Albert Vacas Martínez NIU: 1665473

José Ortín López NIU: 1667573

ATS – Pràctica 1

Observaciones Datasets:

Valores repetidos en restaurants

Diseño del esquema de la base de datos:

1. Observando los datos de restaurantes e inspecciones llegamos a la conclusión que el tipo de relación usado es **One-to-Million.** Este tipo de relación se usa cuando un documento está relacionado con una gran cantidad de datos, en este caso, donde cada restaurante puede tener millones de inspecciones asociadas.

   En lugar de almacenar todas las inspecciones dentro del documento de restaurantes, puedes hacer referencias cruzadas entre la colección restaurantes y una colección separada de inspecciones para almacenar-las.

2. En el caso de añadir una nueva colección que no utilice las referencias e incorpore los documentos embebidos, nos proporciona más flexibilidad y eficiencia en las consultas. Esto es debido a que puedes realizar consultas más eficientes accediendo más fácilmente a los datos.
   Por ejemplo, podríamos tener dentro de la colección restaurantes un objeto llamado inspecciones donde podemos guardar todas las inspecciones realizadas a ese mismo restaurante. Así podemos facilitar la búsqueda de inspecciones sabiendo el restaurante donde se ha realizado.
   Un posible problema de esta configuración es que requiere hacer consultas cruzadas o lookup para obtener detalles completos de la inspección.

3. Hemos diseñado un esquema de validación para garantizar la seguridad y consistencia de los datos en las colecicones inspections y restaurants.

   Por un lado, en la colección inspections hemos considerado los campos restaurant_id, sector, date, result y certificate_number como obligatorios a la hora de instanciar una inspección. Esto es debido a que cada inspección debería especificar obligatoriamente cuando se ha realizado, en que restaurante, en qué sector y cuál es el resultado obtenido. El campo de certificate_number se define como un int y consideramos obligatorio para poder validar la inspección. En cambio, el campo adress no lo consideramos obligatorio ya que tenemos otro campo (restaurant_id) que referencia el restaurante correspondiente que ya contiene la dirección. El campo id no lo consideramos importante ya que cada documento tiene su propio _id.

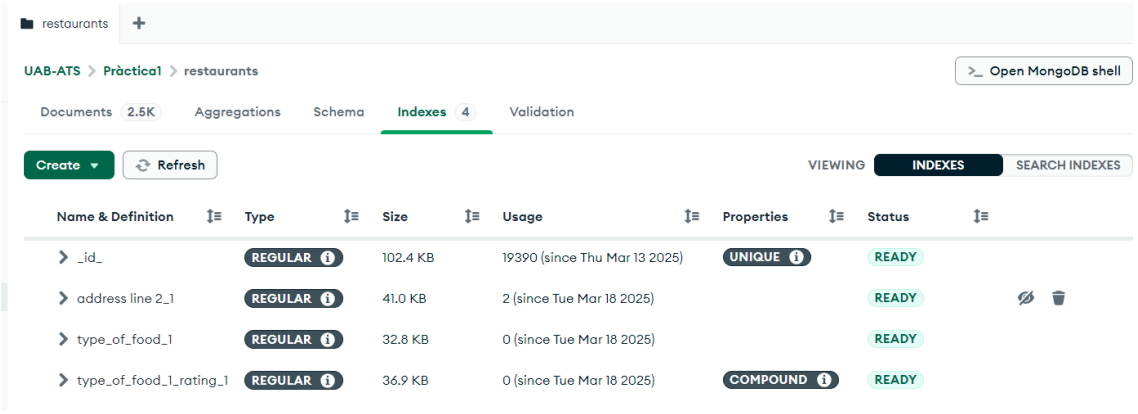
Por otro lado, en la colección restaurants hemos considerado todos los campos obligatorios ya que son imprescindibles para verificar donde se ubica el restaurante y de que tipo és, su nombre, las valoraciones recibidas y el link a la página web de este para consultas de los clientes.

Optimización de rendimiento

Hemos considerado las siguientes consultas como las más frecuentes:

- Restaurantes por el tipo de comida
    - Creemos que esta consulta es muy frecuente porque las personas suelen tener una preferencia clara de un tipo de comida específico, como “pizza”, “sushi”. En lugar de revisar todas las opciones disponibles, los usuarios quieren filtrar rápidamente por restaurantes que ofrezcan el tipo de comido que desean en ese momento
- Restaurantes de la ciudad
    - Creemos que esta consulta es muy frecuente porque muchas personas buscan restaurantes según su ubicación, especialmente si están de viaje o en una ciudad que no conocen. Por lo tanto, el filtrar por ciudad es muy común que tener que revisar-lo manualmente
- Restaurantes según el tipo de comida con un rating mayor a 5
    - Creemos que esta consulta es muy frecuente porque a parte de buscar un tipo de comida en particular, los usuarios suelen buscar que el restaurante tenga buenas valoraciones. Sobre todo, en situaciones de viaje donde no sabes diferenciar bien entre un buen sitio o un sitio malo hecho para que los turistas caigan en la trampa esta consulta es muy recurrente
- Inspecciones con una fecha concreta
    - Creemos que esta consulta es muy frecuente por las empresas o los controles de calidad, ya que permite revisar qué establecimientos han sido inspeccionados en un día en específico. Para la generación de gráficos donde se muestra la media de inspecciones hechas en un día también es de gran utilidad
- Inspecciones de un restaurante en una fecha concreta
    - Creemos que esta consulta es muy frecuente tanto para las empresas que toman un control de las inspecciones de un restaurante como para los usuarios que quieran conocer el estado del restaurante.

Hemos generado los siguientes índices:



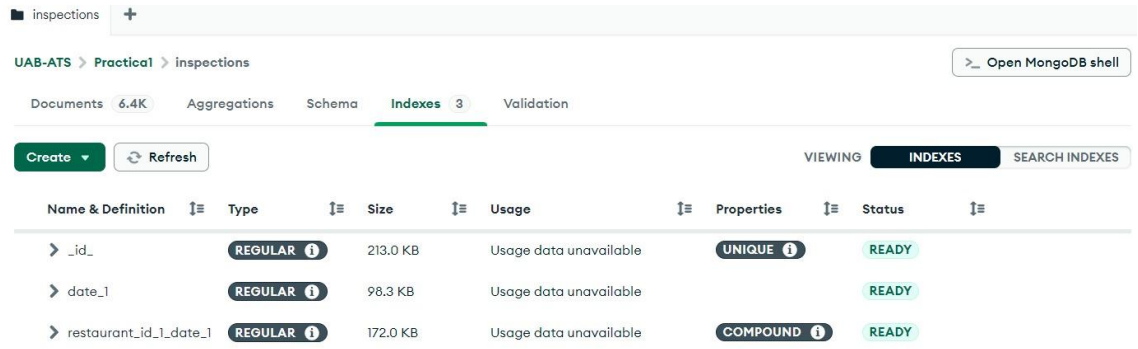


Tabla comparativa de resultados de consultas con índice simple:

Métrica	Sin índice (COLLSCAN)	Con índice (IXSCAN)
totalDocsExamined (Documentos escaneados)	2548	25
totalKeysExamined (Claves de índice examinados)	0	25
executionTimeMillis (Tiempo de ejecución)	1ms	0ms
nReturned (Documentos devueltos)	25	25

Tabla comparativa de consultas con índice compuesto:

Métrica	Sin índice (COLLSCAN)	Con índice (IXSCAN)
totalDocsExamined (Documentos escaneados)	2548	3
totalKeysExamined (Claves de índice examinados)	0	3
executionTimeMillis (Tiempo de ejecución)	1ms	0ms
nReturned (Documentos devueltos)	3	3

Hemos observado que el uso de índices en MongoDB marca una gran diferencia en la eficiencia de las consultas. Una de las principales diferencias que observamos es el tiempo de respuesta. Cuando no usamos índice mongoDB realiza un escaneo completo de la colección (COLLSCAN). Por otro lado, cuando se usa índice mongoDB utiliza un escaneo de índice (IXSCAN) que permite no tener que escanear toda la colección, sino que utiliza el índice para encontrar los documentos y esto se traduce en tiempos de respuesta mas cortos. En la combinación de múltiples condiciones (por ejemplo, buscar restaurantes de una ciudad y con un "rating" mayor a 5), la consulta se vuelve aún más cotosa. Para estos casos, los índices compuestos reducen en mayor cantidad el coste de busca de documentos.

Estrategias de escalabilidad

Para fragmentar una colección, debemos elegir una clave de sharding, que será el campo (o combinación de campos) cuyo valor determinará en qué shard cae cada documento.

Hemos planteado dos distintos shard keys para cada uno de los dos datasets:

1. En restaurants, podemos usar la cuidad como shard key (adress line 2) ya que los restaurantes se distribuyen en distintas ciudades y las consultas frecuentes incluyen filtrado por ciudad. Así se reduce la carga en cada shard, ya que las consultas quedan localizadas.
   Al configurar los shards usamos hashing para distribuir las ciudades uniformemente entre estos shards, esto equilibra la carga de inserciones y evita el sesgo de sharding por ciudades con más restaurantes.

2. En inspections, se puede shardear por el campo restaurant_id ya que se consulta con frecuencia en joins con restaurants. Hasheamos los distintos valores de restaurant_id y así balanceamos mejor las inspecciones de diferentes restaurantes entre los shards.

Mongo usa replica sets para lograr una alta disponibilidad. Un replica set tiene que estar formado mínimo de 3 nodos.

- 1 primario → es el servidor que acepta todas las escrituras (inserciones, actualizaciones, borrados), y replica (copia) esas operaciones a los secundarios.
- 2 secundarios → almacenan copias exactas de los datos del primario, actualizándose constantemente a través del oplog.

A continuación, se analizan los principales cuellos de botella y las estrategias para solucionarlos.

- Un posible cuello de botella ocurre cuando las consultas no incluyen la shard key, lo que obliga a MongoDB a buscar la información en todos los shards, aumentando considerablemente la latencia. Para evitar este error hay que estructurar las consultas de manera que utilicen la shard key. En caso de que otros campos sean consultados frecuentemente hay que crear índices secundarios dentro de cada shard para optimizar la recuperación de datos.

- El proceso de balanceo de shards puede convertirse en un cuello ya que a medida que crecen los datos, MongoDB mueve chunks entre shards para equilibrar la carga. Si este proceso se realiza con mucha frecuencia puede consumir recursos y afectar el rendimiento del clúster. Para para solucionar el problema hay que ajustar el tamaño de los chunks a un valor adecuado y programar los procesos de balanceo fuera de las horas de mayor actividad para minimizar su impacto en el rendimiento de las consultas.

- El Config almacena la metadata del clúster y es esencial para la coordinación de los shards. Si solo se cuenta con un servidor de configuración, puede generarse un cuello de botella y, en caso de fallo, comprometer la disponibilidad del sistema. Para prevenir esto, hay que configurar un Config Server Replica Set (CSRS), asegurando redundancia y tolerancia a fallos.

- Los routers mongos pueden representar un cuello de botella si la cantidad de conexiones simultáneas es alta. Como mongos actúa como intermediario entre la aplicación y los shards, si hay un número elevado de consultas concurrentes, puede saturarse y ralentizar las respuesta. Para solucionar esto se pueden desplegar múltiples instancias de mongos en servidores distribuidos y utilizar un balanceador de carga para distribuir las conexiones entre estas instancias y mejorar la escalabilidad.