



# MongoDB Enterprise 4.0.10

CURSO FUNDAMENTOS DE MONGODB

Instructor: Carlos Carreño  
Email: ccarrenovi@Gmail.com

# Cursores

- Manipulando resultados
- Collections de resultados
- Batches
- BulkWrite
- Indices
- Modelado de Datos

# Cursores Introducción

- Cada vez que llamamos al método `find` de una colección el mismo no retorna un objeto de la clase `Cursor`.
- Si no asignamos el valor a una variable en el shell de MongoDB luego se muestran los documentos recuperados y se nos pide que confirmemos cada vez que se muestran 20.

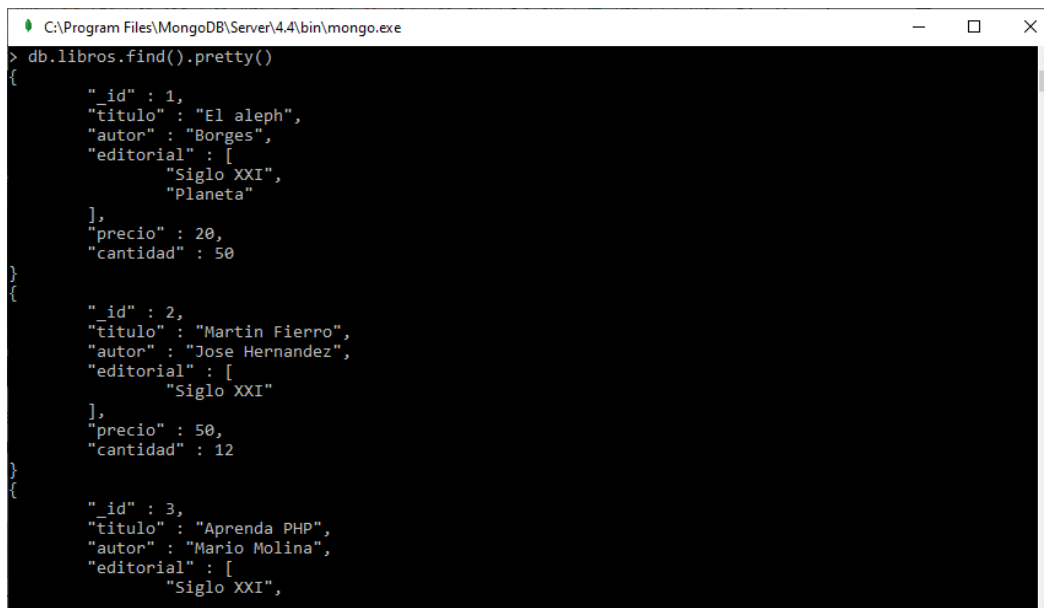
# Manipulando resultados

- A partir del cursor que retorna el método 'find' llamamos al método 'sort' de la clase Cursor y como condición indicamos por el campo que queremos ordenar (si pasamos un 1 se ordena en forma ascendente y si pasamos un -1 se ordena en forma descendente)

```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
> db.libros.find().sort({titulo:1})
{ "_id" : 3, "titulo" : "Aprenda PHP", "autor" : "Mario Molina", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 50, "cantidad" : 20 }
{ "_id" : 1, "titulo" : "El aleph", "autor" : "Borges", "editorial" : [ "Siglo XXI", "Planeta" ], "precio" : 20, "cantidad" : 50 }
{ "_id" : 4, "titulo" : "Java en 10 minutos", "editorial" : [ "Siglo XXI" ], "precio" : 45, "cantidad" : 1 }
{ "_id" : 2, "titulo" : "Martin Fierro", "autor" : "Jose Hernandez", "editorial" : [ "Siglo XXI" ], "precio" : 50, "cantidad" : 12 }
```

# Collections de resultados

- Otro método de la clase Cursor que nos puede ayudar cuando ejecutamos comandos desde el shell de MongoDB es 'pretty', el mismo tiene por objetivo mostrarnos los datos del cursor en forma más legible



```
C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
> db.libros.find().pretty()
{
  "_id" : 1,
  "titulo" : "El aleph",
  "autor" : "Borges",
  "editorial" : [
    "Siglo XXI",
    "Planeta"
  ],
  "precio" : 20,
  "cantidad" : 50
},
{
  "_id" : 2,
  "titulo" : "Martin Fierro",
  "autor" : "Jose Hernandez",
  "editorial" : [
    "Siglo XXI"
  ],
  "precio" : 50,
  "cantidad" : 12
},
{
  "_id" : 3,
  "titulo" : "Aprenda PHP",
  "autor" : "Mario Molina",
  "editorial" : [
    "Siglo XXI",

```

# Batches

- Podemos limitar la cantidad de documentos que retorna MongoDB  
`db.inventory.find().batchSize(10)`

# BulkWrite

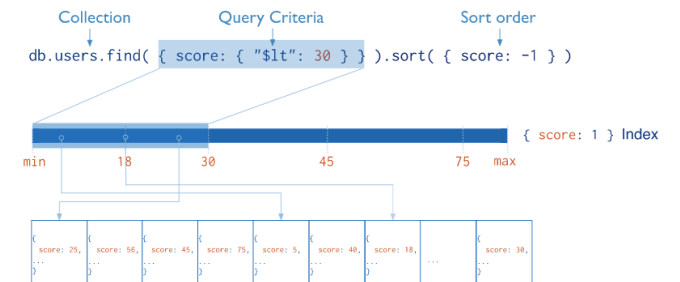
- Permite ejecutar múltiples operaciones de escritura controlando el orden de ejecución.

```
1 try {
2     db.characters.bulkWrite([
3         { insertOne: { "document": { "_id": 4, "char": "Dithras", "class": "barbarian", "lvl": 4 } } },
4         { insertOne: { "document": { "_id": 5, "char": "Taeln", "class": "fighter", "lvl": 3 } } },
5         { updateOne : {
6             "filter" : { "char" : "Eldon" },
7             "update" : { $set : { "status" : "Critical Injury" } }
8         } },
9         { deleteOne : { "filter" : { "char" : "Brisbane"} } },
10        { replaceOne : {
11            "filter" : { "char" : "Meldane" },
12            "replacement" : { "char" : "Tanys", "class" : "oracle", "lvl": 4 }
13        } }
14    ]);
15 } catch (e) {
16     print(e);
17 }
```

```
> show collections
characters
libros
> db.characters.find();
{ "_id" : 4, "char" : "Dithras", "class" : "barbarian", "lvl" : 4 }
{ "_id" : 5, "char" : "Taeln", "class" : "fighter", "lvl" : 3 }
>
```

# Uso de índices

- los índices nos permiten optimizar las consultas que hacemos a nuestra base de datos.
- Los índices son “estructuras de datos especiales que almacenan una pequeña porción de los datos de la colección de tal forma que sea fácil de recorrer. El índice almacena el valor de un campo específico o de un conjunto de campos, ordenados por el valor de dichos campos. El orden de cada una de las entradas del índice permite realizar consultas de igualdad o rango de manera eficiente.”





# Creación de Índices

- Para crear un nuevo índice en nuestra colección usaremos el método `createIndex()` que tiene la siguiente forma:

```
db.collection.createIndex( <key and index type specification>,  
<options> )
```

- Donde `key` representa un documento con los campos de la colección que queremos indexar, `type` hace referencia al tipo de índice que queremos crear, y `options` sirve para indicar con un documento los valores que controlan la creación del índice.

# Creación de índice en orden descendente

- Ejemplo: Para crear un índice por el campo id en tickers, usa el campo con el valor -1

```
db.tickers.createIndex({ id: -1 })
```

- Por defecto mongoDB crea un índice por el campo \_id
- El número puede ser 1 (ascendente) o -1 (descendente).

```
db.people.createIndex({name:1,age:-1});
```

# Índice con nombre

- Se puede asignar un nombre al índice

```
db.people.createIndex({name:-1},"nameDesc");
```

# Eliminando un índice con nombre

- Utiliza `dropIndex()` y el nombre para eliminar un índice nombrado.

```
db.people.dropIndex("nameDesc")
```

# Obtener información de los índices creados

- Usa la función `getIndexes()` en una colección para conocer que índices tiene la colección.

```
db.tickers.getIndexes()
```

# Eliminar índices

- Utiliza la función `dropIndex()`

```
db.tickers.dropIndex('<Nombre del Indice>')
```

# Tipos y opciones en índices

- Tipos

- **Campo individual:** usado para índices de un solo campo donde se facilita el ordenamiento de los documentos.
- **Índice compuesto:** permite crear índices sobre varios campos. El orden en el que está definido cada campo tiene importancia.
- **Índice multi-llave:** son aplicados a campos que tienen información contenida en arreglos. Al usar estos índices, MongoDB crea una entrada para cada uno de los elementos del arreglo.

- Opciones

- **Índices únicos:** con la propiedad unique se encarga de no permitir valores repetidos en los campos indexados.
- **Índices parciales:** solo mantiene un registro de los documentos que coinciden con las condiciones de filtro suministradas, lo cual permite reducir los requerimientos de memoria.
- **Índices con tiempo total de vida:** eliminan automáticamente los documentos de una colección luego de una duración de tiempo especificada.

# Índices Únicos

- Impide que las claves se repitan

```
db.people.createIndex({name:1,age:1},{unique:true})
```



# Índice Compuesto

- Son los índices que emplean más de una clave; no hay que indicarlo explícitamente.

```
db.people.createIndex({name:1,age:1})
```

# Índice en Arrays

- Se pueden crear índices sobre claves que contienen arrays; lo que hace Mongo es crear un valor en el índice para cada valor del array. Se llaman índices multikey.

```
db.people.createIndex({"comments":1});
```

# Índice en subdocumentos

- Se puede crear un índice sobre un subdocumento.

```
db.people.createIndex({"address.primary":1});
```

# Índices dispersos

- Los índices dispersos (sparse), son útiles cuando una clave aparece muy rara vez, es decir cuando la amplia mayoría de los documentos no contienen la clave. En ese caso indicar `sparse:true` hará que el índice sea más eficiente.

# Índices con caducidad

- Son los llamados TTL (Time To Live). Se especifican con la opción `{expireAfterSeconds:<numero de segundos>}` y sirven para borrar documentos de la colección, transcurrido cierto tiempo (el tiempo que pase desde que se ha indexado).

- Ejemplo:

```
db.products.createIndex( { "waiting": 1 }, { expireAfterSeconds: 3600 } )
```

- **Nota:** En el ejemplo, cada documento que se inserte con un ***waiting*** se borrará después de una hora. La llave ***waiting*** tiene que ser una llave de tipo date (ISODate).

# Creación de índice en background

- La opción `{background:true}` hace que el índice se cree sin detener la lectura/escritura. La ventaja que tienen es que, contrariamente a los índices normales, no bloquean la lectura/escritura en la base de datos.

# Plan de ejecución

- Permiten saber si se está usando un índice al ejecutar alguna de las siguientes funciones:
  - find
  - count
  - remove
  - update
  - group

# Modos de Explain

Explain se puede usar de tres modos

- ***executionStats***
- ***queryPlanner***
- ***allPlansExecution***



# Obteniendo información de la ejecución de una consulta

- Usa la función explain

```
db.tickers.find({id: 'bitcoin'}).explain('executionStats')
```

- Tendremos dos valores importantes:
  - “executionTimeMillis”: tiempo total de la ejecución de la consulta en milisegundos.
  - “totalDocsExamined”: número total de documentos analizados.

# Explain con queryPlanner

- El modo por defecto. Para ver el plan tenemos que incluir una llamada al método `explain()` despues del método que queremos probar.

```
db.people.find({name:"Yenny"}).explain();
```

- Podemos distinguir 4 partes dentro del plan:
  - `queryPlanner`: Muestra la query en formato interno (`parsedQuery`), así como el nombre de la base de datos y la colección.
  - `winningPlan`: Es seguramente la parte más importante. Es el plan escogido e Indica cómo ha organizado MongoDB la búsqueda de los documentos.
  - `rejectedPlans`: Indica otros planes de ejecución posibles que han sido desechados por ser menos eficientes
  - `serverInfo`: Lugar donde se ha desarrollado la consulta, y datos sobre la versión de mongo

# Explain con allPlanExecutions

- allPlansExecution muestra todos planes, tanto el ganador como los desechados es muy útil para decidir qué índice va mejor. Muestra un análisis de todos los posibles planes.

# Consideraciones en índices

- Agregar un nuevo índice **afecta a las operaciones de escritura**. Para colecciones que manejan gran número de operaciones de escritura, los índices afectan un poco el rendimiento puesto que con cada inserción de documentos se deben actualizar los índices.
- Cuando un índice está activo **consume espacio en disco y memoria**. Este consumo puede ser considerable por lo cual debe ser analizado y estudiado al momento de trabajar con índices.

# Creación del modelo de datos

- Los modelos de datos mas efectivos dependen del propósito de tu aplicación.
- En mongoDB es clave elegir entre las dos opciones para estructuras tus datos:
  - ***Modelo de Datos Embebidos***
  - ***Modelo de Datos Normalizado***

# Modelo de Datos Embebidos

- Con MongoDB, puede incrustar datos relacionados en una sola estructura o documento. Estos esquemas generalmente se conocen como modelos "desnormalizados".

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

Embedded sub-document

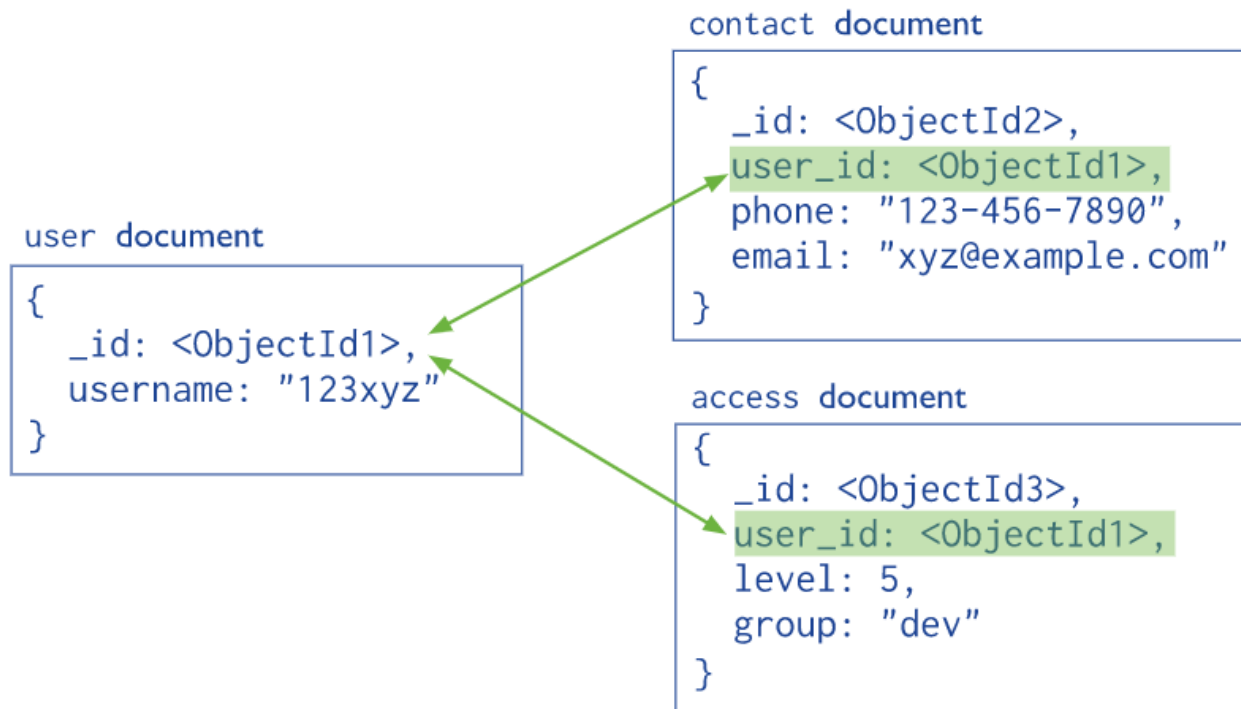
Embedded sub-document

# Cuando usar modelo de datos embebidos

- En general, use modelos de datos embebidos cuando:
  - Tienes relaciones "contiene" entre entidades.
  - Tienes relaciones uno a muchos entre entidades. En estas relaciones, los documentos "muchos" o secundarios siempre aparecen o se ven en el contexto de los documentos "uno" o principal.
- Los documentos en MongoDB deben ser más pequeños que el tamaño **máximo de documento BSON**.

# Modelo de Datos Normalizados

- Los modelos de datos normalizados describen relaciones utilizando referencias entre documentos.





# Modelo de Datos Normalizados

- En general, use modelos de datos normalizados:
  - Cuando la incrustación daría como resultado la duplicación de datos, pero no proporcionaría suficientes ventajas de rendimiento de lectura
  - Para superar las implicaciones de la duplicación.
  - Para representar relaciones de muchos a muchos más complejas.
  - Para modelar grandes conjuntos de datos jerárquicos.

# Patrón de modelado de relaciones uno a uno

- Con el modelo de datos embebido, la aplicación puede recuperar la información completa del usuario con una sola consulta.

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

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

Modelo normalizado

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

Modelo embebido

# Patrón de modelado de relaciones de uno a muchos

- Si los datos son altamente consultados considera el modelo embebido

```
{
  _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"
}
```

Modelo normalizado

```
{
  _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"
    }
  ]
}
```

Modelo embebido

# Patrón de modelado de uno a muchos con referencias

- Se intenta evitar la duplicación de los datos

```
{
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike 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 Developer",
  author: "Kristina Chodorow",
  published_date: ISODate("2011-05-06"),
  pages: 68,
  language: "English",
  publisher: {
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA"
  }
}
```

```
{
  name: "O'Reilly Media",
  founded: 1980,
  location: "CA",
  books: [123456789, 234567890, ...]
}

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

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

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

{
  _id: 123456789,
  title: "MongoDB: The Definitive Guide",
  author: [ "Kristina Chodorow", "Mike Dirolf" ],
  published_date: ISODate("2010-09-24"),
  pages: 216,
  language: "English",
  publisher_id: "oreilly"
}

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

# Modificación del modelo de datos

- El modelado para MongoDB debe considerar varios **factores operativos** que afectan el rendimiento de MongoDB y modificación del modelo de datos.
- Por ejemplo, diferentes modelos de datos pueden permitir consultas más eficientes, aumentar el rendimiento de las operaciones de inserción y **actualización**.

# Factores operacionales y modelo de datos

- Atomicity
- Sharding
- Indexes
- Large Number of Collections
- Collection Contains Large Number of Small Documents
- Storage Optimization for Small Documents
- Data Lifecycle Management

# Atomicity

- En MongoDB, una operación de escritura es atómica en el nivel de un solo documento, incluso si la operación modifica múltiples documentos incrustados dentro de un solo documento. Cuando una sola operación de escritura modifica varios documentos (por ejemplo, `db.collection.updateMany()`), la modificación de cada documento es atómica, pero la operación en su conjunto no es atómica.

# Sharding

- MongoDB utiliza la *fragmentación para proporcionar escalamiento* horizontal. Estos clústeres admiten implementaciones con grandes conjuntos de datos y operaciones de alto rendimiento. *Sharding permite a los usuarios particionar una colección* dentro de una base de datos para distribuir los documentos de la colección en varias instancias o fragmentos mongod.
- Para distribuir datos y tráfico de aplicaciones en una colección fragmentada, MongoDB usa *la clave de fragmento (shard key)*. Seleccionar la clave de fragmento adecuada tiene implicaciones significativas para el rendimiento y puede habilitar o prevenir el aislamiento de consultas y una mayor capacidad de escritura. Es importante considerar cuidadosamente el campo o los *campos que se utilizarán como clave de fragmento*.



# Indexes

- A medida que crea índices, considere los siguientes comportamientos de los índices:
  - Cada índice requiere al menos 8 kB de espacio de datos.
  - Agregar un índice tiene un impacto negativo en el rendimiento de las operaciones de escritura. Para las colecciones con una alta relación de escritura a lectura, los índices son caros, ya que cada inserción también debe actualizar cualquier índice.
  - Las colecciones con una alta relación lectura-escritura a menudo se benefician de índices adicionales. Los índices no afectan las operaciones de lectura no indexadas.
  - Cuando está activo, cada índice consume espacio en disco y memoria. Este uso puede ser significativo y debe rastrearse para la planificación de la capacidad, especialmente para las preocupaciones sobre el tamaño del conjunto de trabajo.

# Large Number of Collections

- Cuando utilice modelos que tengan una gran cantidad de colecciones, tenga en cuenta los siguientes comportamientos:
  - Cada colección tiene una cierta sobrecarga mínima de unos pocos kilobytes.
  - Cada índice, incluido el índice en `_id`, requiere al menos 8 kB de espacio de datos.
  - Para cada base de datos, un solo archivo de espacio de nombres (es decir, `<database> .ns`) almacena todos los metadatos para esa base de datos, y cada índice y colección tiene su propia entrada en el archivo de espacio de nombres. MongoDB pone límites al tamaño de los archivos de espacio de nombres.

# Collection Contains Large Number of Small Documents

- Debe considerar la inclusión por motivos de rendimiento si tiene una colección con una gran cantidad de documentos pequeños. Si puede agrupar estos documentos pequeños por alguna relación lógica y con frecuencia recupera los documentos por esta agrupación, puede considerar "enrollar" los documentos pequeños en documentos más grandes que contienen una serie de documentos incrustados.
- "Enrollar" estos pequeños documentos en agrupaciones lógicas significa que las consultas para recuperar un grupo de documentos implican lecturas secuenciales y menos accesos aleatorios al disco. Además, "enrollar" documentos y mover campos comunes al documento más grande benefician el índice en estos campos. Habría menos copias de los campos comunes y habría menos entradas clave asociadas en el índice correspondiente. Ver Índices para más información sobre el índice

# Storage Optimization for Small Documents

- Considere las siguientes sugerencias y estrategias para optimizar la utilización del almacenamiento para estas colecciones:
  - Use el campo `_id` explícitamente.
  - Los clientes de MongoDB agregan automáticamente un campo `_id` a cada documento y generan un ObjectId único de 12 bytes para el campo `_id`. Además, MongoDB siempre indexa el campo `_id`. Para documentos más pequeños, esto puede representar una cantidad significativa de espacio.
  - Para optimizar el uso del almacenamiento, los usuarios pueden especificar un valor para el campo `_id` explícitamente al insertar documentos en la colección. Esta estrategia permite que las aplicaciones almacenen un valor en el campo `_id` que habría ocupado espacio en otra parte del documento.
  - Puede almacenar cualquier valor en el campo `_id`, pero como este valor sirve como clave principal para los documentos de la colección, debe identificarlos de forma exclusiva. Si el valor del campo no es único, no puede servir como clave principal, ya que habría colisiones en la colección.
  - Use nombres de campo más cortos.

# Data Lifecycle Management

- Las decisiones de modelado de datos deben tener en cuenta la gestión del ciclo de vida de los datos.
- La función Time to Live o TTL de las colecciones vence los documentos después de un período de tiempo. Considere usar la función TTL si su aplicación requiere que algunos datos persistan en la base de datos por un período de tiempo limitado.
- Además, si su aplicación solo usa documentos insertados recientemente, considere Colecciones con límite. Las colecciones con límite proporcionan administración de primero en entrar, primero en salir (FIFO) de los documentos insertados y admiten de manera eficiente las operaciones que insertan y leen documentos según el orden de inserción.

# Practica

- Practica 4 Índices y Modelado de Datos

# Referencias

- <https://docs.mongodb.com/manual/indexes/>
- <https://docs.mongodb.com/manual/core/data-model-operations/>
- <http://gpd.sip.ucm.es/rafa/docencia/nosql/indices.html>
- <https://www.quora.com/What-is-a-good-software-tool-for-designing-a-MongoDB-database-schema>