Práctica Escalabilidad en Almacenes de Datos no Convencionales

REALIZADO POR ARACELI MACIA BARRADO

Indice

[Ejercicio 1 - MongoDB [30 puntos] 3](#_Toc471849693)

[Ejercicio 2 - MongoDB [20 puntos] 8](#_Toc471849694)

[Ejercicio 3 - Cassandra [20 puntos] 9](#_Toc471849695)

[Ejercicio 4 - Cassandra [20 puntos] 11](#_Toc471849696)

[Ejercicio 5 - Cassandra [10 puntos] 11](#_Toc471849697)

# Ejercicio 1 - MongoDB [30 puntos]

En las clases de escalabilidad de MongoDB importamos dos colecciones de documentos. La que no llegamos a usar en clase es la de restaurantes. En este ejercicio usaremos dicha colección. Las instrucciones para cargarla están en [este documento](https://docs.google.com/a/campusciff.net/document/d/1jFScMzXzh7kqBIBJitDaJTApxUtDHKB8RX3RpsvD76g/edit?usp=sharing).

Usa db.restaurants.find().limit(1).pretty() para hacerte una idea del formato de documentos que se usa en la colección de restaurantes.



Ejecuta la consulta de agregación siguiente. Observarás que obtiene el total de restaurantes de cocina brasileña por código postal para el barrio de Queens.

db.restaurants.aggregate(

[

{ $match: { "borough": "Queens", "cuisine": "Brazilian" } },

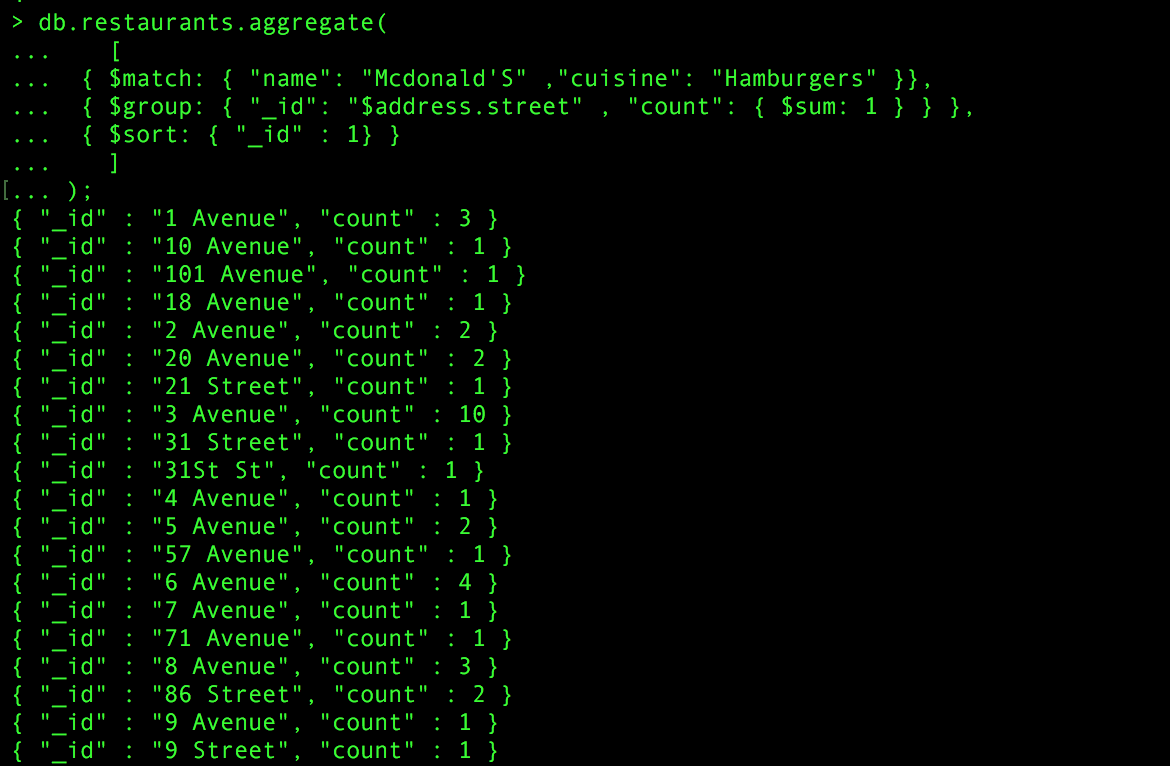
{ $group: { "\_id": "$address.zipcode" , "count": { $sum: 1 } } },

{ $sort: { "\_id" : 1} }

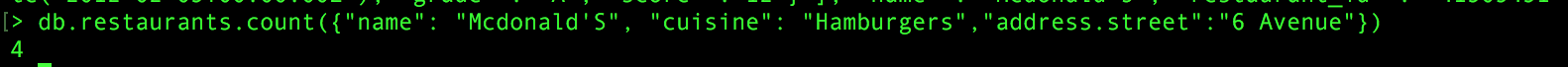
]

);

1. **Modifica la consulta anterior para obtener el número de McDonald’s agregados por calle en todo NYC. ¿Cuántos McDonald’s hay en la Sexta Avenida?**



**Consulta para ver el número de Mc Donald’s de la Sexta Avenida:**



En la Sexta Avenida hay 4 McDonald’s.

1. **Estudia las estadísticas del plan de ejecución de tu consulta de agregación. ¿Qué observas en el *stage* principal?**



Por lo que observo el stage principal es de tipo COLLSCAN, lo que quiere decir que la consulta está recorriendo la colección entera, así que **no** es una consulta optimizada

Si estudio las estadísticas del plan de ejecución de la instrucción:

db.restaurants.explain("executionStats").find({ "name": "Mcdonald'S" ,"cuisine": "Hamburgers" })

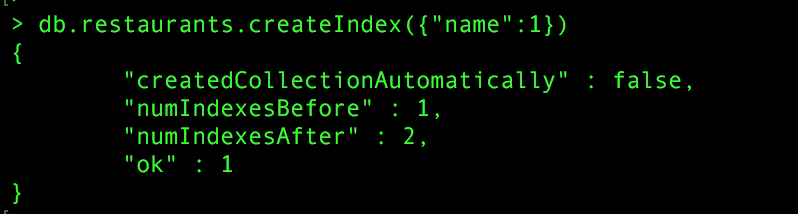
que son los campo que utilizo de query en el agregate me dice lo siguiente:

**

No ha utilizado índice("totalKeysExamined"=0) y ha examinado todos los documentos de la base de datos.

1. **Plantea y crea varios índices con el objetivo de optimizar dicha consulta. ¿Qué índices has creado?**
2. **Estudia de nuevo el plan de ejecución y razona qué índice es el mejor y por qué.**

Creo el índice por nombre del restaurante:



Ahora voy a ver las estadísticas del find anterior:

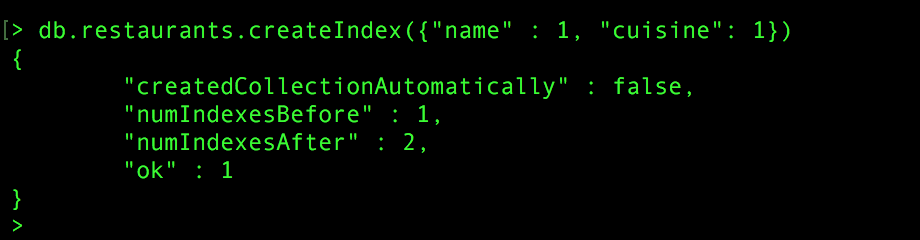


Ahora el stage principal dice IXCAN, que significa que el recorrido lo está haciendo por índice.

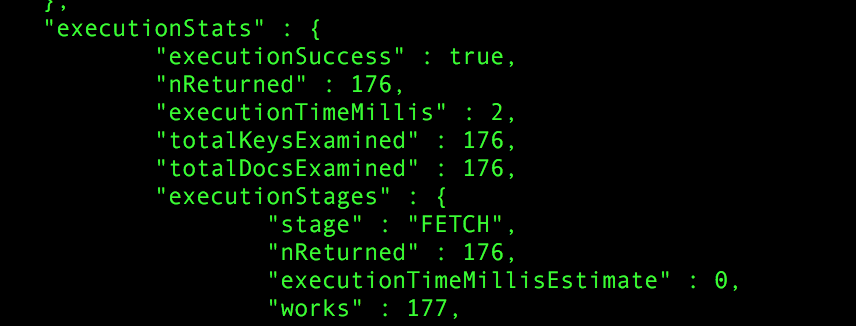


Aquí vemos que efectivamente se ha utilizado el índice para obtener los datos, vemos que se han recorrido muchos menos documentos que en el caso anterior. Pero, el número de documentos devueltos es inferior al número de documentos examinados, lo que quiere decir que la búsqueda la hemos optimizado con respecto a la anterior, en el sentido de que se han recorrido muchos menos registros que en la búsqueda anterior. Pero hay margen de mejora.

Creamos ahora un índice múltiple:

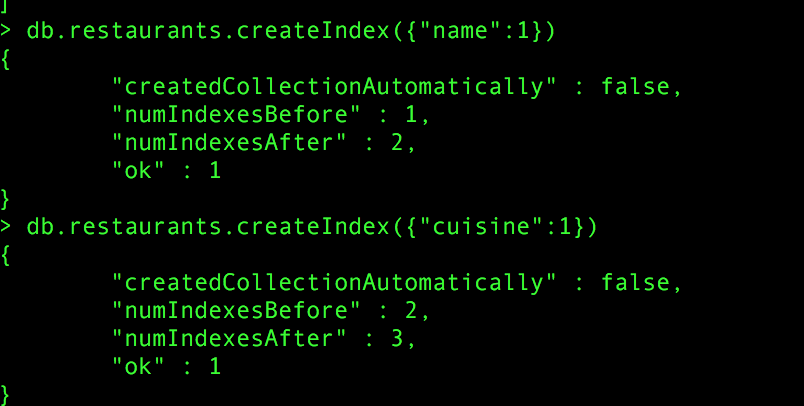


Revisando el plan de ejecución de la consulta:

****

Se puede ver ahora que el número de documentos examinados es igual al número de documentos devueltos, con lo que hemos optimizado el índice, ya que hemos generado un índice que cubre la consulta, todos los campos de la consulta forman parte del índice y los campos de resultado están en el índice con lo que no es necesario acceder a la colección de documentos.

Creamos ahora dos índices independientes:



revisando el plan de ejecución:



En este caso, tener dos índices separados, no mejora la consulta con respecto al índice compuesto. Con lo que efectivamente, lo mejor es tener un índice compuesto por los campos por lo que vamos a generar la consulta.

# Ejercicio 2 - MongoDB [20 puntos]

Habrás observado que el modelo de datos de restaurantes solo cubre el área geográfica de NYC. Supongamos que somos una startup neoyorquina que tiene una aplicación móvil de localización de restaurantes y damos a nuestros usuarios la funcionalidad de valoración. Nuestra aplicación tiene éxito en NYC así que nos planteamos el plan de dominación mundial y nos planteamos extender su uso al resto del mundo.

1. **Haz un juicio crítico del modelo actual, ¿Le encuentras alguna debilidad? ¿Se te ocurre algún modelo mejor?**

El modelo actual:

* No tiene campos para almacenar los datos de restaurantes de otra zona que no sea NYC. De hecho, lo que guarda en el campo *borough* son zonas de NYC, pero no se encuentra el dato NYC en la base de datos.
* No tiene índices para poder realizar búsquedas de datos por localización

Adicionalmente añadir que no se están introduciendo correctamente los datos, por ejemplo, para el restaurante MacDonald’s, se encuentran datos como McDonald y Mc Donald, con lo que la calidad del dato de esta base de datos seria cuestionable.

1. **¿Qué modificaciones al modelo de restaurantes debes realizar para ir más allá de NYC y que sirva para el mundo entero?**

El campo adress lo dejaría asi:

"address" : {

"building" : "469",

"coord" : [

-73.961704,

40.662942

],

"street" : "Flatbush Avenue",

"zipcode" : "11225",

"borough" : "Brooklyn",

"City" : "New York",

"Country" : "USA"

},

El campo de Borough lo incluido dentro de la dirección del restaurante.

Adicionalmente, como la aplicación móvil es de localización de restaurantes, crearía dos índices:

* + Índice Geoespacial con las coordenadas del campo address. De modo que buscaríamos documentos (restaurantes) que estén cerca de una localización dada.
  + Índice por nombre del restaurante y localización: Para que dado un restaurante nos de las localizaciones cercanas de ese restaurante a partir de una localización dada.

1. **Toca ser escalables y montamos un *sharded cluster* de MongoDB. ¿Qué índice usarías para particionar la colección de restaurantes?**

Para responder a esta pregunta sería necesario saber cuáles son las consultas habituales que se hacen a través de la aplicación móvil. Vamos a partir de que lo habitual es que la gente consulte en la aplicación móvil cuales son los restaurantes cercanos a la zona donde están.

Partiendo de esta base, el índice sería el de geolocalización, ya que las consultas habitualmente serian de un rango de valores alrededor de ese índice.

1. **Haz un juicio crítico de tu particionamiento desde el punto de vista de balanceo de los datos en los *shards*.**

Con la partición escogida no vamos a tener balanceados los shards, ya que puede existir localidades que tengan muchos restaurantes para mismas localizaciones (ejemplo de las calles de Manhattan) y pueden existir otras localizaciones con 1 solo restaurante. Así que, aunque las consultas sean más rápidas, si aumentan los restaurantes en grandes ciudades podemos llegar a tener un problema de balanceo.

# Ejercicio 3 - Cassandra [20 puntos]

(Basado en hechos reales) Imaginemos que somos una empresa que tenemos 15 aplicaciones móviles desarrolladas a medida con sus correspondientes componentes del lado servidor. Queremos centralizar los logs de la parte servidora de dichas aplicaciones en una tabla en Cassandra. La base de datos estará desplegada en clúster.

1. **Diseña la tabla que almacenará los mensajes de categoría WARN, ERROR y FATAL.**

CREATE TABLE TablaLogs (

aplicacion TEXT,

mensaje TEXT,

tipo TEXT,

semana TIMESTAMP,

fecha\_hora TIMESTAMP,

PRIMARY KEY ((semana), fecha\_hora, aplicacion, tipo) )

Guardaría el mensaje de log entero, identificando que aplicación lo ha dado, el tipo de error y sobre todo la semana y la fecha y hora en que se ha producido.

1. **¿Cuál imaginas que es el patrón de consulta habitual a tu tabla?**

Imagino que lo importante de estas consultas es saber que problemas está habiendo o hubo entre determinadas fechas de una semana, por lo cual la clave para mí siempre va a estar relacionado con la fecha.

SELECT \* FROM TablaLogs WHERE semana= and fecha\_hora => ?

SELECT \* FROM TablaLogs WHERE semana= and fecha\_hora => ? AND aplicación=?

SELECT \* FROM TablaLogs WHERE semana= and fecha\_hora => ? AND aplicacion=? AND tipo = ?

¿Qué logs hubo ayer?

¿Qué errores hubo de la aplicación X?

¿Qué errores de tipo Fatal hubo ayer?

1. **En función de tu respuesta a la pregunta anterior, ¿cuál es tu estrategia (escalable) de particionamiento de los logs en el clúster?**

Por lo que se, los errores o los logs de una aplicación más o menos se mantienen constantes a lo largo de un día. Una aplicación tiene más o menos los mismos usuarios, y las operaciones a realizar en las aplicaciones seguirían una distribución normal, así que a no ser que haya un error FATAL que desencadena muchos errores (no debería ser habitual).

Además, las consultas entre fechas para ver que ha pasado con las aplicaciones tienen sentido en un corto espacio de tiempo, normalmente se analiza un día, o días distintos para detectar el porqué de los errores o que está sucediendo (analizar muchos días suele ser imposible por volumen de datos). De hecho, cuando se revisa un error Fatal, se mira que errores Warning han ocurrido “alrededor” de ese error Fatal (filtrando por fechas y hora)

Con lo cual, la clave de particionamiento he escogido que sea una semana, creo que por tamaño de datos seria optimo, y además el rango de las fechas considero que debería estar dentro del intervalo de una semana, de modo que siempre se consultaría dentro del mismo particionamiento.

# Ejercicio 4 - Cassandra [20 puntos]

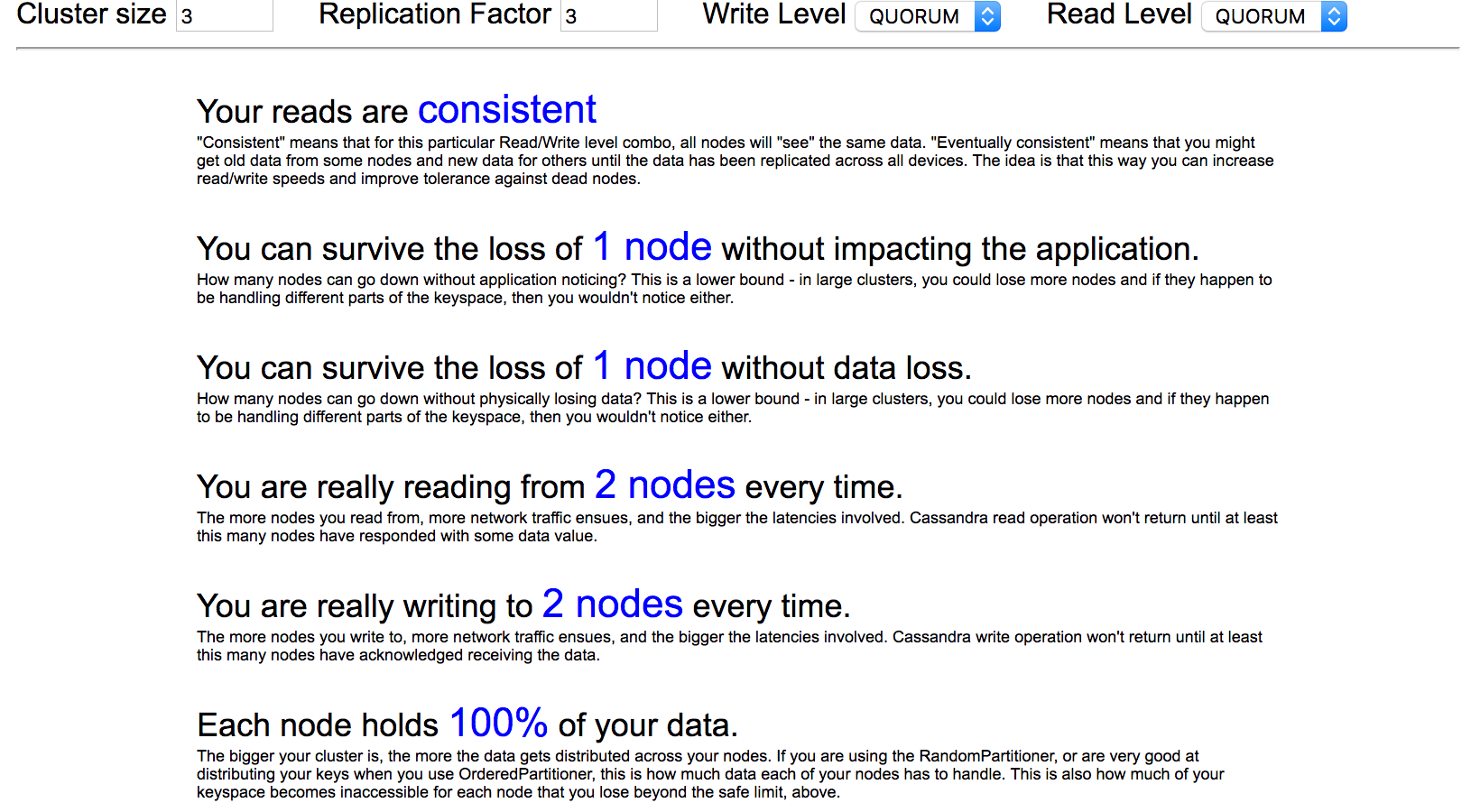
Todos los bancos tienen un sistema informático que se conoce como "diario electrónico", "libro diario" o nombre similar. Se trata de un libro contable en el que se almacenan todas las operaciones que se realizan y sobre el que se realizan las operaciones básicas de cuadre contable. Se vienen usando bases de datos relacionales tradicionales para ello (¡transaccionalidad manda!) pero nos gustaría usar un clúster de 3 nodos de Cassandra para ello porque en nuestro banco somos muy modernos. Sabemos que Cassandra no es transaccional así que deseamos el máximo nivel de garantía de escritura de las operaciones.

1. **¿Qué nivel de CL usarías para las escrituras? ¿Y para las lecturas?**

Para las escrituras: QUORUM

Para las lecturas: QUORUM

Es decir, si tengo 3 nodos y factor de replicación 3, con asegurar la lectura y escritura en dos nodos tendría la base de datos consistente:



Además, tendría contingencia en el sentido de que si se cae un nodo, seguiría teniendo servicio sin pérdida de datos.

1. **¿Qué factor de replicación aconsejas? Haz una valoración crítica de la disponibilidad de tu clúster en base a los valores que has escogido.**

Si tenemos 3 nodos, el factor de replicación tendría que ser 3, para que se escribiera los datos en los 3 nodos.

El problema de esta configuración con máximo nivel de garantía, es que tenemos que asegurar la escritura en dos nodos, lo que ralentizará las escrituras.

Si cambiamos, y ponemos como Write Level 1 y Read Level ALL, haríamos que las escrituras fueran más rápidas, pero no tendríamos contingencia, si se cae un nodo no habría servicio. Por lo que, poniendo ambas circunstancias en la balanza, creo que es mejor asegurar el servicio, y por eso he escogido la configuración con Quorum.

# Ejercicio 5 - Cassandra [10 puntos]

Estudiando Cassandra trabajamos un ejemplo de modelo de datos para IoT sobre el caso de redes de sensores. En el material de Casandra podéis repasar un planteamiento de modelo de datos escalable.

Imaginemos que somos una startup y nuestro negocio se basa en proveer de una infraestructura SaaS de captación, almacenamiento en Cassandra y analítica web de la información de sensores. El modelo SaaS implica que debemos dar servicio a muchos clientes que almacenan en un nuestro clúster de Cassandra los datos de sus redes de sensores. Dicho de otro modo, nuestro software deberá ser [*multitenancy*](https://es.wikipedia.org/wiki/Tenencia_M%C3%BAltiple).

Hay varias opciones para soportar muchos clientes con la misma infraestructura tecnológica.

1. **Imagina y enumera posibles opciones. ¿Cuál te parece mejor?**

Para compartir la misma infraestructura se me ocurren distintos niveles:

* El mismo HW, pero una instancia der servidor de aplicaciones por Cliente, con instancias de BBDD distintas por Cliente.
* El mismo HW, y una combinación de las anteriores, Si compartir servidor web y NO compartir instancias de BBDD y a la inversa.
* Para complicarlo mas, en cuanto a la BBDD podríamos tener la misma instancia, pero una colección de documentos por Cliente.
* Compartirlo todo.

La ventaja que veo de compartir todo es una reducción de costes en el código, ya que solo tendría que implementarse e instalarse en una sola instancia. También veo que habría ahorros de licencia, y aunque el HW en principio debería ser más potente, el tamaño estaría mucho más optimizado en función del uso.

Por el contrario, las pegas que veo de compartirlo todo tiene que ver con la seguridad y la privacidad de los datos. El código seria más complejo, y la administración de la seguridad y privacidad también.

Viendo las ventajas y desventajas, creo que la mejor opción es compartirlo todo, aunque para ello el código sea más complejo y haya que prestar mucha atención en velar por la seguridad y privacidad de los datos de cada cliente.

1. **Si tu mejor opción es conservar tu clúster de Cassandra para dar servicio a todos los clientes, ¿qué modificaciones harías al modelo de datos visto en clase para que fuese *multitenant*?**

* En todas las tablas añadiría un campo de identificador de cliente.
* En todas las claves de particiones añadiría el identificador de cliente.

Ejemplo:

CREATE TABLE sensores\_por\_red (

red INT

sensor TEXT

localizacion TEXT

Cliente TEXT

caracteristicas MAP<TEXT,TEXT>,

PRIMARY KEY ((Cliente, red), sensor)

);

De formas que aseguras de que, si existen redes con el mismo nombre en distintos clientes, van a estar en nodos distintos, es como si hiciéramos una partición virtual por cliente.