Corporación Universitaria Iberoamericana

IBERO

Bases De Datos Avanzadas

(28102024_C2_202434)

Presentado al Profesor:

JORGE CASTAÑEDA

Presentado por:

CRISTIAN SANTIAGO RIVERA GUZMAN

ID 100170863

JHEISSON ALEJANDRO LOZANO CRUZ

ID 100171058

JUAN DAVID UBAQUE JARAMILLO

100171144

MICHAEL FABIAN ROJAS SABOGAL

100171974

Ibagué / Tolima

2024

IBEROAMERICANA
CORPORACIÓN UNIVERSITARIA
P.J. No. 0428 del 28 de Enero 1982 - MEN | VIGILADA MINEDUCACIÓN

Contenido

IBEROAMERICANA
CORPORACIÓN UNIVERSITARIA
P.J. No. 0428 del 28 de Enero 1982 - MEN | VIGILADA MINEDUCACIÓN

Introducción

Escenario:

El escenario involucra una base de datos que gestiona información sobre equipos deportivos, jugadores y eventos deportivos. Esta información incluye detalles como el nombre del equipo, el nombre del entrenador, los jugadores, las posiciones y otros datos asociados.

Requerimientos No Funcionales:

- **Escalabilidad**: La base de datos debe poder manejar un crecimiento significativo en términos de datos (equipos, jugadores, eventos).
- **Alto rendimiento en consultas:** Debe ser eficiente en la ejecución de consultas que involucren la búsqueda de equipos, jugadores y eventos, ya que estos podrían ser accedidos con alta frecuencia.
- **Alta disponibilidad**: El sistema debe garantizar que, incluso en caso de que un shard falle, la disponibilidad de los datos no se vea afectada.

Estrategia de particionamiento

1. **Clave de Fragmentación (Shard Key)**: Se seleccionará un campo que ayude a distribuir de manera uniforme los datos entre los shards. Considerando los requerimientos de las consultas, se utilizará el campo nombre de los equipos para la fragmentación.
2. **Distribución de los equipos**:
    a. Usaremos el campo nombre para dividir los datos de los equipos. Este campo tiene valores relativamente distribuidos alfabéticamente, lo que hace que sea una buena opción para la fragmentación.
    b. **M** es un buen punto de corte porque divide los equipos que comienzan con letras antes de "M" y los que comienzan con letras después de "M".
3. **Distribución de Jugadores y Eventos**: Si se requiere particionar otras colecciones (como los jugadores o eventos), se pueden usar enfoques similares, basados en claves como posición para los jugadores y fecha para los eventos.

Comandos para crear entorno de particionamiento

1. **Crear directorios para los shards:**
   md c:\mongodb\shards\shard1
   md c:\mongodb\shards\shard2
   md c:\mongodb\shards\shard3
   md c:\mongodb\config\config1
   md c:\mongodb\config\config2
   md c:\mongodb\config\config3
   md c:\mongodb\mongos
2. **Configurar los shards con replicación:** Cada shard será un conjunto de réplicas. Usa los siguientes comandos para iniciar los nodos
    a. **Shard 1:**
       start mongod --bind_ip localhost --port 27011 --dbpath c:\mongodb\shards\shard1 --replSet

shard1 --shardsvr
start mongod --bind_ip localhost --port 27012 --dbpath c:\mongodb\shards\shard1 --replSet shard1 --shardsvr
start mongod --bind_ip localhost --port 27013 --dbpath c:\mongodb\shards\shard1 --replSet shard1 –shardsvr

b. **Shard 2:**
start mongod --bind_ip localhost --port 27021 --dbpath c:\mongodb\shards\shard2 --replSet shard2 --shardsvr
start mongod --bind_ip localhost --port 27022 --dbpath c:\mongodb\shards\shard2 --replSet shard2 --shardsvr
start mongod --bind_ip localhost --port 27023 --dbpath c:\mongodb\shards\shard2 --replSet shard2 --shardsvr

c. **Shard 3:**
start mongod --bind_ip localhost --port 27031 --dbpath c:\mongodb\shards\shard3 --replSet shard3 --shardsvr
start mongod --bind_ip localhost --port 27032 --dbpath c:\mongodb\shards\shard3 --replSet shard3 --shardsvr
start mongod --bind_ip localhost --port 27033 --dbpath c:\mongodb\shards\shard3 --replSet shard3 --shardsvr

3. **Configure los config servers:** Los config servers mantienen el estado del clúster y las rutas de los datos. inicia los config servers Como un conjunto de replicas:
start mongod --bind_ip localhost --port 27041 --dbpath c:\mongodb\config\config1 --replSet configReplSet --configsvr
start mongod --bind_ip localhost --port 27042 --dbpath c:\mongodb\config\config2 --replSet configReplSet --configsvr
start mongod --bind_ip localhost --port 27043 --dbpath c:\mongodb\config\config3 --replSet configReplSet --configsvr
4. **Iniciar las configuraciones de replicación:** Conéctate a cada conjunto de réplicas y configúralos

```
Config servers:
        mongosh --port 27041
        rs.initiate({
            _id: "configReplSet",
            configsvr: true,
            members: [
                { _id: 0, host: "localhost:27041" },
                { _id: 1, host: "localhost:27042" },
                { _id: 2, host: "localhost:27043" }
            ]
        })
Shard 1:
    mongosh --port 27011
    rs.initiate({
        _id: "shard1",
        members: [
            { _id: 0, host: "localhost:27011" },
        ]
    })
```

```
    rs.add("localhost:27012"); // Nodo secundario
    rs.add("localhost:27013"); // Nodo secundario

Shard 2:

    mongosh --port 27021
    rs.initiate({
        _id: "shard2",
        members: [
            { _id: 0, host: "localhost:27021" },
        ]
    })
    rs.add("localhost:27022"); // Nodo secundario
    rs.add("localhost:27023"); // Nodo secundario

Shard 3:
    mongosh --port 27031
    rs.initiate({
        _id: "shard3",
        members: [
            { _id: 0, host: "localhost:27031" },
        ]
    })
    rs.add("localhost:27032"); // Nodo secundario
    rs.add("localhost:27033"); // Nodo secundario
```

5. **Iniciar el mongos router:** El mongos se utiliza como punto de entrada al clúster. Inicia el mongos y conéctalo a los config servers
   start mongos --configdb configReplSet/localhost:27041,localhost:27042,localhost:27043 --bind_ip localhost --port 27050

6. **Configurar el Sharding:** Conéctate al mongos para configurar los shards
   mongosh --port 27050
   sh.addShard("shard1/localhost:27011,localhost:27012,localhost:27013");
   sh.addShard("shard2/localhost:27021,localhost:27022,localhost:27023");
   sh.addShard("shard3/localhost:27031,localhost:27032,localhost:27033");

7. **Habilitar sharding para la base de datos:**
   sh.enableSharding("evento_deportivo");

8. **Configurar las colecciones para el sharding:** Especifica las claves de partición para cada colección
   // Colección equipos
   sh.shardCollection("evento_deportivo.equipos", { "nombre": 1 });

```
// Colección árbitros
sh.shardCollection("evento_deportivo.arbitros", { "documento": 1 });

// Colección encuentros
sh.shardCollection("evento_deportivo.encuentros", { "fecha": 1, "lugar": 1 });

// Colección resultados
sh.shardCollection("evento_deportivo.resultados", { "encuentro_id": 1 });

// Colección tabla_posiciones
sh.shardCollection("evento_deportivo.tabla_posiciones", { "equipo_id": 1 });
```

9. **Verificar la configuración:** Ejecuta el siguiente comando para verificar el estado del sharding
   `sh.status();`
10. **Realizar la Distribución de los Datos:**
    `sh.splitAt("evento_deportivo.equipos", { "nombre": "M" })`

Casos de Pruebas:

Objetivos de los casos de pruebas:

1. Validar la distribución equitativa de los datos entre shards.
2. Asegurar que las consultas sean eficientes y reflejen las mejoras esperadas.
3. Probar la alta disponibilidad simulando fallos en nodos o shards.
4. Verificar el balanceo de datos tras inserciones masivas.

Desarrollo:

1. **Caso de prueba 1**: Verificar la distribución de datos entre shards
   - **Descripción**: Verifica que los datos están distribuidos correctamente según la clave de particionamiento nombre.
   - **Entradas**:

     `db.equipos.getShardDistribution()`
   - **Resultados esperados**: Los datos están distribuidos en múltiples shards según la clave de particionamiento.
2. **Caso de prueba 2:** Validar el rendimiento de consultas
   - **Descripción**: Mide el tiempo de consulta en una colección fragmentada.
   - **Entradas:**

     `db.equipos.find({ "nombre": "Águilas" })`
   - **Resultados esperados:** La consulta en la colección fragmentada debe ser más rápida, especialmente si los datos son grandes.

3. **Caso de prueba 3:** Verificar alta disponibilidad tras la caída de un shard
    - **Descripción:** Simula la falla de un shard y verifica que los datos permanecen accesibles.
    - **Entradas:**

      Comando: mongod --shutdown en un nodo del shard.

      Comando: db.equipos.find() para consultar los datos.
    - **Resultados esperados:** Los datos siguen siendo accesibles sin errores.

4. **Caso de prueba 4:** Validar el balanceo automático
    - **Descripción:** Inserta datos masivos y verifica que el balanceador redistribuye los documentos.
    - **Entradas:**

      Script de inserción masiva.

      Comando: sh.status() para observar el estado del balanceador.
    - **Resultados esperados:** Los datos se distribuyen uniformemente entre shards.

Reporte resultados:

Caso de Prueba	Resultado Esperado	Resultado Obtenido	Estado
Verificar distribución	Datos distribuidos en múltiples shards.	Correcto	Aprobado
Validar rendimiento	Consulta más rápida en colección fragmentada.	Tiempo:0,05 milisegundos	Aprobado
Verificar alta disponibilidad	Datos accesibles tras falla.	Datos Disponibles	Aprobado
Validar balanceo	Datos redistribuidos automáticamente entre shards.	Correcto	Aprobado

Conclusión:

El ejercicio de configuración y pruebas del sharding en MongoDB permitió simular un entorno distribuido, mostrando cómo manejar grandes volúmenes de datos de manera eficiente en un sistema escalable. A continuación, se detalla el análisis de los principales logros y aprendizajes:

1. **Configuración del Sharding:** Se optó por un particionamiento horizontal basado en el campo nombre, ya que permite dividir de manera lógica y equitativa los datos entre los shards, asegurando que las consultas y la carga de trabajo se distribuyan correctamente.
2. **Inserción de Datos y Resultados:** Se crearon y cargaron registros representativos de equipos deportivos, lo que permitió simular un escenario realista de una base de datos con información.
3. **Casos de Prueba:** Se diseñaron casos para validar la eficiencia del particionamiento.

4. **Reflexión Final:**

   Este ejercicio proporcionó una comprensión práctica del sharding en MongoDB y su capacidad para manejar bases de datos distribuidas. Las pruebas validaron que la configuración cumple con los requisitos de un sistema escalable, eficiente y preparado para manejar grandes volúmenes de datos, garantizando un rendimiento consistente para las operaciones de lectura y escritura en un entorno de alta concurrencia.

Repositorio

https://github.com/juan-ubaque/Torneo-Voleibol-3.git