PEC FINAL DE ESD: Tania Batista

# **Ejercicio 1 - MongoDB [30 puntos]**

**(1) Modifica la consulta anterior para obtener el número de Macdonald's agregados por calle en todo NYC.**

|  |  |
| --- | --- |
| *db.restaurants.aggregate([*  *{$match:*  *{*  *"name": "Mcdonald'S",*  *"cuisine": "Hamburgers"*  *}*  *},*  *{$group:*  *{*  *"\_id": "$address.street" ,*  *"count": { $sum: 1 }*  *}*  *}, {$sort:*  *{*  *"\_id" : 1*  *}*  *}*  *]);* |  |

**(1b) ¿Cuántos Mcdonald's hay en la Sexta Avenida?**

|  |  |
| --- | --- |
| *db.restaurants.createIndex({"address.street": 1, "name": 1})*  *db.restaurants.aggregate([*  *{$match:*  *{*  *"address.street": "6Th Ave"*  *}*  *},*  *{$group:*  *{*  *"\_id": "$name",*  *"count": { $sum: 1 }*  *}*  *}, {$sort:*  *{*  *"\_id" : 1*  *}*  *}*  *]);*  *db.restaurants.aggregate([*  *{$match:*  *{*  *"address.street": "6 Avenue",*  *"name": "Mcdonald'S"*  *}*  *},*  *{$group:*  *{*  *"\_id": "$name",*  *"count": { $sum: 1 }*  *}*  *}, {$sort:*  *{*  *"\_id" : 1*  *}*  *}*  *]);* | *Existe 1 Mcdonald’s en "6Th Ave"*    *Existen 5 Mcdonald’s en "6 Avenue"* |

**(2) Estudia las estadísticas del plan de ejecución de tu consulta de agregación.**

**¿Qué observas en el *stage* principal?**

|  |  |
| --- | --- |
| *db.restaurants.explain("executionStats").aggregate([*  *{*  *$match: { "name": "Mcdonald'S" } }, {$group: {*  *"\_id": "$address.street" ,*  *"count": { $sum: 1 } } },*  *{$sort: { "\_id" : 1 }*  *}*  *]);* | *COLLSCAN:*  *recorrido de colección - no usa indices* |

**(3) Plantea y crea varios índices con el objetivo de optimizar dicha consulta.**

**¿Qué índices has creado?**

|  |  |
| --- | --- |
| *db.restaurants.createIndex(*  *{*  ***"address.street": 1,***  ***"name": 1,***  ***"cuisine": 1***  *})* |  |

**(4) Estudia de nuevo el plan de ejecución y razona qué índice es el mejor y por qué.**

*Es mucho mas sencillo y rapido CON indices, asi no tiene que examinar todos los documentos. Solo hay que examinar los “Keys” que son muchos menos!*

|  |  |  |
| --- | --- | --- |
| *AGGREGATE() WITHOUT INDEX*  *db.restaurants.explain("executionStats").*  *aggregate([ {$match: { "name": "Mcdonald'S", "cuisine": "Hamburgers" } }, {$group: { "\_id": "$address.street" , "count": { $sum: 1 } } }, {$sort: { "\_id" : 1 } } ]);* | *AGGREGATE() WITH INDEX*    *db.restaurants.explain("executionStats").*  *aggregate([ {$match: { "address.street": "6 Avenue", "name": "Mcdonald'S", "cuisine":"Hamburgers" } }, {$group: { "\_id": "$name", "count": { $sum: 1 } } }, {$sort: { "\_id" : 1 } } ]);* | *FIND() WITH INDEX*    *db.restaurants.explain("executionStats")*  *.find( { "name": "Mcdonald'S", "cuisine": "Hamburgers","address.street": "6 Avenue"} ).count()* |

# **Ejercicio 2 - MongoDB [20 puntos]**

**(1) Haz un juicio crítico del modelo actual, ¿Le encuentras alguna debilidad?**

**¿Se te ocurre algún modelo mejor?**

*El problema mas grande es que “address” y “borough” funcionan solo en New York. Tenemos que incluir una direccion completa (“address”) con “city” y “country”. “Language” podria ser otro “key” a incluir.*

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

|  |  |
| --- | --- |
| *Tendria que ser asi:*  *"\_id" : ObjectId("\_\_\_"),*  *"address" : {*  *"building" : "\_\_",*  *"coord" : [\_\_,\_\_],*  *"street" : "\_\_",*  *"zipcode" : "\_\_",*  ***“city”: \_\_\_,***  ***“country”:\_\_\_\_***  *},*  *"borough" : "\_\_",*  *"cuisine" : "\_\_",*  *"grades" : [*  *{*  *"date" : ISODate("\_\_"),*  *"grade" : "\_\_",*  *"score" : \_\_*  *},*  *],*  *"name" : "\_\_",*  *"restaurant\_id" : "\_\_"* | *> db.restaurants.find( { "name": "Mcdonald'S", "cuisine": "Hamburgers","address.street": "6 Avenue"} ).pretty()* |

**(3) Toca ser escalables y montamos un *sharded cluster* de MongoDB.**

**¿Qué índice usarías para particionar la colección de restaurantes?**

*Si vives en Francia por ejemplo, los restaurantes más relevantes para ti serían los que están en Francia, o quizás incluso en Suiza o Alemania. Entonces yo creo que los Sharded Clusters se deben dividir por region. Entonces* [*se pueden crear 7 zonas de “sharded clusters”*](https://docs.mongodb.com/manual/tutorial/sharding-segmenting-data-by-location/) *- una zona por* [*cada continente*](https://dev.maxmind.com/geoip/legacy/codes/country_continent/)*: EU, NA, SA, OC, AS, AF, AN.*

*Los rangos de cada pais ("Range-Based”) se añaden al sharded cluster correspondiente así:*

*{ "country" : <country>, "restaurant\_id" : MinKey }  
 { "country" : <country>, "restaurant\_id" : MaxKey }*

*Por ejemplo, para añadir los restaurantes franceses al sharded cluster del data center de “EU”:*

*{ "country" : FR, "restaurant\_id" : MinKey }*

*{ "country" : FR, "restaurant\_id" : MaxKey }*

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

*Yo elegí un particionamiento “Range-Based” porque es ideal para consultas basadas en rango de valores, y estoy asumiendo que los restaurant\_id’s ya están divididos en rangos. Si los id’s empiezan con el código numérico de ISO 3166-1, cada país ya tiene un rango distinto. Por ejemplo, si todos los id’s de estados unidos empiezan con* [*840*](https://en.wikipedia.org/wiki/ISO_3166-1_numeric)*, es más eficiente hacer la consulta solo en el shard de Norte America dónde están alemacenados esos datos. También es importante recordar que lo más relevante para un usuario es la zona. Lo demás (como ratings, o cuisine) no es relevante si ni viven en el mismo continente. Solo es un desperdicio de tiempo hacer que la aplicación consulte los datos de restaurantes del otro lado del mundo.*

*Pero si la distribución de id’s es aleatoria, desde el punto de vista de balanceo, podría ser mejor usar “Hash-Based” para evitar una distribución desbalanceada. Por ejemplo si la zona de Australia tiene solo 15 restaurantes registrados, y Norte America tiene 15,000, el Shard de Norte America tiene que procesar mil veces más operaciones. Si los restaurant\_id’s no tienen valor ordinal, no importa se distribuyen aleatoriamente en forma Hash-Based porque no hay rangos.*

# **Ejercicio 3 - Cassandra [20 puntos]**

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

[*Cassandra uses the standard Java log library Log4j.*](https://logging.apache.org/log4j/2.x/manual/customloglevels.html)

*WARN: temporary problems (ex failed user login)*

*ERROR: events that might allow driver to continue running, but should be resolved for proper functioning (ex connecting to DB)*

*FATAL: causing application to crash or fail (ex out of memory)*

*import com.mycompany.myproject.ExtLogger;  
public class MyService {  
 private static final ExtLogger logger = ExtLogger.create(MyService.class);  
 public void demoExtendedLogger() {  
 logger.warn("the built-in WARN level");  
 logger.error("the built-in ERROR level");  
 logger.fatal("the built-in FATAL level");  
 logger.notice("java 8 lambda expression only executed if NOTICE is enabled: {}", () -> someMethod()); }*

[*Log entries have the following attributes in a “Dynamic Table”*](https://www.datastax.com/dev/blog/the-most-important-thing-to-know-in-cassandra-data-modeling-the-primary-key)

*CREATE TABLE log (  
 id UUID,  
 log\_level TEXT,*

*message TEXT,*

*application\_name TEXT,*

*added\_date TIMEUUID,*

*user\_id TEXT,*

*file\_size TEXT,*

*platform TEXT*

*PRIMARY KEY (id) ) WITH CLUSTERING ORDER BY (added\_date DESC, user\_id ASC);*

**(2) ¿Cuál imaginas que es el patrón de consulta habitual a tu tabla?**

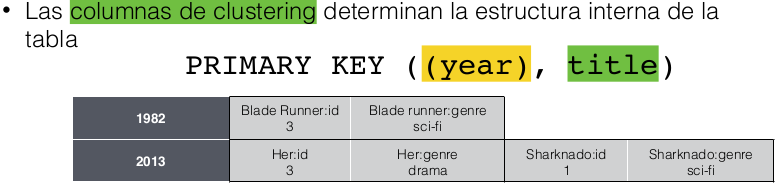
*PRIMARY KEY (id) WITH CLUSTERING ORDER BY (added\_date DESC, user\_id ASC);*

*Or:*

*PRIMARY KEY ((id), added\_date, user\_id)),*

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

*Como no se pueden hacer JOINs en Cassandra, tenemos que crear la tabla “log” para que represente consultas en vez de entidades. (Por ejemplo, usando WHERE). Para lograr esto, lo más importante es la PRIMARY KEY y las columnas de CLUSTERING*

**

*Columna “added\_date” crece horizontalmente todos los días, entonces se puede usar como CLUSTERING-COLUMN1 .*

*“user\_id” no es una clave única, entonces se puede repetir horizontalmente también como CLUSTERING-COLUMN2.*

*Para ver si la platforma de iOS es la que más errores FATALes tuvo ayer en la aplicación de Angry Birds, se logra hacer sin ningún JOIN:*

*SELECT platform  
FROM log  
WHERE application\_name = “Angry Birds” AND log\_level = “FATAL” AND added\_date=\_\_\_  
LIMIT 1000;*

**Ejercicio 4 - Cassandra [20 puntos]**

**(1) ¿Qué** [**nivel de CL**](https://youtu.be/d9NvnMcTVdQ) **usarías para las escrituras?**

*USING CONSISTENCY ALL*

*(porque solo hay 3 nodos, y deseamos máximo nivel de garantía de escritura. Esto asegura consistencia inmediata)*

**(1b) ¿Y para las lecturas?**

*USING CONSISTENCY ONE*

*(porque se realizan operaciones básicas. Aquí sería más importante latencia baja)*

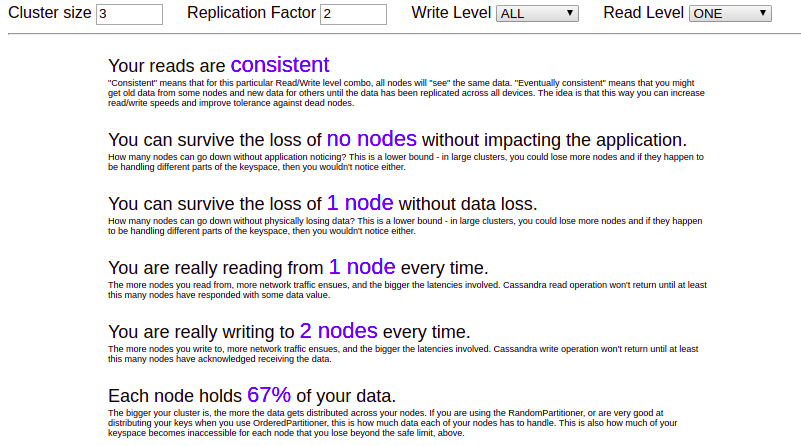
**(2) ¿Qué factor de replicación aconsejas? Haz una valoración crítica de la disponibilidad de tu clúster en base a los valores que has escogido.**

*“if (nodes\_written + nodes\_read) > replication\_factor: then immediate consistency”*

*3 nodes written + 1 node read = 4.*

*Entonces podemos tener RF > 4.*

*Con RF = 1 todavía es consistente, pero si no queremos arriesgar data loss, entonces es mejor RF = 2.*



# 

# 

# 

# 

# **Ejercicio 5 - Cassandra [10 puntos]**

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

*Different approaches to design a multi-tenants database:*

* *Dedicated database: Separate databases per tenant*
* *Dedicated table and different schema: Shared database and separate schema*
* ***BEST OPTION = Share table/schema: Same database and same table***

*SEGURIDAD:*

*• Posibilidad de definición de usuarios para conexión desde drivers, cqlsh, etc.  
• Posibilidad de canal encriptado con SSL entre el cliente y el clúster.  
• Posibilidad de encriptación de tráfico del clúster.  
• Posibilidad de establecimiento de autorizaciones por recurso:  
 • Asignación: GRANT, REVOKE  
 • Permisos: ALTER, AUTHORIZE, CREATE, DROP, MODIFY, SELECT  
 • Recursos: ALL KEYSPACES, KEYSPACE, TABLE*

**(2) 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*?**

*“conservar tu clúster de Cassandra para dar servicio a todos los clientes” ⇒ si cambiamos el modelo de datos para que el PRIMARY KEY sea “client\_id”, así cada cliente tendrá su propia partición. Eso conserva el cluster de Cassandra.*

[*Spread Data Evenly Around the Cluster*](https://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling)

*You want every node in the cluster to have roughly the same amount of data. Rows are spread around the cluster based on a hash of the* [*partition key*](https://www.datastax.com/documentation/cassandra/2.1/cassandra/architecture/architectureDataDistributeHashing_c.html)*, which is the first element of the* [*PRIMARY KEY*](https://www.datastax.com/documentation/cql/3.0/cql/ddl/ddl_compound_keys_c.html)*. So, the key to spreading data evenly is this: pick a good primary key.*

**