NoSQL - MongoDB

Rafael Garrote Hernández
Profesor en NoSQL

Introducción

- Datastore orientado a documentos:
  - Schema less
  - Esquema flexible
- High Performance
  - Soporta modelos de datos embebidos que reducen la actividad de entrada y salida.
  - El sistema de índices acelera las consultas y soporta el indexado de documentos embebidos y arrays.
- Alta disponibilidad
  - Replicación (Replica Set).
  - Automatic failover
  - Redundancia de datos
- Escalado Horizontal
  - Particionado (Sharding).
  - Zonas

Introducción

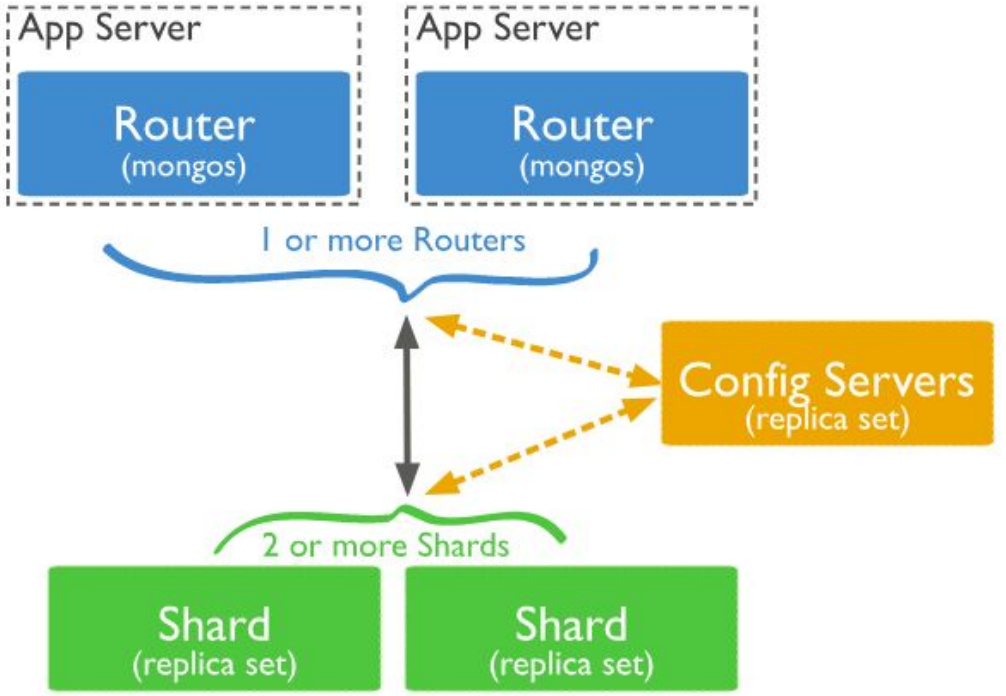
- Lenguaje de consulta potente:
- CRUD
- Agregaciones
- Búsqueda full-text
- Búsqueda geo espacial
- Soporta varios motores de almacenamiento:
  - WiredTiger Storage Engine
  - In-Memory Storage Engine
  - Plugable Storage Engine
- Dos modos de funcionamiento:
  - **Standalone**: Sólo se instala un servicio de MongoDB que aloja el conjunto completo de datos
  - **Sharded Cluster**: Particionado, replicación y tolerancia a fallos.

Particionado

Alta disponibilidad - Particionado

Sharded Cluster: Un cluster de MongoDB con particionado consta de los siguientes componentes:

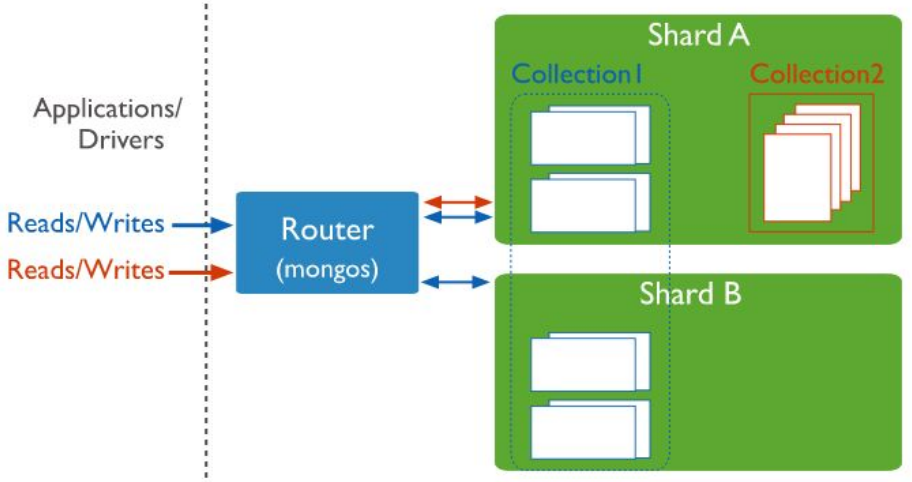
- **Shard**: Cada una de las particiones de los datos. Cada Shard se puede desplegar como un replica set (varios servicios de mongo alojando las distintas réplicas de la partición).
- **mongos**: Actúa como un enrutador de sentencias a modo de interfaz entre los clientes y el cluster particionado.
- **Servidores de configuración**: Almacenan los metadatos de configuración del cluster. Deben ser desplegados como un Replica Set.



Alta disponibilidad - Particionado

Shard: Contiene un subconjunto de los datos compartidos por un cluster particionado. Todos los cluster shards juntos contienen el conjunto total de datos del cluster.

- Cada **base de datos** en un cluster particionado tiene una partición primaria.
- La partición primaria no tiene relación con el nodo primario.
- La partición primaria mantiene también las colecciones no particionadas.
- mongos selecionan la partición primaria al crear la base de datos seleccionado el shard del cluster que tenga menos datos.



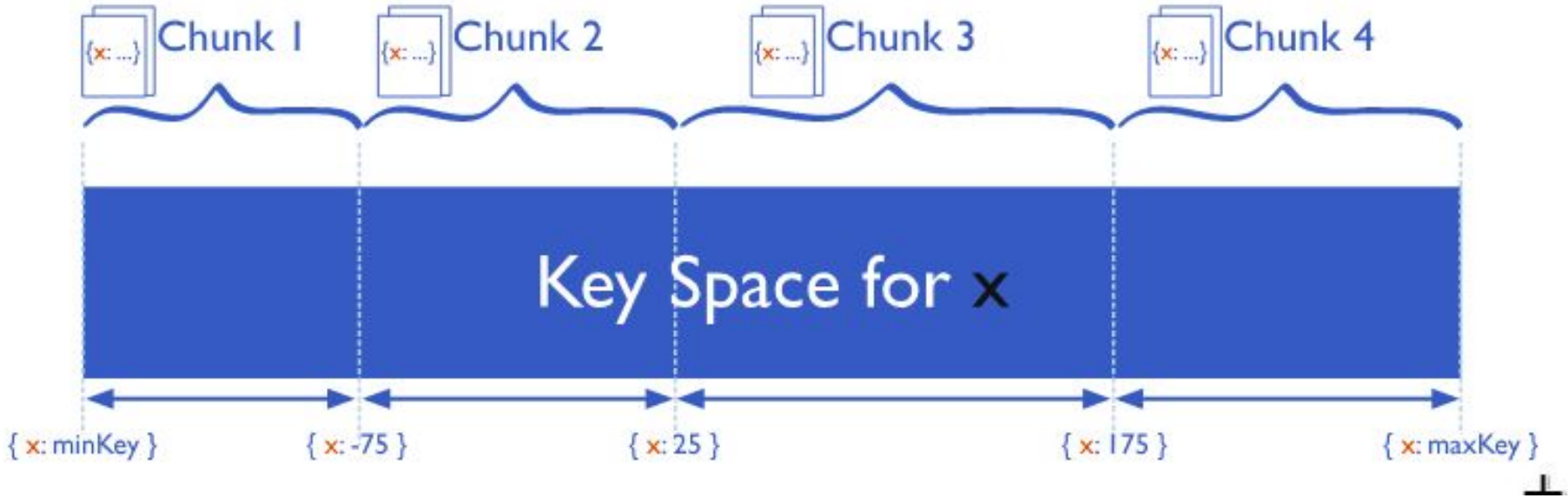
Alta disponibilidad - Particionado

Shard Key: Clave de particionado

- El nivel de particionado en MongoDB es de **colección**. La colección se divide y reparte a lo largo del cluster.
- La Shard Key indica a MongoDB como **repartir** los documentos de las colecciones entre las distintas particiones.
- La Shard Key consiste en uno o varios **campos** que tienen que existir en todos los documentos de una colección.
- La elección de la Shard Key afecta al rendimiento, eficiencia y escalabilidad del cluster particionado.
- MongoDB crea un **índice** para la colección por los campos de la Shard Key.
- Una vez seleccionada la Shard Key **no se puede cambiar**.
  - A partir de la versión 4.2 de MongoDB el valor del campo seleccionado como Shard Key se puede modificar.
- Una colección particionada no se puede convertir en una colección sin particionado.

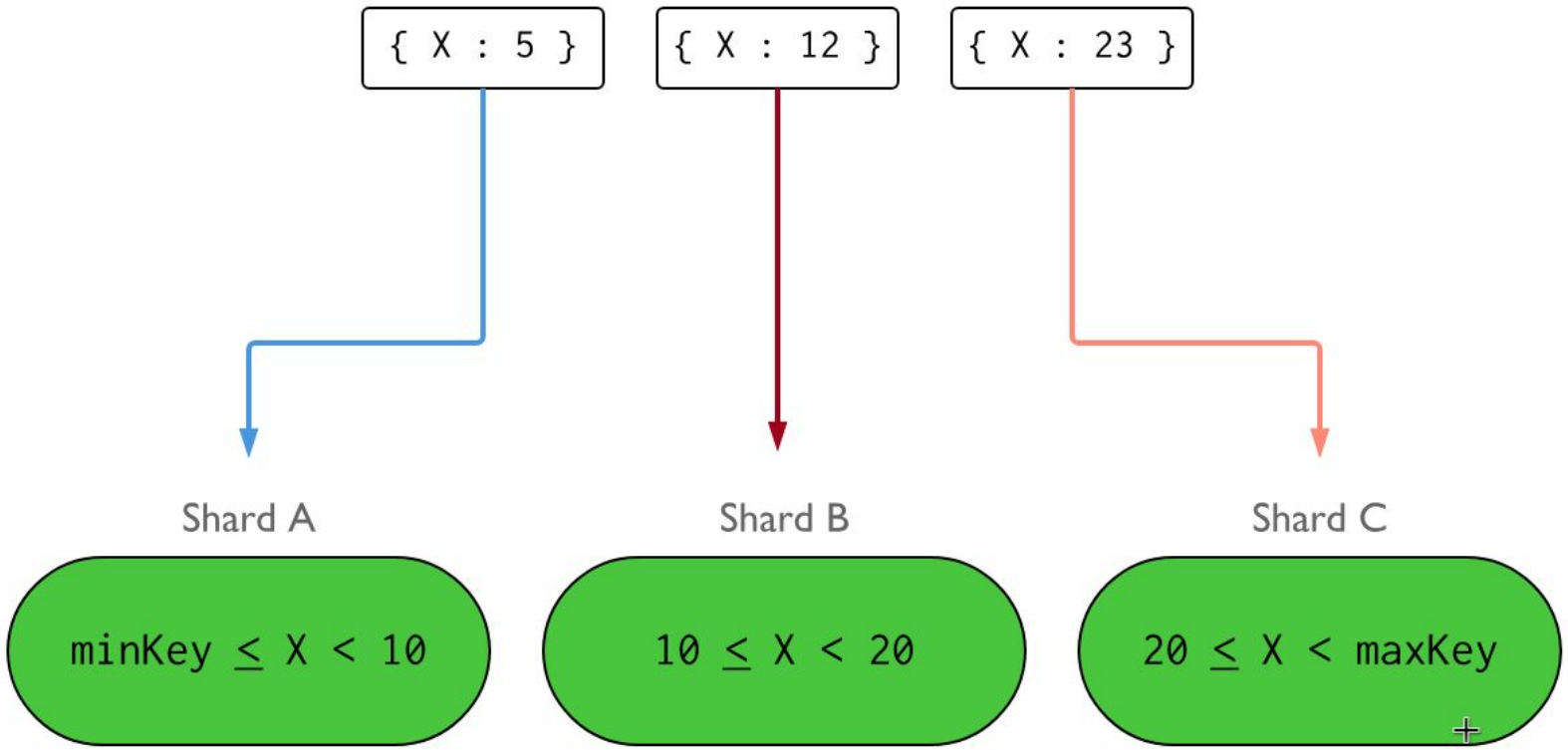
Alta disponibilidad - Particionado

Ranged



Alta disponibilidad - Particionado

Ranged Sharding



Alta disponibilidad - Particionado

Ranged Sharding (por defecto): Utiliza rangos de valores de la Shard Key para crear las particiones.

- **Chunk**: cada uno de los rangos de valores de la Shard Key. Estos rangos de valores no se solapan.
- MongoDB intentará distribuir de forma homogénea los chunks por todos los shards del cluster. La elección de la clave es determinante para que esta distribución sea homogénea.
- Es eficiente cuando la clave cumple las siguientes condiciones:
  - Alta cardinalidad.
  - Baja frecuencia.
  - No incremental.

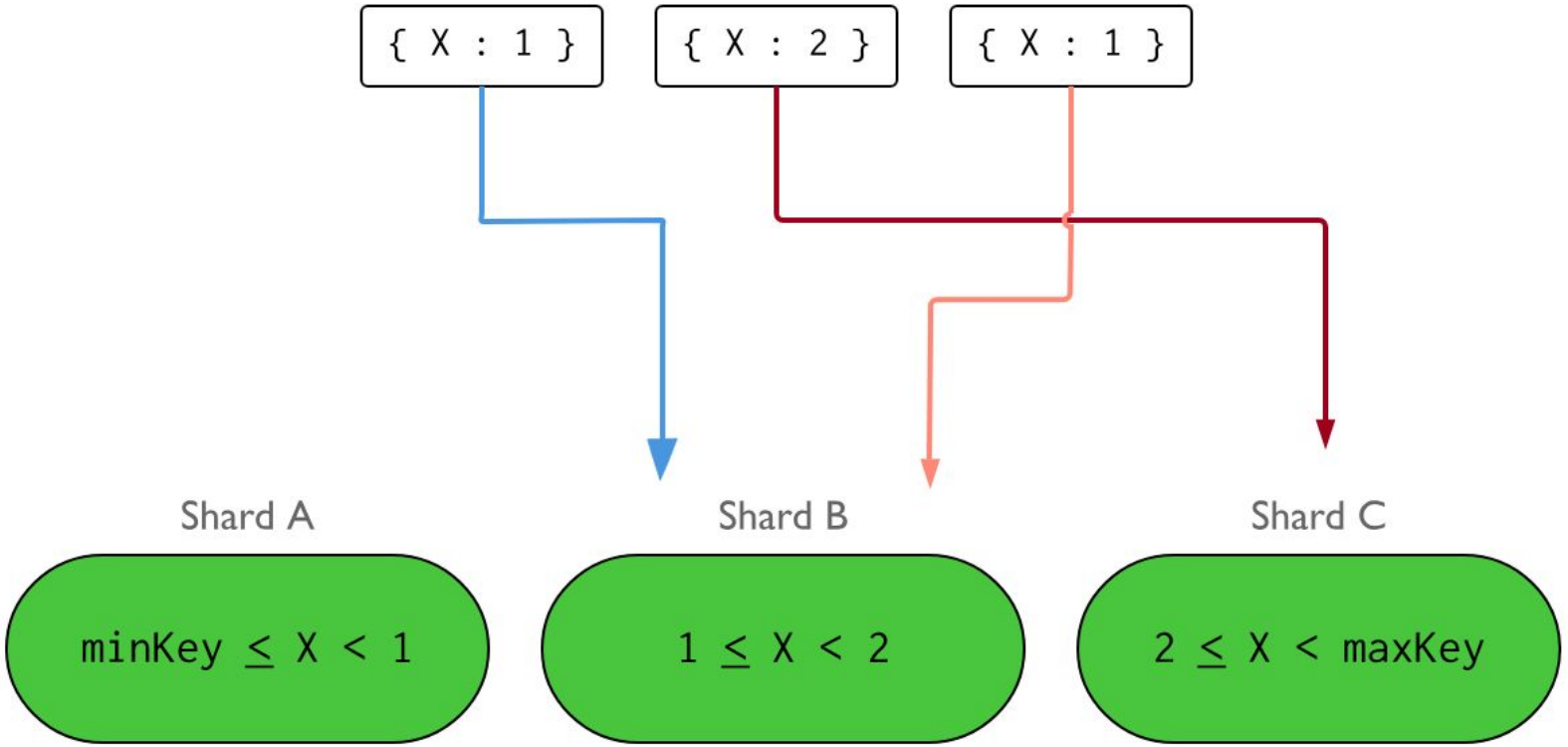
Alta disponibilidad - Particionado

Elección de la clave de particionado: Para una buena elección de clave hay que tener en cuenta:

- **Cardinalidad:** Determina el número máximo de chunks que el balanceador puede crear. Afecta a la eficiencia del escalado horizontal del cluster.
  - Cardinalidad baja: Un valor único de la shard key sólo puede existir en un único chunk. Si la clave tiene una cardinalidad de 4, entonces sólo puede haber 4 chunks en el cluster. Esto limita el número de shards efectivos del cluster. Añadir shards adicionales no genera ningún beneficio. Utilizar claves compuestas puede ayudar a mejorar la distribución de los datos añadiendo un campo con una cardinalidad relativa más alta.
  - Cardinalidad alta: No garantiza una distribución uniforme de los datos, pero facilita el escalado horizontal.

Alta disponibilidad - Particionado

Cardinalidad

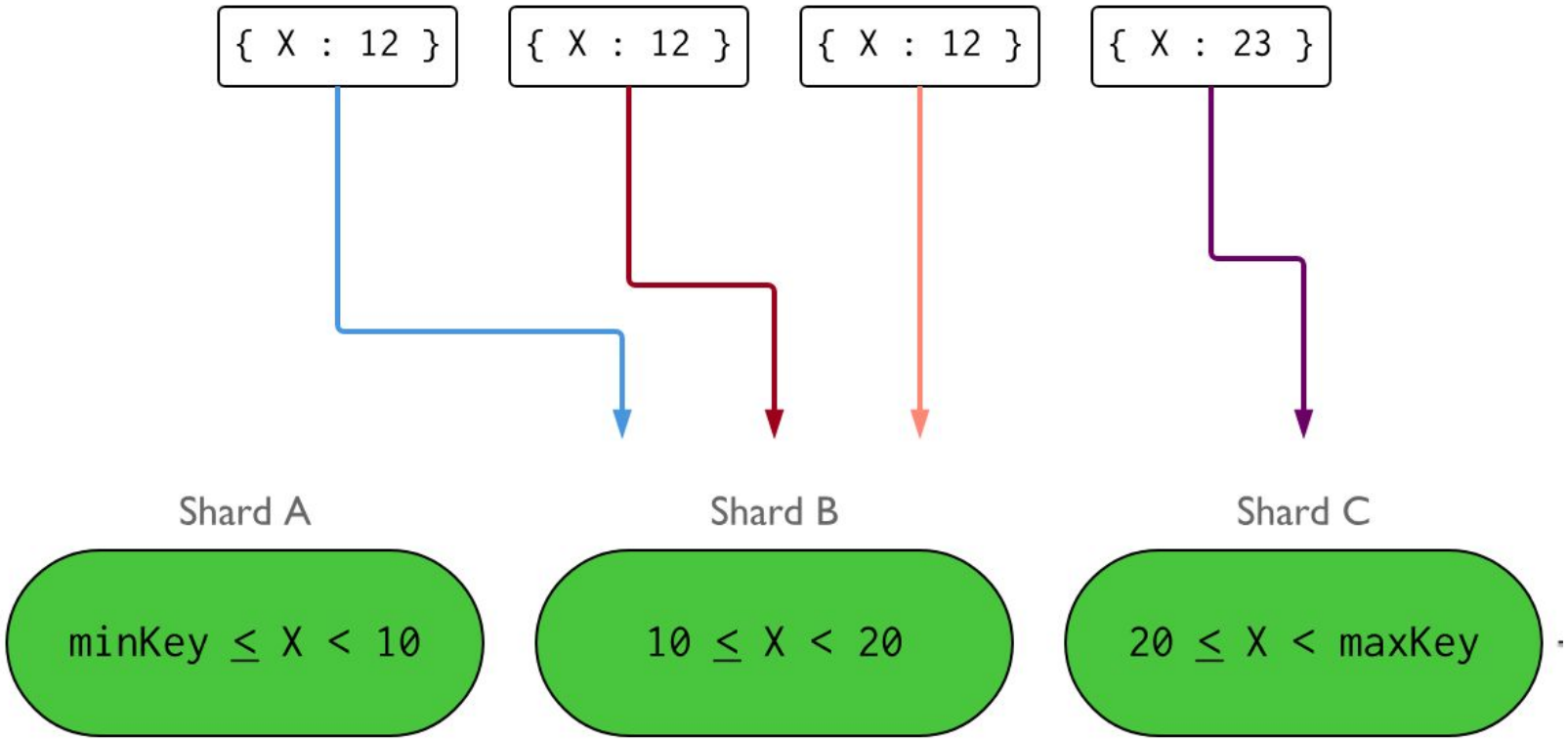


Alta disponibilidad - Particionado

- **Frecuencia**: Hacer referencia a cuántas veces aparece un valor dado en los datos.
  - **Frecuencia alta**: Si la mayoría de los documentos sólo contiene un valor reducido de esos valores, entonces los chunks que almacenan esos valores se convertirán en un cuello de botella en el cluster. Utilizar una clave compuesta con un valor que tenga menor frecuencia puede ayudar a mejorar la distribución de los datos.
  - **Frecuencia baja**: No garantiza una mejor distribución de los datos en el cluster.

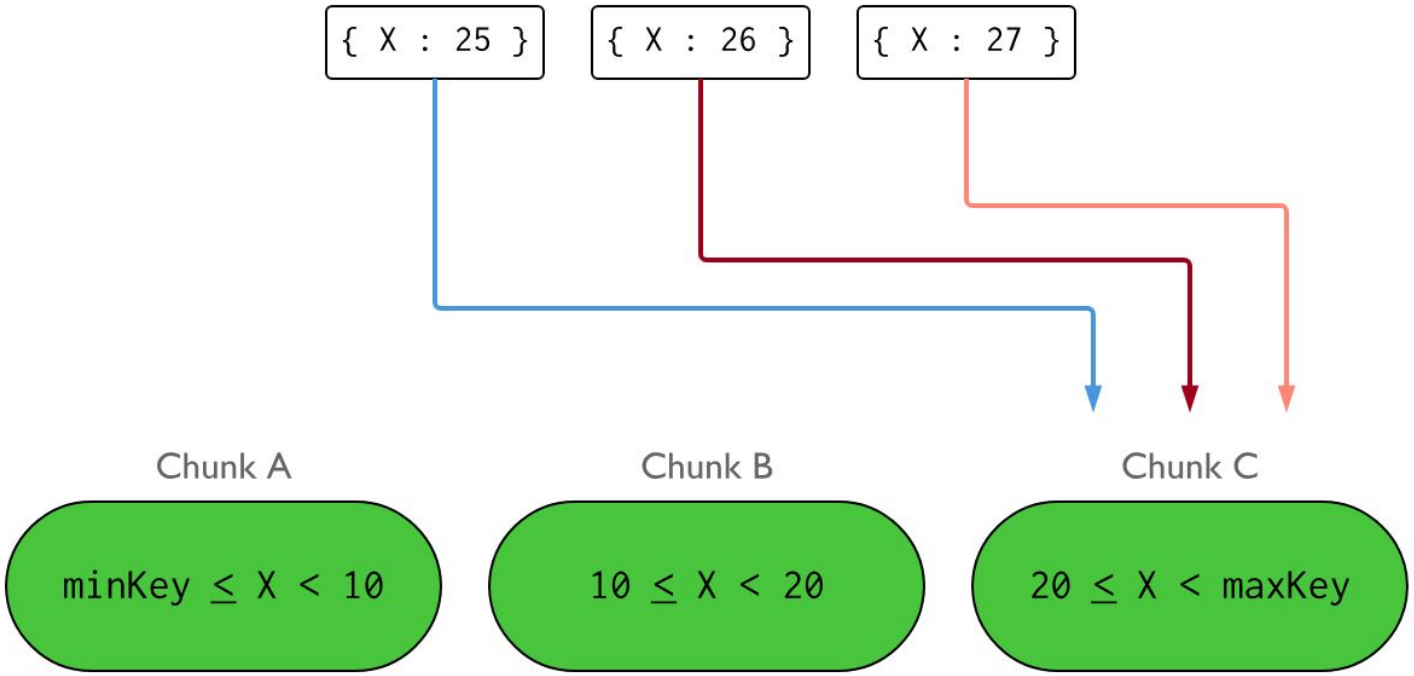
Alta disponibilidad - Particionado

Frecuencia



Alta disponibilidad - Particionado

Crecimiento Incremental: Una clave que crece o disminuye incrementalmente es más probable que distribuya las inserciones en un solo shard del cluster.



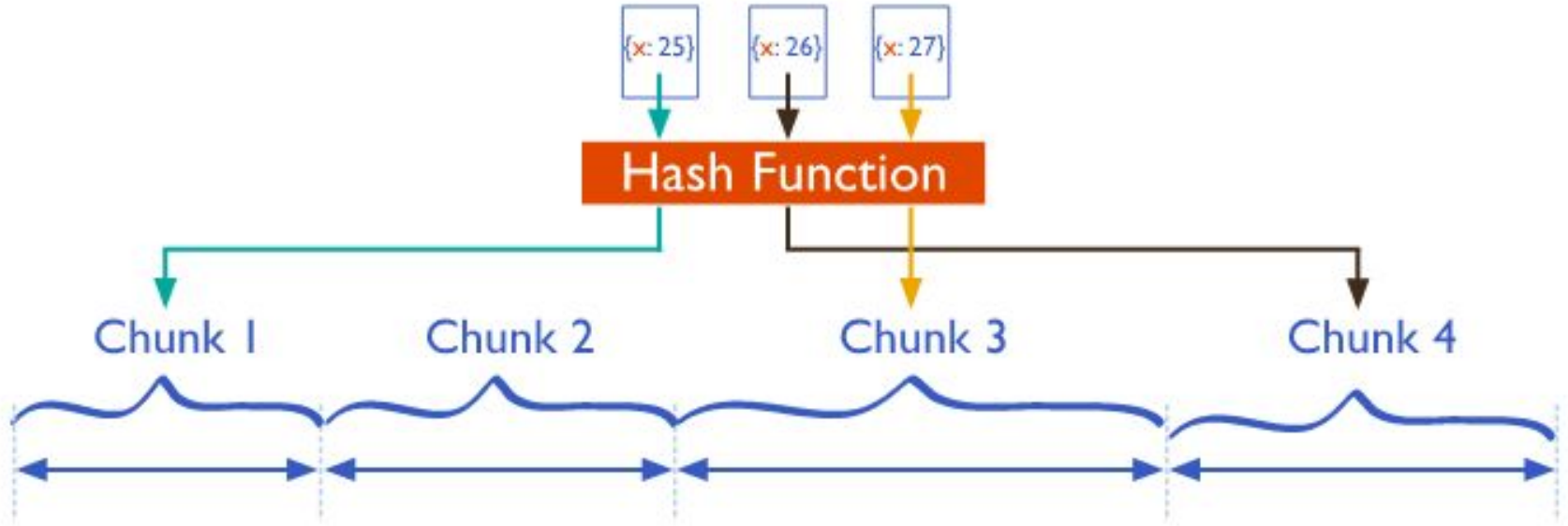
Alta disponibilidad - Particionado

Hashed Sharding: Utiliza una función hash para enrutar los documentos.

- Utilizan sólo un campo como clave de particionado, que es el que introducen en la función hash.
- Proporciona una distribución de los datos en el cluster más uniforme.
- Aunque es más probable que los valores cercanos de clave no estén en el mismo shard.
- Es más probable que los mongos hagan operaciones broadcast para realizar la consulta.

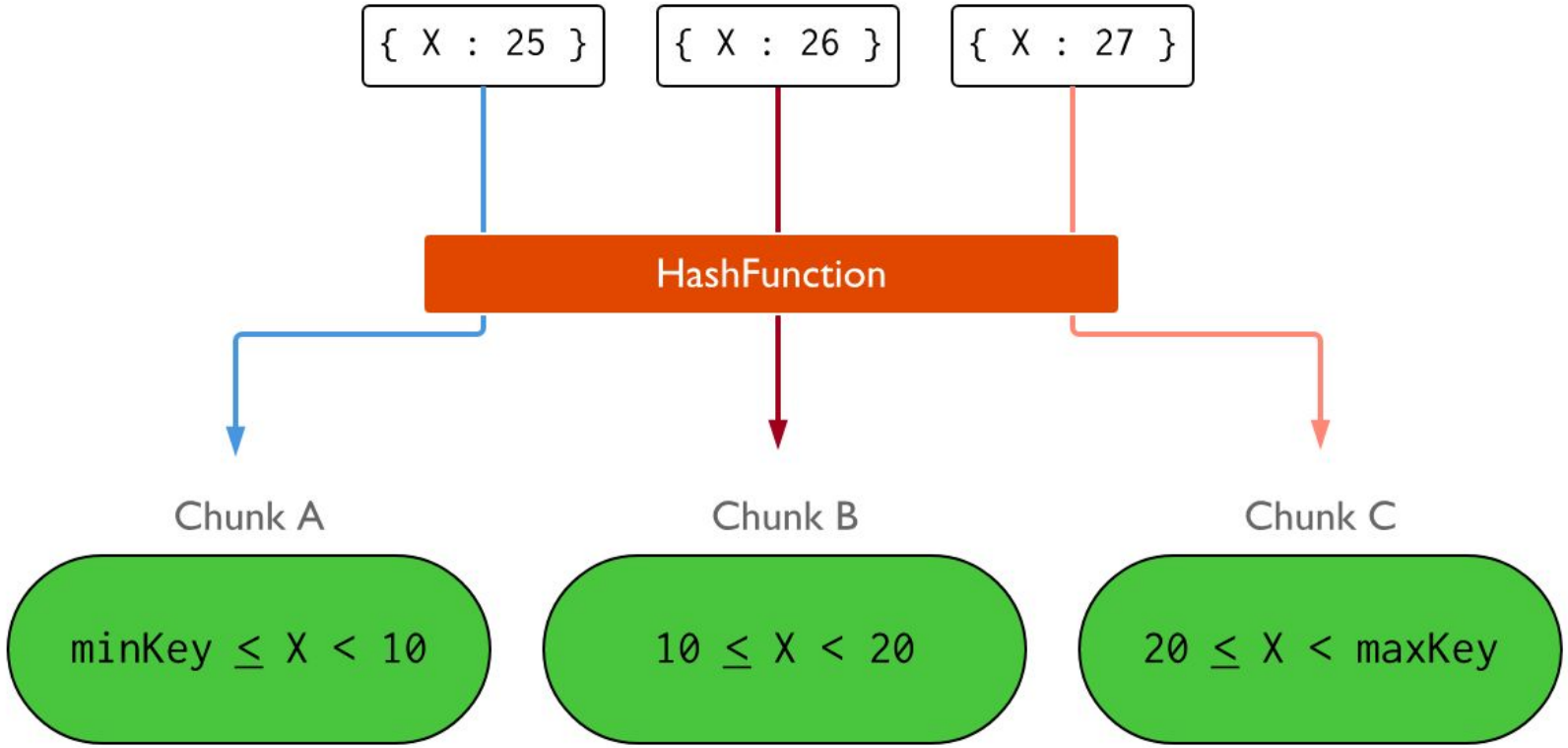
Alta disponibilidad - Particionado

Hashed Sharding



Alta disponibilidad - Particionado

Hashed Sharding



Alta disponibilidad - Particionado

Zonas: En los clusters particionados se pueden crear zonas con datos particionados a través de la Shard Key. Se puede asociar una zona con uno o más shards del cluster.

Se suelen utilizar para:

- Aislar un subconjunto específico de datos en un conjunto específico de shards.
- Asegurarnos que los datos más relevantes para una aplicación están más próximos geográficamente a los servidores de la aplicación.
- Para enrutar los datos a las particiones en función del hardware.

Alta disponibilidad - Particionado

Ranged sharding:

- Cada zona cubre uno o más rangos de una Shard Key.
- Las zonas no pueden compartir rangos o superponer rangos.

Hashed sharding:

- Asocia valores secuenciales del hash a una zona.
- Puesto que utiliza los valores de la función hash se pueden obtener resultados inesperados.

Replicación

Alta disponibilidad - Replicación

MongoDB Replica Set: Es un grupo de procesos de mongodb que mantienen el mismo conjunto de datos. Provee de redundancia y alta disponibilidad.

Varios nodos de datos:

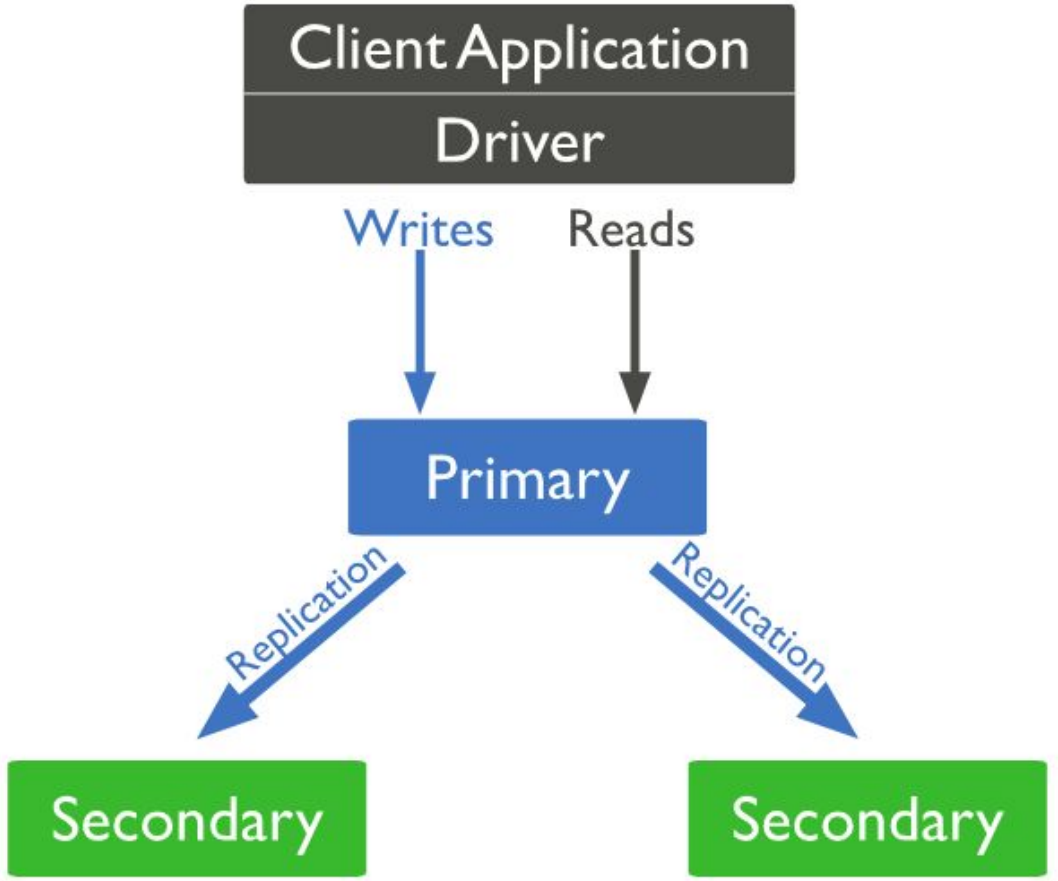
- **Primario** (sólo uno). Recibe todas las peticiones de escritura.
- **Secundarios**. Replican las operaciones del primario para mantener un set de datos idénticos.
- **Nodo árbitro** (opcional): No contienen datos, pero participan en la elección del nodo primario.

La configuración mínima recomendada es de tres nodos de datos. Uno primario y dos secundarios.

Alta disponibilidad - Replicación

Nodo Primario:

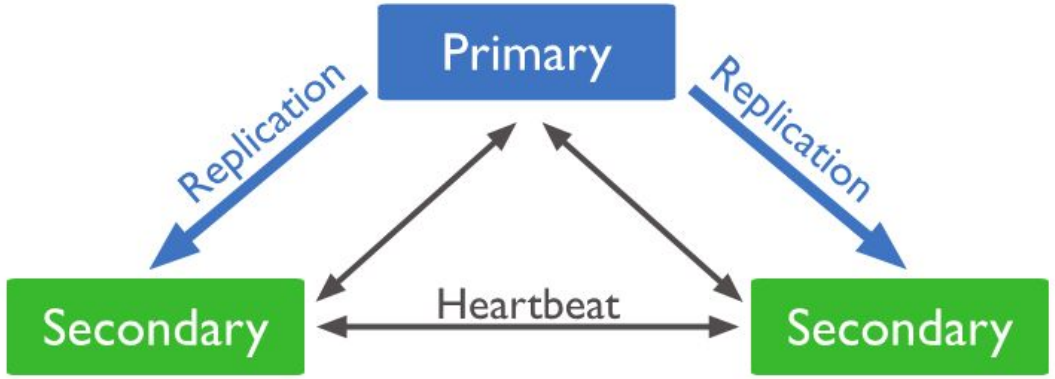
- Único miembro del Replica Set que recibe **operaciones de escritura**.
- MongoDB aplica las operaciones de escritura en el no primario y las registra en su **log operacional** (oplog)
- Los nodos secundarios replican este log y aplican las operaciones en sus nodos.
- Todos los miembros del replica set pueden recibir operaciones de lectura, pero por defecto todas las aplicaciones dirigen esta petición al maestro (configurable).
- Sólo puede haber un nodo primario, pero cuando este deja de estar disponible, un proceso de elección selecciona un nuevo primario.



Alta disponibilidad - Replicación

Nodo Secundario:

- Mantienen una **copia** de los datos del nodo primario.
- Aplican la operaciones del oplog del nodo primario en sus datos de forma **asíncrona**.
- Puede haber uno o más nodos secundarios.
- Los clientes no pueden escribir datos en los nodos secundarios, pero si leer datos de ellos (modificable).
- Un nodo secundario puede convertirse en primario en un proceso de elección, si el nodo primario deja de estar disponible.



Alta disponibilidad - Replicación

Un nodo secundario se puede configurar para un propósito concreto:

- **Priority 0 replica**: Para que no pueda convertirse en nodo primario.
- **Hidden replica**: Las aplicaciones no pueden leer de él.
- **Delayed replica**: Para mantener un histórico que permita recuperarnos de ciertos errores como borrar una base de datos.

Alta disponibilidad - Replicación

Oplog: (Registro operacional) Es una colección especial de datos limitada que mantiene un registro continuo de todas las operaciones que modifican los datos almacenados en una base de datos.

- Se implementa como una **capped collection** (local.oplog.rs): Colección de tamaño fijo que automáticamente sobrescribe los datos más antiguos cuando alcanza su tamaño máximo.
- Las operaciones sobre los datos se envían al nodo primario que las aplica y las escribe en el oplog.
- Los nodos secundarios copian y realizan esas operaciones de forma **asíncrona**.
- Los nodos secundarios obtienen una copia del oplog en la colección local.oplog.rs de sus nodos permitiéndoles mantener el estado actual de la base de datos.
- Un nodo secundario puede obtener una copia del oplog de cualquier miembro del replica set.
- Cada operación registrada en el oplog es **idempotente**: Las operaciones del oplog, producen los mismos resultados independientemente de las veces que se apliquen sobre el conjunto de los datos.

Alta disponibilidad - Replicación

Sincronización: MongoDB tiene dos procesos diferentes para replicar los datos entre los distintos nodos secundarios y mantener las copias de los datos actualizadas:

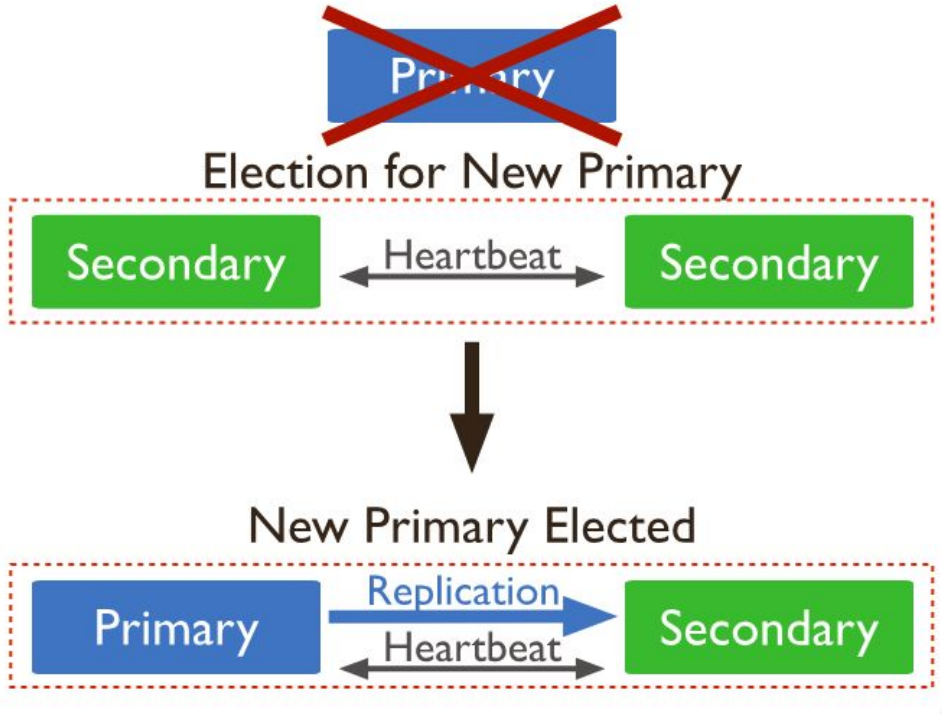
- **Initial Sync:** Para sincronizar todos los datos en un nodo nuevo.
  - mongod escanea cada colección de cada base de datos fuente e inserta los datos de cada colección en su copia local.
  - Aplica todos las operaciones del oplog de la fuente sobre sus datos para tener el estado actual del replica set.
- **Replication**: Para aplicar los cambios sobre los datos según ocurren.
  - Los nodos secundarios replican los datos de forma continua después del Initial Sync.
  - Los nodos secundarios copian el oplog de sus fuentes de sincronía y aplican sus cambios de forma asíncrona.

Fault Tolerance

Alta disponibilidad - Replicación

Elección: Los Replica Set utilizan un proceso de elección para determinar qué nodo se va a convertir en primario. Este proceso de elección se dispara cuando ocurre alguno de esto eventos:

- Añadir un nuevo nodo al Replica Set.
- Inicializar un Replica Set.
- Al ejecutarse una labor de mantenimiento: *rs.stepDown()*, *rs.reconfig()*.
- Los miembros secundarios pierden conectividad con el primario. Ocurre un timeout de 10 s (por defecto) durante el heartbeat.



Alta disponibilidad - Replicación

Heartbeat: Los nodos del Replica Set envían pings al resto de los nodos cada 2 s. Si el resto de nodos no recibe respuesta en 10 s, marcan el nodo como inaccesible.

- Si el nodo primario queda inaccesible se lanza el proceso de elección de un nuevo primario.
- El Replica Set no puede realizar operaciones de escritura hasta que la elección del nuevo primario se complete.
- El Replica Set puede realizar operaciones de lectura si estas operaciones se han configurado para ser ejecutadas en nodos secundarios.

Alta disponibilidad - Replicación

Network Partition: Un fallo en la red puede provocar una división de un sistema distribuido en particiones, de tal forma que los nodos de una partición no pueden comunicarse con los nodos de la otra partición.

- Cuando el nodo primario detecta que sólo puede comunicarse con una minoría de nodos, entonces se convierte en secundario y el Replica Set no admite peticiones que modifiquen los datos.
- Cuando un nodo secundario puede conectarse con la mayoría de los nodos (incluido el mismo) ejecuta una elección para convertirse en primario.

Alta disponibilidad - Replicación

Mayoría: Número de votos mayoritarios necesarios para seleccionar a un nuevo nodo primario en una elección.

- A todos los miembros de un Replica Set se les asigna una prioridad entre 0 y 1000, por defecto 1. Cuanto más prioridad, más posibilidad de ser elegido como primario.
- Para que un nodo pueda participar en la votación tiene que estar configurado como VotingMember.
- Los nodos que no son VotingMember contienen datos del Replica Set y pueden ejecutar operaciones de lectura de los clientes. Tienen prioridad 0.
- Como máximo sólo puede haber 7 VotingMembers en un Replica Set.

Alta disponibilidad - Tolerancia a fallos

Votación: Se realiza por la mayoría de los VotingMembers.

Mayoría: (VotingMembers / 2) + 1.

- La elección se decide primero por prioridad. El nodo con mayor prioridad se selecciona como nuevo primario.
- Si dos nodos tienen la misma prioridad, entonces se selecciona aquel que tenga el oplog más actualizado.

Almacenamiento

Almacenamiento

Almacenamiento: MongoDB soporta dos tipos de almacenamiento:

- **In-Memory Storage Engine**: Disponible en MongoDB Enterprise.
  - Almacena la información en memoria y no en disco.
  - Interesante cuando queremos poca latencia en las operaciones con los datos.

- **WiredTiger Storage Engine**: Por defecto a partir de la versión 3.2 de MongoDB:
  - Concurrencia de operaciones a nivel de documento.
  - Checkpointing
  - Compresión

WiredTiger Storage Engine

- **Concurrencia**:
  - Para las **escrituras**, establece control de concurrencia a **nivel de documento**. Múltiples clientes pueden modificar diferentes documentos de una colección.
  - Para la mayoría de las operaciones de **escritura** y **lectura** implementa un mecanismo de **concurrencia optimista**. Si MongoDB detecta un conflicto de escritura, una de las operaciones incurrirá en un fallo que MongoDB gestionará de forma transparente con un reintento.

WiredTiger Storage Engine

Snapshots y Checkpoint:

- Utiliza MultiVersion Concurrency Control (MVCC). Al iniciar una operación MongoDB crea un snapshot con marca de tiempo del dato:
  - Un Snapshot representa una vista consistente de los datos en memoria.
  - Si hay dos operaciones trabajando sobre el mismo dato, cada una recibirá el mismo snapshot, pero al devolver la nueva copia del dato, la operación que primero termine creará una nueva copia, mientras que la segunda intentará generar una copia nueva sobre un snapshot que ya no es el snapshot que recibió devolviendo un error y generando un reintento.
- MongoDB escribe los snapshots en disco cada 60 s o cuando el journal alcance 2 GB. Estos datos ahora duraderos actúan como un checkpoint en los ficheros de datos.
- MongoDB escribe los checkpoints en disco de forma consistente y no se deben utilizar como checkpoints de recuperación de datos.

Rafael Garrote Hernández

WiredTiger Storage Engine

Journal: MongoDB utiliza un write-ahead log (journal) junto con los checkpoints para garantizar la durabilidad del dato.

- Persiste todas las modificaciones de los datos entre checkpoints.
- Si un nodo falla, MongoDB utiliza el journal para reconstruir la información entre checkpoints.
- Se implementa como un buffer en memoria.
- Se persiste en disco:
  - Cada 100 ms.
  - Cuando una operación específica j:true en su write concern.
  - Cuando alcanza 100MB.
- MongoDB comprime el journal utilizando Snappy.

WiredTiger Storage Engine

Compresión

- Permite comprimir colecciones e índices.
- Implica mayor uso de CPU.
- Utiliza snappy para comprimir las colecciones.
- Utiliza Prefix Compression para los índices.
- Configurable:
  - Otros algoritmos soportados: zlib y zstd
  - Se pueden configurar a nivel global o en su creación, por colección y por índice.

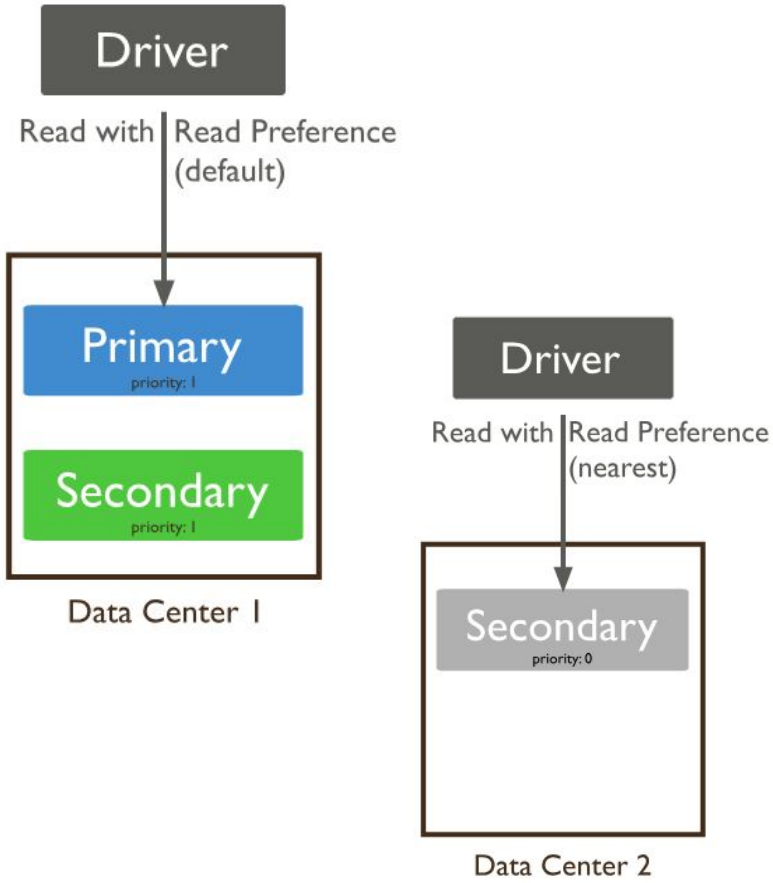
Consistencia

Consistencia

Read Preference: Indica cómo enrutar las lecturas a los miembros del Replica Set.

- **primary** (por defecto): todas las lecturas se enrutan al nodo primario.
- **primaryPreferred**: si el nodo primario no está disponible la operación se envía a un secundario.
- **secondary**: las lecturas se realizan en los nodos secundarios.
- **secondaryPreferred**: si no hay nodos secundarios disponibles, se realiza la lectura sobre el primario.
- **nearest**: la lectura se realiza del nodo con menos latencia independientemente de si es un nodo primario o secundario.

```
db.collection.find({}).readPref( "secondary",  [ { "region": "South" } ] )
```



Rafael Garrote Hernández

Consistencia

Read Concern: permite controlar la consistencia y el aislamiento de las lecturas en un Replica Set y en un Sharded Cluster.

Niveles:

- ***local***: Devuelve el dato de la instancia sin garantizar que el dato se haya escrito en la mayoría de los miembros del replica set (por defecto para lecturas en el primario).
- ***available***: Devuelve el dato de la instancia sin garantizar que el dato se haya escrito en la mayoría de los miembros del replica set (por defecto para lecturas en los secundarios).
- ***majority***: Devuelve los datos que han sido confirmados por la mayoría de los miembros de un Replica Set.
- ***linearizable***: Devuelve los datos que reflejan una escritura exitosa confirmada por la mayoría de los miembros del Replica Set antes de iniciar la lectura.
- ***snapshot***: los datos se devuelven de los snapshots de la mayoría que hayan realizado el commit de los datos.

Consistencia

Write Concern: Describe el nivel de confirmación requerido para una operación de escritura en un mongod standalone en un Sharded Cluster.

Para definir el *write concern* es necesario especificar los siguientes campos:

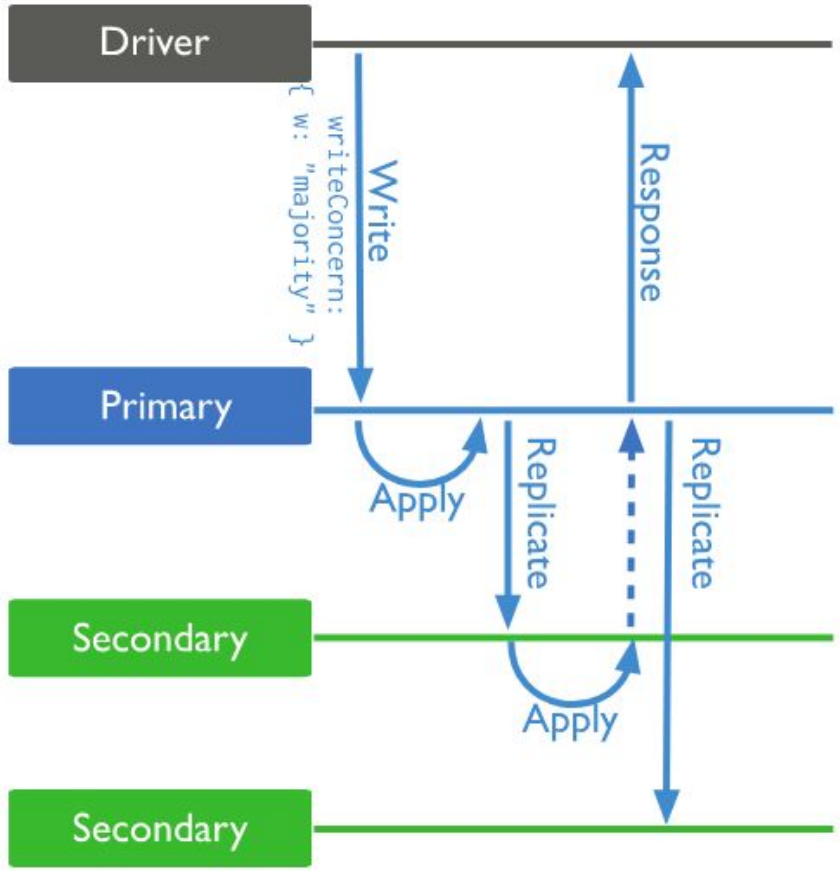
- **w**: indica el número de instancias de mongod que tienen que enviar la confirmación para dar por terminada la escritura.
- **j**: indica que se requiere confirmación de que la operación ha sido escrita en el journal en disco.
- **wtimeout**: para especificar un límite de tiempo para prevenir que las operaciones se bloqueen indefinidamente.

Consistencia

w valores:

- **w:1** (por defecto)se pide confirmación de que la operación se haya realizado al mongod standalone o a la réplica primaria.
- **w:0** no se requiere confirmación.
- **w:majority** solicita confirmación de que la operación se ha realizado en la mayoría calculada de los miembros con derecho a voto.

```
db.products.insert(
   { item: "envelopes", qty : 100, type: "Clasp" },
   { writeConcern: { w: "majority" , wtimeout: 5000 } }
)
```



Rafael Garrote Hernández

Consistencia

j valores:

- **j:true** pide confirmación de la escritura en el journal en disco de la operación de escritura en los nodos especificados en el valor w.
- **j:false** no pide confirmación de escritura en el journal en disco.
- Si se especifica j:true con valor w:0, prevalece j:true sobre el mongod standalone o la réplica primaria.

Consistencia

Comportamiento

	j: no especificado	j: true	j: false
w: <número>	en memoria	journal en disco	en memoria
w: majority	depende del valor por defecto (writeConcernMajorityJournalDefault)	journal en disco	en memoria

Modelado

Modelado de datos

Esquema flexible: Las colecciones de MongoDB por defecto no requieren que los documentos que almacenemos en ellas tengan el mismo esquema.

- Los documentos de una colección no tienen porqué tener el mismo conjunto de campos y los tipos de datos para un campo puede diferir entre los distintos documentos de una colección.
- Para cambiar la estructura de un documento en una colección, ya sea añadir o eliminar campos, modificar el tipo de un campo simplemente hay que modificar el documento con la nueva estructura.
- Para cambiar este comportamiento, se puede añadir validadores a una colección para fijar las reglas que tienen que cumplir los documentos que se almacenen en ella.

Modelado de datos

Estructura de los documentos: A la hora de modelar se trata de representar la estructura de los datos en documentos y las relaciones entre esos documentos.

MongoDB provee de dos mecanismos para relacionar los documentos:

- **Documentos embebidos** (Modelo desnormalizado):
  - Capturan la relación entre documentos almacenando la información relacionada en el mismo documento.
  - Esta información se puede embeber en un documento o en un array de documentos
  - Permite recuperar o modificar información con una sóla operación.
- **Referencias** (Modelo normalizado):
- Almacena la relación añadiendo enlaces o referencias de un documento a otro.

Modelado de datos

Documentos embebidos (Desnormalizado)

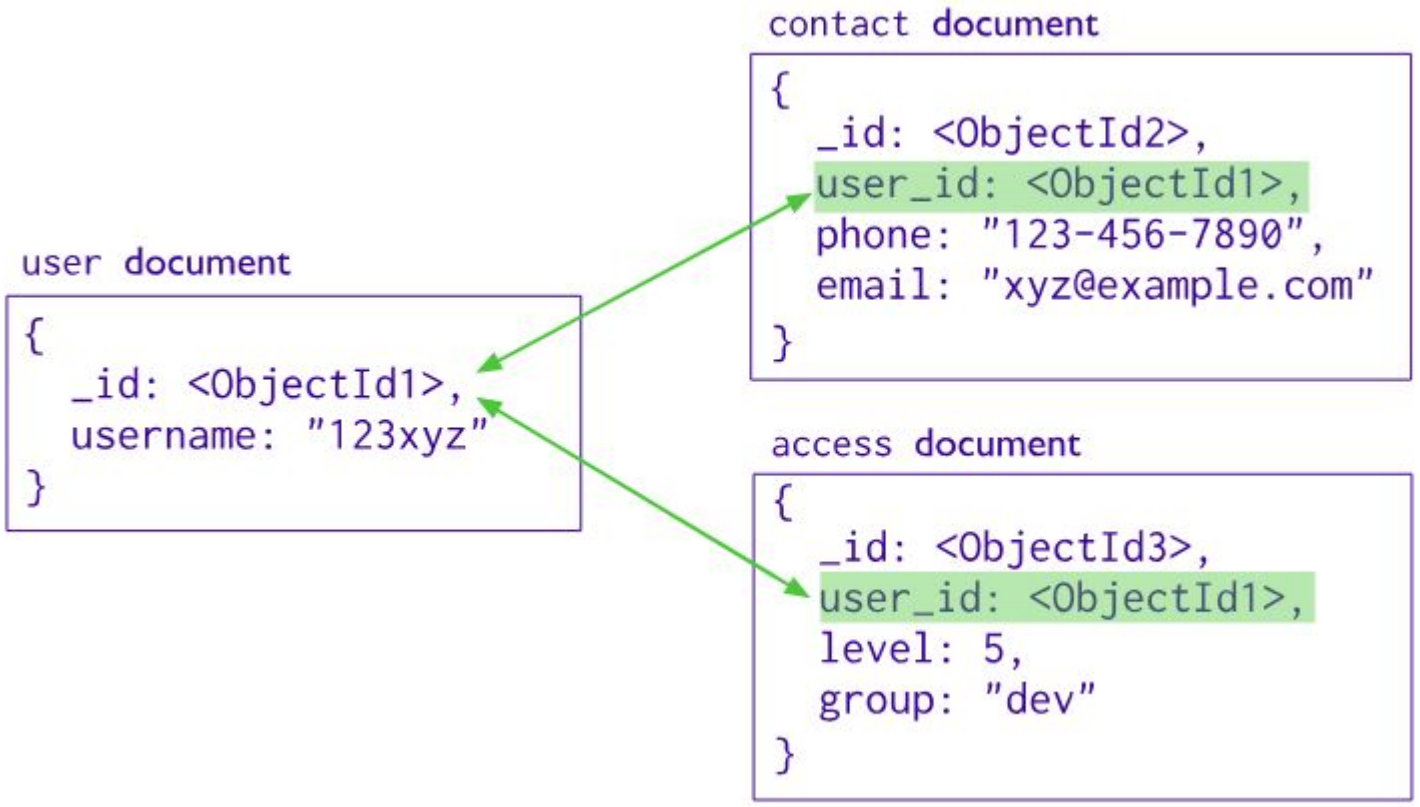
Modelado de datos

Documentos embebidos:

- Utilizar cuando:
  - Cuando se modelan relaciones del tipo “contiene” entre entidades, Relaciones 1:1.
  - Relaciones tipo “uno a muchos” entre entidades, relaciones 1:N.
- Ofrecen un mejor rendimiento para las operaciones de lectura ya que permiten recuperar toda la información relacionada en una sola operación y reduce la carga sobre el cluster.
- Permiten modificar la información relacionada en una única operación atómica de escritura.
- Permite realizar consultas sobre los documentos embebidos utilizando la notación punto y consultas sobre arrays.
- Los documentos no deben exceder del tamaño máximo que puede tener un documento (16MB).

Modelado de datos

Realaciones (Normalizado)



Modelado de datos

Referencias:

- Utilizar cuando:
  - Cuando embeber documentos supone una duplicidad de datos y el rendimiento de las consultas de lectura no supone un beneficio frente a la gestión de esa duplicidad.
  - Con relaciones del tipo "muchos a muchos" entre datos. Relaciones N:M.
  - Modelar data sets jerárquicos muy grandes.
- Para las consultas de agregación, MongoDB provee dos *stages* para hacer join entre colecciones:
  - $lookup (Disponible a partir de MongoDB 3.2)
  - $graphLookup (Disponible a partir de MongoDB 3.4)

Modelado de datos - Transacciones

Transacciones: MongoDB dispone de transacciones multi-documento para los casos en los que se necesita atomicidad leyendo o escribiendo múltiples documentos en una o varias colecciones.

- En MongoDB una operación sobre un único documento es atómica.
- Siempre que se utilicen documentos embebidos y arrays para hacer relaciones, la atomicidad de la operación está garantizada.
- En el caso de que se utilicen estructuras normalizadas con referencias para hacer las relaciones entre documentos, se pueden utilizar las transacciones para hacer atómicas las operaciones sobre múltiples documentos.
- Para lecturas multi-documento se tiene que utilizar **preferencia de lectura primary**.
- Todas las **operaciones** de una transacción se tienen que ***enrutar* en el mismo nodo**.

Modelado de datos - Validadores

Validación de esquemas: MongoDB permite la validación de esquemas en las modificaciones e inserciones de datos en una **colección**.

- Se puede especificar el validador al **crear** una colección:
  - durante la operación **db.createCollection()** especificando la opción **validator**.
  - modificando una colección ya creadas con la operación **collMod** con la opción **validator**.
- Dos tipos de validadores:
  - Especificando un JSON Schema utilizando el operador **$jsonSchema** en la opción **validator**.
  - Utilizando otros **operadores** como **$or, $type, $regex, $in**... en la opción **validator**.

Modelado de datos - Validadores

Comportamiento:

- Si se añade el validador a una colección existente, no se validan los documentos ya existentes hasta que se **modifiquen**.
- **Documentos ya existentes**. La opción *validationLevel* determina a qué operaciones aplica las reglas de validación:
  - **strict** (por defecto): Aplica las reglas a todas las inserciones y modificaciones.
  - **moderate**: Aplica las reglas en las inserciones y modificaciones que cumplen los criterios de validación. A aquellos documentos que en la modificación no cumplen los criterios de validación, no se le aplican las reglas.

Modelado de datos - Validadores

Comportamiento:

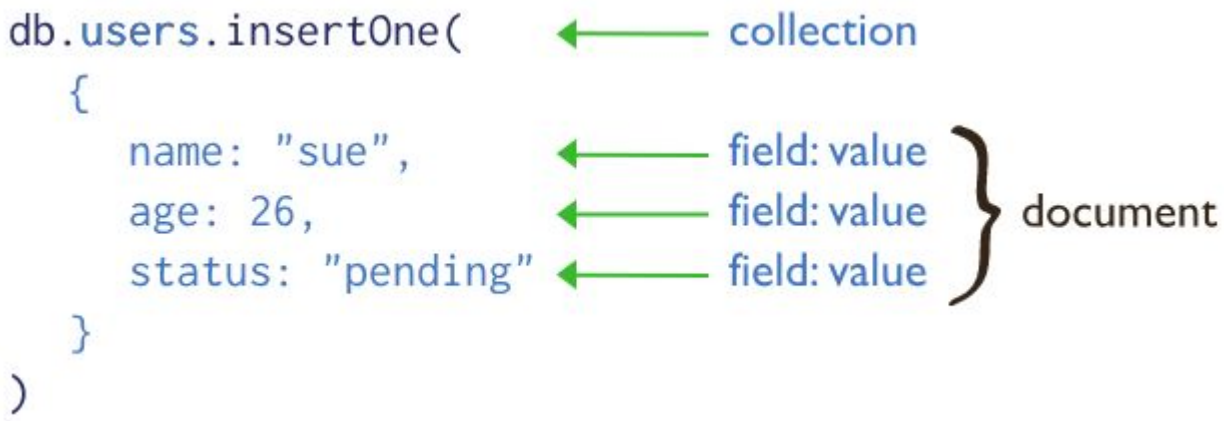
- Aceptar o rechazar documentos validados: la opción *validationAction* determina cómo manejar los documentos que no cumplen los criterios:
  - **error** (por defecto): Rechaza cualquier documento que se intente insertar o modificar que no cumpla los criterios de validación.
  - **warn**: Sólo logea los errores, pero permite la inserción o modificación del documento.

Operaciones CRUD

Operaciones CRUD

Inserción

- Atomicidad a nivel de documento.
- Si la colección no existe la crea.
- Si no se especifica _id, MongoDB lo crea.
- Se puede especificar el write concern.




```
db.inventory.insertOne(
   { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }
)
```

```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } }
   { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
   { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm"
])
```


Operaciones CRUD

Modificación

- Atómica a nivel de documento.
- No se puede modificar el campo _id.
- Opción upsert: true, si no existe el documento lo crea.
- Se puede especificar write concern.

```
db.users.updateMany(                    <-- collection
   { age: { $lt: 18 } },                <-- update filter
   { $set: { status: "reject" } }       <-- update action
)
```

```
db.collection.updateOne(<filter>, <update>, <options>)
db.collection.updateMany(<filter>, <update>, <options>)
db.collection.replaceOne(<filter>, <update>, <options>)
```

```
db.inventory.updateOne(
   { item: "paper" },
   {
     $set: { "size.uom": "cm", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

```
db.inventory.replaceOne(
   { item: "paper" },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ]
)
```


Operaciones CRUD

Delete

- Atómica a nivel de documento.
- Si se borran todos los elementos de una colección no se borran sus índices.
- Se puede especificar el write concern.

```
db.users.deleteMany(          <-- collection
   { status: "reject" }       <-- delete filter
)
```

```
db.collection.deleteMany()
db.collection.deleteOne()
```


Operaciones CRUD

Búsqueda

- El aislamiento de operaciones depende del read concern.

```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

```
db.inventory.find( { status: "A", qty: { $lt: 30 } } )
```

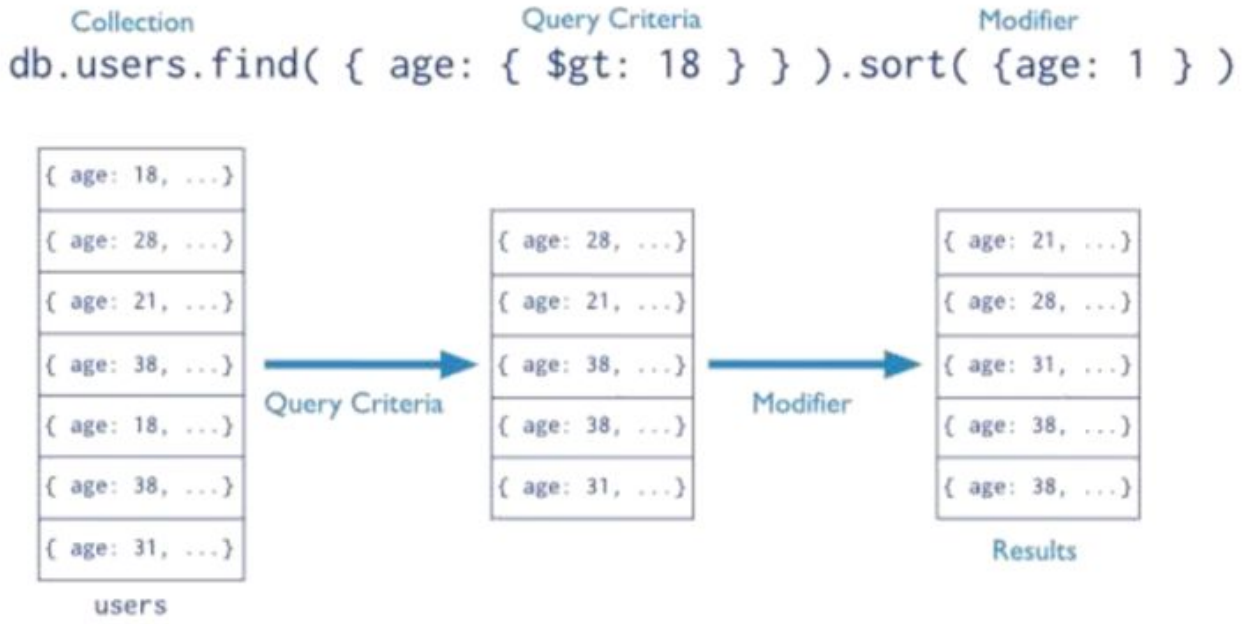
```
db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )
```

```
db.inventory.find( {
     status: "A",
     $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]
} )
```

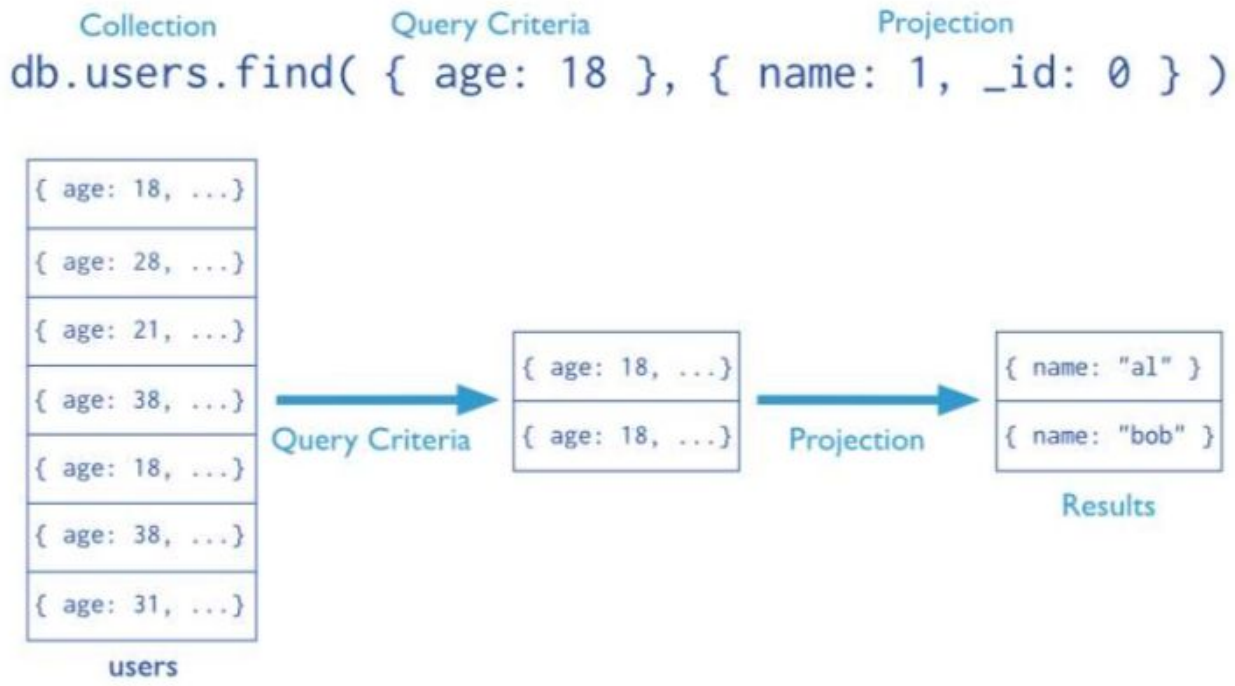
```
db.inventory.find( { status: "A" }, { item: 1, status: 1 } )
```

```
db.inventory.find( { status: "A" }, { status: 0, instock: 0 } )
```


Operaciones CRUD



Operaciones CRUD



Rafael Garrote Hernández

Transacciones

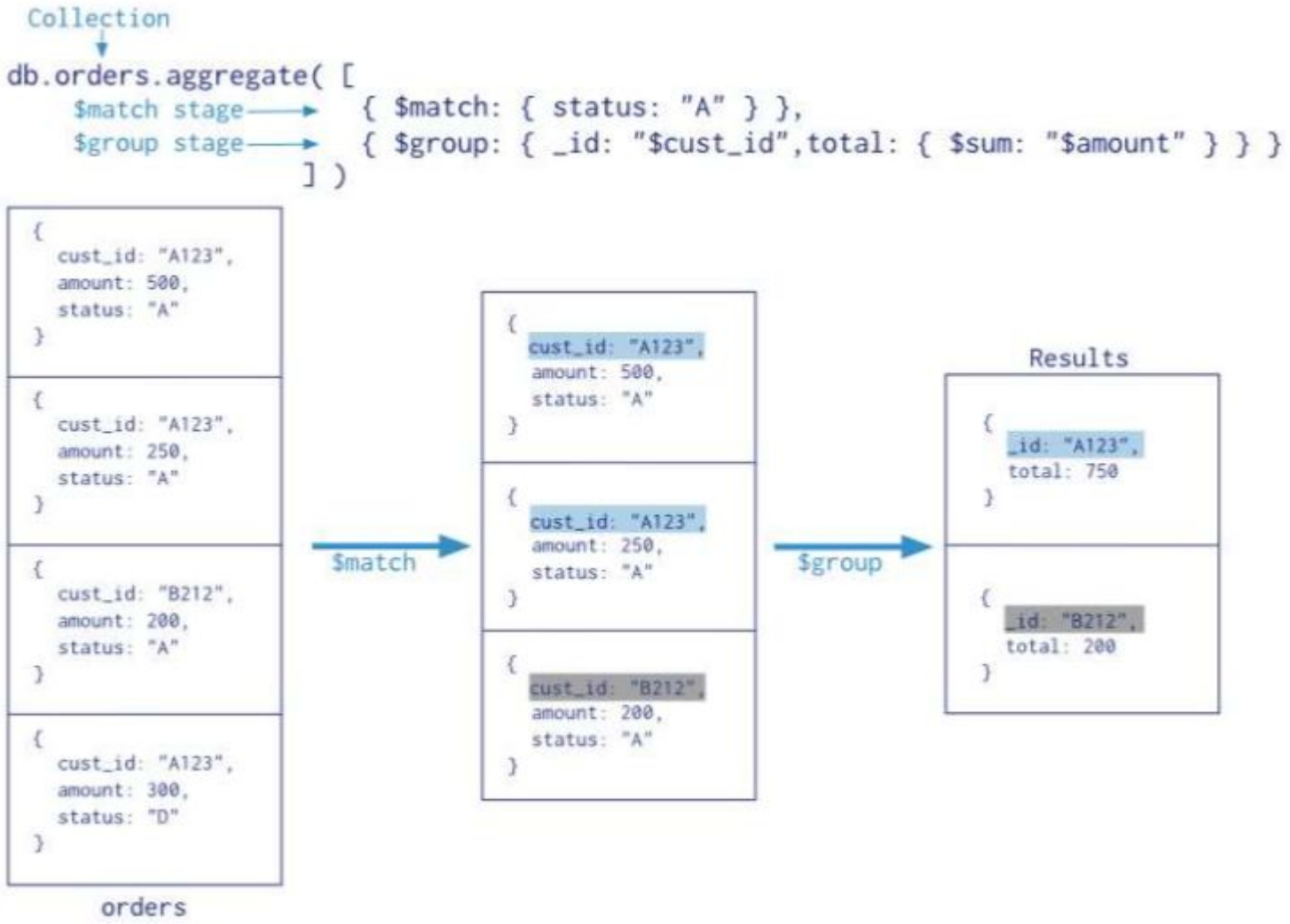
En MongoDB la atomicidad de las operaciones es por documento. Si queremos realizar operaciones multidocumento de forma atómica podemos utilizar transacciones.

```
# Step 1: Define the callback that specifies the sequence of operations to perform inside
def callback(session):
    collection_one = session.client.mydb1.foo
    collection_two = session.client.mydb2.bar

    # Important:: You must pass the session to the operations.
    collection_one.insert_one({'abc': 1}, session=session)
    collection_two.insert_one({'xyz': 999}, session=session)

# Step 2: Start a client session.
with client.start_session() as session:
    # Step 3: Use with_transaction to start a transaction, execute the callback, and comm
    session.with_transaction(
        callback, read_concern=ReadConcern('local'),
        write_concern=wc_majority,
        read_preference=ReadPreference.PRIMARY)
```


Framework de agregación

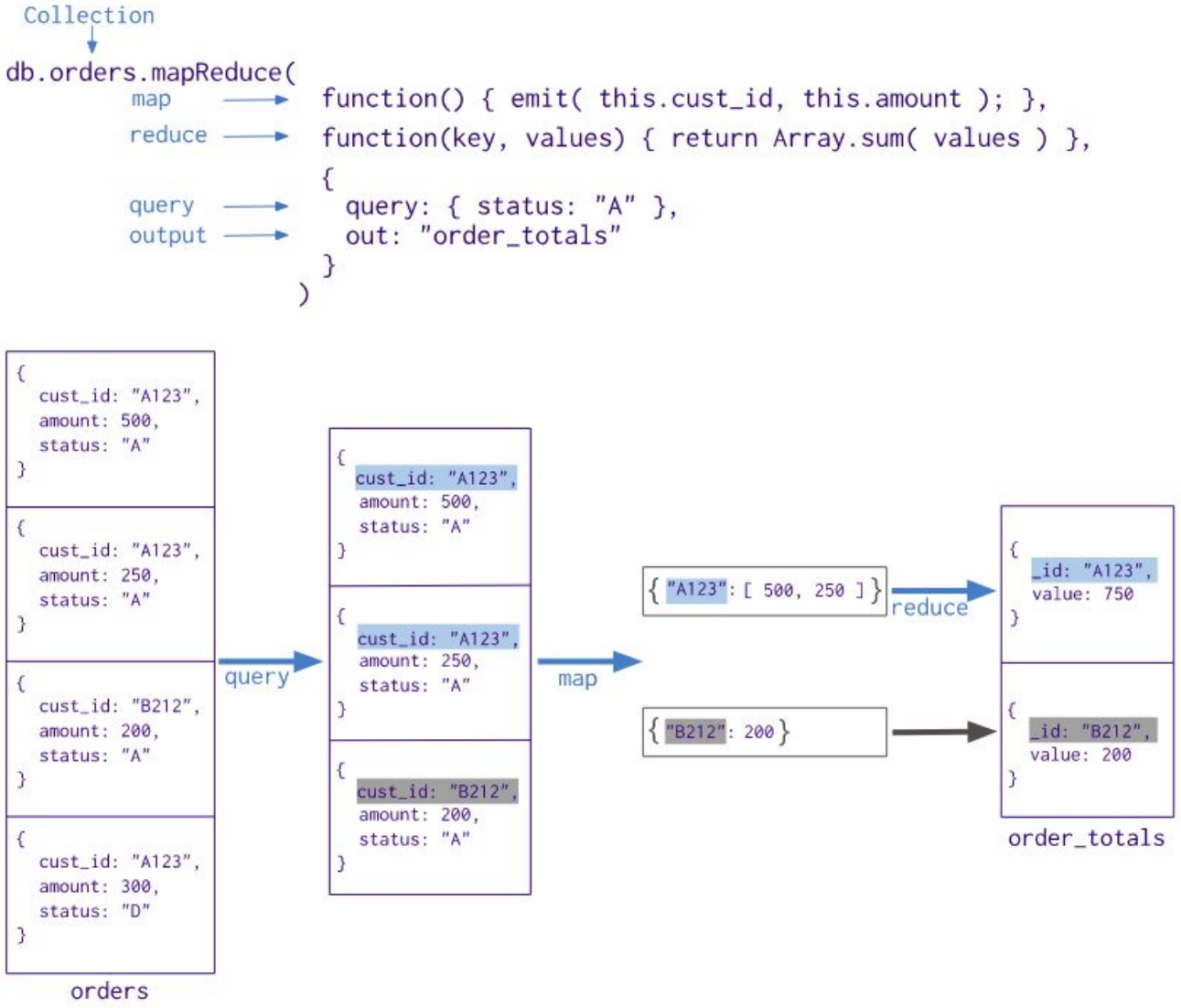
- Basado en pipelines de procesos de datos. Los documentos pasan por una serie de etapas que transforman estos documentos en un resultado agregado.
- Un pipeline está compuesto de etapas.
- Cada etapa procesa los documentos según pasan por ella.
- Cada etapa recibe como entrada los datos resultantes de la etapa anterior.
- Cada etapa emite el resultado de su operación a la siguiente etapa.
- La etapa inicial recibe como conjunto de entrada los documentos de la colección seleccionada.
- Operaciones:
  - Filtrados.
  - Proyecciones.
  - Agregaciones.

Framework de agregación



Rafael Garrote Hernández

Framework Map Reduce



Rafael Garrote Hernández

Rendimiento

Rendimiento - Indexación

MongoDB provee de dos operadores para examinar el plan de consulta sobre una colección:

- cursor.explain("executionStats")
- db.collection.explain("executionStats")

```
db.inventory.find(
   { quantity: { $gte: 100, $lte: 200 } }
).explain("executionStats")
```

```
{
   "queryPlanner" : {
         "plannerVersion" : 1,
         ...
         "winningPlan" : {
            "stage" : "COLLSCAN",
            ...
         }
   },
   "executionStats" : {
      "executionSuccess" : true,
      "nReturned" : 3,
      "executionTimeMillis" : 0,
      "totalKeysExamined" : 0,
      "totalDocsExamined" : 10,
      "executionStages" : {
         "stage" : "COLLSCAN",
         ...
      },
      ...
   },
   ...
}
```

Rendimiento - Indexación

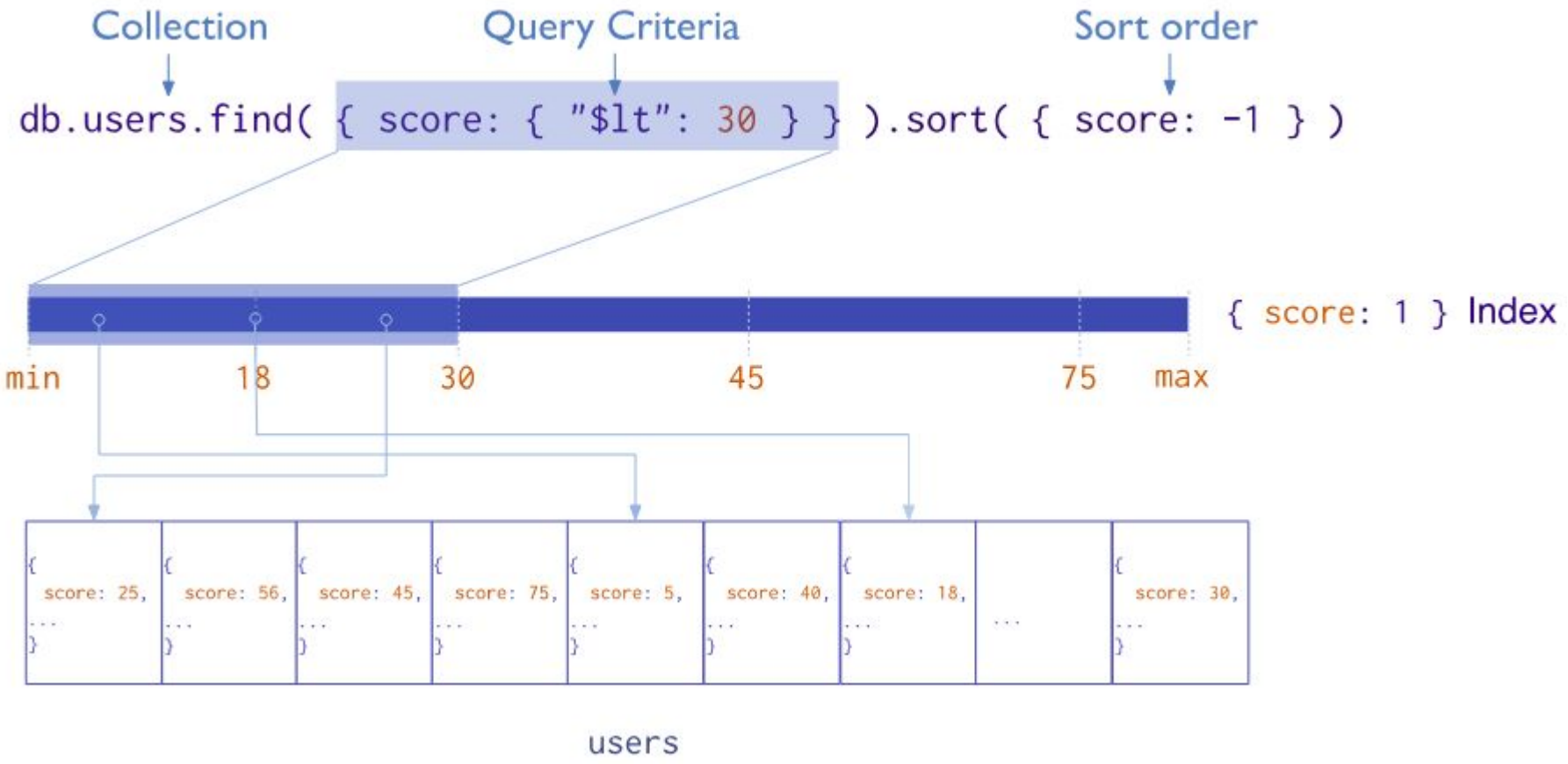
```
{ "_id" : 1, "item" : "f1", type: "food", quantity: 500 }
{ "_id" : 2, "item" : "f2", type: "food", quantity: 100 }
{ "_id" : 3, "item" : "p1", type: "paper", quantity: 200 }
{ "_id" : 4, "item" : "p2", type: "paper", quantity: 150 }
{ "_id" : 5, "item" : "f3", type: "food", quantity: 300 }
{ "_id" : 6, "item" : "t1", type: "toys", quantity: 500 }
{ "_id" : 7, "item" : "a1", type: "apparel", quantity: 250 }
{ "_id" : 8, "item" : "a2", type: "apparel", quantity: 400 }
{ "_id" : 9, "item" : "t2", type: "toys", quantity: 50 }
{ "_id" : 10, "item" : "f4", type: "food", quantity: 75 }
```

```
db.inventory.createIndex( { quantity: 1 } )
```

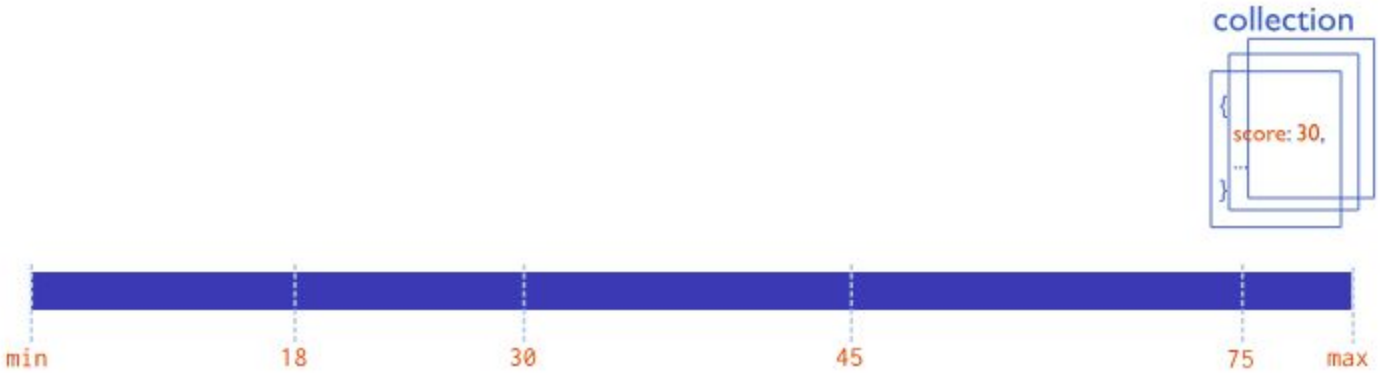
```
db.inventory.find(
   { quantity: { $gte: 100, $lte: 200 } }
).explain("executionStats")
```

```
{
   "queryPlanner" : {
         "plannerVersion" : 1,
         ...
         "winningPlan" : {
               "stage" : "FETCH",
               "inputStage" : {
                  "stage" : "IXSCAN",
                  "keyPattern" : {
                     "quantity" : 1
                  },
                  ...
               }
         },
         "rejectedPlans" : [ ]
   },
   "executionStats" : {
         "executionSuccess" : true,
         "nReturned" : 3,
         "executionTimeMillis" : 0,
         "totalKeysExamined" : 3,
         "totalDocsExamined" : 3,
         "executionStages" : {
            ...
         },
         ...
   },
   ...
}
```

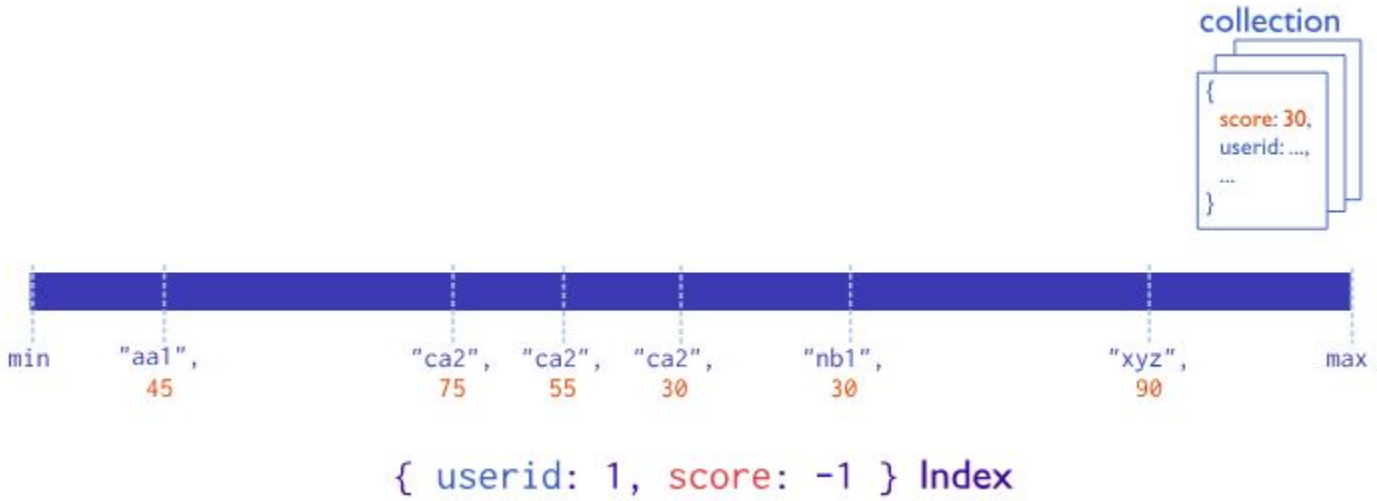
Rendimiento - Indexación



Rendimiento - Indexación

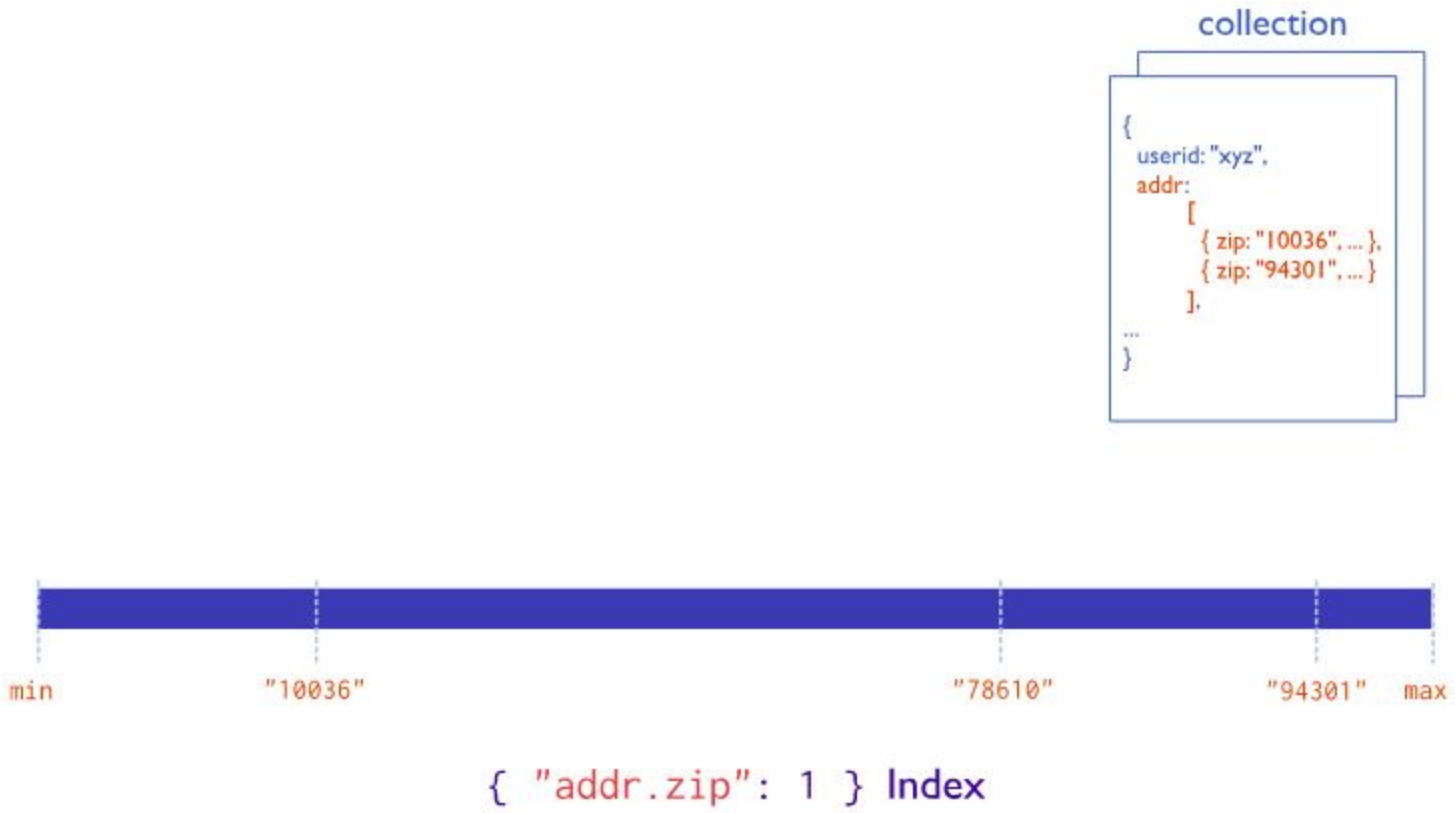


Rendimiento - Indexación



Rendimiento - Indexación

índices multiclave



Rendimiento - Indexación

Índices

- Permiten una ejecución eficiente de las consultas.
- Sin índices MongoDB realizaría *full scan* sobre la colección para seleccionar los documentos que cumplen la query.
- Almacenan el valor de uno o varios campos ordenados por el valor del campo.
- La ordenación facilita la búsqueda de coincidencias y de rangos de valores.
- Devuelve el resultado ordenado utilizando el orden del índice.

```
db.collection.createIndex( { name: -1 } )
```

```
db.products.createIndex(
  { item: 1, quantity: -1 } ,
  { name: "query for inventory" }
)
```


Muchas gracias

datahack
INSPIRING BIG DATA