

Actividad 3 - Conceptos y Comandos básicos del particionamiento en bases de datos NoSQL

Helio David Espinosa Contreras

1010235183

julior José osorio pacheco

100090230

Corporación Universitaria Iberoamericana, Facultad De Ingeniería

Ingeniería De Software.

Bases de datos avanzadas

Prof. William Ruiz

17 de diciembre 2023

BASE DE DATOS PARA EVENTOS DE FUTBOL.

Requerimientos no funcionales para la base de datos.

Algunos de los requerimientos no funcionales que son fundamentales para garantizar que la base de datos pueda satisfacer las demandas de un evento de fútbol, desde la gestión de boletos y transacciones hasta la recopilación de datos en tiempo real durante el evento, encontramos los siguientes.

Rendimiento:

Alta velocidad de acceso a los datos: Acceda a los datos de manera rápida y eficiente para admitir consultas en tiempo real durante el evento de fútbol.

Escalabilidad: Capacidad para manejar un gran volumen de transacciones simultáneas durante eventos de alta concurrencia, como partidos importantes o venta de boletos.

Disponibilidad:

Alta disponibilidad: Garantizar que la base de datos esté disponible en todo momento, incluso en situaciones de alta demanda durante eventos en vivo.

Tolerancia a fallos: Capacidad para recuperarse rápidamente de posibles fallos del sistema para evitar interrupciones durante el evento.

Seguridad:

Seguridad de los datos: Implementar medidas de seguridad robustas para proteger la integridad y confidencialidad de los datos del evento, incluyendo información de boletos, transacciones y datos personales.

Cumplimiento normativo: Cumplir con las regulaciones de privacidad y protección de datos, especialmente al procesar información personal de los asistentes al evento de fútbol.

Escalar

Capacidad de crecimiento: La base de datos debe ser capaz de escalar para manejar un aumento en la cantidad de datos a medida que crece la popularidad del evento de fútbol y la participación de los aficionados.

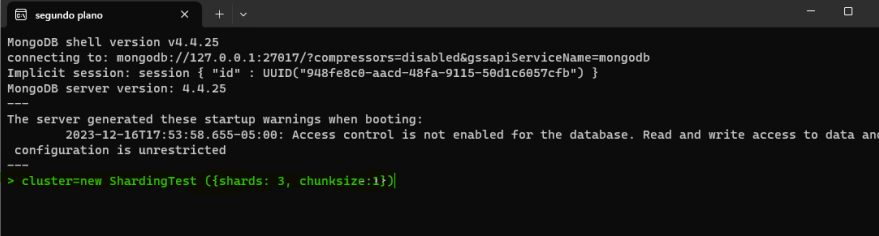
Mantenibilidad:

Facilidad de mantenimiento: La base de datos debe ser fácil de mantener y actualizar para garantizar un rendimiento óptimo antes, durante y después del evento de fútbol.

Particionamiento.

En este caso en MongoDB, podría ser necesario utilizar el particionamiento de la base de datos para mejorar el rendimiento y la escalabilidad del sistema. El particionamiento es una técnica que divide los datos en fragmentos más pequeños llamados particiones, que se distribuyen en diferentes servidores o nodos. Esto permite que la carga de trabajo se distribuya entre los nodos y se pueda acceder a los datos de manera más eficiente, el particionamiento en MongoDB se puede realizar de diferentes formas, como el particionamiento horizontal o el particionamiento por rango. El particionamiento horizontal implica dividir los datos en fragmentos basados en un criterio, como una clave de partición. Por ejemplo, se podría particionar la base de datos del evento futbolístico por equipos, de manera que los datos de cada equipo se almacenen en un fragmento separado. Esto permite que las consultas se realicen de manera más rápida y eficiente, ya que solo se accede a los datos relevantes, para la consulta de particionamiento también puede ayudar a mejorar la disponibilidad y la tolerancia a fallos del sistema. Si un nodo falla, los datos todavía estarán disponibles en otros nodos, lo que garantiza la continuidad del servicio, también es importante tener en cuenta que el particionamiento puede introducir cierta complejidad en el diseño y la implementación de la base de datos.

Se deben considerar aspectos como la distribución equitativa de la carga de trabajo, la gestión de las particiones y la sincronización de los datos entre los nodos, en conclusión, el particionamiento de la base de datos en MongoDB puede ser necesario en un evento de fútbol para mejorar el rendimiento, la escalabilidad y la disponibilidad del sistema. Esto permite distribuir la carga de trabajo entre los nodos y acceder a los datos de manera más eficiente.

- 
- The screenshot shows a terminal window titled "segundo plano" with the following content:
- ```

MongoDB shell version v4.4.25
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongod
Implicit session: session { "id" : UUID("948fe8c0-aacd-48fa-9115-50d1c6057cfb") }
MongoDB server version: 4.4.25

The server generated these startup warnings when booting:
 2023-12-16T17:53:58.655-05:00: Access control is not enabled for the database. Read and write access to data and
 configuration is unrestricted

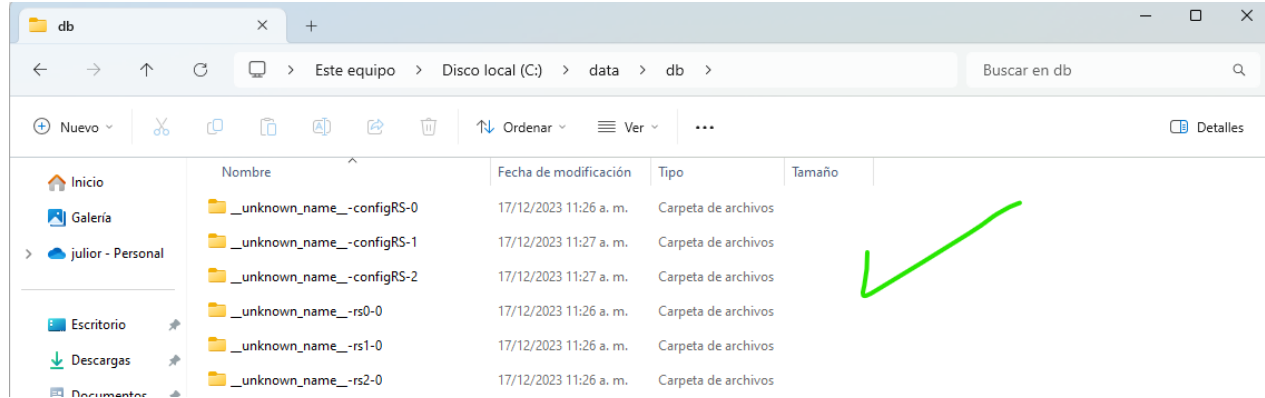
> cluster=new ShardingTest({shards: 3, chunksize:1})

```

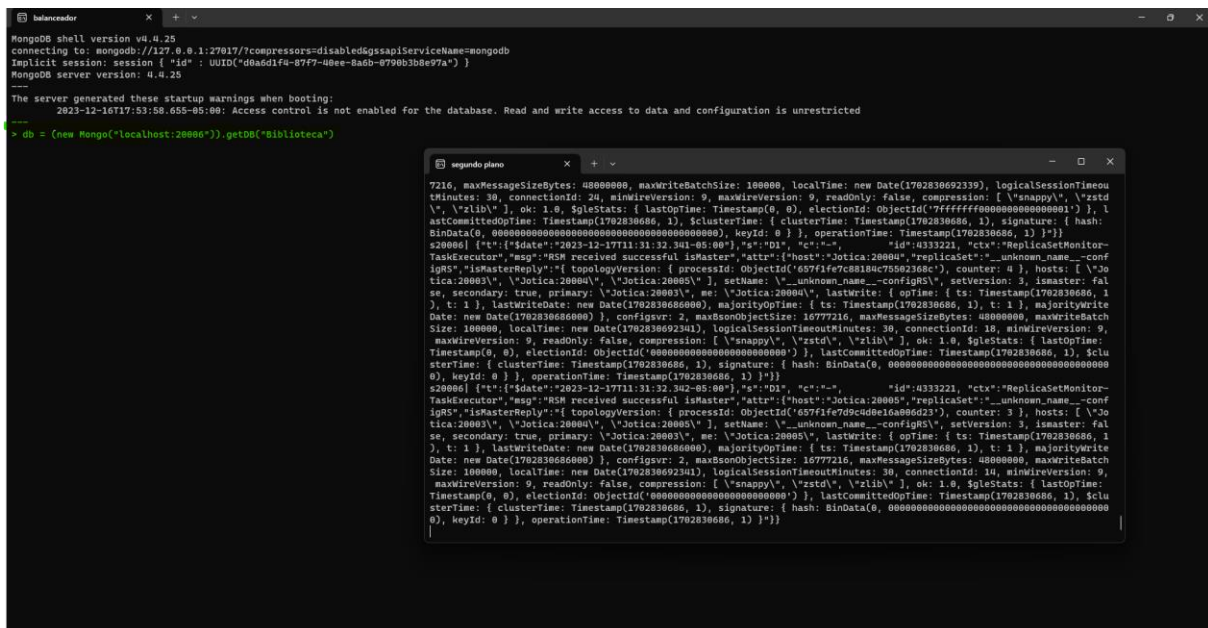
[illegible]

Esperamos hasta que las instancias se creen por completo.

- Verificamos en **C:\data\db**



- Ahora creamos una nueva ventana en consola BALANCEADOR y escribimos el siguiente comando, como evidenciamos la ventana de inicio que se llama segundo plano sigue ejecutándose.



con este comando `db = (new Mongo("localhost:20006")).getDB("DEPORTIVOS")` establecemos la conexión con la base de datos "DEPORTIVOS" Y ahora nuestra base de datos se encuentra particionada y lista para insertar datos.

```
balanceador x + v
MongoDB shell version v4.4.25
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("34a1ca02-230f-498d-b955-acb4d374f7e1") }
MongoDB server version: 4.4.25

The server generated these startup warnings when booting:
 2023-12-16T17:53:58.655-05:00: Access control is not enabled for the database. Read and write access to data and
 configuration is unrestricted

> db = (new Mongo("localhost:20006")).getDB("DEPORTIVOS")
DEPORTIVOS
mongos> |
```

Se insertan 50,000 registros en la colección "encuentros" utilizando el siguiente código:

```
balanceador x + v
MongoDB shell version v4.4.25
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("ec761891-e7de-4e24-b79a-5c39e11d049c") }
MongoDB server version: 4.4.25

The server generated these startup warnings when booting:
 2023-12-16T17:53:58.655-05:00: Access control is not enabled for the database. Read and write access to data and
 configuration is unrestricted

> db = (new Mongo("localhost:20006")).getDB("DEPORTIVOS")
DEPORTIVOS
mongos> for (var i = 0; i < 5000; i++) {
... var encuentro = {
... equipo_local: "Equipo " + i,
... equipo_visitante: "Equipo " + (i + 1),
... fecha: new Date()
... };
... db.encuentros.insertOne(encuentro);
... }
{
 "acknowledged" : true,
 "insertedId" : ObjectId("657f2c07d7399855e7e4a380")
}
mongos> |
```

Se ingresan 25,000 registros en la colección "posiciones" utilizando el siguiente código.

```
mongos> for (var i = 0; i < 25000; i++) {
... var posicion = {
... campo1: "valor1",
... campo2: "valor2",
... };
... db.posiciones.insertOne(posicion);
... }
{
 "acknowledged" : true,
 "insertedId" : ObjectId("657f3310d7399855e7e50528")
}
```

vamos a ingresar 20,000 datos a la colección "resultados" utilizando el siguiente código.

```
mongos> for (var i = 0; i < 20000; i++) {
... var resultado = {
... campo1: "valor1",
... campo2: "valor2",
... };
... db.resultados.insertOne(resultado);
... }
{
 "acknowledged" : true,
 "insertedId" : ObjectId("657f36fad7399855e7e55348")
}
```

Verificamos los datos ingresados.

```
mongos> db.encuentros.count()
5000
mongos> db.posiciones.count()
25000
mongos> db.resultados.count()
20000
mongos> |
```

### Comprobación de la distribución de datos en los nodos

```
comprobar
MongoDB shell version v4.4.25
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("f65d8a1e-80f2-412f-abfa-894b0b1117d0") }
MongoDB server version: 4.4.25

The server generated these startup warnings when booting:
2023-12-16T17:53:58.655-05:00: Access control is not enabled for the database. Read and write access to data and
configuration is unrestricted

> shard1 = new Mongo("localhost:20000")
connection to localhost:20000
> shard1DB = shard1.getDB("DEPORTIVOS")
DEPORTIVOS
> shard1DB.encuentros.count()
5000
> shard2 = new Mongo("localhost:20001")
connection to localhost:20001
> shard2DB = shard2.getDB("DEPORTIVOS")
DEPORTIVOS
> shard2DB.encuentros.count()
0
> shard3 = new Mongo("localhost:20002")
connection to localhost:20002
> shard3DB = shard3.getDB("DEPORTIVOS")
DEPORTIVOS
> shard3DB.encuentros.count()
0
> |
```

En resumen, al examinar la inserción de registros en la colección "encuentros" en el nodo fragmentado indica una distribución de datos desequilibrada. La conexión a cada nodo se establece desde el balanceador, pero se observa que todos los documentos insertados se almacenan únicamente en el primer nodo. Esto indica que la partición de datos no funciona porque la distribución entre nodos es desigual. Es importante tener en cuenta que, de forma predeterminada, la fragmentación no está habilitada al crear objetos para pruebas de fragmentación. Por tanto, la falta de distribución entre nodos se puede atribuir a esta configuración predeterminada. Antes de concluir que la fragmentación en MongoDB no funciona, es necesario habilitar la partición para garantizar que la carga de datos se distribuya correctamente entre todos los nodos del conjunto de fragmentación.

### Activación del Sharding.

Utilizamos los siguientes comandos

```
shard1 = new Mongo("localhost:20006")
```

```
sh.status()
```

```
mongos> shard1 = new Mongo("localhost:20006")
connection to localhost:20006
mongos> sh.status()
--- Sharding Status ---
 sharding version: {
 "_id" : 1,
 "minCompatibleVersion" : 5,
 "currentVersion" : 6,
 "clusterId" : ObjectId("657f1fe9185650b0438da868")
 }
 shards:
 { "_id" : "__unknown_name__-rs0", "host" : "__unknown_name__-rs0/Jotica:20000", "state" : 1 }
 { "_id" : "__unknown_name__-rs1", "host" : "__unknown_name__-rs1/Jotica:20001", "state" : 1 }
 { "_id" : "__unknown_name__-rs2", "host" : "__unknown_name__-rs2/Jotica:20002", "state" : 1 }
 active mongoses:
 "4.4.25" : 1
 autosplit:
 Currently enabled: no
 balancer:
 Currently enabled: no
 Currently running: no
 Failed balancer rounds in last 5 attempts: 0
 Migration Results for the last 24 hours:
 No recent migrations
 databases:
 { "_id" : "DEPORTIVOS", "primary" : "__unknown_name__-rs0", "partitioned" : false, "version" : { "uuid" : U
 UID("a1cf577c-ff7a-4e91-b245-fc4ed931bb59"), "lastMod" : 1 } }
 { "_id" : "config", "primary" : "config", "partitioned" : true }
mongos>
```



```
mongos> sh.enableSharding("DEPORTIVOS")
{
 "ok" : 1,
 "operationTime" : Timestamp(1702838341, 3),
 "$clusterTime" : {
 "clusterTime" : Timestamp(1702838341, 3),
 "signature" : {
 "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
 "keyId" : NumberLong(0)
 }
 }
}
mongos> |
```

Crearemos un índice en la clave que se utilizará como shard con el siguiente comando:  
`db.partidos.ensureIndex({fecha: 1}).`

```
mongos> db.encuentros.ensureIndex({fecha: 1})
{
 "raw" : {
 "__unknown_name__-rs0/Jotica:20000" : {
 "createdCollectionAutomatically" : false,
 "numIndexesBefore" : 1,
 "numIndexesAfter" : 2,
 "commitQuorum" : "votingMembers",
 "ok" : 1
 }
 },
 "ok" : 1,
 "operationTime" : Timestamp(1702838603, 5),
 "$clusterTime" : {
 "clusterTime" : Timestamp(1702838603, 5),
 "signature" : {
 "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
 "keyId" : NumberLong(0)
 }
 }
}
mongos> |
```

Definimos la colección "partidos" para el Sharding con el comando:  
`sh.shardCollection("DEPORTIVOS.partidos", {fecha: 1}).`

Después de unos minutos, ejecutamos `sh.status ()` nuevamente, y vemos que esta información de salida es más explícita

Para activar el balanceador de carga, aunque el shard0000 está en modo particionado (partitioned: true), el balanceador aún no se ha ejecutado. Puedes verificar su estado de ejecución con el siguiente comando.

```
sh.getBalancerState()
```

```
mongos> sh.getBalancerState()
false
mongos> |
```

Iniciamos la ejecución del balanceador. Para lograrlo, utilizamos la función `setBalancerState(boolean)`.

```
mongos> sh.setBalancerState(true)
{
 "ok" : 1,
 "operationTime" : Timestamp(1702839549, 3),
 "$clusterTime" : {
 "clusterTime" : Timestamp(1702839549, 3),
 "signature" : {
 "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA="),
 "keyId" : NumberLong(0)
 }
 }
}
mongos> |
```

Video [https://laiberocol-my.sharepoint.com/personal/josorio5\\_iberocol\\_edu\\_co/\\_layouts/15/stream.aspx?id=%2Fpersonal%2Fjosorio5%5Fiberocol%5Fedu%5Fco%2FDocuments%2FGrabaciones%2FGrabaciones%2Fdb%20actividad%203%2D20231217%5F143223%2DGrabaci%C3%B3n%20de%20la%20reuni%C3%B3n%2Emp4&ga=1&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview](https://laiberocol-my.sharepoint.com/personal/josorio5_iberocol_edu_co/_layouts/15/stream.aspx?id=%2Fpersonal%2Fjosorio5%5Fiberocol%5Fedu%5Fco%2FDocuments%2FGrabaciones%2FGrabaciones%2Fdb%20actividad%203%2D20231217%5F143223%2DGrabaci%C3%B3n%20de%20la%20reuni%C3%B3n%2Emp4&ga=1&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview)

## **Conclusión**

En resumen, el particionamiento es una técnica fundamental en las bases de datos NoSQL para lograr escalabilidad y rendimiento. Proporciona flexibilidad en el esquema, permite distribuir los datos en múltiples nodos y mejorar la capacidad de manejar grandes volúmenes de datos distribuidos. Sin embargo, es importante considerar las características de consistencia y tolerancia a la partición al elegir una base de datos NoSQL.