Bases de Datos No Relacionales

- Índices

- Los índices permiten la ejecución eficiente de consultas. **Sin índices**, MongoDB **examina todos los documentos de una colección** para devolver los resultados de la consulta. **Si existe un índice** adecuado para una consulta, MongoDB utiliza el índice y **limita el número de documentos que debe analizar.** → **COLLSCAN vs. IXSCAN**

- Aunque los índices mejoran el rendimiento de las consultas, agregar un índice tiene un impacto negativo en el rendimiento de las operaciones de escritura. En las colecciones con una alta proporción de escritura a lectura, los índices son costosos porque cada inserción también debe actualizar los índices.

- Los índices son estructuras de datos especiales que almacenan una pequeña parte del conjunto de datos de la colección en un formato fácil de recorrer. En MongoDB la estructura de datos utilizada es la del **árbol B+.**

- El índice almacena **el valor de un campo específico o un conjunto de campos, ordenados por el valor del campo.** El orden de las entradas de índice **admite coincidencias de igualdad** y **operaciones de consulta basadas en intervalos**.

- **Casos de uso**
  - Índice de atributo único:
    Crearemos un índice sobre un único atributo cuando regularmente se requiera consultar ese atributo en particular.

  - Índice de atributo único sobre un documento embebido
    Crearemos un índice sobre un atributo de tipo **documento embebido** cuando regularmente se requiera consultar por el documento completo, **no** por alguno de sus atributos en particular. En esos casos no se utilizará el índice.
    Esto puede provocar comportamientos inesperados si el modelo de esquema cambia y agrega o elimina campos del documento indexado.
    Cuando se consulta documentos embebidos, **el orden en que se especifican los campos en la consulta importa.**

- **Casos de uso**

  - Índice en un atributo de un documento embebido:
    Se pueden crear índices en campos dentro de documentos incrustados. Los índices en campos incrustados pueden satisfacer consultas que utilizan notación de puntos.

  - Índice compuesto
    Crearemos un índice compuesto cuando las consultas a responder se realicen por más de atributo. Cuando se crea un índice por un atributo dentro de array, MongoDB almacena ese índice como un índice multiclave o compuesto, sea un array de documentos o de elementos escalares. Se pueden especificar hasta 32 atributos. Los índices compuestos admiten operaciones de ordenación que coinciden con el orden del índice o con el orden de clasificación inverso del índice.

- **Default**

  MongoDB crea un índice único en el campo **_id** durante la creación de una colección. El índice _id impide que se inserten dos documentos con el mismo valor para el campo _id.
  No se puede quitar este índice.

- **Nomenclatura de los índices**

  El nombre predeterminado de un índice será la concatenación de las claves indexadas y la dirección de cada clave en el índice (1 o -1, es decir ascendente o descendente) utilizando el carácter **underscore** como separador.
  Por ejemplo, un índice creado en { item : 1, quantity: -1 } tendrá el nombre
  item_1_quantity_-1

- ## Propiedades de los índices

(No todos los tipos de índice son compatibles con todas las propiedades de índice.)

Las propiedades de índice afectan la forma en que el ***planificador de consultas*** utiliza un índice y ***la forma en que se almacenan los documentos indizados.***

Las propiedades del índice se especifican como **parámetros opcionales** al momento de crear un índice.

Propiedades de los índices

Índices ocultos (Hidden)

No son visibles para el planificador de consultas y no se pueden utilizar para respaldar una consulta. Su utilidad radica en poder evaluar el impacto potencial de eliminar un índice sin eliminarlo realmente. Si el impacto es negativo, puede “mostrar” el índice en lugar de tener que volver a crear un índice eliminado. Los índices ocultos se mantienen en su totalidad y se pueden usar inmediatamente una vez que se muestran.

Índices dispersos (sparse)

Los índices dispersos solo tendrán entradas para documentos que tienen el atributo indexado (aunque el valor sea nulo). Estos índices omiten documentos que no tienen el campo indexado.

Propiedades de los índices

Índices parciales
Solo indexan los documentos de una colección que cumplen con una expresión de filtro especificada. Tienen menores requisitos de almacenamiento y costos de rendimiento reducidos para la creación y el mantenimiento de índices. Ofrecen un superconjunto de la funcionalidad de los índices dispersos y deben preferirse a los índices dispersos.

Índices TTL (Time to Live)
Eliminan automáticamente documentos de una colección después de un cierto período de tiempo. Se utilizan para datos que solo necesitan persistir durante un período de tiempo finito, como datos de eventos, registros e información de sesiones, etc.

Índices únicos
Los índices únicos hacen que MongoDB rechace valores duplicados para el atributo indexado. Estos índices son útiles cuando los documentos contienen un identificador único, como un ID de usuario, producto, evento, etc.

- ## Estrategias de indexación
  - Para crear los mejores índices para una aplicación se deben tener en cuenta una serie de factores:
    - los tipos de consultas que se esperan,
    - la proporción de lecturas y escrituras
    - la cantidad de memoria libre en el sistema.

Al desarrollar la estrategia de indexación, se debe tener un conocimiento profundo de las consultas que hará la aplicación. Los índices tienen un costo de rendimiento, pero valen más que el costo de las consultas **frecuentes** en grandes conjuntos de datos. Se debe tener en cuenta la frecuencia relativa de cada consulta de la aplicación y si la consulta justifica un índice. La mejor estrategia en general para diseñar índices es **crear perfiles de una variedad de configuraciones de índices con conjuntos de datos similares a los que se ejecutarán en producción** para ver qué configuraciones funcionan mejor.

- **Algunas operaciones sobre índices**
  - db.collname.**createIndex**(…)
  - db.collname.**dropIndex**(…)
  - db.collname.**dropIndexes**()
  - db.collname.**getIndexes**()
  - db.collname.**reIndex**() *(deprecated)*
  - db.collection.**hideIndex**()
  - db.collection.**unhideIndex**()

- **Algunas operaciones sobre índices**

Sintaxis:

db.collection.createIndex(<keys>, <options>, <commitQuorum>)

Cada parámetro será un documento

Ejemplos:

- `db.collection.createIndex( { orderDate: 1 } )`

  1 Indica orden ascendente / -1 descendente

- ```
  db.collection.createIndex(
  {
      atributo: 1
  },
  {
  unique: true,
  sparse: true,
  expireAfterSeconds: 3600
  … })
  ```

  Con opciones de propiedades
  ```

- **Algunas operaciones sobre índices**

Ejemplos:

- `db.facturas.createIndex( { orderDate: 1, zipcode: -1 } )`

El orden de los campos en un índice compuesto es muy importante.

- `db.facturas.createIndex({"item.producto":1, "item.cantidad":1})`

Índice compuesto creado con dos atributos correspondientes al mismo array.

No se puede crear un índice utilizando dos atributos de distintos arrays del mismo documento.

- **Algunas operaciones sobre índices**

Ejemplos:

- `db.facturas.createIndex({"cliente.region":1,condPago:-1})`

Aquí se crea un índice compuesto utilizando un atributo de un documento embebido y un atributo de la raíz del documento.

- `db.facturas.createIndex({"cliente":1})`

Aquí se crea un índice sobre un documento embebido. Si se consulta por algún atributo del documento en particular, no se usará el índice. (Suponer el caso de consulta por cliente.dni)

- **Algunas operaciones sobre índices**

- ```
  db.collection.createIndex(
                    { category: 1 },
                    { name: "category_IDX" } }
                )
  ```

  Se crea un índice y **se nombra específicamente**

Antes de especificar un nombre de índice, **considerar**:

- Los nombres de los índices deben ser únicos.
- No se puede cambiar el nombre de un índice existente.
  Se debe eliminar y volver a crear el índice con un nombre nuevo.
  ```

- Algunas operaciones sobre índices

Prefijos de índice

Los prefijos de índice son los subconjuntos iniciales de campos indexados. Los índices compuestos admiten consultas en todos los campos incluidos en el prefijo del índice.

Por ejemplo, suponiendo este índice compuesto:

{ "artículo": 1, "ubicación": 1, "stock": 1 }

Los prefijos del índice serán:

{ artículo 1 } / {artículo: 1, ubicación: 1}

MongoDB utilizará el índice compuesto para admitir consultas en estas combinaciones de campos:

artículo / artículo y ubicación/ artículo, ubicación y stock

También puede utilizar el índice para admitir una consulta sobre artículo y stock, pues artículo corresponde a un prefijo. Sin embargo la consulta no puede utilizar el campo de stock que sigue a la ubicación.

- Algunas operaciones sobre índices

Prefijos de índice (continuación)

Los campos de índice se analizan en orden; si una consulta omite un prefijo de índice, no podrá utilizar ningún campo de índice que siga a ese prefijo.

Entonces en el ejemplo anterior, MongoDB no puede utilizar el índice compuesto para admitir consultas en estas combinaciones de campos:

ubicación
stock
ubicación y stock

Sin el campo de artículo, ninguna de las combinaciones de campos anteriores corresponde a un índice de prefijo → no se usa el índice :(

- Algunas operaciones sobre índices

Prefijos de índice

TIP : *Eliminar índices redundantes !!*

Si se tiene una colección que tiene un índice compuesto y además un índice por su prefijo, supongamos : {a: 1, b: 1} y {a: 1}), si ninguno de los índices tiene una restricción única o sparse, puede eliminar el índice del prefijo ({ a: 1 }).

MongoDB utilizará el índice compuesto
en todas las situaciones en las que habría utilizado el índice de prefijo ;)

- **Algunas operaciones sobre índices**

Sintaxis:

db.collection.**dropIndex**("<indexName>")

```
Borra el indice especificado
```

db.<collection>.**dropIndexes**(["<index1>", "<index2>", "<index3>"])

```
Borra todos los indices contenidos en el array pasado como parámetro
```

db.collection.**dropIndex**( )

```
Borra todos los indices menos el _id
```

- **Algunas operaciones sobre índices**

Sintaxis:

db.collection.**getIndexes**()

Retorna los índices existentes en <collection>

db.collection.**reIndex**()

Elimina todos los índices en una colección y los recrea.
Esta operación puede resultar costosa para colecciones que tienen una gran cantidad de datos y/o una gran cantidad de índices.
Obtiene un bloqueo exclusivo (W) en la colección, bloqueando otras operaciones hasta que se completa.

- **Algunas operaciones sobre índices**

Sintaxis:

db.collection.**hideIndex**() // db.collection.**unhideIndex**()

Restricciones :

No se puede ocultar el índice _id.
Ocultar un índice no oculto restablece sus $indexStats.

Para ver estadísticas sobre el uso del índice se utiliza la siguiente operación de agregación:

```
db.coleccion.aggregate( [ { $indexStats: { } } ] )
```

- ## Otros tipos de índices

  - ### Comodín (wildcards)

    Estos tipos de índices se utilizan cuando los atributos que se desea indexar son desconocidos o puedan cambiar. Los índices comodín no funcionan tan bien como los índices de atributos específicos. Si la colección contiene nombres de atributos arbitrarios que impiden los índices específicos, se debería considerar remodelar el esquema para que tenga nombres de atributos consistentes.

- **Otros tipos de índices…**
  - Utilizar un índice comodín en los siguientes escenarios:

    Si la aplicación consulta una colección donde los nombres de los atributos varían entre documentos.

    Si la aplicación consulta repetidamente un atributo de documento incrustado donde los subcampos no son consistentes.

    Si la aplicación consulta documentos que comparten características comunes. Un índice comodín compuesto puede cubrir de forma eficaz muchas consultas de documentos que tienen atributos comunes.

- ## Otros tipos de índices

La siguiente sintaxis crea un índice para todos los campos de los documentos de la colección, incluyendo los atributos de documentos embebidos y de arrays:

db.coleccion.createIndex({ "$**" : 1 }) *(Se excluye el _id por default)*

Si se quisiera incluir o excluir atributos en un índice comodín, se debe setear la opción ***wildcardProjection***:

```
db.<collection>.createIndex(
  {
    "$**" : <sortOrder>
  },
  {
    "wildcardProjection" : {
      "<field1>" : < 0 | 1 >,
        ...
      "<fieldN>" : < 0 | 1 >
    }  } )
```

- **Otros tipos de índices**
  - Crear **un índice comodín sobre un atributo específico**:

    db.collection.createIndex({ "<field>.$**": <sortOrder> })

    Supongamos la siguiente colección:

```
{"product_name" : "Spy Coat",
"attributes" : {
      "material" : [ "Tweed", "Wool", "Leather" ],
      "size" : {
      "length" : 72,
      "units" : "inches"
}}},
{"product name" : "Spy Pen",
"attributes" : {
      "colors" : [ "Blue", "Black" ],
      "secret_feature" : {
      "name" : "laser",
      "power" : "1000",
      "units" : "watts"
}}}
```

- **Otros tipos de índices**

El siguiente índice:

```
db.products.createIndex( { "attributes.$**" : 1 } )
```

soportará las siguientes consultas:

- db.products.find({ "attributes.size.length" : { $gt : 60 } })
- db.products.find({ "attributes.material" : "Leather" })
- db.products.find(
  { "attributes.secret_feature.name" : "laser" },
  { "_id": 0, "product_name": 1, "attributes.colors": 1 }
)

- Crear **un índice comodín compuesto**:

```
db.products.createIndex( { "product_name":1,"attributes.$**" : 1 } )
```

- ## Otros tipos de índices

  - **Hash :** Considerar lo siguiente al crear índices de este tipo:

El atributo que se elija como clave para aplicar la función hash, debe tener una cardinalidad alta, lo que significa una gran cantidad de valores diferentes.

Si el modelo de datos no contiene un solo atributo con alta cardinalidad, considerar crear un **índice hash compuesto.** Un índice hash compuesto proporciona valores indexados más exclusivos y puede aumentar la cardinalidad. Se debe especificar hashed como el valor de una única clave del índice. Para las otras claves de índice, se especifica el orden de clasificación (1 o -1).

Si el atributo a utilizar debe admitir solo patrones de consulta comunes. Las consultas de rango (como $gt y $lt) no pueden utilizar un índice hash.

Un índice hash puede contener hasta 32 campos.

- ## Otros tipos de índices

  - **Indice Hash Simple**

    ```
    db.facturas.createIndex( { "nroFactura": "hashed" }
    ```

  - **Indice Hash Compuesto**

    ```
    db.facturas.createIndex( { "pago" : 1, "nroFactura" : "hashed" } )
    ```

- **Otros tipos de índices…**

  - **De Texto :**
  - Los índices de texto admiten consultas de búsqueda de texto en **atributos con tipo string**, incluso un array con datos tipo string.
  - Los índices de texto mejoran el rendimiento al buscar **palabras o frases específicas** dentro del contenido de un atributo de tipo cadena.
  - Una colección **sólo puede tener un índice de texto**, pero ese índice **puede cubrir varios atributos**.
  - Para realizar consultas que utilicen el índice text se debe utilizar el operador ***$text***.

- ## Otros tipos de índices…

Se puede crear un índice de texto que contenga todos los campos del documento con datos de string en una colección.

Estos índices de texto se denominan **índices de texto comodín**.

Los índices de texto comodín admiten la búsqueda de texto en campos desconocidos, arbitrarios o generados dinámicamente.

```
db.<collection>.createIndex( { "$**": "text" } )
```

Luego de crear un índice de texto comodín, cuando se insertan o actualizan documentos, **el índice se actualiza para incluir cualquier valor de campo de cadena nuevo.** Como resultado, **los índices de texto comodín afectan negativamente al rendimiento de las inserciones y actualizaciones.**

- **Otros tipos de índices…**
  - **De texto Simple**

```
db.<collection>.createIndex(
  {
        <field2>: "text",
  })
```

  - **De texto Compuesto**

```
db.<collection>.createIndex(
  {
    <field1>: "text",
    <field2>: "text",
    ...
  })
```

- **Otros tipos de índices…**
  - Ejemplos de uso de los indices de texto:
    - Para *buscar una palabra* :

      db.blog.**find**({ $text: { $search: "palabra" } })
    - Para *buscar varias palabras* :

      db.blog.**find**({ $text: { $search: "palabra1 palabra2" } }
    - Para *buscar una frase exacta*:

      db.blog.**find**({ $text: { $search: "\"la frase de interés\"" } })

- **Otros tipos de índices…**
  - **TTL (Time to live):**

    Los índices TTL son índices especiales de un solo campo que MongoDB puede usar para eliminar automáticamente documentos de una colección después de una cierta cantidad de tiempo o en un tiempo específico.
    La expiración de datos es útil para ciertos tipos de información, como datos de eventos generados por la máquina, registros e información de sesión que solo necesitan persistir en una base de datos por un tiempo finito.

- **Otros tipos de índices…**
  - **TTL :**

Creación de Índice TTL con expiración al llegar a una determinada fecha.

db.eventos.createIndex({ "fechaVencimiento": 1 }, { expireAfterSeconds: 0 })

db.eventos.insert({ "fechaCreado": new Date(), "fechaVencimiento": new Date("2020-01-01"), "eventoNro": 2, "mensajeLog": "Perfercto!!!" })

El Motor de BD borrará el mismo automáticamente cuando se alcance la fecha de Vencimiento

- **Otros tipos de índices…**
  - **TTL :**

Creación de Índice TTL con expiración de n segundos a partir de una fecha determinada.

db.eventos.createIndex({ "fechaCreado": 1 }, { expireAfterSeconds: 30})

Dado el siguiente insert:
db.eventos.insert({ "fechaCreado": new Date(), "eventoNro": 2, "mensajeLog": "Perfecto!!!" })
El Motor de BD borrará el mismo automáticamente pasados 30 segundos de su creación

- **Otros tipos de índices…**
  - **TTL parcial:**

Se pueden expirar documentos con condiciones de filtro: Para esto se debe crear un índice que sea parcial y un índice TTL.

Supongamos que insertamos los siguientes documentos:

```
db.foo.insertMany( [    { "Fecha" : new Date(), "D" : 3},
                        { "Fecha" : new Date(), "D" : 1}
                    ] )
```

y creamos el siguiente indice:

```
db.foo.createIndex(
  { Fecha: 1 },
  { name: "Partial-TTL-Index",
    partialFilterExpression: { D : 1 },
    expireAfterSeconds: 10
  })
```

Luego de 10 segundos solo se eliminará el segundo documento

- **Otros tipos de índices…**
  - **Sparse :**

    Los índices sparse solo contienen entradas para documentos que contienen el atributo correspondiente a la clave del índice, con valor o con nulo.

    El índice saltea los documentos que no contienen dicho atributo y no incluye entradas con la referencia a dichos documentos.

    Creación de un Indice Sparse

    ```
    db.facturas.createIndex( { estado: 1 }, { sparse: true } )
    ```

    El atributo estado estaría contenido por algunos documentos, no por la totalidad de los mismos. El índice sólo contendrá los documentos que contengan a dicho atributo.

- **explain()**

  - Con el comando explain() se puede ejecutar el *explain plan* de una determinada consulta con find, obteniendo un documento que informará las decisiones tomadas por el optimizador para ejecutar de manera más óptima la consulta solicitada.
  - Dada la siguiente consulta:

    ```
    db.facturas.find({"cliente.region":"CENTRO"}).explain()
    ```

    Observar el resultado (... winningPlan..) antes y después de crear un índice, por ejemplo, sobre el atributo cliente.region.

    Si sólo quiero saber el tiempo que toma el motor en realizar una query, indico luego del explain lo siguiente:

    ```
    db.facturas.find().explain("executionStats").executionStats.executionTimeMillis
    ```

- **hint()**
  - Utilizando la cláusula hint() se podría forzar al motor de base de datos a realizar una determinada consulta utilizando cierto índice.

No es conveniente utilizar este tipo de cláusulas,

lo ideal es que el optimizador del motor tome la mejor decisión.

Dadas las siguientes consultas:

db.facturas.find({},{}).explain() y db.facturas.find({},{}).hint("_id_").explain()

Observar el resultado (... winningPlan..) en el resultado de ambas