George Lucas”,
    year: 1977,
    ratings: [
        {
            _id: 999999,
            rating: 10
        },
        {
            _id: 888888,
            rating: 8
        }
    ]
}
```

```
{
    _id: 222222,
    title: “La Vida de Brian”,
    director: “Terry Jones”,
    year: 1979,
    ratings: [
        {
            _id: 999999,
            rating: 7
        }
    ]
}
```

Users

```
{
    _id: 999999,
    nick: “juan007”,
    email: “juan007@gmail.com”
}

{
    _id: 888888,
    nick: “aruiz”,
    email: “aruiz@mtr.com”
}
```

Ejemplo

- Pros:
  - Acceso inmediato a los votos de cada película
- Contras:
  - Recuperar los votos de un usuario es lento
  - Actualizar un voto es lento
  - Si una película tiene muchos votos el tamaño del objeto en disco puede ser demasiado grande

Ejemplo

- Modelo orientado a usuarios:

Movies

```
{
      _id: 111111,
      title: “Star Wars”,
      director: “George Lucas”,
      year: 1977
}

{
      _id: 222222,
      title: “La Vida de Brian”,
      director: “Terry Jones”,
      year: 1979
}
```

Users

```
{
      _id: 999999,
      nick: “juan007”,
      email: “juan007@gmail.com”
      ratings: [
            {
                  _id: 111111,
                  title: “Star Wars”,
                  director: “George Lucas”,
                  year: 1977,
                  rating: 10
            },
            {
                  _id: 222222,
                  title: “La Vida de Brian”,
                  director: “Terry Jones”,
                  year: 1979,
                  rating: 7
            }
      ]
}
```

```
{
      id: 888888,
      _id: 888888,
      nick: “aruiz”,
      email: “aruiz@mtr.com”,
      ratings: [
            {
                  _id: 111111,
                  title: “Star Wars”,
                  director: “George Lucas”,
                  year: 1977,
                  rating: 8
            }
      ]
}
```

Ejemplo

- Pros:
  - Acceso inmediato a los votos del usuario
  - Acceso inmediato a las fichas de las películas votadas por el usuario
- Contras:
  - Duplica información
  - El objeto usuario puede ser muy grande si vota muchas películas
  - Un cambio en una ficha de una película implica actualizar información en los usuarios

Ejemplo

- Modelo mixto:

Users

```
{
        _id: 999999,
        nick: “juan007”,
        email: “juan007@gmail.com”
        ratings: [
             {
                  _id: 111111,
                  title: “Star Wars”,
                  director: “George Lucas”,
                  year: 1977,
                  rating: 10
             },
             {
                  _id: 222222,
                  title: “La Vida de Brian”,
                  director: “Terry Jones”,
                  year: 1979,
                  rating: 7
             }
        ]
}
```

```
{
        _id: 888888,
        nick: “aruiz”,
        email: “aruiz@mtr.com”,
        ratings: [
             {
                  _id: 111111,
                  title: “Star Wars”,
                  director: “George Lucas”,
                  year: 1977,
                  rating: 8
             }
        ]
}
```

Ejemplo

Movies

```
{
     _id: 111111,
     title: “Star Wars”,
     director: “George Lucas”,
     year: 1977,
     ratings: [
          {
               _id: 999999,
               nick: “juan007”,
               email: “juan007@gmail.com”
               rating: 10
          },
          {
               _id: 888888,
               nick: “aruiz”,
               email: “aruiz@mtr.com”,
               rating: 8
          }
     ]
}
```

```
{
     _id: 222222,
     title: “La Vida de Brian”,
     director: “Terry Jones”,
     year: 1979,
     ratings: [
          {
               _id: 999999,
               nick: “juan007”,
               email: “juan007@gmail.com”
               rating: 7
          }
     ]
}
```

Ejemplo

- Pros:
  - Acceso inmediato a la información de los votos de las películas
  - Acceso inmediato a la información de los votos de los usuarios
- Contras:
  - Mucha información duplicada
  - Objetos muy grandes

Ejemplo

- Debemos responder a las siguientes preguntas:
  - ¿Es frecuente actualizar los votos?
  - ¿Es necesario conocer quién votó cada película?
  - ¿Cada cuanto cambiamos la ficha de una película?
  - ¿Puede un usuario modificar su nick?
  - …

Aspectos clave

- MongoDB es flexible
- No existen normas para modelar la base de datos
  - Solamente existen una serie de buenos consejos
- Debemos pensar en el uso de los datos
- Se puede (y se aconseja) duplicar información

Operaciones

- MongoDB ofrece soporte para:
  - Escritura (Create)
  - Lectura (Read)
  - Modificación (Update)
  - Borrado (Delete)

Lectura

```
db.movies.find( {title: “Toy Story”} )

db.movies.find( {year: {$lt: 1980}} )

db.movies.find( {“director.surname”: “Smith”} )

db.movies.find( {genres: {$in: [ “Action” , “Comedy” ]}} )

db.movies.find().sort( {year: 1} )

db.users.find( {gender: “female”, age:{$gte:20, $lt:30}} )

db.users.find( {name: /Joh?n/} )
```

Escritura

```
db.coches.insert(
    {model:'Audi A5', km:28000, year:2011}
  )

db.coches.save(
    {_id:700, model:'Toyota Auris', km:10000, year:2014}
  )
```

ObjectId

- Tipo de datos de 12 bytes especial que garantiza ser único en cada colección
- Se genera en base a:
  - Timestamp
  - ID de la máquina
  - ID del proceso
  - Contador incremental local del proceso
- MongoDB usa valores de ObjectId como tipo por defecto en el campo _id
- Los valores del ObjectID son únicos

```
ObjectId("507f1f77bcf86cd799439011")
```


Fernando Ortega (fernando.ortega@upm.es)

Actualizaciones

```
db.users.update(                          ← collection
   { age: { $gt: 18 } },                  ← update criteria
   { $set: { status: "A" } },             ← update action
   { multi: true }                        ← update option
)
```

```
db.coches.update(
   { model: 'Toyota Auris' },
   { $inc: { km:1000 } },
   { multi: true }
)
```

Borrado

```
db.users.remove(          ← collection
   { status: "D" }        ← remove criteria
)
```

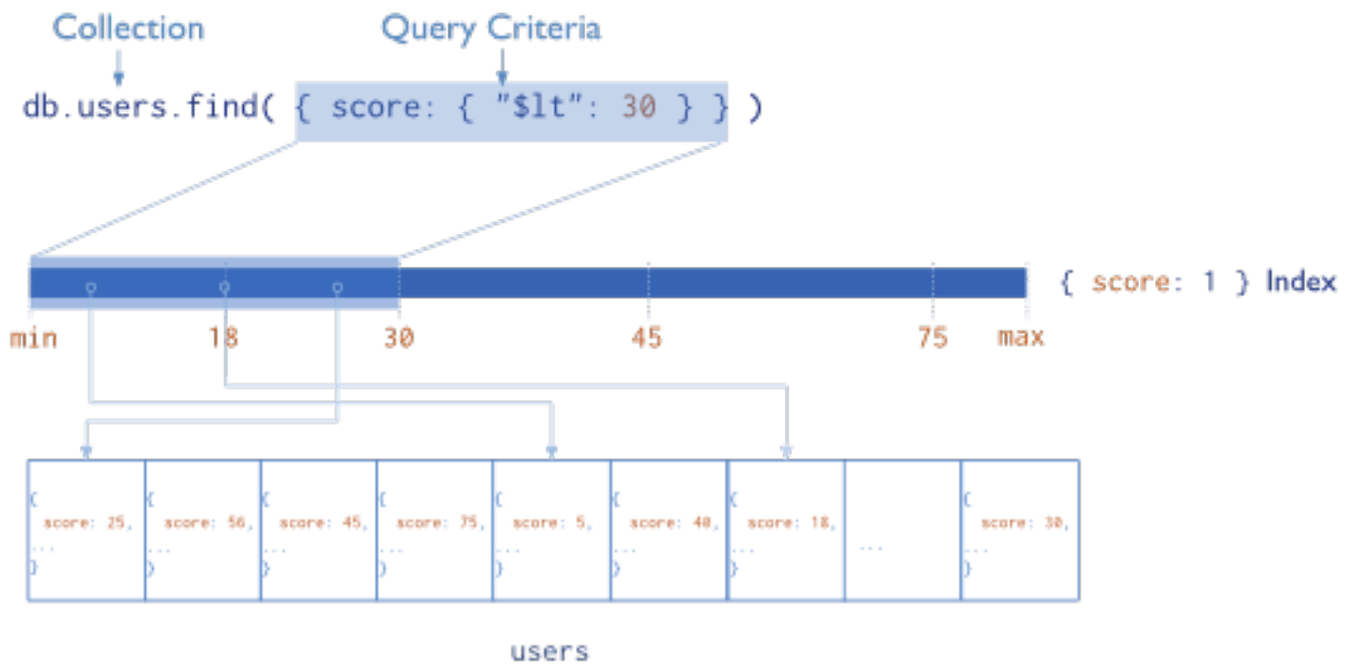
```
db.coches.remove( {
  $or : [
    { year: { $lt: 2010 } },
    { km: { $gt: 40000} }
  ]
} )
```

Características

¿Qué hace a MongoDB diferente?

Índices

- Los índices incrementan el rendimiento de las operaciones de lectura más frecuentes
  - Sin los índices habría que recorrer todos los documentos para encontrar los que coincidan con la consulta realizada



Fernando Ortega (fernando.ortega@upm.es)

Índices

Default _id	All MongoDB collections have an index on the _id field that exists by default
Single Field	MongoDB supports user-defined indexes on a single field of a document
Compound Index	MongoDB supports user-defined indexes on multiple fields
Multikey Index	MongoDB uses multikey indexes to index the content stored in arrays. If you index a field that holds an array value, MongoDB creates separate index entries for every element of the array
Geospatial Index	MongoDB provides two special indexes: 2d indexes that uses planar geometry when returning results and 2sphere indexes that use spherical geometry to return results
Text Indexes	MongoDB provides a text index type that supports searching for string content in a collection. These text indexes do not store language-specific stop words (e.g. “the”, “a”, “or”) and stem the words in a collection to only store root words
Hashed Indexes	MongoDB provides a hashed index type, which indexes the hash of the value of a field

Replica Set

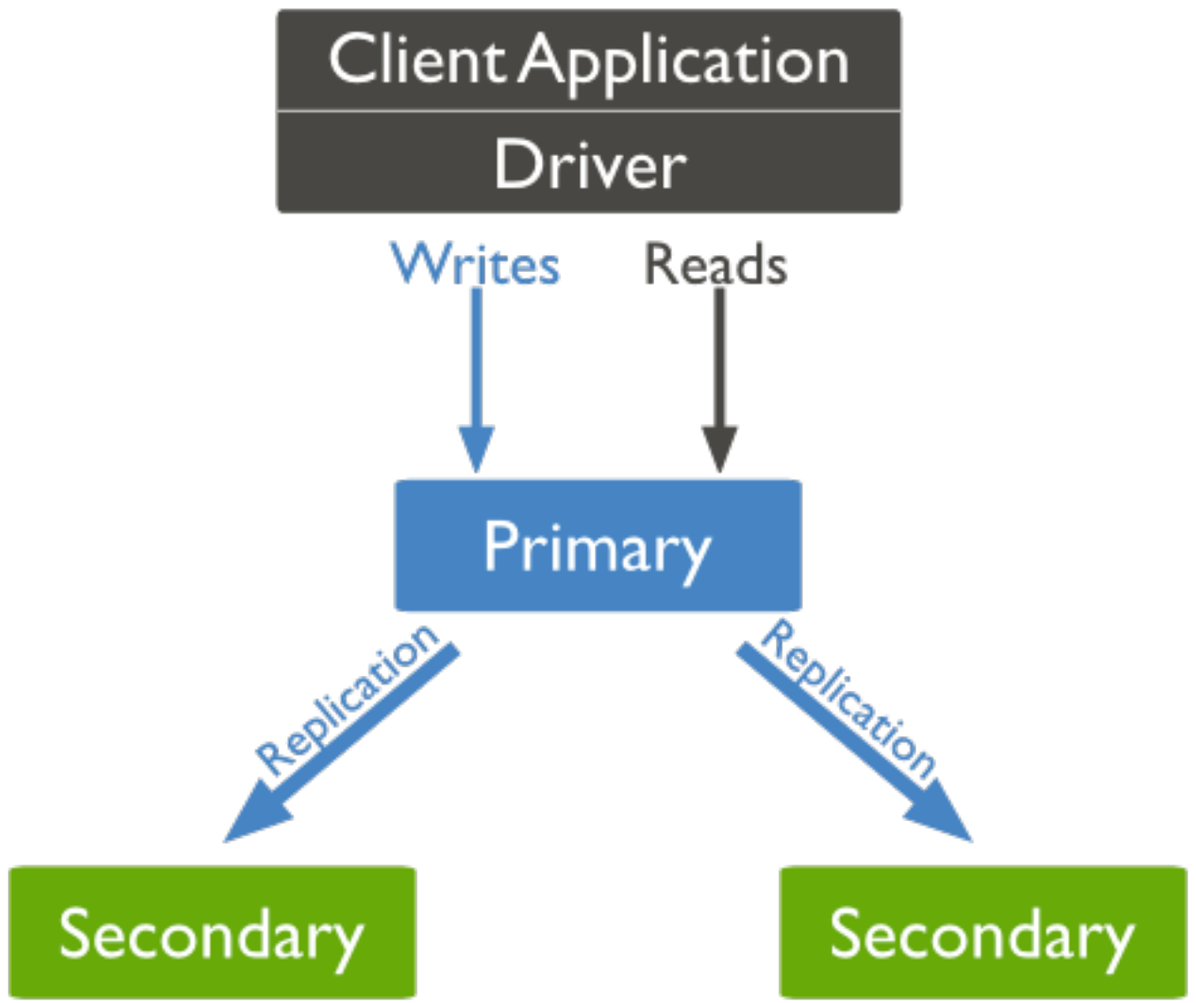
- Las replicas permiten sincronizar datos entre múltiples servidores

- Proporciona redundancia e incrementa la disponibilidad y la tolerancia a fallos

- Las replicas pueden mejorar el rendimiento de las lecturas

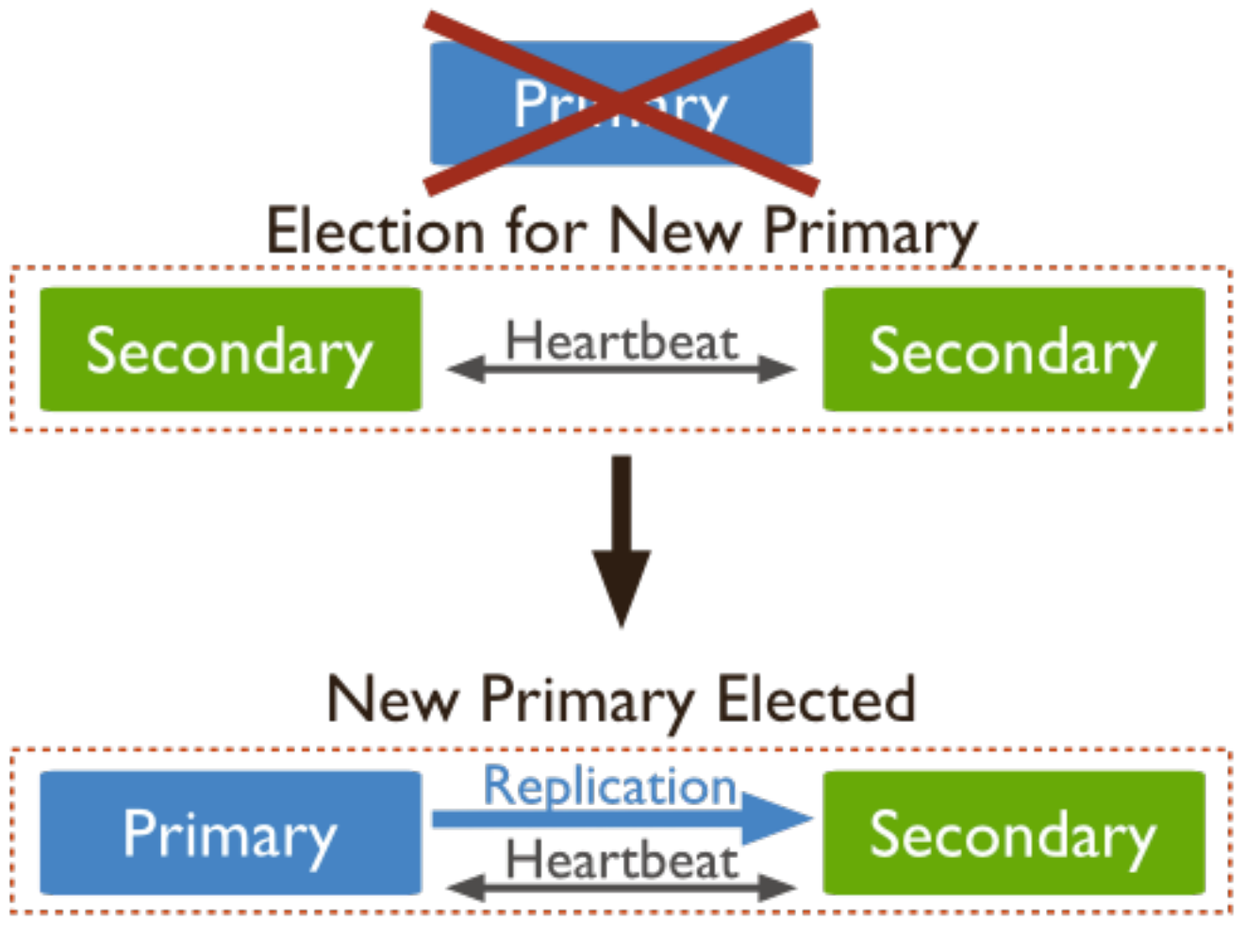
Replica Set

- El primario acepta las operaciones de los clientes
- Los secundarios replican las operaciones del primario en sus datos



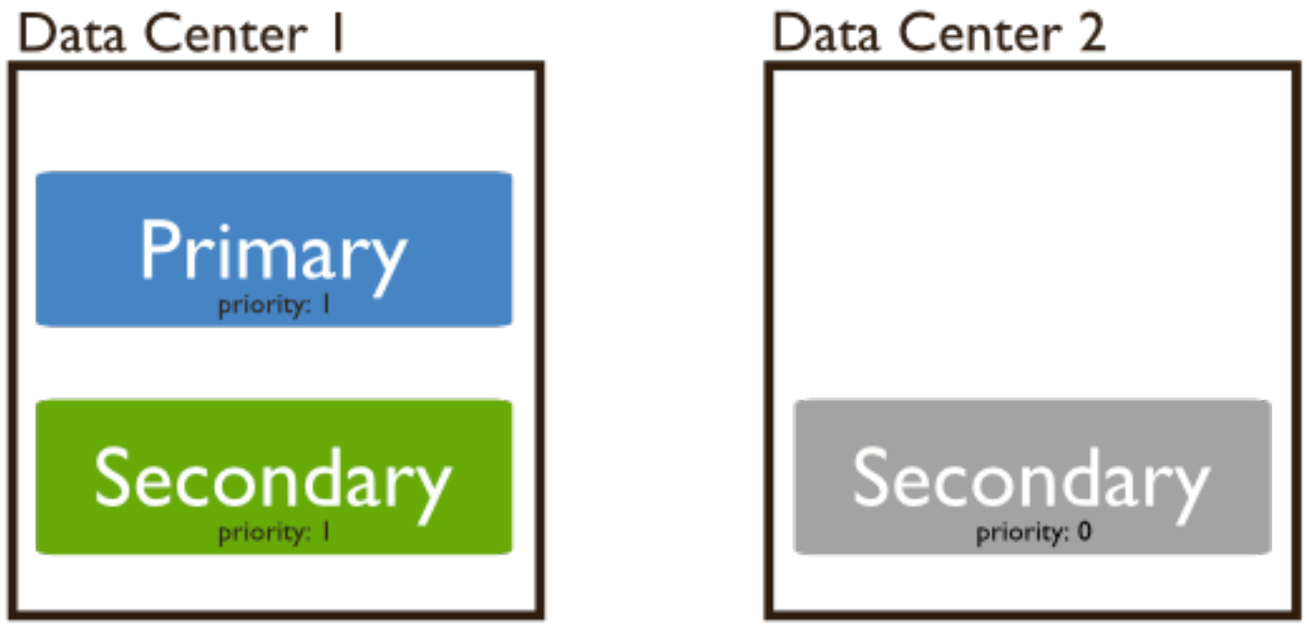
Replica Set

- Cuando el primario no se comunica con los otros miembros durante más de 10 segundos, el conjunto de replicas busca un nuevo primario. El secundario que recibe más votos se convierte en primario.



Replica Set

- Priority 0 Replica Set Members
  - A priority 0 member is a secondary that cannot become primary

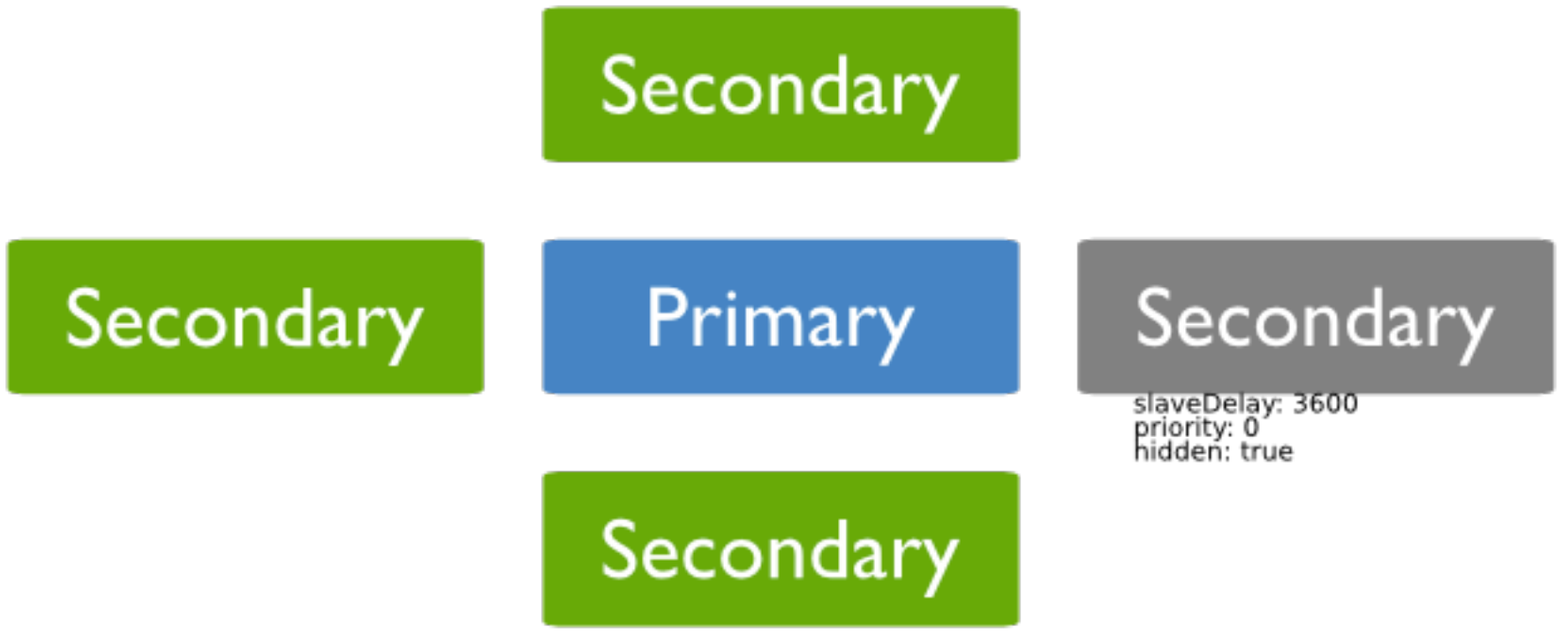


Replica Set

- Hidden Replica Set Members
  - A hidden member maintains a copy of the primary's data set but is invisible to client applications
  - Hidden members must always be priority 0 members and so cannot become primary

Replica Set

- Delayed Replica Set Members
  - Delayed members contain copies of a replica set's data set
  - A delayed member's data set reflects an earlier, or delayed, state of the set
  - Must be priority 0 and shroud be hidden members



Replica Set

- Replica Set puede mejorar el rendimiento de las lecturas
- MongoDB soporta 5 modos de lectura:

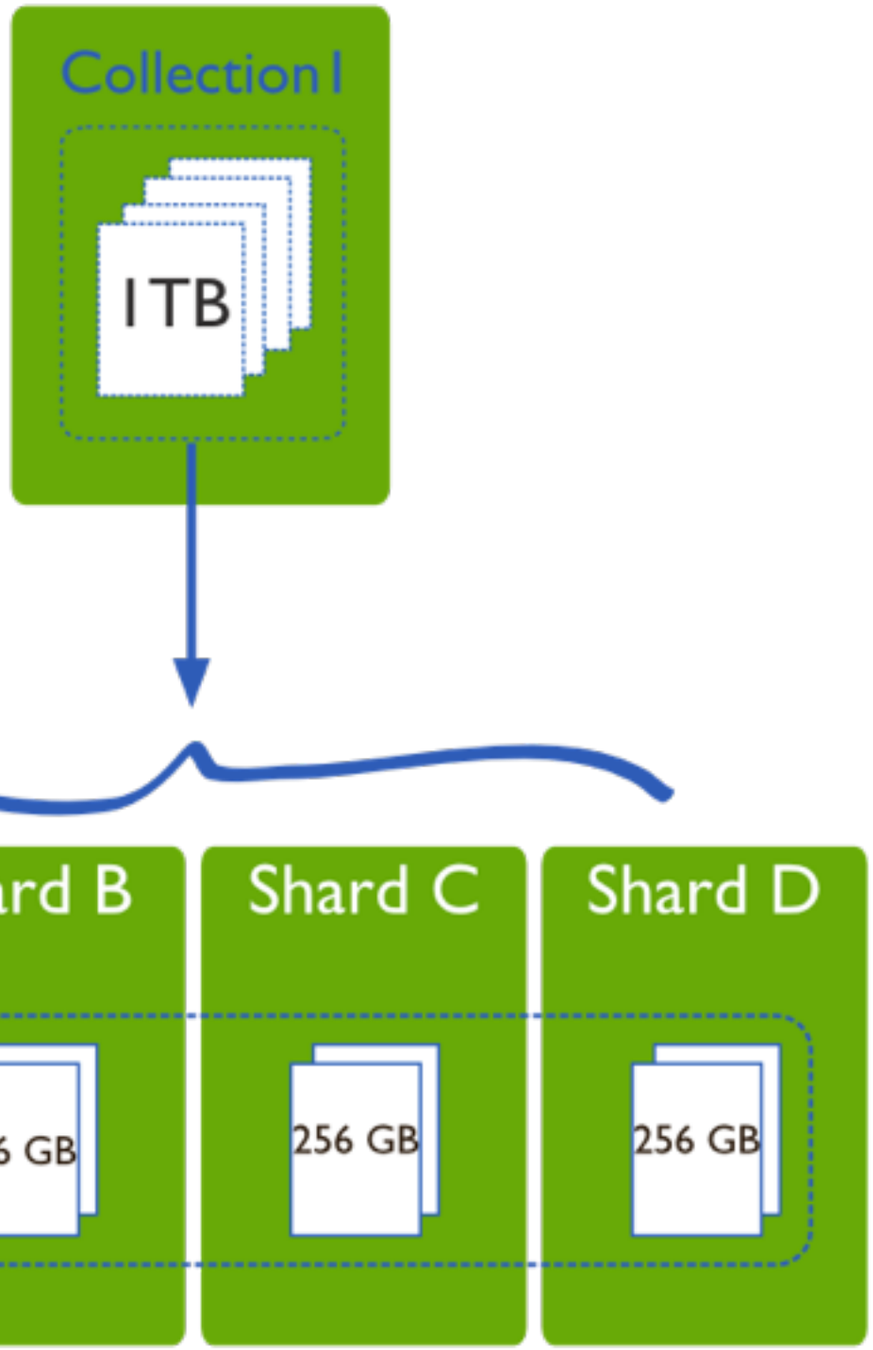
primary	Default mode. All operations read from the current replica set primary.
primaryPreferred	In most situations, operations read from the primary but if it is unavailable, operations read from secondary members.
secondary	All operations read from the secondary members of the replica set.
secondaryPreferred	In most situations, operations read from secondary members but if no secondary members are available, operations read from the primary.
nearest	Operations read from member of the replica set with the least network latency, irrespective of the member's type.

Sharding

- Sharding es una forma de guardar datos en diferentes máquinas
- MongoDB utiliza sharding para sistemas con conjuntos de datos enormes y mucha carga de entrada-salida
- Existen dos formas de escalar una base de datos:
  - La escalabilidad vertical añade más CPU y memoria para mejorar la capacidad del sistema
  - Sharding, o escalabilidad horizontal, divide los datos y los distribuyen entre different servidores (shards). Cada shard es una base de datos independiente y todos los shards juntos forman una base de datos.

Fernando Ortega (fernando.ortega@upm.es)

Sharding


Collection I
1 TB
Shard A
Collection I
256 GB
Shard B
256 GB
Shard C
256 GB
Shard D
256 GB

Sharding

- Sharding abordar el reto de la escalabilidad para soportar altas tasas de entreda/salidad en grades conjuntos de datos:

- Sharding reduce el número de operaciones que maneja cada shard.

- Sharding reduce la cantidad de información que cada shard tiene que almacenar.

Fernando Ortega (fernando.ortega@upm.es)

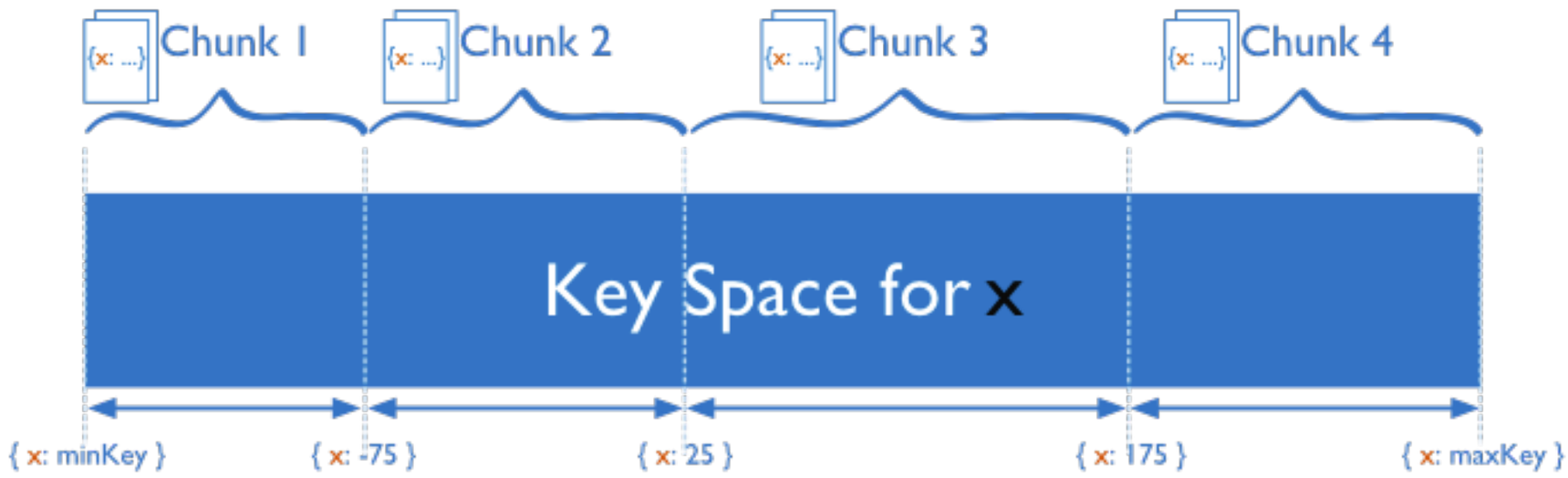
Sharding

- MongoDB distribuye los datos a nivel de colección
- Sharding distribuye los datos basándose en una shard key
- Una shard key es un campo indexable que existe en cualquier documento de la colección
- MongoDB divide los valores de la shard key en chunks y distribuye los chunks entre los shards
- Existen dos formas de dividir la shard key: range based partitioning o hash based partitioning

Fernando Ortega (fernando.ortega@upm.es)

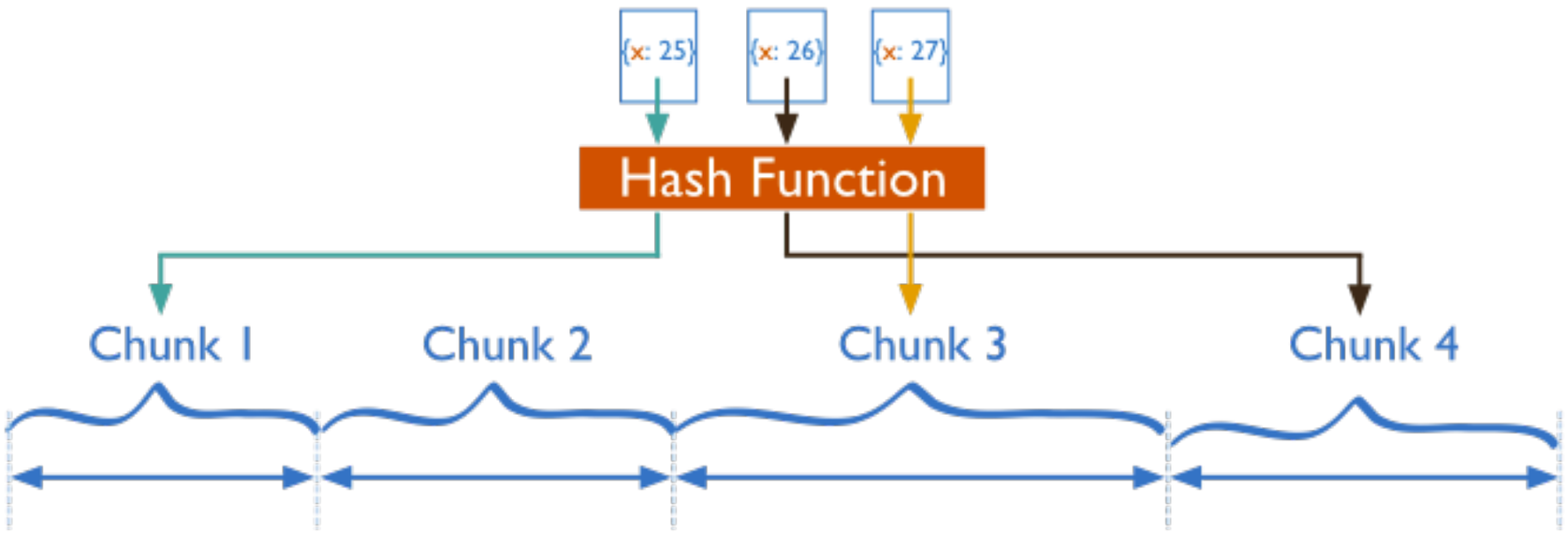
Sharding

- Range-based sharding: MongoDB divide los datos en rangos basándose en los valores que toma la shard key.



Sharding

- Hash based partitioning: MongoDB calcula una función resumen de la shard key y usa ese valor para elegir el chunk.



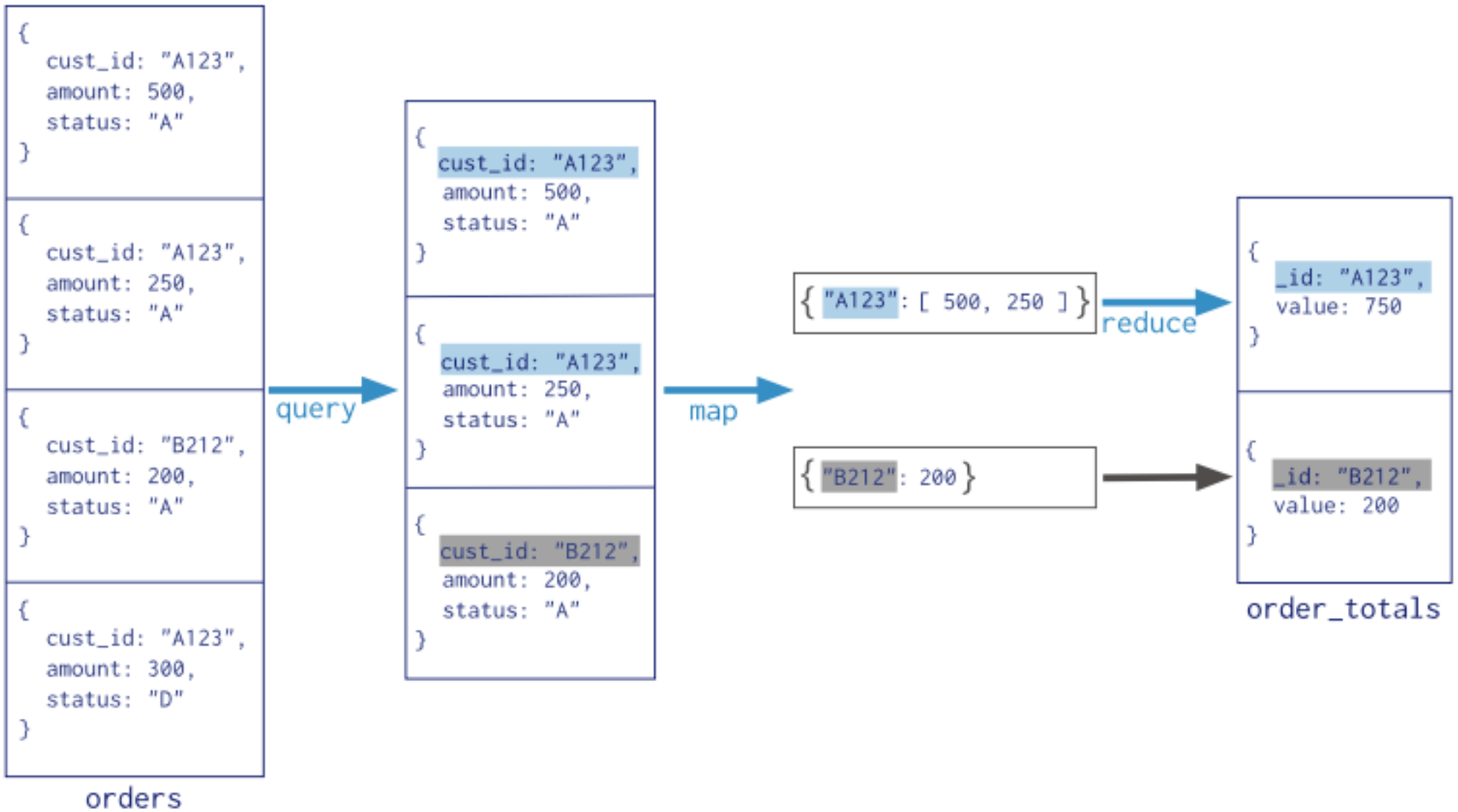
MongoDB orientado a grafos

Importando relaciones al mundo no relacional

Agregación de resultados

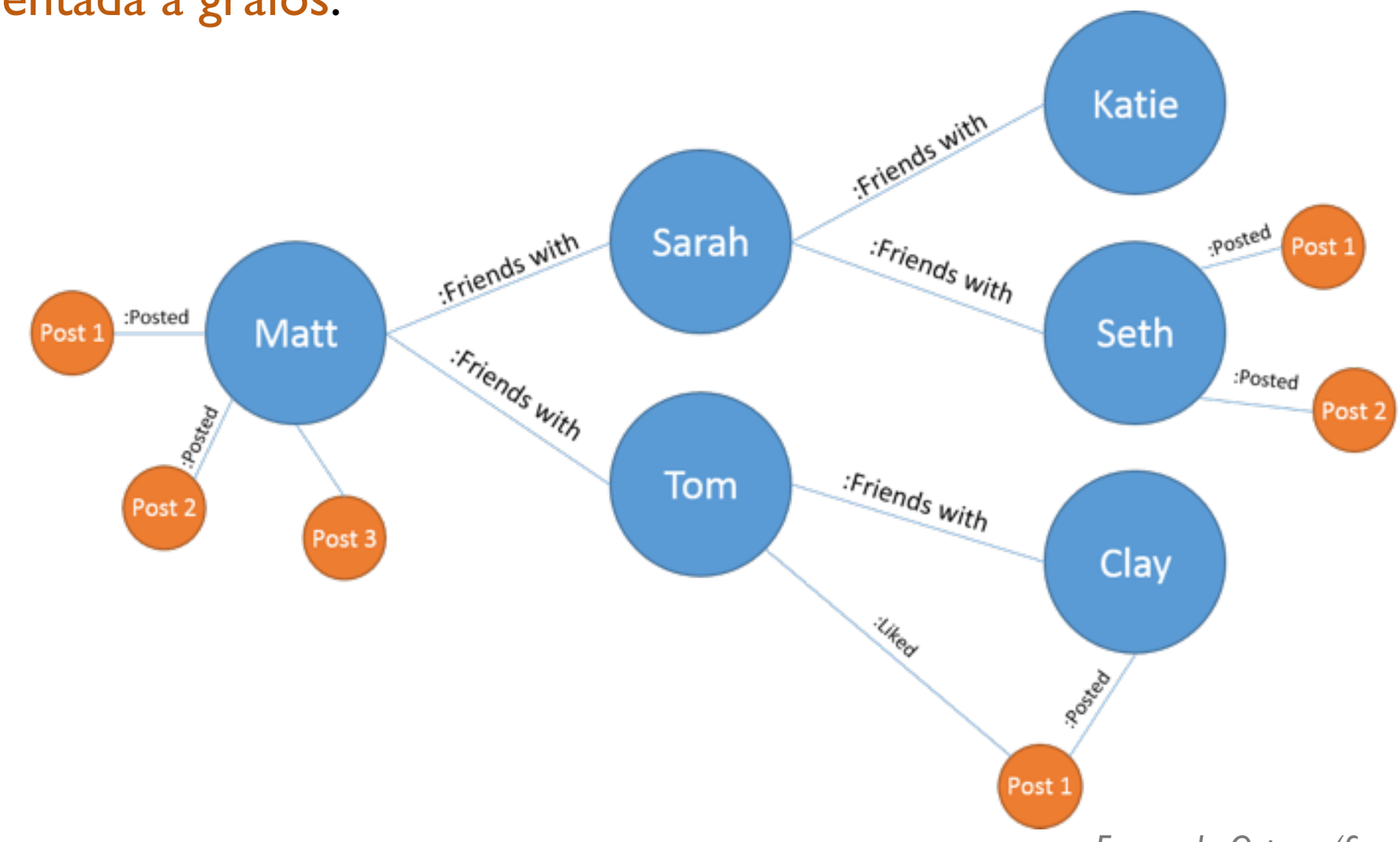
- MongoDB dispone de un motor de consultas avanzado denominado aggregation pipeline.
- Se van enlazando etapas para transformar los datos almacenados.
  - Por ejemplo: map-reduce

Ejemplo map-reduce



$graphLookUp

- En la versión 3.4 se ha añadido la operación $graphLookUp para el pipeline.
- Permite trabajar con MongoDB como si se tratara de una base de datos orientada a grafos.



Ejemplo de $graphLookUp

airports

```
{ "_id" : 0, "airport" : "JFK", "connects" : [ "BOS", "ORD" ] }
{ "_id" : 1, "airport" : "BOS", "connects" : [ "JFK", "PWM" ] }
{ "_id" : 2, "airport" : "ORD", "connects" : [ "JFK" ] }
{ "_id" : 3, "airport" : "PWM", "connects" : [ "BOS", "LHR" ] }
{ "_id" : 4, "airport" : "LHR", "connects" : [ "PWM" ] }
```

travelers

```
{ "_id" : 1, "name" : "Dev", "nearestAirport" : "JFK" }
{ "_id" : 2, "name" : "Eliot", "nearestAirport" : "JFK" }
{ "_id" : 3, "name" : "Jeff", "nearestAirport" : "BOS" }
```

```
db.travelers.aggregate( [
  {
    $graphLookup: {
      from: "airports",
      startWith: "$nearestAirport",
      connectFromField: "connects",
      connectToField: "airport",
      maxDepth: 2,
      depthField: "numConnections",
      as: "destinations"
    }
  }
] )
```

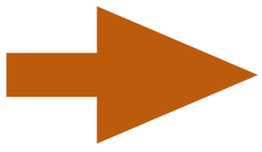
Ejemplo de $graphLookUp

```
{
  "_id" : 1,
  "name" : "Dev",
  "nearestAirport" : "JFK",
  "destinations" : [
    { "_id" : 3,
      "airport" : "PWM",
      "connects" : [ "BOS", "LHR" ],
      "numConnections" : NumberLong(2) },
    { "_id" : 2,
      "airport" : "ORD",
      "connects" : [ "JFK" ],
      "numConnections" : NumberLong(1) },
    { "_id" : 1,
      "airport" : "BOS",
      "connects" : [ "JFK", "PWM" ],
      "numConnections" : NumberLong(1) },
    { "_id" : 0,
      "airport" : "JFK",
      "connects" : [ "BOS", "ORD" ],
      "numConnections" : NumberLong(0) }
  ]
}
```

$graphLoohUp en el pipeline

- La operación $graphLookUp nos permite formar pipelines más complejos.

```
db.people.aggregate( [
  { $match: { "name": "Tanya Jordan" } },
  { $graphLookup: {
      from: "people",
      startWith: "$friends",
      connectFromField: "friends",
      connectToField: "name",
      as: "golfers",
      restrictSearchWithMatch: { "hobbies" : "golf" }
    }
  },
  { $project: {
      "name": 1,
      "friends": 1,
      "connections who play golf": "$golfers.name"
    }
  }
] )
```



```
{
  "_id" : 1,
  "name" : "Tanya Jordan",
  "friends" : [
    "Shirley Soto",
    "Terry Hawkins",
    "Carole Hale"
  ],
  "connections who play golf" : [
    "Joseph Dennis",
    "Tanya Jordan",
    "Angelo Ward",
    "Carole Hale"
  ]
}
```

Búsquedas geoespaciales

MongoDB fue diseñado para trabajar con mapas

Tipos de superficies

- MongoDB esta preparado para trabajar con las siguientes superficies:
  - Esferas: funcionan mediante latitud y longitud
  - Planos: superficies bidimensionales
- Se almacenan en documentos del tipo GeoJSON
- Se utilizan índices para optimizar las operaciones sobre campos geoespaciales
  - 2dsphere index para esferas
  - 2d index para planos

Fernando Ortega (fernando.ortega@upm.es)

GeoJSON

- Permiten definir las siguientes localizaciones:
  - Point
  - LineString
  - Polygon
  - MultiPoint
  - MultiLineString
  - MultiPolygon

```
{
   type: "<GeoJSON type>" ,
   coordinates: <coordinates>
}
```

```
{
  type: "Polygon",
  coordinates: [ [ [ 0 , 0 ] , [ 3 , 6 ] , [ 6 , 1 ] , [ 0 , 0  ] ] ]
}
```

Operaciones geoespaciales

- $geoWithin: permite comprobar si una localización está completamente incluida dentro de otra
  - Funciona tanto para planos como esferas y no require índices (se recomiendan)
- $geoIntersects: comprueba si una localización está parcialmente incluida dentro de otra
  - Sólo funciona con esferas y requiere indices
- $near: permite comprobar los puntos más cercanos a uno dado
  - Funciona tanto para planos como esferas y require indices

Fernando Ortega (fernando.ortega@upm.es)

$near

```
{
   <location field>: {
     $near: {
       $geometry: {
          type: "Point" ,
          coordinates: [ <longitude> , <latitude> ]
       },
       $maxDistance: <distance in meters>,
       $minDistance: <distance in meters>
     }
   }
}
```

Ejemplo de $near

```
db.places.find(
   {
     location:
       { $near :
         {
           $geometry: { type: "Point",  coordinates: [ -73.9667, 40.78 ] },
           $minDistance: 1000,
           $maxDistance: 5000
         }
       }
   }
)
```

Ventajas del operador $near

- Permite encontrar rápidamente puntos cercanos a uno dado
- Permite controlar la distancia máxima y minima
- Permite ordenar los resultados por distancia
  - Dame los usuario más cercanos a mi
- Puede combinarse con otros operadores

Muchas gracias

¿Preguntas?