ACTIVIDAD 3

Conceptos y Comandos básicos del particionamiento en bases de datos NoSQL

ISABEL DELGADO CORREAL, ISMAEL CARVAJAL, Y LIZZET PAOLA BUITRAGO

Introducción

El escenario en el que se necesitaría particionamiento es cuando el torneo deportivo tiene un gran número de participantes y encuentros. En este caso, la cantidad de datos que se deben gestionar y almacenar puede ser considerable, lo que podría afectar el rendimiento del sistema si no se implementa una adecuada estrategia de particionamiento.

El particionamiento consiste en dividir la información en fragmentos más pequeños y distribuirlos en diferentes servidores o bases de datos. Esto permite distribuir la carga de trabajo y mejorar la eficiencia en el acceso a los datos. En el contexto del torneo deportivo, se podrían considerar los siguientes aspectos para el particionamiento:

1. Particionamiento de datos de deportistas, entrenadores y árbitros: Si hay un gran número de participantes en el torneo, es recomendable particionar los datos de deportistas, entrenadores y árbitros en función de algún criterio relevante, como el equipo al que pertenecen o la posición que ocupan. De esta manera, se pueden asignar diferentes particiones a diferentes servidores, lo que facilita la gestión de los datos y mejora el rendimiento en las consultas y actualizaciones.

2. Particionamiento de datos de encuentros y resultados: Considerando que habrá una programación de encuentros deportivos y se registrarán los resultados de cada uno, es conveniente particionar los datos de encuentros y resultados en función de la fecha o período de tiempo en el que se llevan a cabo. Por ejemplo, se puede particionar por meses, temporadas o rondas del torneo. Esto permite distribuir los datos en diferentes particiones y agilizar la consulta y actualización de los resultados.

Especificación de los requerimientos no funcionales para dicho escenario:

1. Escalabilidad: El sistema debe ser capaz de manejar un gran número de participantes, equipos y encuentros sin degradar su rendimiento. El particionamiento de los datos debe garantizar la escalabilidad del sistema al distribuir la carga de trabajo entre múltiples servidores o bases de datos.

2. Seguridad y confidencialidad: El sistema debe garantizar la seguridad de los datos de los participantes, como deportistas, entrenadores y árbitros. Se deben implementar medidas de seguridad, como cifrado de datos, autenticación y control de acceso, para proteger la confidencialidad de la información.

3. Usabilidad: El sistema debe ser intuitivo y de fácil uso para los usuarios. Se debe diseñar una interfaz amigable que permita a los administradores, entrenadores y árbitros gestionar y consultar la información de manera sencilla. Además, se deben proporcionar instrucciones claras y documentación para facilitar la comprensión y el uso del sistema.

4. Robustez y fiabilidad: El sistema debe ser robusto y minimizar el riesgo de fallos o errores. Se deben realizar pruebas exhaustivas de funcionalidad y rendimiento para

garantizar que el sistema pueda manejar situaciones adversas y responder adecuadamente a las solicitudes de los usuarios. Además, se deben implementar mecanismos de copia de seguridad y recuperación de datos para asegurar la disponibilidad de la información en caso de fallos.

5. Tiempo de respuesta rápido: El sistema debe tener un tiempo de respuesta rápido para proporcionar una experiencia fluida a los usuarios. El particionamiento de datos debe contribuir a mejorar el rendimiento del sistema al distribuir la carga de trabajo de manera eficiente y permitir consultas y actualizaciones rápidas.

En resumen, el particionamiento es necesario en un escenario de torneo deportivo con un gran número de participantes y encuentros para garantizar la escalabilidad y el rendimiento del sistema. Los requerimientos no funcionales, como la seguridad, usabilidad, robustez y tiempo de respuesta rápido, deben ser considerados y cumplidos para asegurar el éxito del sistema.

Codigo- instrucciones

Paso 1

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

Paso 2

Verificamos creación carpetas

Paso 3

Db = (new Mongo("localhost:20002")).getdb("torneo_deportivo")

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.getbalancerstate()**
- **True**

- **Sh.isbalancerrunning()**
- **True**

Paso 15

Se para el cluster

- Cluster.stop()

Paso 16

Miramos shard activos

- db.adminCommand({ listShards: 1 })

Repositorio

https://github.com/Lizeth-Buitrago/Bases-de-datos-avanzadas-No_SQL