# Práctica Final - NoSQL

### Después de investigar y comparar las diferentes tecnologías NOSQL y evaluar cuál de ellas se ajustaba mejor al enunciado planteado, se ha decidido seleccionar MongoDB, a continución se exponen los motivos de la elección.

### Según el problema planteado, se cuenta con un volumen de información entorno a 500K de registros diarios, lo cual hace que sea lógico utilizar un modelo de base de datos el cual nos facilite hacer operaciones sobre estos datos con el menor coste posible.

### La principal razón de elegir MongoDB es que nos permite utilizar indices compuestos, no podemos pensar en tener una colección con millones de documentos, sin tener índices sobre uno o varios campos. Las diferencia entre realizar una consulta sobre campo con índice, y realizarla sin él, puede ser crítica. Además, en términos generales, MongoDB es mejor para cargas de trabajo con muchos datos altamente desestructurados, puede ser una gran opción si necesita escalabilidad y almacenamiento en caché para análisis en tiempo real. MongoDB se utiliza con frecuencia para aplicaciones móviles, administración de contenido, análisis en tiempo real y aplicaciones relacionadas con el Internet de las cosas.

### Algunas de estas características las comparte con Cassandra, pero, teniendo en cuenta que las ventajas de Cassandra sobre MongoDB son la alta disponibilidad y el lenguaje CQL, que es muy parecido al SQL, y que el enunciado no dice nada acerca de la insfractustura y el conociemiento de los empleados, se ha elegido MongoDB.

### Se ha optado por crear una colección de documentos llamada pasajeros, donde se almacenará un documento por cada pasajero, en el cual se irán añadiendo los datos obtenidos en las diferentes estancias del aeropuerto.

### Este diseño tiene todas las ventajas y desventajas de la incrustación. La principal ventaja es que no tiene que realizar una consulta por separado para obtener los detalles incrustados; la principal desventaja es que no tiene forma de acceder a los detalles integrados como entidades independientes, pero teniendo en cuenta que para el problema estamos seguros de que no se va a necesitar más datos de ninguna de las posibles entidades como ‘arcoSeguridad’ o ‘embarque’, es una posible solución.

### En este caso es útil saber que para crear índices que solo incluyan los documentos cuyo campo indexado existe, se utiliza la opción sparse, ya que nuestro modelo tiene datos que pueden existir o no en los diferentes espacios tiempo en el documento.

### También hay que tener en cuenta, que para que el sistema tenga éxito a la hora de almacenar los datos y tratarlos para generar información, es importante que los pasos de cada pasajero se almacen fielmente a la realidad, se podría utilizar la tarjeta de embarque como medio para efectuar cualquier compra, además de para pasar por el arco de serguridad y la puerta de embarque. En cada uno de estos puntos el sistema haría un find del usuario y completaría la información que falta.

db.createCollection(“pasajeros“);

db.pasajeros.insert( {

fecha\_arcoSeguridad,

hora\_arcoSeguridad,

hora\_embarque,

puerta\_embarque,

destino,

equipaje\_facturado,

edad,

movilidad\_reducida

});

db.users.ensureIndex( { "cualquiercampo" : 1 }, {"sparse":true} )

db.pasajero.findOne({\_id : 'id'})

### Las consultas realizadas de forma nativa serían algo complicadas, no obstante, se podrían simplificar utilizando un ORM en Java, o en cualquier otra tecnología. Se ha intentado hacer una aproximación a lo que sería la consulta.

### // tiempo medio que pasan los pasajeros que se dirigen a un determinado aeropuerto calculado para cada mes.

db.pasajeros.aggregate([

{

$match: {"aeropuerto\_destino": "AEROPUERTO"}

}

{

"$project": {

"auxMonth": { "$month": "$fecha\_arcoSeguridad" },

}

},

{

"$group": {

"\_id": “$auxMonth”,

"average": { "$avg": { $subtrack: [ "$hora\_embarque", "$hora\_arcoSeguridad" ] }},

}

}

])

### // tiempo medio que pasan los pasajeros en la zona de embarque que se dirigen a una determinada puerta de embarque, pudiendo especificar la consulta un destino concreto o no.

db.pasajeros.aggregate([

{

$match: {"puerta\_embarque": "PUERTA\_EMBARQUE"}, {"aeropuerto\_destino": "AEROPUERTO"}

}

{

"$group": {

"\_id": “null”,

"average": { "$avg": { $subtrack: [ "$hora\_embarque", "$hora\_arcoSeguridad" ] }},

}

}

])

### Para esta última consulta se han tenido muchas dudas sobre si era mejor seguir con el planteamiento seguido hasta ahora e insertar un campo más en el documento de pasajero, o era mejor crear una colección nueva (‘tiendas’) y después realizar una una agregación entre dos documentos, con el id del pasajero como dato común.

### // gasto medio realizado en una tienda o establecimiento en concreto por los pasajeros que se dirigen a un destino concreto, diferenciando entre los que han facturado equipaje y los que no.

db.pasajeros.aggregate([

])

### ANEXO: TABLAS RESUMEN COMPARACIÓN MONGODB VS CASSANDRA

### cassandra vs mongodb

### Cassandra-MongoDB-Panoply

Mateo Bernal Motnoya

Samuel Andreo Vilaplana