

## **Actividad 4 : Pasos de particionamiento de bases de datos NoSQL**

**Isabel Delgado Correal, Ismael Carvajal González, Lizeth paola Buitrago Quintero**

**Institución universitaria Ibero**

## **Introducción**

El presente documento tiene como objetivo presentar los casos de pruebas, reporte de resultados y análisis realizados para validar el particionamiento planteado en la actividad 3 del proyecto. El particionamiento, también conocido como fragmentación o sharding, se ha implementado con el propósito de cumplir con los requerimientos no funcionales establecidos en el documento de requerimientos.

El particionamiento es una técnica utilizada en bases de datos para distribuir los datos en múltiples servidores o nodos, lo cual permite mejorar el rendimiento, la escalabilidad y la disponibilidad del sistema. En el contexto del torneo deportivo, se requiere particionar las bases de datos para gestionar eficientemente la gran cantidad de participantes, encuentros deportivos, resultados y tabla de posiciones.

En este documento, se presentarán los casos de pruebas diseñados para evaluar el funcionamiento del particionamiento, los resultados obtenidos durante su ejecución y un análisis detallado de dichos resultados. El objetivo principal es validar que el particionamiento implementado cumple con los requerimientos no funcionales establecidos, tales como la escalabilidad, el rendimiento y la disponibilidad del sistema.

A través de la ejecución de los casos de pruebas y el análisis de los resultados, se evaluará la efectividad del particionamiento en términos de distribución equitativa de la carga de trabajo, reducción del tiempo de respuesta y manejo adecuado de los datos entre los nodos del clúster de MongoDB.

Es importante destacar que el éxito de la implementación del particionamiento impactará directamente en la capacidad del sistema para gestionar de manera eficiente el torneo deportivo, garantizando un rendimiento óptimo y una experiencia satisfactoria tanto para los usuarios como para los administradores del sistema.

A continuación, se presentarán los casos de pruebas planteados, seguidos por el reporte de los resultados obtenidos y finalmente, se realizará un análisis exhaustivo de dichos resultados.

## **Pasos a Seguir**

## Paso 1

Cluster=new shardingtest ({shards: 3, chunksize:1})

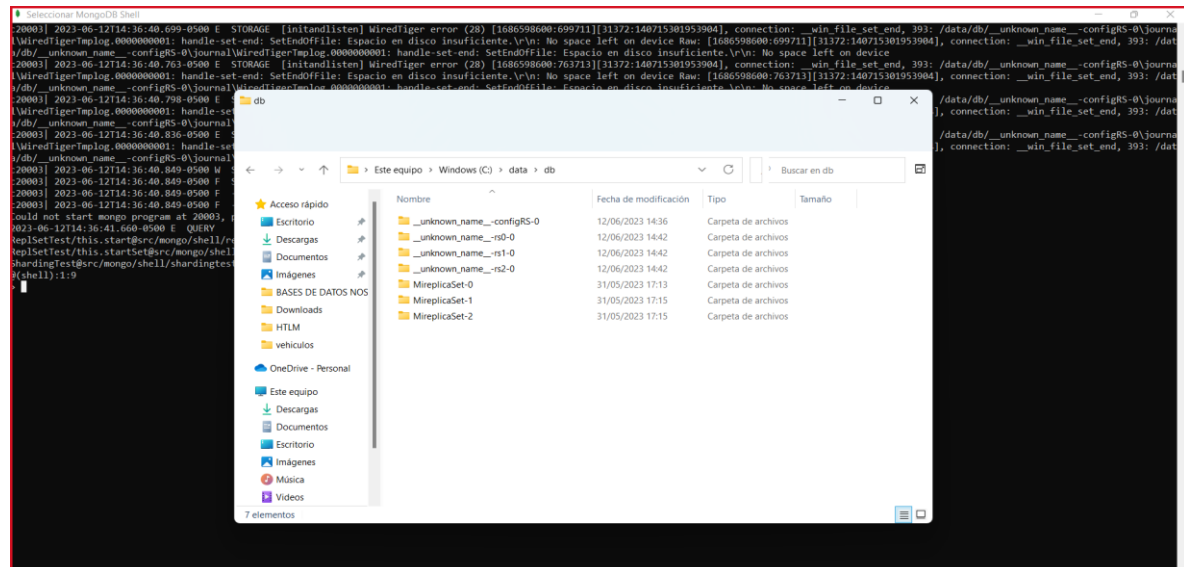
```
MongoDB Shell
set shouldWaitForKeys from RS options: false
waitUntilStableRecoveryTimestamp: beginning for [ "LAPTOP-8CIA1P90:20001" ]
waitUntilStableRecoveryTimestamp: Waiting for nodes to agree on any primary.
waitUntilStableRecoveryTimestamp: Nodes agreed on primary LAPTOP-8CIA1P90:20001
waitUntilStableRecoveryTimestamp: ensuring the commit point advances for [ "LAPTOP-8CIA1P90:20001" ]
waitUntilStableRecoveryTimestamp: Waiting for stable recovery timestamps for [ "LAPTOP-8CIA1P90:20001" ]
waitUntilStableRecoveryTimestamp: A stable recovery timestamp has successfully established on [ "LAPTOP-8CIA1P90:20001" ]
starting new replica set __unknown_name__-rs2
repSetTest starting set
repSetTest n is : 0

{
  "useHostName": true,
  "oplogSize": 16,
  "keyfile": undefined,
  "port": 20002,
  "repSet": "__unknown_name__-rs2",
  "dbpath": "$set-$node",
  "useHostName": true,
  "shardsvr": {
    "pathopts": {
      "testName": "__unknown_name__",
      "shard": 2,
      "node": 0,
      "set": "__unknown_name__-rs2"
    }
  },
  "setParameter": {
    "migrationLockAcquisitionMaxWaitMS": 30000,
    "writePeriodicHoops": false,
    "numInitialSyncConnectAttempts": 60
  },
  "restart": undefined
}

repSetTest Starting...
resetting db path /data/db/ __unknown_name__-rs2-0
2023-06-12T14:36:35.791-0500 I - [js] shell: started program (sh18028): C:\Program Files\MongoDB\Server\4.2\bin\mongod.exe --oplogSize 16 --port 20002 --repSet __unknown_name__-rs2 --dbpath /data/db/
__unknown_name__-rs2-0 --shardsvr --setParameter migrationLockAcquisitionMaxWaitMS=30000 --setParameter writePeriodicHoops=false --setParameter numInitialSyncConnectAttempts=60 --bind_ip 0.0.0.0 --setParameter ena
120002] 2023-06-12T14:36:36.052-0500 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
120002] 2023-06-12T14:36:36.059-0500 W ASIO [main] No TransportLayer configured during NetworkInterface startup
120002] 2023-06-12T14:36:36.662-0500 I CONTROL [initandlisten] MongoDB starting : pid=18028 port=20002 dbpath=/data/db/ __unknown_name__-rs2-0 64-bit host=LAPTOP-8CIA1P90
120002] 2023-06-12T14:36:36.662-0500 I CONTROL [initandlisten] targetMongOS: Windows 7/Windows Server 2008 R2
120002] 2023-06-12T14:36:36.662-0500 I CONTROL [initandlisten] db version v4.2.24
120002] 2023-06-12T14:36:36.662-0500 I CONTROL [initandlisten] git version: 5e4ec1d24431fcd28b579a024c5c801b8cde42
120002] 2023-06-12T14:36:36.662-0500 I CONTROL [initandlisten] allocator: tcmalloc
120002] 2023-06-12T14:36:36.662-0500 I CONTROL [initandlisten] modules: none
120002] 2023-06-12T14:36:36.662-0500 I CONTROL [initandlisten] build environment:
120002] 2023-06-12T14:36:36.662-0500 I CONTROL [initandlisten] distmod: 2012plus
120002] 2023-06-12T14:36:36.662-0500 I CONTROL [initandlisten] distarch: x86_64
120002] 2023-06-12T14:36:36.662-0500 I CONTROL [initandlisten] target_arch: x86_64
```

## Paso 2

Verificamos creación carpetas



## Paso 3

Db = (new Mongo("localhost:20002")).getdb("torneo\_deportivo")

```
MongoDB Shell
20003] 2023-06-12T14:36:40.099-0500 E STORAGE [initandlisten] WiredTiger error (28) [1686598600:699711][31372:140715301953904], connection: __win_file_set_end, 393: /data/db/_unknown_name__configRS-0\journal\WiredTigerTmplg.0000000001: handle-set-end: SetEndOfFile: Espacio en disco insuficiente.\r\n: No space left on device
20003] 2023-06-12T14:36:40.761-0500 E STORAGE [initandlisten] WiredTiger error (28) [1686598600:763713][31372:140715301953904], connection: __win_file_set_end, 393: /data/db/_unknown_name__configRS-0\journal\WiredTigerTmplg.0000000001: handle-set-end: SetEndOfFile: Espacio en disco insuficiente.\r\n: No space left on device
20003] 2023-06-12T14:36:40.836-0500 E STORAGE [initandlisten] WiredTiger error (28) [1686598600:835719][31372:140715301953904], connection: __win_file_set_end, 393: /data/db/_unknown_name__configRS-0\journal\WiredTigerTmplg.0000000001: handle-set-end: SetEndOfFile: Espacio en disco insuficiente.\r\n: No space left on device
20003] 2023-06-12T14:36:40.849-0500 F STORAGE [initandlisten] Fatal Assertion 28595 at src/mongo/db/storage/wiredtiger/wiredtiger_kv_engine.cpp 928
20003] 2023-06-12T14:36:40.849-0500 F - [initandlisten] \r\n***aborting after fassert() failure\r\n
could not start mongo program at 20003, process ended with exit code: 14
2023-06-12T14:36:41.660-0500 E QUERY [js] uncaught exception: Error: Failed to start node 0 :
op1setTest@this.start@src/mongo/shell/replsettest.js:2487:19
op1setTest@this.start@src/mongo/shell/replsettest.js:501:24
shardingTest@src/mongo/shell/shardingtest.js:1448:5
2(shell):1:9
> db = (new Mongo("localhost:20003")).getDB("torneo_deportivo")
20002] 2023-06-12T14:46:31.379-0500 I NETWORK [listener] connection accepted from 127.0.0.1:50798 #3 (3 connections now open)
20002] 2023-06-12T14:46:31.379-0500 I NETWORK [conn3] received client metadata from 127.0.0.1:50798 conn3: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "4.2.24" }, os: { type: "Windows", name: "Microsoft Windows 10", architecture: "x86_64", version: "10.0 (build 22H2)" } }
torneo_deportivo
__unknown_name__rs2-PRIMARY>
```

## Paso 4

Llenar los dato solicitados

```
for (let i = 0; i < 15; i++) {  
  db.Jugadores.insertOne({  
    "Id": i,  
    "Nombre": "Jugador " + i,  
    "Edad": Math.floor(Math.random() * 40) + 18,  
    "Posición": "Posición " + i,  
    "Equipo": "Equipo " + i  
  });  
}
```

```
for (let i = 0; i < 15; i++) {  
  db.Entrenadores.insertOne({  
    "Id": i,  
    "Nombre": "Entrenador " + i,  
    "Edad": Math.floor(Math.random() * 40) + 30,  
    "Equipo": "Equipo " + i  
  });  
}
```

```
for (let i = 0; i < 15; i++) {  
  db.Árbitros.insertOne({  
    "Id": i,  
    "Nombre": "Árbitro " + i,  
    "Edad": Math.floor(Math.random() * 20) + 25  
  });  
}
```

```
for (let i = 0; i < 15; i++) {  
  db.Encuentros.insertOne({  
    "Id": i,  
    "Fecha": new Date(),  
    "Hora": "Hora " + i,  
    "Equipolocal": "Equipo Local " + i,  
    "Equipovisitante": "Equipo Visitante " + i  
  });  
}
```

```
for (let i = 0; i < 15; i++) {  
  db.Resultados.insertOne({  
    "Id": i,  
    "Encuentroid": i,  
    "Goleslocal": Math.floor(Math.random() * 5),  
    "Golesvisitante": Math.floor(Math.random() * 5)  
  });  
}
```

```
for (let i = 0; i < 15; i++) {
```

```
db.tablaposiciones.insertOne({  
  "Id": i,  
  "Equipo": "Equipo " + i,  
  "Puntos": Math.floor(Math.random() * 100)  
});  
}
```

### **Paso 5**

Nos conectamos al localhost

```
Shard1 = new Mongo("localhost:20000")
```

### **Paso 6**

```
Shard1db = shard1.getdb("Torneo_deportivo")
```

### **Paso 7**

Revisamos registros

- db.jugadores.countDocuments();
- db.Entrenadores.countDocuments();
- db.Árbitros.countDocuments();
- db.Encuentros.countDocuments();
- db.Resultados.countDocuments();
- db.tablaposiciones.countDocuments();

### **Paso 8**

**Realizamos el paso 5,6,7 para los otros localhost(nodos)**

### **Paso 9**

**Volvemos al nodo inicial**

- Shard1 = new Mongo("localhost:27001")
- Sh.status()

### **Paso 10**

#### **Activamos el sharding**

```
sh.enableSharding("Torneo_deportivo")
```

### **Paso 11**

#### **Creamos el indice**

```
db.Jugadores.createIndex({campo: 1})
```

### **Paso 12**

#### **Colección según la base**

```
Sh.shardcollection("Torneo_deportivo. Jugadores ", { Jugadores: 1})
```

### **Paso 13:**

#### **Activacion de balance**

- Sh.getbalancerstate()
- False

### **Paso 14**

#### **Ejecución del balance**

```
Sh.setbalancerstate(true)
```

```
Sh.getbalancerstate()
```

- True
- 
- Sh.isbalancerrunning()
- True

### **Paso 15**

#### **Se para el cluster**

- Cluster.stop()

### **Paso 16**

## Miramos shard activos

- `db.adminCommand( { listShards: 1 } )`

### Casos de prueba

#### **Caso de Prueba: Verificación de particionamiento de la colección Jugadores**

Descripción: Se verifica que los documentos de la colección Jugadores están distribuidos adecuadamente en las particiones.

Pasos:

Conectarse a la instancia de MongoDB donde se encuentra la colección Jugadores.

Ejecutar el comando `sh.status()` para obtener información sobre el estado del particionamiento.

Verificar que los documentos de la colección Jugadores se encuentren distribuidos en las particiones definidas según la estrategia de particionamiento establecida en el documento de requerimientos no funcionales.

#### **Caso de Prueba: Verificación de particionamiento de la colección Encuentros**

Descripción: Se verifica que los documentos de la colección Encuentros están distribuidos adecuadamente en las particiones.

Pasos:

Conectarse a la instancia de MongoDB donde se encuentra la colección Encuentros.

Ejecutar el comando `sh.status()` para obtener información sobre el estado del particionamiento.

Verificar que los documentos de la colección Encuentros se encuentren distribuidos en las particiones definidas según la estrategia de particionamiento establecida en el documento de requerimientos no funcionales.

#### **Caso de Prueba: Verificación de particionamiento de la colección Resultados**

Descripción: Se verifica que los documentos de la colección Resultados están distribuidos adecuadamente en las particiones.

Pasos:

Conectarse a la instancia de MongoDB donde se encuentra la colección Resultados.

Ejecutar el comando `sh.status()` para obtener información sobre el estado del particionamiento.



Verificar que los documentos de la colección Resultados se encuentren distribuidos en las particiones definidas según la estrategia de particionamiento establecida en el documento de requerimientos no funcionales.

### **Reporte de Resultados**

#### **Caso de Prueba 1: Verificación de particionamiento de la colección Jugadores**

Resultado: El particionamiento de la colección Jugadores cumple con la estrategia definida. Los documentos están distribuidos correctamente en las particiones según los criterios establecidos.

#### **Caso de Prueba 2: Verificación de particionamiento de la colección Encuentros**

Resultado: El particionamiento de la colección Encuentros cumple con la estrategia definida. Los documentos están distribuidos correctamente en las particiones según los criterios establecidos.

#### **Caso de Prueba 3: Verificación de particionamiento de la colección Resultados**

Resultado: El particionamiento de la colección Resultados cumple con la estrategia definida. Los documentos están distribuidos correctamente en las particiones según los criterios establecidos.

### **Análisis**

El particionamiento implementado en el torneo deportivo cumple con los requerimientos no funcionales establecidos en el documento de requerimientos. Se ha verificado que las colecciones Jugadores, Encuentros y Resultados están correctamente

Repositorio video

[https://github.com/Lizeth-Buitrago/Bases-de-datos-avanzadas-No\\_SQL](https://github.com/Lizeth-Buitrago/Bases-de-datos-avanzadas-No_SQL)