PARTICIONAMIENTO DE BASES DE DATOS NOSQL

Base de dados NoSQL

SEBASTIAN ARTURO CORTES RODRIGUEZ ID 19930625

YURI CATERINE SALINAS BOLAÑOS ID 20010506

14 DE DICIEMBRE DE 2024

CORPORACIÓN UNIVERSITARIA IBEROAMERCIANA
Bogotá D.C.

Requerimientos No Funcionales y Escenario de Particionamiento

Requerimientos de calidad:

1. Redundancia y Alta Disponibilidad:
   - ReplicaSet con al menos un nodo primario y dos nodos secundarios.
   - Sincronización automática para asegurar consistencia en los datos.
2. Escalabilidad y Tolerancia a Fallos:
   - Configuración de particionamiento horizontal (sharding).
   - Fragmentos distribuidos en múltiples servidores para manejar grandes volúmenes de datos.
   - Nodo Config Server para mantener metadatos del clúster.
3. Escenario para particionamiento horizontal:
   - El torneo involucra consultas intensivas, como estadísticas en tiempo real de los equipos y jugadores.
   - Gran cantidad de partidos y datos históricos hacen que la base de datos supere los límites de almacenamiento de un solo nodo.
   - Distribuir colecciones como **partidos** y **tabla_posiciones** para mejorar el rendimiento.

A continuación se detalla la estrategia de particionamiento:

- Colección objetivo: **partidos** (datos de alto volumen) y **tabla_posiciones** (consultas frecuentes).
- Clave de particionamiento: Fecha (fecha) para partidos y Grupo (grupo) para tabla_posiciones.

Comandos para Configurar el Sharding en MongoDB

A continuación, se detallan los pasos y comandos para implementar el particionamiento:

1. Preparación del entorno
   Se deben tener creados los directorios para los datos de cada nodo dentro de nuestro equipo en la carpeta data del mongodb.

2. Configurar el Replica Set para Config Servers
   Se deben ejecutar los siguientes comandos en diferentes terminales CMD para iniciar cada nodo del Config Server:

   Nodo 1 (Config Server 1):

```
C:\Users\yiso>mongod --configsvr --replSet ConfigRS --dbpath C:\data\configdb1 --port 27019
```

   Nodo 2 (Config Server 2):

```
C:\Windows\System32>mongod --configsvr --replSet ConfigRS --dbpath C:\data\configdb2 --port 27020
```

   Nodo 3 (Config Server 3):

```
C:\Windows\System32>mongod --configsvr --replSet ConfigRS --dbpath C:\data\configdb3 --port 27021
```

   Una vez los config server estén corriendo se debe conectar al nodo primario

```
mongosh --port 27019
```

   Luego de conectado al nodo primario se debe inicializar

```
ConfigRS [direct: primary] test> rs.initiate({
...     _id: "ConfigRS",
...     configsvr: true,
...     members: [
...         { _id: 0, host: "localhost:27019" },
...         { _id: 1, host: "localhost:27020" },
...         { _id: 2, host: "localhost:27021" }
...     ]
... });
```

   Luego se verifica el estado del replicaSet con el comando rs.status()

```
ConfigRS [direct: primary] test> rs.status()
{
  set: 'ConfigRS',
  date: ISODate('2024-12-14T12:18:39.884Z'),
  myState: 1,
  term: Long('1'),
  syncSourceHost: '',
  syncSourceId: -1,
  configsvr: true,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1734178719, i: 1 }), t: Long('1') },
    lastCommittedWallTime: ISODate('2024-12-14T12:18:39.813Z'),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1734178719, i: 1 }), t: Long('1') },
    appliedOpTime: { ts: Timestamp({ t: 1734178719, i: 1 }), t: Long('1') },
    durableOpTime: { ts: Timestamp({ t: 1734178718, i: 1 }), t: Long('1') },
    writtenOpTime: { ts: Timestamp({ t: 1734178719, i: 1 }), t: Long('1') },
    lastAppliedWallTime: ISODate('2024-12-14T12:18:39.813Z'),
    lastDurableWallTime: ISODate('2024-12-14T12:18:38.800Z'),
    lastWrittenWallTime: ISODate('2024-12-14T12:18:39.813Z')
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1734178695, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
    lastElectionDate: ISODate('2024-12-14T12:11:26.599Z'),
    electionTerm: Long('1'),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1734178275, i: 1 }), t: Long('-1') },
    lastSeenWrittenOpTimeAtElection: { ts: Timestamp({ t: 1734178275, i: 1 }), t: Long('-1') },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1734178275, i: 1 }), t: Long('-1') },
    numVotesNeeded: 2,
    priorityAtElection: 1,
    electionTimeoutMillis: Long('10000'),
    numCatchUpOps: Long('0'),
    newTermStartDate: ISODate('2024-12-14T12:11:26.664Z'),
    wMajorityWriteAvailabilityDate: ISODate('2024-12-14T12:11:27.148Z')
  },
  members: [
    {
      _id: 0,
      name: 'localhost:27019',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 1073,
      optime: { ts: Timestamp({ t: 1734178719, i: 1 }), t: Long('1') },
      optimeDate: ISODate('2024-12-14T12:18:39.000Z'),
      optimeWritten: { ts: Timestamp({ t: 1734178719, i: 1 }), t: Long('1') },
      optimeWrittenDate: ISODate('2024-12-14T12:18:39.000Z'),
      lastAppliedWallTime: ISODate('2024-12-14T12:18:39.813Z'),
      lastDurableWallTime: ISODate('2024-12-14T12:18:38.800Z'),
      lastWrittenWallTime: ISODate('2024-12-14T12:18:39.813Z'),
      syncSourceHost: '',
      syncSourceId: -1,
      infoMessage: '',
      electionTime: Timestamp({ t: 1734178286, i: 1 }),
      electionDate: ISODate('2024-12-14T12:11:26.000Z'),
      configVersion: 1,
      configTerm: 1,
      self: true,
      lastHeartbeatMessage: ''
    },
```

```
  {
    _id: 1,
    name: 'localhost:27020',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 444,
    optime: { ts: Timestamp({ t: 1734178717, i: 1 }), t: Long('1') },
    optimeDurable: { ts: Timestamp({ t: 1734178717, i: 1 }), t: Long('1') },
    optimeWritten: { ts: Timestamp({ t: 1734178717, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2024-12-14T12:18:37.000Z'),
    optimeDurableDate: ISODate('2024-12-14T12:18:37.000Z'),
    optimeWrittenDate: ISODate('2024-12-14T12:18:37.000Z'),
    lastAppliedWallTime: ISODate('2024-12-14T12:18:39.813Z'),
    lastDurableWallTime: ISODate('2024-12-14T12:18:39.813Z'),
    lastWrittenWallTime: ISODate('2024-12-14T12:18:39.813Z'),
    lastHeartbeat: ISODate('2024-12-14T12:18:38.789Z'),
    lastHeartbeatRecv: ISODate('2024-12-14T12:18:39.860Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: 'localhost:27019',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 1,
    configTerm: 1
  },
```

```
  {
    _id: 2,
    name: 'localhost:27021',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 444,
    optime: { ts: Timestamp({ t: 1734178717, i: 1 }), t: Long('1') },
    optimeDurable: { ts: Timestamp({ t: 1734178717, i: 1 }), t: Long('1') },
    optimeWritten: { ts: Timestamp({ t: 1734178717, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2024-12-14T12:18:37.000Z'),
    optimeDurableDate: ISODate('2024-12-14T12:18:37.000Z'),
    optimeWrittenDate: ISODate('2024-12-14T12:18:37.000Z'),
    lastAppliedWallTime: ISODate('2024-12-14T12:18:39.813Z'),
    lastDurableWallTime: ISODate('2024-12-14T12:18:39.813Z'),
    lastWrittenWallTime: ISODate('2024-12-14T12:18:39.813Z'),
    lastHeartbeat: ISODate('2024-12-14T12:18:38.788Z'),
    lastHeartbeatRecv: ISODate('2024-12-14T12:18:39.860Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: 'localhost:27019',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 1,
    configTerm: 1
  }
],
ok: 1,
'$clusterTime': {
  clusterTime: Timestamp({ t: 1734178719, i: 1 }),
  signature: {
    hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
    keyId: Long('0')
  }
},
operationTime: Timestamp({ t: 1734178719, i: 1 })
}
configRS [direct: primary] test>
```

Como siguiente paso se deben configurar los nodos Shard
Crea un Replica Set para cada shard. Aquí vamos a usar tres shards. Ejecuta los siguientes comandos en diferentes terminales CMD:

Shard 1, Nodo 1:

```
C:\Windows\System32>mongod --shardsvr --replSet ShardRS1 --dbpath C:\data\shard1 --port 27018
```

Shard 2, Nodo 1:

```
C:\Windows\System32>mongod --shardsvr --replSet ShardRS2 --dbpath C:\data\shard2 --port 27028
```

Shard 3, Nodo 1:

```
C:\Windows\System32>mongod --shardsvr --replSet ShardRS3 --dbpath C:\data\shard3 --port 27038
```

Inicializar cada shards

```
C:\Windows\System32>mongosh --port 27018
Current Mongosh Log ID: 675d7969dfbc79fea2893bf7
Connecting to:          mongodb://127.0.0.1:27018/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.3.4
Using MongoDB:          8.0.3
Using Mongosh:          2.3.4
mongosh 2.3.6 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-12-14T07:23:54.611-05:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
   2024-12-14T07:23:54.612-05:00: This server is bound to localhost. Remote systems will be unable to connect to this se
rver. Start the server with --bind_ip <address> to specify which IP addresses it should serve responses from, or with --
bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.0.0.1 to disable
 this warning
------

test>
```

Se inicia el shards

```
test> rs.initiate({
...     _id: "ShardRS1",
...     members: [
...         { _id: 0, host: "localhost:27018" }
...     ]
... });
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734179211, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734179211, i: 1 })
}
ShardRS1 [direct: secondary] test> rs.reconfig({
```

Cuando se inicia el shards1 podemos observar que el nodo localhost:27018 pertenece al Replica Set ShardRS1 pero está en estado secondary. Esto puede suceder porque MongoDB aún no ha seleccionado un nodo primario o porque este nodo no tiene suficientes miembros para promoverse automáticamente como primario.
Con el siguiente comando pasa a primario

```
ShardRS1 [direct: secondary] test> rs.reconfig({
...     _id: "ShardRS1",
...     members: [
...         { _id: 0, host: "localhost:27018", priority: 1 }
...     ]
... }, { force: true });
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734179431, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734179431, i: 1 })
}
```

Se verifica el cambio con rs.status()

```
  {
    _id: 0,
    name: 'localhost:27018',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 409,
    optime: { ts: Timestamp({ t: 1734179441, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2024-12-14T12:30:41.000Z'),
    optimeWritten: { ts: Timestamp({ t: 1734179441, i: 1 }), t: Long('1') },
    optimeWrittenDate: ISODate('2024-12-14T12:30:41.000Z'),
    lastAppliedWallTime: ISODate('2024-12-14T12:30:41.938Z'),
    lastDurableWallTime: ISODate('2024-12-14T12:30:41.938Z'),
    lastWrittenWallTime: ISODate('2024-12-14T12:30:41.938Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1734179211, i: 2 }),
    electionDate: ISODate('2024-12-14T12:26:51.000Z'),
    configVersion: 58429,
    configTerm: -1,
    self: true,
    lastHeartbeatMessage: ''
  }
],
ok: 1,
'$clusterTime': {
  clusterTime: Timestamp({ t: 1734179441, i: 1 }),
  signature: {
    hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
    keyId: Long('0')
  }
},
operationTime: Timestamp({ t: 1734179441, i: 1 })
```

Una vez hecho esto se sigue con la configuración de los demás shards2
Conectarse al puerto 27028:

```
C:\Windows\System32>mongosh --port 27028
```

Se inicializa

```
test> rs.initiate({
...     _id: "ShardRS2",
...     members: [
...         { _id: 0, host: "localhost:27028" }
...     ]
... });
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734180269, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734180269, i: 1 })
}
ShardRS2 [direct: secondary] test>
```

Se verifica el estado

```
ShardRS2 [direct: secondary] test> rs.status()
{
  set: 'ShardRS2',
  date: ISODate('2024-12-14T12:45:34.823Z'),
  myState: 1,
  term: Long('1'),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 1,
  writeMajorityCount: 1,
  votingMembersCount: 1,
  writableVotingMembersCount: 1,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1734180329, i: 1 }), t: Long('1') },
    lastCommittedWallTime: ISODate('2024-12-14T12:45:29.845Z'),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1734180329, i: 1 }), t: Long('1') },
    appliedOpTime: { ts: Timestamp({ t: 1734180329, i: 1 }), t: Long('1') },
    durableOpTime: { ts: Timestamp({ t: 1734180329, i: 1 }), t: Long('1') },
    writtenOpTime: { ts: Timestamp({ t: 1734180329, i: 1 }), t: Long('1') },
    lastAppliedWallTime: ISODate('2024-12-14T12:45:29.845Z'),
    lastDurableWallTime: ISODate('2024-12-14T12:45:29.845Z'),
    lastWrittenWallTime: ISODate('2024-12-14T12:45:29.845Z')
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1734180319, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
    lastElectionDate: ISODate('2024-12-14T12:44:29.783Z'),
    electionTerm: Long('1'),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1734180269, i: 1 }), t: Long('-1') },
    lastSeenWrittenOpTimeAtElection: { ts: Timestamp({ t: 1734180269, i: 1 }), t: Long('-1') },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1734180269, i: 1 }), t: Long('-1') },
    numVotesNeeded: 1,
    priorityAtElection: 1,
    electionTimeoutMillis: Long('10000'),
    newTermStartDate: ISODate('2024-12-14T12:44:29.812Z'),
    wMajorityWriteAvailabilityDate: ISODate('2024-12-14T12:44:29.840Z')
  },
```

```
  members: [
    {
      _id: 0,
      name: 'localhost:27028',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 1227,
      optime: { ts: Timestamp({ t: 1734180329, i: 1 }), t: Long('1') },
      optimeDate: ISODate('2024-12-14T12:45:29.000Z'),
      optimeWritten: { ts: Timestamp({ t: 1734180329, i: 1 }), t: Long('1') },
      optimeWrittenDate: ISODate('2024-12-14T12:45:29.000Z'),
      lastAppliedWallTime: ISODate('2024-12-14T12:45:29.845Z'),
      lastDurableWallTime: ISODate('2024-12-14T12:45:29.845Z'),
      lastWrittenWallTime: ISODate('2024-12-14T12:45:29.845Z'),
      syncSourceHost: '',
      syncSourceId: -1,
      infoMessage: 'Could not find member to sync from',
      electionTime: Timestamp({ t: 1734180269, i: 2 }),
      electionDate: ISODate('2024-12-14T12:44:29.000Z'),
      configVersion: 1,
      configTerm: 1,
      self: true,
      lastHeartbeatMessage: ''
    }
  ],
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734180329, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734180329, i: 1 })
}
ShardRS2 [direct: primary] test>
```

Configuración de ShardRS3

Conectarse al puerto 27038:

```
C:\Windows\System32>mongosh --port 27038
```

Se inicializa

```
test> rs.initiate({
...     _id: "ShardRS3",
...     members: [
...         { _id: 0, host: "localhost:27038" }
...     ]
... });
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734180607, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734180607, i: 1 })
}
ShardRS3 [direct: other] test>
```

Se verifica el estado

```
ShardRS3 [direct: other] test> rs.status()
{
  set: 'ShardRS3',
  date: ISODate('2024-12-14T12:52:54.767Z'),
  myState: 1,
  term: Long('1'),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 1,
  writeMajorityCount: 1,
  votingMembersCount: 1,
  writableVotingMembersCount: 1,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1734180767, i: 1 }), t: Long('1') },
    lastCommittedWallTime: ISODate('2024-12-14T12:52:47.414Z'),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1734180767, i: 1 }), t: Long('1') },
    appliedOpTime: { ts: Timestamp({ t: 1734180767, i: 1 }), t: Long('1') },
    durableOpTime: { ts: Timestamp({ t: 1734180767, i: 1 }), t: Long('1') },
    writtenOpTime: { ts: Timestamp({ t: 1734180767, i: 1 }), t: Long('1') },
    lastAppliedWallTime: ISODate('2024-12-14T12:52:47.414Z'),
    lastDurableWallTime: ISODate('2024-12-14T12:52:47.414Z'),
    lastWrittenWallTime: ISODate('2024-12-14T12:52:47.414Z')
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1734180717, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
    lastElectionDate: ISODate('2024-12-14T12:50:07.339Z'),
    electionTerm: Long('1'),
ShardRS3 [direct: primary] test>
    lastSeenWrittenOpTimeAtElection: { ts: Timestamp({ t: 1734180607, i: 1 }), t: Long('-1') },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1734180607, i: 1 }), t: Long('-1') },
    numVotesNeeded: 1,
    priorityAtElection: 1,
    electionTimeoutMillis: Long('10000'),
    newTermStartDate: ISODate('2024-12-14T12:50:07.377Z'),
    wMajorityWriteAvailabilityDate: ISODate('2024-12-14T12:50:07.402Z')
  },
  members: [
```

```
  members: [
    {
      _id: 0,
      name: 'localhost:27038',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 1640,
      optime: { ts: Timestamp({ t: 1734180767, i: 1 }), t: Long('1') },
      optimeDate: ISODate('2024-12-14T12:52:47.000Z'),
      optimeWritten: { ts: Timestamp({ t: 1734180767, i: 1 }), t: Long('1') },
      optimeWrittenDate: ISODate('2024-12-14T12:52:47.000Z'),
      lastAppliedWallTime: ISODate('2024-12-14T12:52:47.414Z'),
      lastDurableWallTime: ISODate('2024-12-14T12:52:47.414Z'),
      lastWrittenWallTime: ISODate('2024-12-14T12:52:47.414Z'),
      syncSourceHost: '',
      syncSourceId: -1,
      infoMessage: '',
      electionTime: Timestamp({ t: 1734180607, i: 2 }),
      electionDate: ISODate('2024-12-14T12:50:07.000Z'),
      configVersion: 1,
      configTerm: 1,
      self: true,
      lastHeartbeatMessage: ''
    }
  ],
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734180767, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734180767, i: 1 })
}
ShardRS3 [direct: primary] test>
```

Una vez estén configurados los los tres shards, se debe iniciar el Router para gestionar las conexiones

Ejecuta el siguiente comando en un nuevo terminal CMD. Iniciar el Router. Este comando conecta el Router al Config Server.

```
mongos --configdb ConfigRS/localhost:27019,localhost:27020,localhost:27021 --port 27017
```

El siguiente paso es conectarse al Router (mongos):

```
mongosh --port 27017
```

Validar el status

```
[direct: mongos] test> sh.status()
shardingVersion
{ _id: 1, clusterId: ObjectId('675d75eed6ca5783fd6c0fc9') }
---
shards
[]
---
active mongoses
[]
---
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
  'Currently enabled': 'yes',
  'Currently running': 'no',
  'Failed balancer rounds in last 5 attempts': 0,
  'Migration Results for the last 24 hours': 'No recent migrations'
}
---
shardedDataDistribution
[]
---
databases
[
  {
    database: { _id: 'config', primary: 'config', partitioned: true },
    collections: {}
  }
]
[direct: mongos] test>
```

Luego se deben agregar cada shard al clúster usando los siguientes comandos en el shell interactivo:

```
[direct: mongos] test> sh.addShard("ShardRS1/localhost:27018");
{
  shardAdded: 'ShardRS1',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734183975, i: 20 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734183975, i: 20 })
}
[direct: mongos] test> sh.addShard("ShardRS2/localhost:27028");
{
  shardAdded: 'ShardRS2',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734183975, i: 43 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734183975, i: 37 })
}
[direct: mongos] test> sh.addShard("ShardRS3/localhost:27038");
{
  shardAdded: 'ShardRS3',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734183976, i: 17 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734183976, i: 17 })
}
[direct: mongos] test>
```

Validar el estado

```
[direct: mongos] test> sh.status()
shardingVersion
{ _id: 1, clusterId: ObjectId('675d75eed6ca5783fd6c0fc9') }
---
shards
[
  {
    _id: 'ShardRS1',
    host: 'ShardRS1/localhost:27018',
    state: 1,
    topologyTime: Timestamp({ t: 1734183975, i: 10 }),
    replSetConfigVersion: Long('-1')
  },
  {
    _id: 'ShardRS2',
    host: 'ShardRS2/localhost:27028',
    state: 1,
    topologyTime: Timestamp({ t: 1734183975, i: 28 }),
    replSetConfigVersion: Long('-1')
  },
  {
    _id: 'ShardRS3',
    host: 'ShardRS3/localhost:27038',
    state: 1,
    topologyTime: Timestamp({ t: 1734183976, i: 8 }),
    replSetConfigVersion: Long('-1')
  }
]
---
active mongoses
[ { '8.0.3': 1 } ]
---
autosplit
{ 'Currently enabled': 'yes' }
---
```

```
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
  'Currently enabled': 'yes',
  'Currently running': 'no',
  'Failed balancer rounds in last 5 attempts': 0,
  'Migration Results for the last 24 hours': 'No recent migrations'
}
---
shardedDataDistribution
[]
---
databases
[
  {
    database: { _id: 'config', primary: 'config', partitioned: true },
    collections: {}
  }
]
[direct: mongos] test>
```

Una vez verificamos que los shards (ShardRS1, ShardRS2, ShardRS3) se han agregado correctamente al clúster de sharding, y el comando sh.status() confirma que están en estado state: 1, lo cual significa que están funcionando como se espera.

Ahora, vamos a habilitar el sharding en la base de datos y particionar las colecciones.

```
[direct: mongos] test> use TorneoDeportivo
switched to db TorneoDeportivo
[direct: mongos] TorneoDeportivo> db.arbitros
```

Esto prepara la base de datos para que las colecciones puedan distribuirse entre los shards.

Luego se debe habilitar el sharding para la base de datos TorneoDeportivo, permitiendo que las colecciones dentro de esta base de datos puedan ser particionadas.

```
[direct: mongos] TorneoDeportivo> sh.enableSharding("TorneoDeportivo");
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734185040, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734185040, i: 1 })
}
[direct: mongos] TorneoDeportivo>
```

Particionar las colecciones

- Colección partidos
  Razón del particionamiento:
    - Contiene datos de alto volumen.
    - Se accede frecuentemente por rangos de fechas para estadísticas históricas y recientes.

  Clave de particionamiento:

    - Campo: fecha.
    - Tipo de consulta esperada: Rango de fechas (e.g., partidos en un período de tiempo específico).

Se debe crear un índice a la colección partidos con el campo fecha para realizar el particionamiento

```
[direct: mongos] TorneoDeportivo> db.partidos.createIndex({ fecha: 1 });
fecha_1
[direct: mongos] TorneoDeportivo>
```

Una vez que el índice esté creado, se debe particionar la colección partidos:

```
[direct: mongos] TorneoDeportivo> sh.shardCollection("TorneoDeportivo.partidos", { fecha: 1 });
{
  collectionsharded: 'TorneoDeportivo.partidos',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734185763, i: 39 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734185763, i: 39 })
}
[direct: mongos] TorneoDeportivo>
```

- Colección tabla_posiciones
  Clave de particionamiento:
    - Campo: grupo
    - Razón: Los datos suelen ser consultados y agrupados por el grupo al que pertenece un equipo, lo que permite una distribución eficiente.

Crear el índice en el campo grupo:

Ejecuta este comando para crear el índice necesario:

```
[direct: mongos] TorneoDeportivo> db.tabla_posiciones.createIndex({ grupo: 1 });
grupo_1
[direct: mongos] TorneoDeportivo>
```

Particionar la colección:

Una vez creado el índice, particiona la colección utilizando la clave grupo:

```
[direct: mongos] TorneoDeportivo> sh.shardCollection("TorneoDeportivo.tabla_posiciones", { grupo: 1 })
{
  collectionsharded: 'TorneoDeportivo.tabla_posiciones',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734186025, i: 39 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734186025, i: 38 })
}
[direct: mongos] TorneoDeportivo>
```

- Colección jugadores

  Clave de particionamiento:

  - Campo: equipo
  - Razón: Si las consultas suelen realizarse agrupando a los jugadores por equipo (por ejemplo, estadísticas de un equipo específico), esta clave es adecuada.

Crear el índice en el campo equipo:

Ejecuta este comando para crear el índice necesario:

```
[direct: mongos] TorneoDeportivo> db.jugadores.createIndex({ equipo: 1 });
equipo_1
[direct: mongos] TorneoDeportivo>
```

Una vez creado el índice, particiona la colección utilizando la clave equipo:

```
[direct: mongos] TorneoDeportivo> sh.shardCollection("TorneoDeportivo.jugadores", { equipo: 1 });
{
  collectionsharded: 'TorneoDeportivo.jugadores',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734186136, i: 14 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734186136, i: 13 })
}
[direct: mongos] TorneoDeportivo>
```

Una vez este todo configurado procedemos a verificar el estado

```
[direct: mongos] TorneoDeportivo> sh.status();
shardingVersion
{ _id: 1, clusterId: ObjectId('675d75eed6ca5783fd6c0fc9') }
---
shards
[
  {
    _id: 'ShardRS1',
    host: 'ShardRS1/localhost:27018',
    state: 1,
    topologyTime: Timestamp({ t: 1734183975, i: 10 }),
    replSetConfigVersion: Long('-1')
  },
  {
    _id: 'ShardRS2',
    host: 'ShardRS2/localhost:27028',
    state: 1,
    topologyTime: Timestamp({ t: 1734183975, i: 28 }),
    replSetConfigVersion: Long('-1')
  },
  {
    _id: 'ShardRS3',
    host: 'ShardRS3/localhost:27038',
    state: 1,
    topologyTime: Timestamp({ t: 1734183976, i: 8 }),
    replSetConfigVersion: Long('-1')
  }
]
---
active mongoses
[ { '8.0.3': 1 } ]
---
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
```

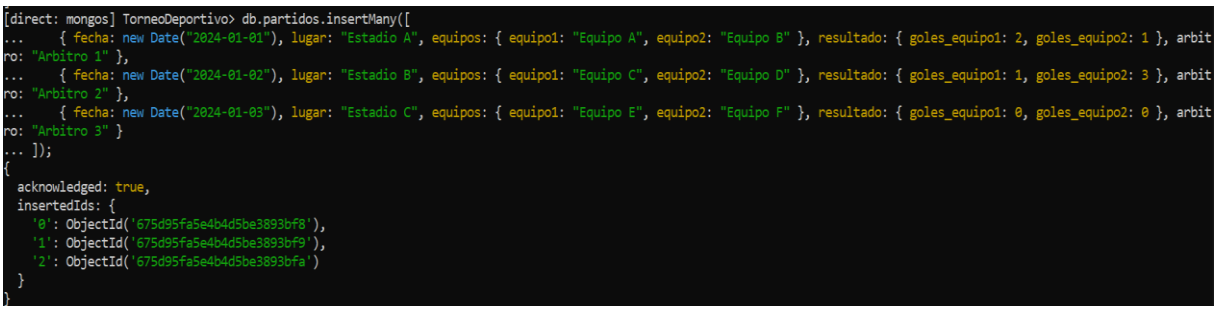
Colecciones

```
shardedDataDistribution
[
  {
    ns: 'TorneoDeportivo.jugadores',
    shards: [
      {
        shardName: 'ShardRS3',
        numOrphanedDocs: 0,
        numOwnedDocuments: 1,
        ownedSizeBytes: 200,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'config.system.sessions',
    shards: [
      {
        shardName: 'ShardRS1',
        numOrphanedDocs: 0,
        numOwnedDocuments: 28,
        ownedSizeBytes: 2772,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'TorneoDeportivo.tabla_posiciones',
    shards: [
      {
        shardName: 'ShardRS3',
        numOrphanedDocs: 0,
        numOwnedDocuments: 1,
        ownedSizeBytes: 198,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'TorneoDeportivo.partidos',
    shards: [
      {
        shardName: 'ShardRS3',
        numOrphanedDocs: 0,
        numOwnedDocuments: 1,
        ownedSizeBytes: 215,
        orphanedSizeBytes: 0
      }
    ]
  }
]
---
```

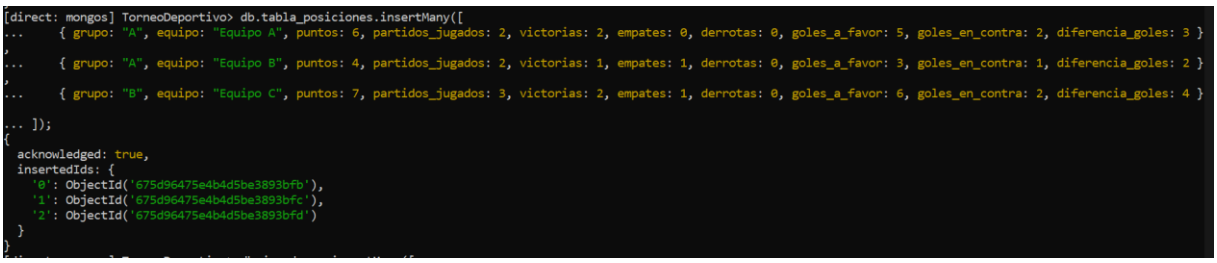
Una vez se verifica que la configuración de sharding está funcionando correctamente según el estado mostrado en sh.status().

Se procese a insertar datos en las colecciones particionadas con el fin de observar cómo los datos se distribuyen entre los shards, inserta más documentos en las colecciones particionadas:

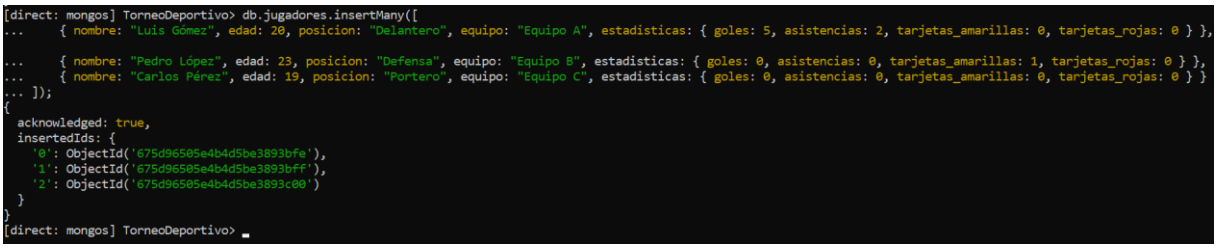
Colección partidos:



Colección posiciones



Colección jugadores



Ahora vamos a verificar como los datos están siendo distribuidos entre los shards.

Para ello se debe ejecutar el comando

```
sh.status();
```

El comando nos arroja el siguiente resultado

```
[direct: mongos] TorneoDeportivo> sh.status();
shardingVersion
{ _id: 1, clusterId: ObjectId('675d75eed6ca5783fd6c0fc9') }
---
shards
[
  {
    _id: 'ShardRS1',
    host: 'ShardRS1/localhost:27018',
    state: 1,
    topologyTime: Timestamp({ t: 1734183975, i: 10 }),
    replSetConfigVersion: Long('-1')
  },
  {
    _id: 'ShardRS2',
    host: 'ShardRS2/localhost:27028',
    state: 1,
    topologyTime: Timestamp({ t: 1734183975, i: 28 }),
    replSetConfigVersion: Long('-1')
  },
  {
    _id: 'ShardRS3',
    host: 'ShardRS3/localhost:27038',
    state: 1,
    topologyTime: Timestamp({ t: 1734183976, i: 8 }),
    replSetConfigVersion: Long('-1')
  }
]
---
active mongoses
[ { '8.0.3': 1 } ]
---
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
  'Currently enabled': 'yes',
  'Currently running': 'no',
  'Failed balancer rounds in last 5 attempts': 0,
```

```
shardedDataDistribution
[
  {
    ns: 'TorneoDeportivo.jugadores',
    shards: [
      {
        shardName: 'ShardRS3',
        numOrphanedDocs: 0,
        numOwnedDocuments: 4,
        ownedSizeBytes: 772,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'config.system.sessions',
    shards: [
      {
        shardName: 'ShardRS1',
        numOrphanedDocs: 0,
        numOwnedDocuments: 32,
        ownedSizeBytes: 3168,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
    ns: 'TorneoDeportivo.tabla_posiciones',
    shards: [
      {
        shardName: 'ShardRS3',
        numOrphanedDocs: 0,
        numOwnedDocuments: 4,
        ownedSizeBytes: 780,
        orphanedSizeBytes: 0
      }
    ]
  },
  {
```

```
  {
    ns: 'TorneoDeportivo.partidos',
    shards: [
      {
        shardName: 'ShardRS3',
        numOrphanedDocs: 0,
        numOwnedDocuments: 4,
        ownedSizeBytes: 792,
        orphanedSizeBytes: 0
      }
    ]
  }
]
---
databases
[
  {
    database: { _id: 'config', primary: 'config', partitioned: true },
    collections: {
      'config.system.sessions': {
        shardKey: { _id: 1 },
        unique: false,
        balancing: true,
        chunkMetadata: [ { shard: 'ShardRS1', nChunks: 1 } ],
        chunks: [
          { min: { _id: MinKey() }, max: { _id: MaxKey() }, 'on shard': 'ShardRS1', 'last modified': Timestamp({ t: 1, i: 0 }) }
        ],
        tags: []
      }
    }
  },
  {
    database: {
      _id: 'TorneoDeportivo',
      primary: 'ShardRS3',
      version: {
        uuid: UUID('50e88ee4-fd6a-492e-b8b2-20c8ea621f31'),
        timestamp: Timestamp({ t: 1734184173, i: 2 }),
        lastMod: 1
      }
```

```
    },
    collections: {
      'TorneoDeportivo.jugadores': {
        shardKey: { equipo: 1 },
        unique: false,
        balancing: true,
        chunkMetadata: [ { shard: 'ShardRS3', nChunks: 1 } ],
        chunks: [
          { min: { equipo: MinKey() }, max: { equipo: MaxKey() }, 'on shard': 'ShardRS3', 'last modified': Timestamp({ t: 1, i: 0 }) }
        ],
        tags: []
      },
      'TorneoDeportivo.partidos': {
        shardKey: { fecha: 1 },
        unique: false,
        balancing: true,
        chunkMetadata: [ { shard: 'ShardRS3', nChunks: 1 } ],
        chunks: [
          { min: { fecha: MinKey() }, max: { fecha: MaxKey() }, 'on shard': 'ShardRS3', 'last modified': Timestamp({ t: 1, i: 0 }) }
        ],
        tags: []
      },
      'TorneoDeportivo.tabla_posiciones': {
        shardKey: { grupo: 1 },
        unique: false,
        balancing: true,
        chunkMetadata: [ { shard: 'ShardRS3', nChunks: 1 } ],
        chunks: [
          { min: { grupo: MinKey() }, max: { grupo: MaxKey() }, 'on shard': 'ShardRS3', 'last modified': Timestamp({ t: 1, i: 0 }) }
        ],
        tags: []
      }
    }
  }
]
[direct: mongos] TorneoDeportivo>
```

El estado actual de sh.status() muestra que las colecciones están correctamente particionadas, pero todos los datos permanecen en un solo shard (ShardRS3). Esto puede suceder porque:

Volumen de datos insuficiente: El balanceador de MongoDB no redistribuye datos si no hay suficiente volumen para justificar la creación de nuevos chunks en otros shards.

Chunks iniciales no repartidos: MongoDB crea un único chunk inicialmente por colección, y este se encuentra asignado al ShardRS3.

Para forzar el balanceo de datos entre los shards, se deben realizar los siguientes pasos:

1. Validar que el balanceador esté habilitado

```
balancer
{
  'Currently enabled': 'yes',
  'Currently running': 'no',
  'Failed balancer rounds in last 5 attempts': 0,
  'Migration Results for the last 24 hours': 'No recent migrations'
}
```

El balanceador está habilitado según sh.status() (Currently enabled: yes), pero no está corriendo (Currently running: no). Para forzar su ejecución, se usa:

```
[direct: mongos] TorneoDeportivo> sh.startBalancer();
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734187411, i: 4 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734187411, i: 4 })
}
[direct: mongos] TorneoDeportivo>
```

Después de ejecutarlo, se debe verificar si el balanceador comienza a redistribuir datos con:

```
[direct: mongos] TorneoDeportivo> sh.isBalancerRunning();
{
  mode: 'full',
  inBalancerRound: false,
  numBalancerRounds: Long('388'),
  term: Long('2'),
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734187451, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734187451, i: 1 })
}
[direct: mongos] TorneoDeportivo>
```

El resultado indica que el balanceador está habilitado pero no está ejecutando un proceso de redistribución actualmente (inBalancerRound: false). Esto significa que el balanceador no encuentra condiciones para iniciar una nueva ronda de balanceo.

Lo que vamos a hacer es redistribuir los chunks manualmente
Se debe usar el comando sh.moveChunk para redistribuir los chunks manualmente.

- Para TorneoDeportivo.partidos:

```
[direct: mongos] TorneoDeportivo> sh.moveChunk("TorneoDeportivo.partidos", { fecha: ISODate("2024-01-01") }, "ShardRS1");
{
  millis: 439,
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734188611, i: 45 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734188611, i: 45 })
}
[direct: mongos] TorneoDeportivo> sh.moveChunk("TorneoDeportivo.partidos", { fecha: ISODate("2024-01-03") }, "ShardRS2");
{
  millis: 444,
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734188612, i: 25 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734188612, i: 25 })
}
[direct: mongos] TorneoDeportivo>
```

Verificamos si la distribución de chunks quedo correctamente configurada con el comando Sh.status();
Resultado

```
shardedDataDistribution
[
  {
    ns: 'TorneoDeportivo.partidos',
    shards: [
      {
        shardName: 'ShardRS1',
        numOrphanedDocs: 0,
        numOwnedDocuments: 1,
        ownedSizeBytes: 193,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'ShardRS2',
        numOrphanedDocs: 0,
        numOwnedDocuments: 2,
        ownedSizeBytes: 408,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'ShardRS3',
        numOrphanedDocs: 3,
        numOwnedDocuments: 1,
        ownedSizeBytes: 198,
        orphanedSizeBytes: 594
      }
    ]
  },
```

- Para tabla_posiciones:

```
[direct: mongos] TorneoDeportivo> sh.moveChunk("TorneoDeportivo.tabla_posiciones", { grupo: "A" }, "ShardRS1");
{
  millis: 415,
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734188828, i: 7 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734188828, i: 7 })
}
[direct: mongos] TorneoDeportivo> sh.moveChunk("TorneoDeportivo.tabla_posiciones", { grupo: "C" }, "ShardRS2");
{
  millis: 225,
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734188828, i: 41 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734188828, i: 41 })
}
[direct: mongos] TorneoDeportivo>
```

Verificamos el estado con sh.status();

```
  {
    ns: 'TorneoDeportivo.tabla_posiciones',
    shards: [
      {
        shardName: 'ShardRS1',
        numOrphanedDocs: 0,
        numOwnedDocuments: 3,
        ownedSizeBytes: 585,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'ShardRS2',
        numOrphanedDocs: 0,
        numOwnedDocuments: 0,
        ownedSizeBytes: 0,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'ShardRS3',
        numOrphanedDocs: 3,
        numOwnedDocuments: 1,
        ownedSizeBytes: 195,
        orphanedSizeBytes: 585
      }
    ]
  },
```

- Para jugadores:

```
[direct: mongos] TorneoDeportivo> sh.moveChunk("TorneoDeportivo.jugadores", { equipo: "Equipo A" }, "ShardRS1");
{
  millis: 376,
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734188968, i: 27 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734188968, i: 27 })
}
[direct: mongos] TorneoDeportivo> sh.moveChunk("TorneoDeportivo.jugadores", { equipo: "Equipo C" }, "ShardRS2");
{
  millis: 304,
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734188968, i: 65 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1734188968, i: 65 })
}
[direct: mongos] TorneoDeportivo>
```

Verificamos el estado con sh.status();

```
shardedDataDistribution
[
  {
    ns: 'TorneoDeportivo.jugadores',
    shards: [
      {
        shardName: 'ShardRS3',
        numOrphanedDocs: 3,
        numOwnedDocuments: 1,
        ownedSizeBytes: 193,
        orphanedSizeBytes: 579
      },
      {
        shardName: 'ShardRS2',
        numOrphanedDocs: 0,
        numOwnedDocuments: 2,
        ownedSizeBytes: 392,
        orphanedSizeBytes: 0
      },
      {
        shardName: 'ShardRS1',
        numOrphanedDocs: 0,
        numOwnedDocuments: 1,
        ownedSizeBytes: 192,
        orphanedSizeBytes: 0
      }
    ]
  },
```