Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

Introducción a las bases de datos NoSQL: comparativa MongoDB vs Cassandra

Autor: Alejandro Díaz Moreno

Tutor/es: Iván González Diego

Julio de 2019

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

Introducción a las bases de datos NoSQL: comparativa MongoDB vs Cassandra

Autor: Alejandro Díaz Moreno

Tutor/es: Iván González Diego

TRIBUNAL:

Presidente:

Vocal 1º:

Vocal 2º:

CALIFICACIÓN: ……

FECHA: ……

Agradecimientos

En primer lugar me gustaría darle las gracias a mi tutor, Iván, por su ayuda, su simpatía y por siempre tener la puerta de su despacho abierta para mí y mis dudas, que no han sido pocas.

Agradecerle a mi familia todo lo que hacen por mí, a mi madre, por todo el apoyo que me da, he de confesar que me encantan tus mensajes de WhatsApp deseándome suerte minutos antes de un examen, a mi padre, ya sé que siempre me enfado cuando entras a mi habitación a ver cómo voy en momentos de estrés, a pesar de mi enfado es tu manera de mostrarme tu preocupación y tu apoyo, así que puedes seguir haciéndolo, y a mi hermana Gemma, ahora me pongo en tu lugar y entiendo cuando me decías que ya me tocaría vivir esto, infinitas gracias por tu ayuda, tu apoyo y tu preocupación por que me vaya bien en la vida.

He conocido durante estos cuatro años a grandes personas, en especial quiero darles las gracias a María, Paula y Daniela, son muchos los buenos momentos y risas que hemos vivido juntos y sin duda haberos conocido es lo mejor que me llevo.

Ortega, tú te mereces una mención especial, ya nos conocíamos antes de entrar aquí, pero en estos cuatro años posiblemente hayas sido la persona que mejor me ha entendido y la que más me ha sufrido, hemos estado muchas horas juntos, codo con codo, haciendo prácticas y trabajos, también hemos vivido muchísimos momentos buenos y risas que no se me olvidarán, gracias por todo y siempre estaré para lo que necesites.

Por último gracias a mis amigos, llevamos ya muchos años juntos y seguís a mi lado apoyándome en todo, gracias también por ser unos sosos y no salir nunca de fiesta, a vuestra manera me habéis ayudado a aprobar la carrera.

Índice

Índice de tablas

Índice de ilustraciones

Resumen

Este proyecto trata de comprender el concepto de las tecnologías NoSQL en un plano general y una comparación con las bases de datos relacionales como uno de sus objetivos principales. Otro de estos objetivos, es la comparación de dos de las bases de datos NoSQL más utilizadas, MongoDB y Apache Cassandra. Para poder realizar esta comparación de manera muy detallada, se han descrito técnicamente, pero no solo interesa saber qué características tiene cada una, sino también, es importante conocer cómo se comportan en aplicaciones a nivel de usuario y el rendimiento que pueden proporcionar, para esto se ha desarrollado una implementación real llamada BIBLIOTEQ.

Abstract

This Project is about to understand the concept of NoSQL technologies in a general level and a comparison with relational databases as one of their main objectives. Another objective is the comparison of two of the most used NoSQL databases, MongoDB and Apache Cassandra. In order to make this comparison of both in a very detailed way they have been described technically, but there is not only interest in knowing what features each one has, but also, it is important to know how they behave in applications at the user level and the performance they can provide us, for this has been developed a real implementation called BIBLIOTEQ.

Palabra clave

NoSQL, bases de datos relacionales, BIBLIOTEQ, MongoDB, Apache Cassandra, driver, cliente Java, API, publicaciones, arquitectura.

1. Introducción a las bases de datos NoSQL

1.1. Origen del NoSQL

Actualmente el manejo de la información cada vez se está haciendo más complejo debido a las grandes cantidades de datos que hay que recopilar y tratar. En estos últimos años ha aumentado el interés de los sistemas de bases de datos NoSQL, ya que facilitan el manejo de la información, y las organizaciones cada vez almacenan una mayor cantidad de datos no estructurados.

Las bases de datos relacionales han sido las más utilizadas hasta el momento, pero con el crecimiento de las aplicaciones Web, a las que acceden un gran número de usuarios simultáneamente, se ha dado paso a este nuevo concepto.

Para aquellas aplicaciones para las que no sirve una base de datos relacional, ya que no realizan una buena gestión de los datos a gran escala, aparecen las bases de datos NoSQL. Así que se puede afirmar que este tipo de bases de datos se crean para abordar las limitaciones de los sistemas de administración de bases de datos relacionales, pero esto no implica que las bases de datos relacionales vayan a desaparecer.

En conclusión las dos principales causas de su aparición son [1]:

- **Cantidad de la información**. Las grandes compañías manejan altos volúmenes de información, por lo que se dieron cuenta de que con la infraestructura que contaban no podrían manejar esa cantidad. Además el crecimiento de esta información es exponencial, por eso, son estas compañías las que están invirtiendo y creando soluciones NoSQL. A parte de esto hay que tener en cuenta que las bases de datos relacionales presentan problemas de escalabilidad, por eso se buscan otro tipo de soluciones.
- **Velocidad.** Actualmente, como usuarios, cuando se accede a una página Web o a una aplicación se busca que se muestre la información por pantalla de forma rápida, las consultas en bases de datos relacionales con un modelo muy complejo no lo permiten.

1.2. ¿Qué es NoSQL?

Para comenzar hay que tener claro qué significa NoSQL, no es una tarea fácil definirlo y su propio nombre suele llevar a confusión.

Con NoSQL no nos referimos a “no SQL”, sino a “no solo SQL”, por lo tanto es una alternativa al SQL y no es una exclusión de este, son complementarios y se pueden aplicar conceptos de consulta similares al SQL.

Este término engloba cualquier base de datos que no siga el modelo tradicional denominado sistema de gestión de bases de datos relacionales (RDBMS), lo que de verdad se busca con este concepto es tener un diseño más simplificado, un procesamiento rápido y una especialización para tratar los datos, alcanzando un mayor

rendimiento, puesto que en un RDBMS los datos tienen que adaptarse a las tablas y cumplir una serie de reglas de diseño, entre las que está la normalización.

Por lo tanto, NoSQL se puede definir como "un conjunto de conceptos que permite el procesamiento rápido y eficiente de conjuntos de datos con un enfoque en el rendimiento, la confiabilidad y la agilidad"[2].

1.3. Características de los sistemas NoSQL

A pesar de encontrar varias tecnologías NoSQL diferentes, se puede encontrar características comunes [1]:

- **Escalabilidad horizontal y habilidad de distribución.** En este tipo de sistemas existe una gran facilidad para añadir, eliminar o realizar operaciones con sus elementos (hardware) sin afectar a su rendimiento. Gracias a esta escalabilidad se puede realizar replicas y distribuir datos de manera más sencilla sobre servidores. Este almacenamiento distribuido aumenta la fiabilidad y escalabilidad, pero en contraposición hace que los costos aumenten.
- **Libertad de esquema.** Al tener un esquema flexible permite mayor libertad para modelar los datos. Esto facilita su integración con los lenguajes de programación orientados a objetos.
- **Modelo de concurrencia débil.** No implementan ACID (Atomicity, Consistency, Isolation and Durability), por lo que hay que tener un especial cuidado para asegurar la consistencia de las transacciones. En su lugar implementa BASE que se explicará de forma más extensa a continuación.
- **Consultas simples.** Las consultas necesitan de menos operaciones y son más sencillas, lo que hace que se gane en eficiencia y en simplicidad.

1.4. Teorema CAP (Consistency, Availability and Partition Tolerance)

Eric Brewer en el año 2000 propuso una percepción sobre los entornos distribuidos en la que comenta que estos no pueden garantizar de manera simultánea una consistencia, disponibilidad y tolerancia a particiones [2].

- **Consistencia.** Cuando se realiza una consulta o una inserción en la base de datos, siempre se tiene que recibir la misma información independientemente del nodo o del servidor desde que se realiza. Es decir, los usuarios tienen la posibilidad de acceder al mismo registro de manera simultánea.
- **Disponibilidad.** Es la garantía de que cada solicitud a un nodo reciba una respuesta de si ha sido o no resuelta satisfactoriamente.
- **Tolerancia a particiones.** Como los sistemas distribuidos están divididos en particiones, el sistema debe continuar funcionando en el caso de que se produzca una caída o un fallo parcial en el sistema.

Ilustración 1: Teorema de CAP

Para ser distribuidas y escalables, las bases de datos no pueden cumplir las tres características simultáneamente. En la imagen se pueden ver las condiciones que se cumplen en el teorema de CAP [18], se puede hacer una clasificación de las bases de datos NoSQL en función de las condiciones que cumplen.

Hay que tener en cuenta que esta clasificación no es definitiva, puesto que algunas bases de datos NoSQL pueden configurarse de tal manera que deje de cumplir unas condiciones por otras distintas.

Condiciones que se cumplen en el teorema de CAP [3]:

- **AP:** Garantiza disponibilidad y tolerancia a particiones, pero no la consistencia de manera total, en algunas de ellas se puede lograr la consistencia de manera parcial mediante las replicaciones. Esto quiere decir que no se puede garantizar que los datos sean iguales en todos los nodos todo el tiempo.
- **CP:** Garantiza la consistencia y la tolerancia a particiones, pero para ello hay que sacrificar la disponibilidad. A pesar de que no se pueda garantizar un 100% de disponibilidad tampoco quiere decir que no se pueda lograr, o alcanzar un buen nivel.
- **CA:** Garantiza la consistencia y la disponibilidad, debido a esto surgen problemas con la tolerancia a particiones ya que se garantiza que los datos son iguales siempre; y que el sistema estará disponible respondiendo a todas las peticiones.

Clasificación de las bases de datos en función de la condición que cumplen [4]		
Condición	**Características**	**Ejemplos**
CA. Consistencia y disponibilidad.	Confirmaciones dobles. Protocolos de validación de cachés.	Relacionales (Oracle, Mysql, SQL Server), Neo4J.
CP. Consistencia y particiones	Bloqueos "pesimistas". Ignorar las particiones más pequeñas.	MongoDB, HBase, Redis.
AP. Disponibilidad y particiones	Invalidaciones de caché. Resolución de conflictos	DynamoDB, CouchDB, Cassandra.

Tabla 1: Clasificación de las bases de datos en función del teorema de CAP

1.5. BASE: Basically Available, Soft State, Eventually Consistent

- **BA es por Basically Available (Básicamente Disponible).** Significa que puede haber una falla en alguna de las partes del sistema distribuido y el resto del sistema sigue funcionando correctamente. Por ejemplo, se tiene una base de datos NoSQL distribuida en 5 servidores (en ninguno hay replicación de datos) si uno falla el 20% de las consultas no funcionaran y el 80% restante sí. Esto permite que el sistema siga funcionando a pesar de que parte de él esté caído, para que ese 20% siga funcionando, una medida muy utilizada es tener réplicas de los datos en diferentes servidores, de esta manera siempre se podrán realizar el 100% de las consultas.
- **S es por Soft state (Estado suave).** Hace referencia al hecho de que los datos pueden sobrescribirse por otros datos más recientes. Esta propiedad se solapa junto con la tercera propiedad de las transacciones BASE que se va a explicar a continuación.
- **E es por eventually consistent (Eventualmente consistente).** Esto significa que en algunos momentos la base de datos puede tener un estado inconsistente. Como hay copias de los datos en diferentes servidores, puede haber en algunos de ellos en los que no tengan los mismos datos que en otros hasta que se realiza la replicación, y en ese momento se obtiene la consistencia de la base de datos. El tiempo de replicación depende de muchos factores como la carga del sistema, la velocidad de la red, incluso la disposición geográfica de los servidores [5].

2. Ventajas y desventajas de NoSQL frente a RDBMS

El primer paso para poder elegir uno de los dos sistemas en un proyecto es conocer las ventajas y desventajas de cada uno de los sistemas, de esta manera se sabrá en que entornos se desenvuelve mejor cada uno:

Ventajas de RDBMS	Desventajas de RDBMS
Como son más antiguas está más adaptado su uso.	Mal rendimiento de este tipo de bases de datos.
Tienen un gran soporte en la comunidad.	No son fácilmente escalables.
Tienen gran cantidad de herramientas complementarias para su gestión.	Sus consultas no son óptimas si se tienen gran cantidad de datos.
Aseguran la atomicidad de las transacciones.	No es conveniente modificar el esquema previamente establecido después de su puesta en marcha.

Tabla 2: Ventajas y desventajas de usar RDBMS

Ventajas de NoSQL	Desventajas de NoSQL
Es fácilmente escalable.	No aseguran la atomicidad. Se soporta la consistencia eventual.
Tiene carácter descentralizado.	No todas las bases de datos NoSQL son multiplataforma.
Son flexibles, lo que permiten adaptarse a diversas necesidades.	No se encuentran muchas herramientas de gestión que sean usables, por lo que al final se acaban gestionando por consola.
En vez de ejecutarse en máquinas con muchos recursos se pueden ejecutar en varias máquinas con pocos recursos.	
Optimizan las consultas de grandes cantidades de datos.	
No genera cuellos de botella como pasa en los sistemas RDBMS.	

Tabla 3: Ventajas y desventajas de usar NoSQL

Después de definir las ventajas y desventajas de los sistemas NoSQL y RDBMS, se va a poner en contexto estas dos tablas. Para ello hay que definir las principales diferencias de los sistemas NoSQL con los RDBMS [6]:

- No suelen utilizar el lenguaje SQL para realizar consultas, como mucho lo usan de apoyo. Hay varias bases de datos que utilizan su propio lenguaje.
- No utilizan estructuras fijas para realizar el almacenamiento de los datos, pueden utilizar tablas, aunque también existen otro tipo de almacenamiento como el de clave-valor, los grafos, etc.
- No se pueden realizar JOINs, esto es debido a que realizar este tipo de operaciones con grandes cantidades de datos puede llegar a ser muy costoso o incluso puede llegar a colapsar el sistema.
- Arquitectura distribuida: mientras que las bases de datos relaciones suelen tener una arquitectura maestro-esclavo, las NoSQL pueden estar implementadas en varias máquinas a la vez.

Una vez llegado aquí, hay que hacer la siguiente pregunta: **¿cuándo debería usar una base de datos NoSQL en el proyecto?**

Si en el proyecto se necesita escalabilidad, si no se disponen de grandes máquinas, los datos van a tener una estructura variable, se tienen que analizar grandes cantidades de

datos y respetar la integridad de estos datos no es una necesidad primordial, utilizar una base de datos NoSQL puede ser la mejor opción, como pueden ser en aplicaciones IoT que tienen que almacenar una gran cantidad de datos de los sensores que van a reportar valores en tiempo real, también es útil usarlas para redes sociales y para aplicaciones Big Data.

Por otro lado, si la integridad de los datos es primordial para el proyecto tendremos que usar las bases de datos relacionales como en las aplicaciones para los bancos [7].

3. Tipos de bases de datos NoSQL

3.1. Bases de datos clave-valor

Es uno de los modelos más populares dentro de las BBDD NoSQL y es uno de los más sencillos. Este tipo funciona de la siguiente manera: cada elemento está identificado por una clave única, lo que hace que las operaciones de escritura y lectura sean mucho más rápidas. La información se suele almacenar como un objeto binario (BLOB) [8,19].

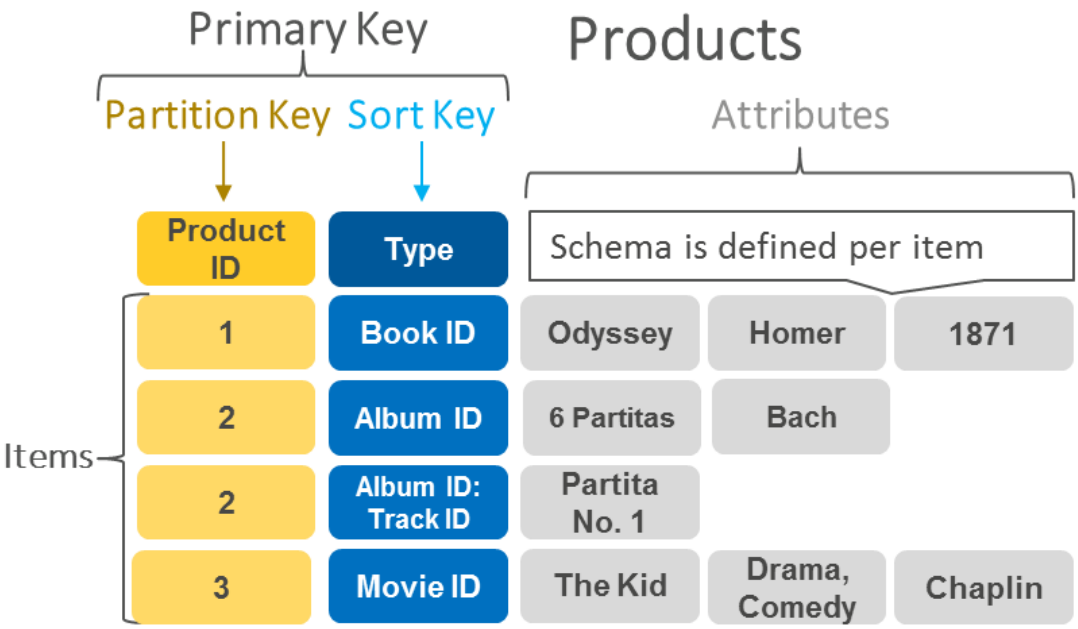


Ilustración 2: Estructura de una base de datos clave-valor

Las características esenciales de las bases de datos clave-valor son [5]:

- **Simplicidad.** Las bases de datos clave-valor utilizan una estructura de datos muy sencilla. A veces no se necesitan bases de datos con estructuras complejas y si se puede ahorrar el trabajo de definición de un esquema siempre es algo positivo. Un buen ejemplo sería implementarla para tener un registro de los alumnos de GSI con sus datos y el año que están cursando. Para la manipulación de los datos simplemente hay que establecer un nombre de colección, en nuestro ejemplo "Alumnos". Cada vez que se inserta un valor en la colección Alumnos se le asignará una clave, para cada clave puede haber varios atributos como pueden ser el nombre, email, DNI, apellidos del alumno en cuestión. Para hacer una

consulta simplemente hay que acceder al registro con clave 001 y mostrará todos los valores asociados a esa clave, también se pueden modificar valores.

- **Velocidad.** Esta es una de sus principales características, debido a su simple estructura de datos pueden ofrecer operaciones de alto rendimiento en poco tiempo.
  Una de las maneras para que la ejecución de las operaciones sea rápida es mantener los datos en la memoria, ya que leer y escribir datos en RAM es mucho más rápido que en disco. Pero al guardar los datos en RAM, y no en disco, existe el inconveniente de que la RAM no tiene un almacenamiento persistente y pueden surgir problemas que hagan que se pierdan datos, como cuando surge una pérdida de energía y se cae el servidor.
  Si una aplicación cambia el valor asociado a una clave, la BBDD de clave-valor lo que hace es actualizar la entrada de la RAM en la que se almacena ese valor. Luego envía un mensaje a la aplicación de que ese valor se ha actualizado, y la aplicación puede continuar realizando otras acciones mientras que la BBDD escribe en disco el valor actualizado en RAM para que no haya pérdida de datos.

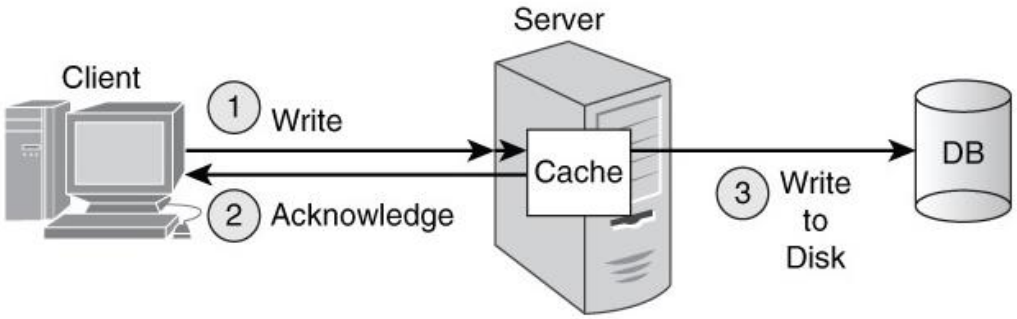


Ilustración 3: Ejemplo de escritura en una BBDD clave-valor

Para realizar operaciones de lectura también es más rápida si se almacenan los datos en memoria. El problema que puede aparecer es que la memoria posiblemente tenga un tamaño bastante menor al disco, por tanto, hay que usar un algoritmo que elija qué parte de la memoria tiene que ser liberada para que pueda ser ocupado ese espacio por otros datos. Normalmente se suele usar el algoritmo LRU (Least Recently Used), con éste los datos que llevan más tiempo sin ser utilizados serán borrados y se libera espacio para dar paso al almacenamiento de nuevos datos.

- **Escalabilidad**. Se necesita la capacidad de agregar o eliminar servidores de un grupo en función de la carga del sistema. Cuando se agrega o elimina un servidor hay que replicar datos y operaciones. Dos maneras para escalar son replicación con maestro-esclavo y la replicación sin maestro-esclavo.

3.1.1. Diseño de bases de datos clave-valor

La forma en la que se diseñan las claves es muy importante debido a que éstas deben tener algo de sentido para facilitar las operaciones en la base de datos. Para adoptar este sentido, las claves deben de seguir una nomenclatura. Para obtener una buena nomenclatura de la clave se pueden seguir una serie de reglas o consejos:

- Utilizar nombres inequívocos que deriven del nombre de la entidad o el atributo, por ejemplo 'lib' para libro o 'alum' para alumno. Para que el nombre no sea inequívoco se debe realizar bien la abreviatura, pues si no, pueden englobar muchas más entidades como es el caso de usar 'l' para libro.
- Cuando con una clave se tiene que recuperar rangos es mejor definir la clave con componentes basados también en rangos como fechas o contadores.
- Se deben añadir delimitadores en caso de que sea necesario, el que se suele utilizar comúnmente es ':' pero puede valer cualquier otro.
- Hay que intentar mantener las claves lo más cortas posibles, pero intentar cumplir a su vez con los puntos anteriores.

También hay que tener en cuenta que algunas bases de datos tienen limitaciones a la hora de implementar las claves, como pueden ser restringir el tamaño de éstas o incluso el tipo de datos que pueden usarse.

Hay que prestar especial atención a la forma en la que se agrupan los pares clave-valor para ser asignados a un nodo u otro de la base de datos descentralizada. Un método muy usado es el Hash, que usa una función matemática que se puede elegir para distribuir de manera uniforme las claves y valores en todos los nodos. Otro método utilizado es la partición por rango para almacenar los datos por orden de su clave.

El inconveniente que se produce a la hora de definir un esquema de particiones es el crecimiento masivo, ya que en el caso de que la base de datos crezca mucho y de manera rápida, se va a tener que añadir otro servidor y reestructurarlo.

En cuanto a los valores, es importante indicar también que este tipo de bases de datos son flexibles, es decir, un atributo podría almacenarse por ejemplo como Integer o como String en función de lo que se necesite en cada momento. Esto es muy útil cuando el tipo de dato puede cambiar a lo largo del tiempo [5].

3.2. Bases de datos documentales

"Una base de datos de documentos es un tipo de base de datos no relacional que está diseñada para almacenar datos semiestructurados."

Las bases de datos de documentos son intuitivas, los desarrolladores las utilizan debido a que los datos en el nivel de la aplicación generalmente se representan mejor en formato de documento que en una tabla. Cada documento puede tener la misma estructura de datos o no, cada documento es autodescriptivo, teniendo su posible esquema único, y no depende necesariamente de ningún otro documento. Los

documentos se agrupan en colecciones, que tienen un propósito similar al de una tabla en una base de datos relacional.

Este tipo de bases de datos utilizan documentos para almacenar información. Con documentos no se refiere a documentos de texto, sino archivos XML o JSON normalmente, aunque se pueden utilizar otros tipos como YAML.

```xml
<writer>
   <writername>Gabriel García Márquez</writername>
   <books>
     <book>
       <bookname>Crónicas de una muerte anunciada</bookname>
       <datereleased>1981</datereleased>
       <genre>Novela de ficción</genre>
       <numpag>144</numpag>
     </book>
     <book>
       <bookname>Cien años de soledad</bookname>
       <datereleased>1967</datereleased>
       <genre>Novela de ficción</genre>
       <numpag>496</numpag>
     </book>
   </books>
 </writer>
```

Ilustración 4: Ejemplo de un documento XML

```json
{
    "id" : 1,
    "writername": "Gabriel García Márquez",
    "books" : [
        {
            "bookname" : "Crónicas de una muerte anunciada",
            "datereleased" : 1981,
            "genre" : "Novela de ficción",
            "numpag" : 144
        },
        {
            "bookname" : "Cien años de soledad",
            "datereleased" : 1967,
            "genre" : "Novela de ficción",
            "numpag" : 496
        }
    ]
}
```

Ilustración 5: Ejemplo de documento JSON

Dentro de las bases de datos se crean colecciones y en éstas se almacenan los documentos cuando son insertados. Normalmente los documentos almacenados dentro de una colección están relacionados con la misma entidad, es decir, en la colección 'Libros' se almacenan documentos con datos sobre libros y no sobre animales, no es recomendable guardar documentos en una colección con la que no guardan relación [9].

Estas bases de datos están diseñadas para permitir que surjan variaciones en los distintos documentos que se encuentran en una colección, por lo tanto, se pueden encontrar gran cantidad de documentos con diferente estructura, es por eso por lo que este tipo de base de datos es muy flexible o carece de esquema, ya que no es necesario predefinirlo antes de comenzar a guardar datos como ocurre con otros tipos [5].

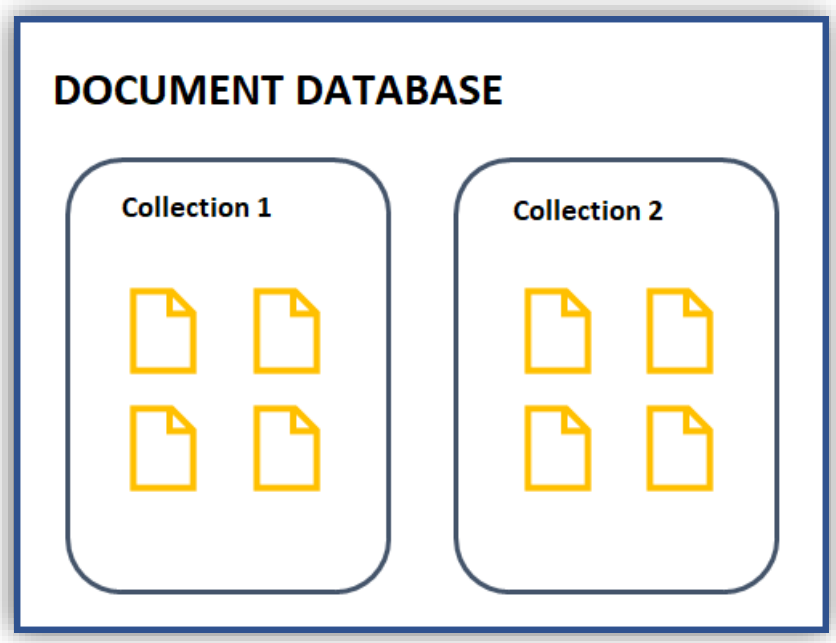


Ilustración 6: Estructura de una base de datos documental

En las bases de datos documentales los IDs de los documentos no se tienen ni por qué usar en contraposición de las de clave-valor. Aquí se pueden buscar documentos a partir de valores, por ejemplo, en las ilustraciones 4 y 5, si se buscase los libros escritos en 1981 el resultado sería 'Crónicas de una muerte anunciada'. Esto es debido al uso de índices para mejorar la eficiencia de las operaciones en grupos de documentos. Todos los datos que aparecen en un documento dentro de la base de datos son indexados, de esta manera se pueden encontrar rápidamente grupos de documentos que tienen una misma propiedad [2].

Las operaciones básicas son iguales que en el resto de los tipos:

- Insertar documentos en una colección
- Eliminar documentos de una colección
- Modificar datos de los documentos de una colección
- Realizar búsquedas para mostrar datos de documentos que se encuentran en una colección.

3.2.1. Diseño de una base de datos documental

Como ya se ha comentado anteriormente las bases de datos documentales son flexibles. Para comenzar con el modelado de la base de datos hay que tener un conocimiento

sobre la normalización y la desnormalización. Estos conceptos en este tipo de bases de datos se interpretan de una manera diferente a cuando se habla de bases de datos relacionales.

Los documentos estarán normalizados si tienen referencia a otros documentos en los que se pueden buscar información adicional. Por ejemplo, se tiene una colección que almacena personas y otra que almacena expedientes académicos, para no repetir en cada registro de una persona su expediente lo que se hace es referenciar al expediente almacenado en la otra colección indicando su identificador único de expediente. Por lo tanto, cuando se usan varias colecciones para almacenar datos relacionados se considera que estamos normalizando.

La normalización por lo tanto evita redundancia en los datos, pero produce problemas de rendimiento. Para mejorar el rendimiento se puede aplicar la desnormalización, que es todo lo contrario a la normalización, en la que se utilizará la redundancia de datos. Si los datos no están normalizados se recuperan solo de un documento, esto es más rápido que recuperarlos de varias colecciones, sobre todo si se utilizan índices.

Estas bases de datos ofrecen diversas opciones para recuperar los datos, como mostrar todos los documentos posteriores a una fecha, los que son de un tipo específico o los que tienen un atributo determinado, etc.

Para poder realizarse las búsquedas, se utiliza un procesador de consultas que coge las consultas de entrada, los datos de los documentos y de las colecciones, y genera una serie de operaciones que recuperan los datos solicitados.

Este procesador de consultas puede tomar decisiones propias con el fin de optimizar el rendimiento de la consulta.

Hay que tener en cuenta también que los documentos son mutables, por tanto, cuando se crea un documento el sistema de administración de base de datos le asigna un espacio. El tamaño de este espacio va a estar compuesto del tamaño del documento más el otro tamaño que se asigna por si el documento es modificado y crece. Si el documento crece más del tamaño que se ha reservado, el sistema tendrá que moverlo en otra ubicación y esto producirá una perdida en el rendimiento, pues se tendrá que leer el documento, copiarlo en otra ubicación y liberar el espacio que ocupaba anteriormente.

Para evitar esta pérdida de rendimiento hay que asignarle un espacio suficiente para su crecimiento y, de esta manera, no tener que cambiar la ubicación del documento.

Cuando se diseña una base de datos de estas características es importante elegir un correcto número de índices en función de las necesidades. En el caso de que se tenga una aplicación en la que la mayoría de las operaciones son de lectura, habrá que añadir un mayor número de índices e indexar la mayoría de los atributos de los documentos, pues lo más probable es que se usen varios campos para filtrar búsquedas. Por otro lado,

si en nuestra base de datos la gran mayoría de operaciones son de escritura se implantarán un menor número de índices, ya que éstos también consumen recursos. Dicho esto, lo correcto es buscar un equilibrio en el número de índices a implementar en la base de datos según la funcionalidad que se le va a dar [5].

3.3. Bases de datos de grafos

Como introducción es bueno saber y entender lo que es un grafo. Un grafo es un conjunto de objetos llamados nodos y que están unidos entre ellos mediante arcos o aristas, de esta manera se consigue representar relaciones entre estos nodos.

Por tanto, este tipo de base de datos adquiere ese nombre porque la información que almacena se representa como nodos de un grafo y las relaciones como aristas, de esta forma se puede utilizar la teoría de grafos para recorrerla. Cada nodo tiene diversas propiedades.

El mayor inconveniente es que tiene que estar completamente normalizada pero la navegación entre nodos es muy eficiente [8].

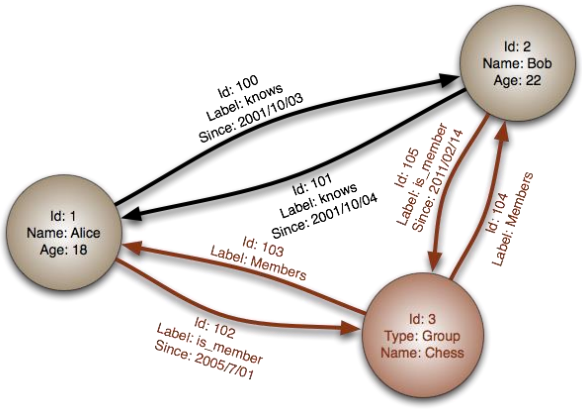


Ilustración 7: Estructura de una base de datos de grafos

Son muy útiles cuando se tienen elementos relacionados entre sí de manera compleja y estas relaciones tienen propiedades al igual que los nodos. Son capaces de realizar desde consultas muy simples, como consultar propiedades de un nodo, a consultas muy complejas de las realciones entre nodos para encontrar patrones. Se les atribuyen un identificador único a los nodos y a las relaciones.

El inconveniente que tienen es que son difíciles de escalar a múltiples servidores, sí se pueden replicar en diversos servidores para mejorar las operaciones de lectura, pero las escrituras en múltiples servidores son difíciles de llevar a cabo debido a la relaciones entre nodos.

Las consultas que se realizan son más rápidas que en las bases de datos relacionales, esto se debe a que en lugar de realizar uniones, sigue los arcos de nodo a nodo, esto es una operación mucho más sencilla y menos costosa. Su modelado es sencillo, a pesar de que haya que normalizar. Cabe destacar que se pueden modelar múltiples relaciones entre dos nodos y que estas relaciones también tienen propiedades [5].

3.3.1. Diseño de una base de datos de grafos

A la hora de diseñar estas bases de datos es importante saber que las entidades pueden ser prácticamente cualquier cosa, no solo estas bases de datos son las únicas que son adecuadas para implementar entidades.

Lo que hace elegir este tipo de bases de datos son las relaciones entre entidades y toda la información que se puede extraer de éstas mediante consultas específicas, estas bases de datos aportan una flexibilidad a la hora de realizar consultas, es lo que hace que se priorice su uso frente a otras, si no se va a hacer uso de esta flexibilidad que aportan y solo se va a obtener información de las entidades, y no de las relaciones, no tiene mucho sentido su uso.

Una de las cosas más importantes de estas bases de datos es que las relaciones entre entidades no tienen por qué ser del mismo tipo.

Los pasos principales para un diseño de base de datos de grafo son los siguientes:

- Identificar las consultas que se quieren realizar en la base de datos.
- Identificar las entidades, con sus propiedades.
- Identificar las relaciones entre entidades, con sus propiedades.
- Asignar las consultas específicas del primer paso a consultas más específicas orientadas a grafos.

En estas bases de datos se pueden utilizar índices para los nodos y, de esta manera, mejorar el tiempo de recuperación de la información.

En cuanto a los arcos, estos pueden ser dirigidos o no dirigidos según la relación que queremos representar con éstos. Con los arcos dirigidos la relación que se representa no es simétrica, por ejemplo, en una red social cuando un usuario cualquiera sigue a un famoso normalmente éste no seguirá a un usuario cualquiera, por tanto, la relación entre ambos no es simétrica. En contraposición, cuando las relaciones son simétricas tendremos que utilizar los arcos no dirigidos, un ejemplo de este tipo de relaciones es la distancia entre ciudades.

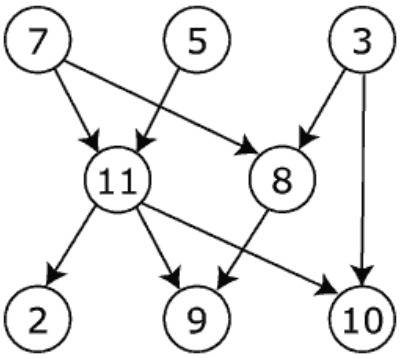


Ilustración 9: Ejemplo de grafo con arcos dirigidos

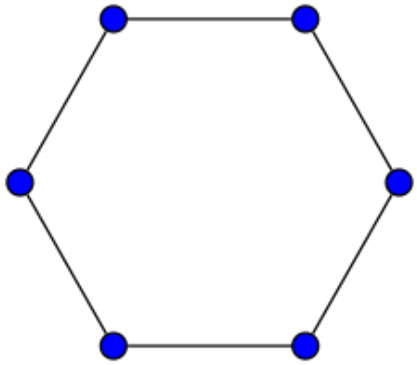


Ilustración 8: Ejemplo de grafo con arcos no dirigidos

Al recorrer los grafos hay que estar atentos en no caer en un ciclo, ya que se puede consumir muchos recursos [5].

3.4. Bases de datos columnares

Usan el almacenamiento basado en columnas para reducir el coste de operaciones E/S del disco. Son utilizadas en tipos de aplicaciones analíticas, ya que permiten guardar un gran volumen de datos a gran velocidad y facilidad de recuperar estos datos.

Estas bases de datos tienen un cierto parecido a las bases de datos relacionales, de hecho parte de sus características son compartidas por ambas. Un ejemplo es que en ambas se usan claves primarias, conocidas como claves de fila en el caso de que se esté usando una base de datos columnar. En ambas bases de datos estas claves son indexadas para una recuperación rápida de los registros.

Hay otras características que parecen idénticas, pero la forma de implementarse es diferente. En este caso se encuentra el almacenamiento en columnas. Aunque se esté viendo los datos en columnas, realmente los datos se almacenan en mapas de mapas que apuntan a los valores de las columnas.

Las características que son diferenciales de una base de datos y otra, son que en las columnares las relaciones no existen y no se pueden realizar transacciones de múltiples filas.

3.4.1. Diseño de una base de datos columnar

Antes de comenzar con el modelado de la base de datos hay que tener claro una serie de conceptos importantes:

- No existe la normalización en estas bases de datos.
- No existen las relaciones.
- No se utilizan uniones.
- Se pueden hacer uso de columnas sin valor.
- Para almacenar los datos se usan el nombre o el valor de la columna.

Es muy importante entender que las columnas pueden variar entre filas, es decir, puede haber columnas que tienen un valor para un registro mientras que para otro tiene otro valor o incluso puede ser nula y no existir.

Hay casos en los que se puede utilizar el nombre de la columna para almacenar también valores.

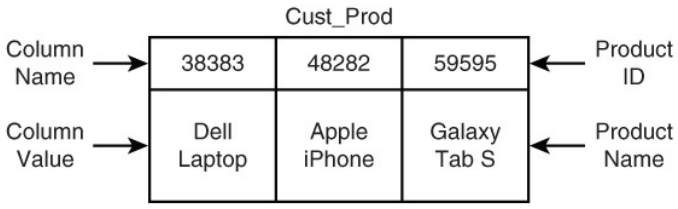


Ilustración 10:Ejemplo de tabla de una BBDD columnar

Los pasos para diseñar un modelo para una base de datos siguiendo estos consejos son:

- Definir las entidades en nuestra solución.
- Definir la tabla en función de estas entidades. Hay que tener en cuenta que una entidad puede ser una fila de la tabla o una columna de la tabla, pero nunca la propia tabla.
- Para definir las columnas se tendrá que analizar cómo es mejor almacenar los datos, si por el nombre de columna o por el valor de columnas. Ambas opciones son válidas, pero para algunos casos es mejor utilizar un procedimiento y para otros es mejor utilizar el otro.

Tras a haber definido todas las tablas del modelo ya se podrá comenzar con la utilización de la base de datos.

Estas bases de datos permiten modificar el modelo posteriormente de haberlo definido [5].

4. MongoDB VS Apache Cassandra

En este apartado se va a realizar una comparación exhaustiva de aspectos técnicos. Antes de comenzar con el análisis, es bueno recordar que MongoDB es una base de datos documental, pero también hay que definir de qué tipo de base de datos es Cassandra. Apache Cassandra hay opiniones que la clasifican como una base de datos clave-valor y otras que la clasifican como una base de datos columnar.

Lo cierto es que Cassandra es un híbrido entre estas bases de datos, ya que la rapidez de lectura/escritura y la distribución del clúster en nodos se basan en una base de datos clave-valor, pero tiene un almacenamiento de los datos basado en grupos de columnas.

4.1. Arquitectura

Es importante puntualizar un aspecto visto anteriormente, en el teorema de CAP. Cassandra proporciona tolerancia a fallos y disponibilidad, todo ello a cambio de ser consistente de manera eventual, en contraposición, MongoDB lo que sacrifica es la disponibilidad.

A pesar de que ambas sean tolerantes a las particiones, su arquitectura es bastante distinta.

Cassandra presenta una arquitectura Peer to Peer, es decir, todos los nodos tienen la misma importancia, todos los nodos serán tratados como nodos primarios y tendrán la misma información que el resto. La base de datos de manera interna será la que decida qué nodo va a tener el rol coordinador en la ejecución de una consulta.

Los nodos del clúster formarán un anillo, los datos se repartirán a un nodo u a otro en base a un token único definido por una función hash. Posteriormente, para que los nodos tengan la misma información, se replicará los datos entre ellos, esta replicación se hará en base a la política y cantidad de nodos que se tenga [10].

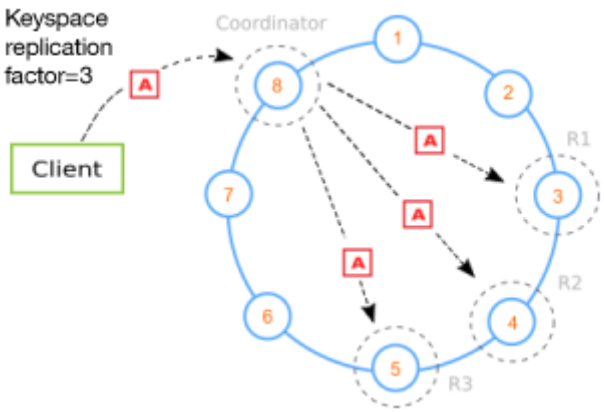


Ilustración 11: Arquitectura de Cassandra

Como se puede ver en la imagen, la mitad de los nodos no tendrán la última actualización de los datos, por esa misma razón es eventualmente consistente [11].

La arquitectura de MongoDB se basa en la replicación y en la fragmentación del clúster.

El primer paso para entender la arquitectura de MongoDB es entender la replicación, y cómo funciona ante caídas de los nodos.

Un conjunto de réplicas es un grupo de instancias o nodos que contienen el mismo conjunto de datos, opcionalmente puede haber algún nodo que ejerza la función de árbitro. Dentro de este conjunto solo un nodo va a ser el primario, y los restantes serán secundarios. Este nodo primario es el que recibe las operaciones de escrituras y el que las realiza, tras esto registra todos los cambios realizados en su conjunto de datos en un registro de operación llamado 'oplog'. Los secundarios proceden a replicar el 'oplog' y ejecutar todas estas operaciones sobre sus conjuntos de datos, de esta manera se tendrá siempre el mismo conjunto de datos en todos los nodos.

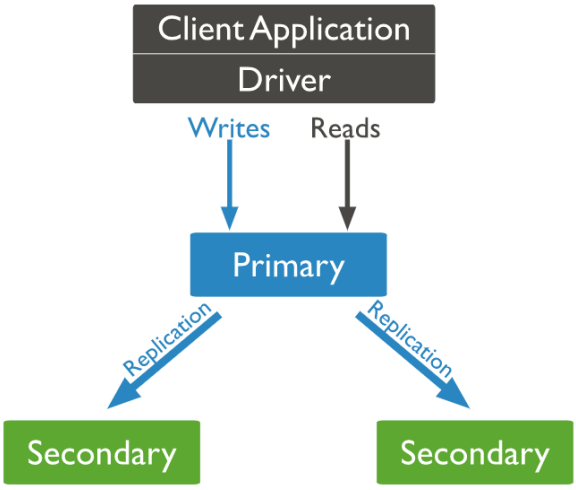


Ilustración 12: Conjunto de réplicas en MongoDB

Si el nodo primario se cae y no está disponible, será el resto de los nodos secundarios quienes elijan el que actuará, a partir de ese momento como nodo primario, a través de

una votación. Como ya se ha comentado antes, puede haber un nodo árbitro que cumpla esta función, pero ese nodo no almacenará una copia del conjunto de datos. Por ejemplo, si se tiene un número par de nodos se puede introducir un nodo árbitro para que siempre se cumpla una decisión en caso de que haya un empate, y de esta manera intentar tener el máximo de disponibilidad posible aunque no sea el 100%.

Cabe destacar que un nodo primario puede renunciar a serlo y convertirse en secundario, y un secundario puede ser elegido primario, pero un nodo árbitro siempre actuará como tal.

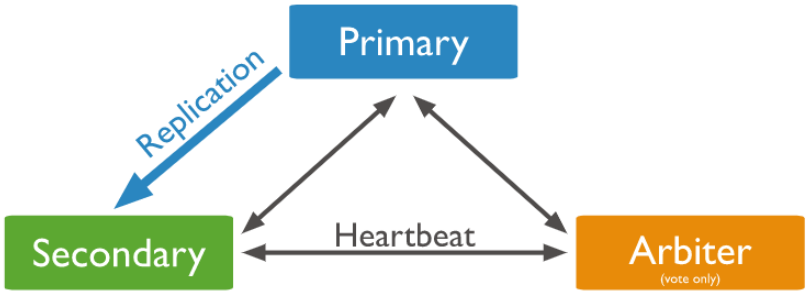


Ilustración 13: Ejemplo de uso de un nodo árbitro

El funcionamiento del conjunto de réplicas es la causa por la cual MongoDB es una base de datos que descuida la disponibilidad, pero es consistente.

Ahora bien, esta base de datos para ser tolerante a particiones y permitir un escalado horizontal de esta necesita fragmentar el clúster.

Esta fragmentación o Sharding consiste en compartir los datos de la colección distribuyéndose a través del clúster.

Un clúster fragmentado tiene tres componentes:

- Shard: Es un subconjunto del total de los datos que se han fragmentado. Es conveniente implementarlos como un conjunto de réplicas para siempre tener esta consistencia.
- Mongos: Enruta consultas a los Shards, sirve de interfaz entre el clúster y el cliente.
- Config Servers: Almacena metadatos y ajustes de configuración del clúster, también implementado como un conjunto de réplicas.

El resultado de un clúster de MongoDB fragmentado es el siguiente:

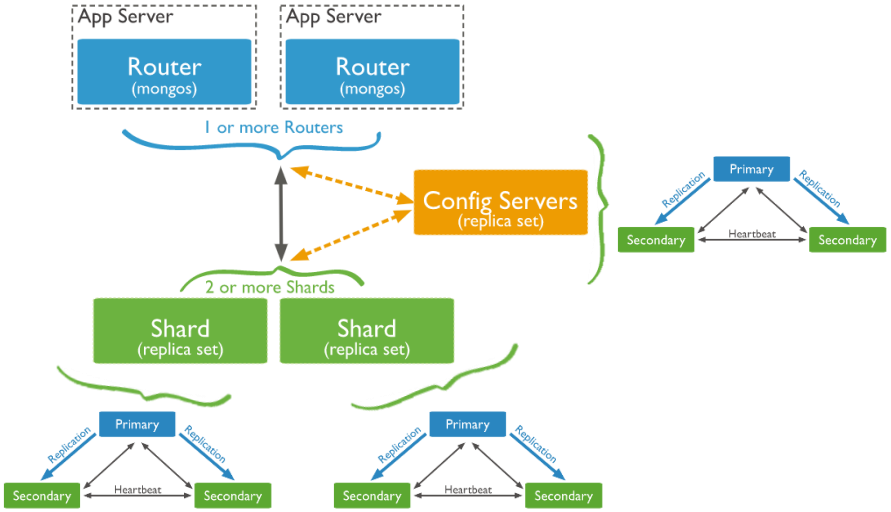


Ilustración 14: Clúster de MongoDB fragmentado

Para distribuir los documentos en un Shard se utiliza la clave de fragmentación, que es un campo indexado o campos compuestos indexados que se encuentran en cada documento de la colección. Primero se escoge una clave de fragmentación y, posteriormente, se particiona en función de los rangos de valores de esta clave de fragmentación. Hay que tener cuidado al elegir estos rangos de valores porque puede afectar directamente sobre el rendimiento, la eficiencia y la futura escalabilidad. Una vez elegida la clave de fragmentación y sus rangos de valores estos son inmutables, es decir, es imposible modificar la clave de fragmento en esa colección ni tampoco los valores de los campos seleccionados de esta clave [12].

Como se puede ver en estas bases de datos, y siguiendo el teorema de CAP, aparte de estar clasificadas en diferentes grupos ya que Cassandra prioriza la disponibilidad y MongoDB prioriza la consistencia, la tolerancia a particiones se realiza de manera distinta provocando que sus arquitecturas sean totalmente distintas.

4.2. Modelado de los datos

El modelado de los datos de ambos sistemas difiere el uno con el otro en bastantes cosas. Aparte de que estructuralmente tienen distinta forma de almacenar los datos, cada base de datos tiene unas utilidades extras que no tiene la otra.

Para definir un modelo en Cassandra hace falta conocer los distintos apartados en los que se sustenta [13]:

- **Clúster:** Es una colección de nodos y están organizados en forma de anillo y puede escalar horizontalmente, de manera transparente para el cliente. La arquitectura del clúster se ha podido observar en el punto anterior.
- **Keyspace:** Es el esquema y el contenedor más externo de una base de datos en Cassandra. Sus atributos son el factor de replicación (número de nodos sobre los que se realizará cada replicación) , estrategia de ubicación de réplica (si se dispondrá de solo un anillo, o de un anillo de anillos) y la familia de columnas.

- **Familia de columnas:** Equivale a tablas dentro de un modelo relacional. Contiene una colección de filas y cada fila tiene una clave para dar acceso al resto de datos de la fila.
- **Columna:** Estructura que tiene un nombre, un valor y una marca de tiempo. Estas pueden variar en cada fila ya que los datos no tienen una estructura definida.

Los pasos para modelar una base de datos son los siguiente:

- Establecer un espacio de claves con su estrategia y factor de replicación.
- Definir las entidades que se quieren almacenar en la base de datos.
- Crear las familias de columnas en función de esas entidades.

Aparte de almacenar tipos de datos nativos, también se pueden almacenar colecciones, conjuntos, listas y mapas.

Otro aspecto muy positivo son los **UDT,** un tipo de datos definidos por nosotros, en el que se puede indicar de qué campos se desea que se componga y solo ocuparía una columna de la familia.

Los registros o filas no tienen por qué tener la misma estructura unos de otros aunque pertenezcan la misma familia, ya que se pueden dejar valores de columnas vacíos para un registro y no ocuparían espacio, esto genera que un registro pueda tener algunas columnas con valores rellenos que el siguiente registro no tiene por qué tener [14].

En Cassandra lo primordial es definir el modelo antes que construir una solución conectada a ella, en nuestro caso es una aplicación. A pesar de definir el modelo antes, este se puede modificar a diferencia de otras bases de datos. Para poder modelar la base de datos hay que tener muy claro que, aunque tenga cierta similitud con una base de datos relacional no lo es, ya que no existen las relaciones en esta. Un error muy habitual en el modelado de datos de Cassandra es simular relaciones a nivel de aplicación.

En Cassandra se tienen las denominadas claves primarias, estas claves pueden estar presentes e incluso formar parte de otra clave primaria en otra familia de columna. Estrictamente hablando no existe una relación entre ambas familias de columnas, pero a nivel de aplicación gracias al modelo se puede definir esta relación.

Para entenderlo mejor se va a ejemplificar este proceso:

Suponer que se tiene una familia de columnas con un usuario cuya clave primaria es su DNI, y una familia de columnas llamada reservas que tiene una columna denominada DNI, que almacenará el DNI del usuario que ha realizado una reserva. Si se dedea buscar las reservas que ha realizado un usuario determinado, simplemente hay que coger el DNI de ese usuario y buscar todos los registros de reservas que tienen ese DNI asignado.

Cuando se tiene que buscar un registro en una familia de columnas con Cassandra, debería buscarse a partir de su clave primaría, es decir la finalidad de esta base de datos es la devolución rápida de registros accediendo a ellos a partir de su clave.

Siguiendo el ejemplo anterior, si se busca en la familia de columnas de reservas por el DNI del usuario reservado, se está incumpliendo este aspecto comentado y se estaría simulando una base de datos relacional.

También es incorrecto tratar de solucionar este error definiendo que ese campo que relaciona una familia con otra sea clave primaria o parte de una clave primaria en ambas familias.

El modelo resultante del ejemplo sería el siguiente:

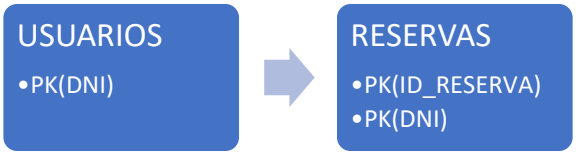


Ilustración 15: Modelo relacional en Cassandra

En este caso siempre se buscaría por la clave y no se estaría incumpliendo ninguna regla de una base de datos clave-valor, además con la clave compuesta un mismo usuario podrá tener varias reservas distintas y consultar los datos a través de una clave primaria o parte de ella.

La razón del porqué sigue siendo incorrecto este modelo es la siguiente: si se quiere introducir relaciones en la base de datos de Cassandra directamente es mejor que no se utilices y elegir para la aplicación una base de datos relacional pensada para ello, se puede definir el modelo desde un principio usando un modelo entidad relación sin tener que definir las relaciones a nivel de aplicación.

Por otra parte MongoDB se compone de los siguientes apartados de mayor a menor nivel:

- **Clúster:** Es algo más que una colección de nodos, su estructura se ha visto en el punto anterior.
- **Colección:** Es el contenedor de los documentos con sus respectivos datos.
- **Documento:** Un documento es un conjunto de campos con sus respectivos valores.

No es necesario modelar un esquema previamente, se puede ir definiendo a nivel de aplicación, esto es gracias a la flexibilidad que tiene. Una colección en MongoDB no necesita ni que sus documentos tengan el mismo esquema, es decir, los documentos de esta colección no necesitan tener el mismo conjunto de campos o, incluso, puede variar el tipo de dato que se almacena en un mismo campo. Aparte se pueden modificar documentos independientemente del esquema que tenga.

Estas propiedades que tienen los documentos hacen que sea muy simple almacenar entidades. Esta base de datos es muy útil para almacenar datos de aplicaciones orientadas a objetos.

El punto clave en la implementación de esta base de datos, es la definición de la estructura que debe tener los documentos o parte de ellos, porque como se ha comentado son independientes unos de otros.

Si la estructura del documento es simple no hace falta incidir en este tema, pero cuando se quiere relacionar dos documentos es cuando hay que definir un modo de los dos que proporciona MongoDB [12].

- **Datos embebidos:** Consiste en introducir la estructura de un documento dentro de otro documento, es decir, permite guardar datos relacionados en el mismo registro. A este método se le llama método desnormalizado y gracias a él se pueden manipular los datos realizando solo una operación.

Ilustración 16:Documento con datos embebidos en MongoDB

- **Referencias:** A este modelo se le denomina normalizado. Consiste en introducir en un documento enlaces que apuntan a otro documento para acceder a esos datos. Este modelo se puede implementar usando referencias manuales, en las que se guarda en un campo de mi documento el **_id** del documento con el que tiene relación, y para consultar estos datos simplemente hay que ejecutar una segunda consulta.

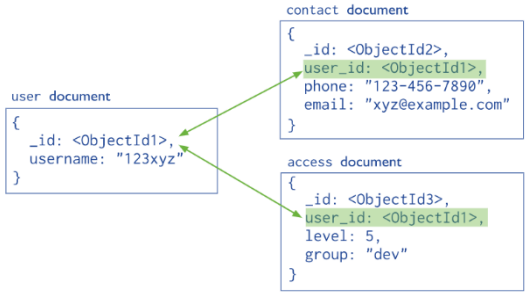


Ilustración 17: Referencias entre documentos en MongoDB

Otra opción es más formal y trata de usar DBRefs. Se deben de utilizar en casos muy específicos, como cuando se tiene un documento que se relaciona con

documentos de diversas colecciones. La notación consiste en indicar el **_id** del documento en cuestión y la colección a la que pertenece, opcionalmente se puede añadir la base de datos a la que pertenece también [12].

Las conclusiones que se pueden obtener son que en MongoDB el modelado no se prioriza, se realiza a la vez que se definen las entidades, que es muy flexible y que permite relaciones entre sus datos. Por otro lado, para Cassandra hay que definir un esquema previo, y no hay que caer en el error de que se puedan simular relaciones, porque se perdería la esencia de Cassandra e incluso, en algunos casos, se podría llegar a reducir su rendimiento o convertirla en una base de datos relacional. También hay que destacar que Cassandra permite cambiar el modelo previamente definido.

En definitiva, en MongoDB el modelo se ajusta a los datos y en Cassandra los datos se ajustan al modelo.

4.3. Gestión de archivos

En Cassandra almacenar un archivo es muy costoso debido a que tiene que fragmentarlo en trozos de muy pocos bytes, pues no permite el formato de almacenamiento BLOB (Binary Large Objects). A pesar de que se permita, hay que realizar un tratamiento del archivo a nivel de aplicación y se reduciría considerablemente el rendimiento, por lo que es mejor utilizar un sistema de ficheros tradicional y recuperar los ficheros a nivel de aplicación.

En cambio, con MongoDB se tiene una utilidad llamada GridFS, esta funcionalidad permite almacenar y recuperar archivos que superan los 16MB.

Su funcionamiento consiste en particionar el archivo en fragmentos de 255 KB a excepción del último fragmento, que será tan grande como se necesite, y guardar cada fragmento como un documento individual en una colección. Los archivos que no tienen un gran tamaño se almacenarán como un único fragmento en un documento.

Para recuperar el archivo almacenado en la base de datos, el controlador vuelve a unir todos los fragmentos.

GridFS crea en nuestra base de datos de manera automática dos colecciones [12]:

- **Chunks:** Cada documento de esta colección es una parte distinta de un archivo. Los documentos de esta colección se componen de la siguiente estructura:

```
_id: ObjectId("5cf8e6fe5e0cac5b248b456b")
files_id: ObjectId("5cf8e6fe5e0cac5b248b456a")
n: 0
data: Binary('/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAUDBAQEAwUEBAQFBQUGBwwIBwcHBw8LCwkMEQ8SEhEPERETFhwXExQaFRERGCEYGh0d...')
```

Ilustración 18: Ejemplo de chunk en GridFS

El **ID** es el identificador único del documento en el que se encuentra almacenado el fragmento, el **files_id** es el ID del documento que está almacenado en la colección fs.files, la **n** es el número secuencial del fragmento, en este ejemplo es

el primer fragmento en el que se ha dividido el archivo, **data** es el contenido de archivo que almacena ese fragmento.

- **Files:** Almacena en un documento los metadatos del archivo. Se insertará en esta colección un documento por archivo.

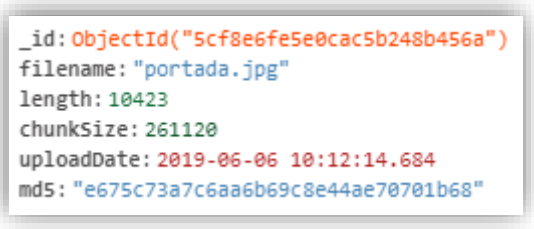


Ilustración 19: Ejemplo de file en GridFS

Se puede ver que se almacenan campos como el nombre del archivo o la fecha en la que se ha creado ese documento.

4.4. Indexación de los datos

Como ya se ha comentado anteriormente, Cassandra es una base de datos clave-valor, aun así, no implica que permita buscar resultados a partir de campo distinto a su clave debido a su condición de base de datos columnar, para ello usa los índices. La finalidad es la realización de búsquedas rápidas y eficientes de registros que cumplen una condición dada. Estos índices no se pueden implementar sobre columnas de tipo contador.

Se pueden encontrar los índices primarios que se crean por defecto, son únicos por cada fila, se utilizan para asignar cada registro de una familia de columnas a su nodo correspondiente. Como cada nodo conoce qué rango de valores del índice primario almacena, las consultas pueden realizarse de forma eficiente escaneando únicamente los índices de las réplicas de las filas buscadas.

Por otro lado, se tiene los índices secundarios que se crean sobre una columna tras definir la familia de columnas. Se utilizan para consultar un registro utilizando una columna que no es consultable. Este índice se crea como una tabla oculta y se almacena en cada nodo. Por esto hay que tener un especial cuidado, ya que si se realiza una consulta que acceda a varios nodos se puede tener problemas muy serios de rendimiento [14].

Las situaciones en las que no conviene usar índices son:

- Para consultas que haya que comprobar un gran número de valores y devuelva pocos resultados.
- En columnas que sean frecuentemente actualizados o eliminados sus valores.
- Para buscar una fila en una partición grande.

En MongoDB los índices son importantes, sin ellos al realizar una consulta se realiza un escaneo completo de la colección, documento a documento, con un índice adecuado para una consulta el número de documentos que se deben de inspeccionar es menor.

El índice almacena el valor de manera ordenada de un campo específico o conjunto de campos. Se puede realizar consultas de igualdad o basadas en rango, y proporciona la opción de devolver los valores ordenados.

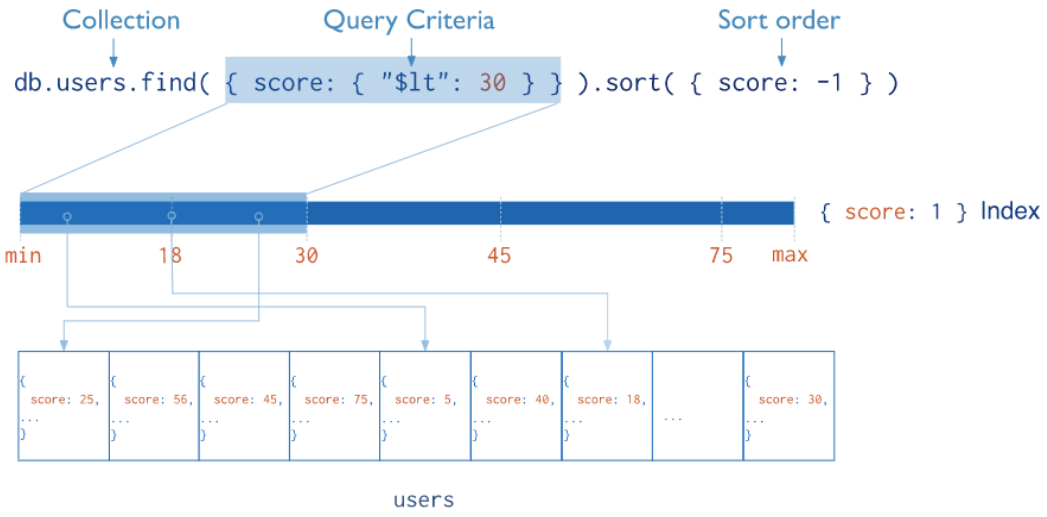


Ilustración 20: Ejemplo de índice en MongoDB

Es trabajo del administrador de base de datos construir índices en los campos que sean frecuentemente accedidos. Por defecto MongoDB crea un índice sobre el campo _id, se utiliza para asegurar de que no se pueden insertar dos documentos distintos con el mismo _id. Este índice no se puede eliminar.

Las consideraciones que hay que tener cuando se crean índices son las siguientes:

- Cada índice necesita al menos 8 KB de espacio en disco.
- Cuando está activo, cada índice consume espacio en disco y memoria.
- Tener índices supone un impacto negativo en operaciones de escritura. En colecciones sobre las que se realizan muchas escrituras un índice es caro porque hay que actualizarlo con cada operación.
- Los índices no afectan a consultas sobre campos no indexados.

Hay varios tipos de índices en función de los tipos de datos y consultas que admiten:

- **Campo único:** Es igual que el índice que viene predefinido sobre el campo _id, pero se puede escoger el campo que se quiera y definir si se quiere que sea ascendente o descendente.
- **Compuesto:** Son índices sobre varios campos a la vez. Establecer el orden en estos índices es importante. Al ser de múltiples campos primero hay que indicar por cuál campo se ordenan y el orden que se desea que siga, y posteriormente,

en caso de que haya varios valores iguales hay que decidir el orden que deben seguir en función de otro de los campos.

- **Multiclave:** Sirven para indexar datos almacenados en un array.
- **Geoespacial:** Se crea para admitir consultas para datos de coordenadas. Pueden ser de tipo d2, que usan geometría plana, o d2sphere, que usan una geometría esférica para devolver resultados.
- **Índice de texto:** Sirve para admitir consultas de búsqueda de texto dentro de una cadena de caracteres. Es insensible a las mayúsculas y a caracteres con signos diacríticos, como puede ser una tilde. Muy utilizado como sustitutivo del comando LIKE en SQL.
- **Índice de hash:** Indexa el hash del valor de un campo. Solo admite coincidencias de igualdad.
- **Índice parcial:** Solo indexan documentos de una colección que cumplen una expresión de filtro que se especifica.
- **Índice TTL:** Se usan para borrar información determinada después de un período específico de tiempo. Muy útil para datos de eventos, logs e información sobre sesiones.

Como se ver en MongoDB hay gran cantidad de índices, y es importante definirlos a partir de una cantidad de datos media en nuestra base de datos.

Se puede ver el rendimiento de las consultas y verificar que están usando el índice creado con el comando **explain()**.

Como se ha podido comprobar en MongoDB, el uso de índices es bastante útil, sin embargo, se nota que Cassandra es una base de datos clave-valor, porque está preparada para encontrar registros a partir de su clave primaria y los índices es solo un plus para alcanzar a realizar algún tipo más de búsqueda [12].

4.5. Operaciones CRUD

Ambas bases de datos pueden realizar este tipo de operaciones, la única diferencia es que cada una utiliza su propio lenguaje. A pesar de que el lenguaje que usa MongoDB es muy simple, quizás se tenga algo más afianzado el lenguaje que usa Cassandra que es el CQL, esto se debe a que es muy similar al SQL.

```
db.usuarios.find({
    nombre_user: "alejandro.diazm"});

SELECT * FROM usuarios WHERE nombre_user='alejandro.diazm';
```

Ilustración 21: Ejemplo de operación de lectura en MongoDB y Cassandra

Para MongoDB hay que indicar el nombre de la colección sobre la que se quiere realizar la búsqueda, en este ejemplo se ha utilizado una llamada usuarios.

4.6. Transacciones

Cassandra ofrece transacciones atómicas, aisladas y duraderas. La consistencia que ofrece es eventual pero no es completa. Son atómicas porque las inserciones, actualizaciones o la eliminación de dos o más filas en la misma partición se consideran como una sola operación de escritura, aparte permite realizar operaciones por lotes, en las que si no se cumple cualquiera dentro del lote no se cumple ninguna de las restantes.

Las operaciones están aisladas a nivel de fila o registro, y en caso de realizar una operación en una sola partición, solo es visible para el cliente que realiza la operación hasta que se complete. Cuando se ejecuta un lote con varias operaciones sobre la misma partición éstas son aisladas, sin embargo, si se realizan sobre varias particiones no se produce este aislamiento.

Por último, todas las escrituras en un nodo se registran tanto en la memoria, como en un registro de confirmación (commit log) en el disco antes de que se cumplan exitosamente. Cuando se produce un fallo del nodo y antes de que se vacíen las tablas de memoria, el registro de confirmación se reproduce al reiniciar el nodo para recuperar cualquier operación perdida.

Existen un tipo de transacciones en Cassandra llamadas transacciones ligeras, consisten en añadir a la operación la cláusula “IF NOT EXIST”, de esta manera si se va a insertar un registro, primero comprueba si existe y en el caso de que se cumpla y existe la operación, no se realiza, por el contrario, si no existe se la operación se lleva a cabo [14].

Por otro lado, en MongoDB una operación de un solo documento es atómica, ya que existen métodos como los documentos incrustados para capturar las relaciones entre éstos, haciendo que tengan una estructura única.

Pero antes se ha visto que si se opta por el modelo normalizado, se tienen varios documentos relacionados. Para resolver esto, MongoDB ha desarrollado en sus últimas versiones la capacidad de realizar transacciones de varios documentos e, incluso, contra varios conjuntos de réplicas. Estas transacciones se fundamentan en el principio de “todo o nada”, cuando se comienza una transacción se guardan todos los cambios de datos realizados, al fallar alguna de las operaciones toda la transacción se cancela y todos los cambios de datos realizados se descartan, durante este proceso los cambios no son visibles cumpliendo el aislamiento. Hasta que no se ha producido la confirmación no se pueden ver los cambios en los datos.

El costo de rendimiento de las transacciones que afectan a múltiples documentos es mayor que las transacciones que afectan a un solo documento. Debido a esto hay que modelar los datos de la forma más apropiada posible.

Las transacciones hay que configurarlas a nivel de aplicación, están asociadas a una sesión, es decir, se inicia una transacción para una sesión. Solo se puede tener como máximo una transacción por sesión. En caso de que se finalice una sesión y esta tenga una transacción ejecutándose, se cancela automáticamente.

MongoDB a diferencia de Cassandra proporciona tres métodos para la realización de transacciones:

- Session.startTransaction()
- Session.commitTransaction()
- Session.abortTransaction()

El proceso que se sigue es el siguiente:

- Se ejecuta “session.startTransaction()”.
- Se definen las operaciones de las que va a constar la transacción. Utilizar un bloque “try/catch” para captar excepciones.
- En caso de capturar un error registrarlo en nuestro log y ejecutar “session.abortTransaction()”.
- Por último fuera del bloque “try/catch” ejecutar session.commitTransaction() y cerrar la sesión.

Una vez se confirma una transacción, todos los cambios realizados en esta se vuelven visibles, en caso de que se cancele la transacción no se llevarán a cabo estos cambios y, por lo tanto, no serán visibles.

Es muy importante a nivel de aplicación también tomar medidas para manejar posibles errores durante la realización de transacciones, e incorporar alternativas como reintentar la transacción.

Las operaciones de escritura individuales no se pueden reintentar, simplemente devolverá el error por el que se aborta la transacción.

Las confirmaciones de transacciones sí se pueden reintentar, cuando se encuentra un error en la confirmación se reintenta una sola vez, y si vuelve a fallar se aborta y no se completa la confirmación [12].

En conclusión, la posibilidad de realización de transacciones que ofrece MongoDB es mucho mayor y más completa que Cassandra, ofrece una serie de métodos que suenan y son muy habituales en las bases de datos relacionales. Por otro lado, el contenido de Cassandra para realizar transacciones es muy pobre y no ofrece ningún método para realizar confirmaciones o abortar transacciones.

5. BIBLIOTEQ

5.1. ¿Qué es?

BIBLIOTEQ es una aplicación enfocada en gestionar la demanda de publicaciones que tienen alumnos en las universidades. Nuestra idea principal es hacer de la manera más cómoda posible que todos los alumnos tengan acceso a los libros, revistas o proyectos fin de grado, con el objetivo de que puedan aprovecharse de todo el material que la universidad tiene disponible para ellos.

Proporciona una gestión de las reservas a distancia sin tener que pasar por la facultad y ver si disponen de la publicación concreta, desde BIBLIOTEQ se puede ver si quedan ejemplares disponibles para su reserva, a parte, en caso de que estuvieran todos los ejemplares reservados, si la universidad tiene esa publicación en formato digital se encarga de proporcionar el acceso a esa versión digital, de esta manera no tienen que esperar a que se haga efectiva la próxima devolución de la publicación en la biblioteca.

5.2. Entidades

5.2.1. Usuarios

BIBLIOTEQ tiene dos tipos de usuario; los administradores y los alumnos.

Hay una gran diferencia entre ambos, ya que los alumnos, como se ha comentado anteriormente, podrán reservar publicaciones o visualizarlas de manera online. Sin embargo, los administradores son los encargados de dar de alta o baja publicaciones o usuarios, así como gestionar las devoluciones de las publicaciones tras el fin del plazo de reserva.

Los campos que se van a tener sobre un usuario son los siguiente:

- Nombre
- Apellidos
- NIF
- Cargo (Administrado o Alumno)
- Nombre de usuario
- Contraseña

Las consultas que se realizarán en base de datos que guarden relación con los usuarios serán las siguientes:

- Insertar o eliminar registros en base de datos para dar de alta/baja un usuario en el sistema.
- Consultar todos los datos de un usuario.
- Consultar el nombre de usuario y contraseña para el acceso a la aplicación.
- Modificar datos de un usuario.

Los administradores se encargarán también de gestionar las reservas, finalizándolas una vez se haya entregado la publicación en la facultad correspondiente.

5.2.2. Publicaciones

Va a haber tres tipos de publicaciones: libros, revistas y proyectos, estos difieren en algunos campos.

Al crear una publicación hay que definir a qué tipo pertenecerá.

Los campos que las publicaciones van a tener en común son los siguientes:

- ISBN

- Título
- Lista de autores
- Materia
- Portada
- Número de ejemplares disponibles en la biblioteca
- Fecha de publicación
- URL para el acceso online.

Si la publicación es un libro se tendrá adicionalmente los siguientes campos:

- Edición
- Editorial
- Localidad en la que se ha escrito

Si la publicación es una revista los campos adicionales serán los siguientes:

- Periodicidad (Semanal, Mensual, etc)
- Volumen
- Ejemplar

Por último si es un proyecto los campos serán:

- Titulación
- Tribunal
- Nota

Las consultas que se van a realizar en base de datos que guardan relación con las publicaciones son:

- Consulta de todos los datos de una publicación a partir de su título.
- Consulta de todos los datos de una publicación a partir de una combinación de campos según quiera el usuario que la búsqueda sea más o menos exacta.
- Insertar registros de publicaciones para dar de alta una nueva publicación.
- Eliminar registros de publicaciones para dar de baja publicaciones.

A los usuarios se les permitirá buscar publicaciones en base a los campos que tienen en común.

5.2.3. Reservas

En BIBLIOTEQ, como se ha comentado anteriormente, los usuarios pueden realizar reservas de publicaciones desde casa sin la necesidad de ir a la biblioteca para comprobar si la publicación está disponible.

Por lo tanto, las reservas son una relación directa entre usuarios y publicaciones, por ello deben tener campos que pertenecen a ambos. Los campos de las reservas son:

- Nombre de usuario que realiza la reserva

- NIF del usuario que realiza la reserva
- Título de la publicación reservada
- Fecha en la que se realiza la reserva
- Fecha en la que se devuelve la reserva
- Estado de la reserva (finalizada o activa)

La fecha de devolución se puede modificar desde el momento que se crea la reserva, puesto que cuando un usuario realiza la reserva la fecha de devolución del sistema va a ser un mes después de la fecha de realización de esta. En el caso de que un usuario entregue la publicación antes de la fecha de devolución marcada, esta se actualizará al instante en el que el administrador finaliza la reserva.

La fecha de devolución no puede ser superior a un mes, es decir, el usuario tiene que devolver la publicación en una fecha tope de un mes tras la realización de la reserva para que otros usuarios puedan reservar esa misma publicación.

Las consultas a base de datos que guardan relación con las reservas son:

- Insertar registros de reservas en la base de datos con todos los campos especificados anteriormente.
- Buscar reservas a partir de los campos que las componen
- Modificar su estado y su fecha de devolución

5.3. Desarrollo

En este apartado se va a explicar todas las decisiones tomadas para realizar el desarrollo de BIBLIOTEQ, dentro de este desarrollo se incluye el modelado de las bases de datos y las conexiones con estas.

5.3.1. MongoDB

Para comenzar con el desarrollo de BIBLIOTEQ primero hay descargarse el controlador que proporciona MongoDB e incluirlo en nuestro proyecto Java. Aparte del controlador, es imprescindible añadir un par de librerías que se van a necesitar para la realización de consultas dentro de la aplicación, ya que el controlador simplemente sirve para una conexión síncrona o asíncrona con nuestra base de datos. Estas librerías se pueden encontrar para su descarga dentro de la documentación de MongoDB Java driver.

Como se ha explicado anteriormente una base de datos documental no tiene modelo de datos definido, pero a pesar de no tener modelo, se tiene que definir qué estructura se quiere que sigan los documentos que se almacenan. Esta estructura se va a definir a nivel de aplicación.

En nuestra base de datos se van a crear tres colecciones principales: usuarios, publicaciones y reservas.

Los documentos que se van a guardar en la colección usuarios van a tener la siguiente estructura:

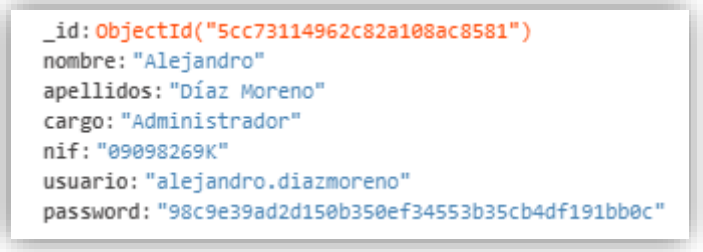


Ilustración 22: Ejemplo de documento en la colección usuarios

El campo id se introduce automáticamente cada vez que se inserta un nuevo documento en la colección.

Las contraseñas para mayor seguridad se almacenan encriptadas.

En la colección "publicaciones" se van a almacenar tres tipos de documento con estructuras ligeramente distintas, como en una colección se pueden insertar documentos con variaciones en su estructura, dentro de "publicaciones" se van a insertar libros, revistas y proyectos.

Ilustración 23: Ejemplo de un documento que contiene los datos de un libro

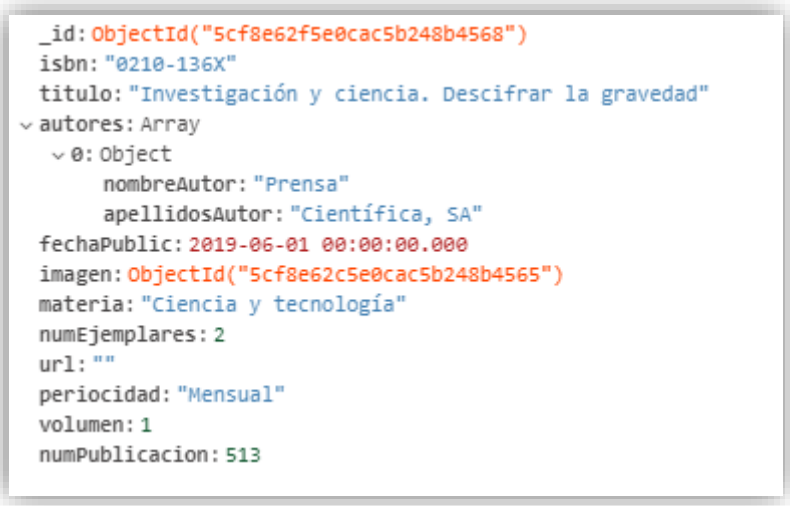


Ilustración 24: Ejemplo de un documento que contiene los datos de una revista

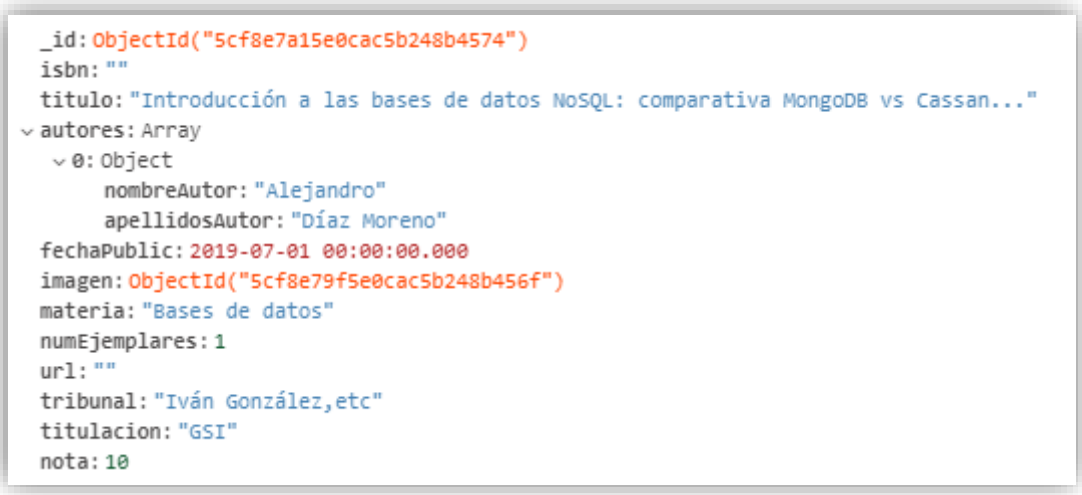


Ilustración 25: Ejemplo de un documento que contiene los datos de un proyecto

Para almacenar los autores existían varias opciones:

- Utilizar documentos embebidos, es decir, dentro del documento de la publicación tener tantos documentos dentro como autores. Ha quedado descartada porque aumentaría la complejidad de formación del documento.
- Guardar los autores en otra colección y referenciarlos con su ID. También se ha descartado, ya que habría un déficit del rendimiento.
- Almacenar los autores en una lista de objetos, de tal manera que cada objeto tendrá un campo nombre de autor y otro con los apellidos.

La última es la mejor opción ya que no es necesario utilizar documentos embebidos o almacenar documentos con los campos de los autores en otra colección. Como los autores no tiene gran complejidad, solo se componen de nombre y apellidos se puede transformar fácilmente en un “BasicDBObject”, soportado por MongoDB, e incluirse en

una lista de estos objetos para que en el caso de que haya más de un autor se almacenen todos en la base de datos.

```
public BasicDBObject toDBObject(){
    BasicDBObject autor = new BasicDBObject("nombreAutor",this.nombre)
            .append("apellidosAutor", this.apellidos);
    return autor;
}
```

Ilustración 26: Método que convierte un objeto Autor en un BasicDBObject

```
ArrayList<DBObject> lista = new ArrayList<>();
for (Autor autor: this.autores){
    lista.add(autor.toDBObject());
}
```

Ilustración 27: Ejemplo de creación de una lista de autores que se almacenará en un documento

Para el campo “imagen” (portada) se utiliza una referencia a un id de otro documento almacenado en la propia base de datos. Esto es debido a que se está utilizando la propia base de datos para el almacenamiento de archivos, en este caso de imágenes de la portada de las publicaciones. Se va a realizar a través de GridFS explicado anteriormente.

Por último, queda por ver la estructura de los documentos en la colección reservas, estos documentos van a almacenar datos tanto de publicaciones como de usuarios, y así tener una buena identificación de quien realiza la reserva y que publicación se reserva:

```
_id: ObjectId("5cf8e8f25ace05460630409b")
titulo: "Investigación y ciencia. Descifrar la gravedad"
fechaReserva: 2019-06-06 00:00:00.000
fechaDevolucion: 2019-07-06 00:00:00.000
usuario: "pruebas"
nif: "09098269K"
finalizada: false
```

Ilustración 28: Ejemplo de un documento que contiene los datos de una reserva

A continuación se va a mostrar algunas de los requisitos fundamentales para conectarse a la base de datos desde Java.

Primero hay que tener todos los archivos con extensión “.jar” descargados e importados en el proyecto:

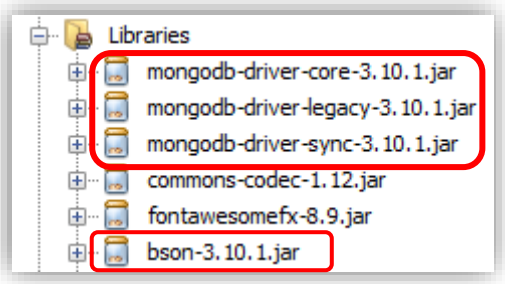


Ilustración 29: Librerías necesarias para usar el controlador de MongoDB

Tras tener listo el entorno, lo primero que hay que hacer es establecer una conexión con el servidor, que en nuestro caso será nuestro equipo o localhost. Para conectarse a nuestro host es muy importante que esté escuchando y aceptando conexiones, para ello una vez instalado MongoDB hay que usar desde la consola el comando "**mongod**".

Para crear una conexión con Mongo simplemente habría que crearse un cliente de la siguiente manera:

```
MongoClient mongoClient = MongoClients.create();
```

Ilustración 30: Creación de una conexión con el servicio mongod usando Java

Siempre que se haga una operación de escritura en la base de datos hay que cerrar la sesión del cliente que está abierta en ese momento y volver a establecer una nueva conexión, porque si se sigue con la misma conexión, al intentar hacer una operación de lectura sobre el registro insertado, no encontrará ningún resultado. Por lo tanto, cada vez que se realice una operación de escritura se necesita ejecutar lo siguiente o no se reflejará ese registro aún en nuestra base de datos:

```
mongoClient.close();
```

Ilustración 31: Cerrar la conexión con la base de datos

Para poder realizar consultas a la base de datos se tiene primero que especificar a que base de datos se accede y a que colección:

```
MongoDatabase database = mongoClient.getDatabase("biblioteca");
MongoCollection<Document> coleccion = database.getCollection("usuarios");
```

Ilustración 32: Recuperar una base de datos y una colección existente en MongoDB

Las operaciones de lectura y de escritura se realizan sobre las colecciones de la base de datos. En MongoDB en vez de realizar consultas a mano, es decir, escribir la query, el propio driver las genera a un nivel más bajo basándose en los parámetros que se les pase.

```
Bson query = and(eq("usuario","nombre_usuario"),eq("password", "contrasenna"));
```

Ilustración 34: Definición de una consulta usando filtros

```
Document documento = new Document("usuario","nombre_usuario")
        .append("password", "contrasena");
```

Ilustración 33: Definición de una consulta usando la clase Document

Los distintos tipos de operaciones son los siguientes:

- Lectura: Las operaciones de lectura se pueden realizar de dos maneras, o bien mediante filtros de búsqueda, o pasando como parámetro de búsqueda un objeto Java llamado Document que está disponible gracias a las librerías importadas al proyecto.

  Es importante tener en cuenta como están formados los documentos en nuestra colección, ya que si se busca un documento con el campo "usuario", y en los documentos en vez de llamar a ese campo "usuario" se llama de otra manera, nunca encontrará nada.

  Tras tener formados los parámetros se pasa a realizar la búsqueda en la colección:

```
ArrayList<Document> resultados = new ArrayList<>();
FindIterable iterator = collection.find(query);
if(iterator!=null){
    iterator.forEach(new Consumer<Document>(){
        @Override
        public void accept(Document t) {
            resultados.add(t);
        }
    });
```

Ilustración 35: Realizar una consulta en una colección de MongoDB y recuperar los resultados de esta

  El método "find" devuelve todos los documentos que ha encontrado en la colección que cumplan lo que se ha indicado y pasado por parámetro, ya sea el objeto "Bson" o el objeto "Document" definidos anteriormente.
  Como se puede ver en la imagen hay que recorrer el iterador resultante para obtener todos los documentos encontrados en esa colección.

- Escritura: Las operaciones de escritura pueden ser de modificación, de inserción o de eliminación de documentos. Para ambas operaciones hay que trabajar con la clase “Document”. Un ejemplo sería el siguiente:

```
public Document toDocument(){
    Document documento = new Document("nombre",this.nombre)
            .append("apellidos", this.apellidos)
            .append("cargo",this.cargo)
            .append("nif", this.nif)
            .append("usuario", this.nombreUsuario)
            .append("password",this.password);
    return documento;
}
```

Ilustración 36: Convertir la clase Usuario de Java en un Document para poder ser almacenado en una colección

Este método convierte una instancia de la clase Usuario, creada en nuestro proyecto, en un “Document” para poder ser insertado en la colección.
Para insertarlo en nuestra colección simplemente hay que recuperar la colección como ya se ha visto anteriormente, y usar el método “insertOne” pasándole por parámetro el documento a insertar:

```
usuarios.insertOne(usuario_nuevo.toDocument());
```

Ilustración 37: Insertar un documento en una colección

Para las modificaciones, en vez de utilizar el método anterior, se utiliza el método “updateOne”, y por parámetro se le pasa el documento que se quiere modificar y el documento con los cambios realizados.

```
reservas.updateOne(oldDocument, newDocument);
```

Ilustración 38: Actualizar un documento de una colección

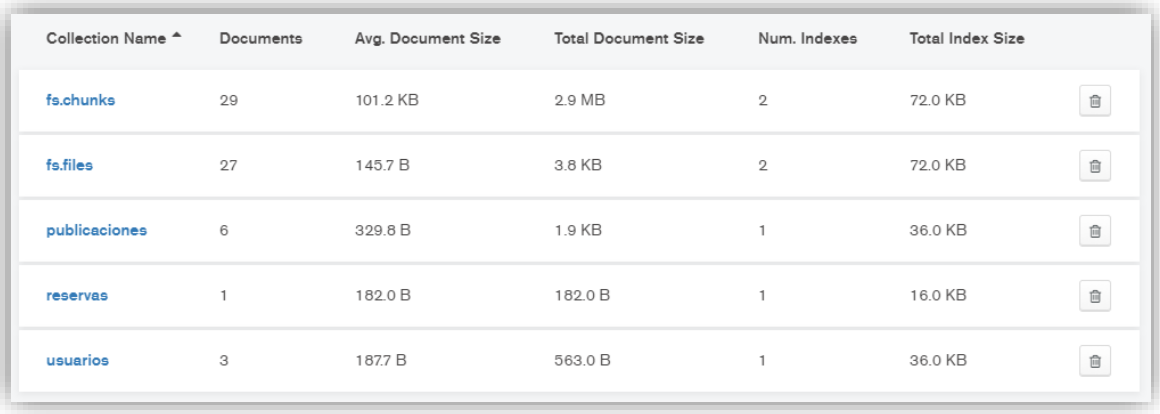
Por último, para eliminar un documento de una colección se usa el método “deleteOne”.

```
usuarios.deleteOne(documento);
```

Ilustración 39: Eliminar un documento de una colección

En el caso de que se quiera realizar estas operaciones de escritura de forma masiva, en vez de utilizar los métodos “intertOne”, “updateOne” y “deleteOne”, se utilizan los métodos “insertMany”, “updateMany” y “deleteMany”, y se pasa por parámetro una lista de “Document”.

Como resultado final se tendría una base de datos con las siguientes colecciones:



Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
fs.chunks	29	101.2 KB	2.9 MB	2	72.0 KB
fs.files	27	145.7 B	3.8 KB	2	72.0 KB
publicaciones	6	329.8 B	1.9 KB	1	36.0 KB
reservas	1	182.0 B	182.0 B	1	16.0 KB
usuarios	3	187.7 B	563.0 B	1	36.0 KB

Ilustración 40:Base de datos resultante en MongoDB

5.3.2. Apache Cassandra

Antes de comenzar con la realización de la aplicación, hay que tener en cuenta que Cassandra no proporciona una API para conectarse a la base de datos con Java, hay que identificar los clientes disponibles y elegir uno teniendo en cuenta varios factores como la documentación de este, si continúa actualizándose y se añaden nuevas funcionalidades, o si es de bajo nivel o de alto nivel.

Como primer paso se debe de tener en cuenta que la API de bajo nivel llamada **Thrift** dejó de actualizarse e incluir funcionalidades hace un tiempo. A pesar de ser esta de bajo nivel, lo que dificulta su utilización por lo que no es recomendable su uso, había otras APIs de alto nivel que funcionaban muy bien y eran muy estables, pero tenían el inconveniente de que dependían de esta anterior y al dejar de actualizarse se quedaron también obsoletas, ahora solo se pueden usar con versiones de Cassandra antiguas por lo que no es viable su utilización. Dos ejemplos son **Hector** y **Astyanax** creada por Netflix, pero hay más que han ido desapareciendo [15].

Tras este suceso, el cerco de clientes viables para utilizarlos en nuestro proyecto se cierra bastante y las opciones más interesantes son dos:

- **DataStax:** Este cliente está actualizado, tiene una buena documentación y está disponible para versiones de Apache Cassandra 2.1 o superiores, en nuestro caso se usará el último lanzamiento 3.11.4. Utiliza exclusivamente el protocolo binario de Cassandra y el lenguaje CQL versión 3. Para obtener ambos módulos se pueden realizar a través de dependencias de Maven o descargándose los archivos ".jar" que proporcionan. Contiene muchas de las funcionalidades que se implementaron en Astyanax y se incluyeron en este controlador, lo bueno es que este no depende del protocolo Thrift y que, actualmente, en Cassandra se utiliza el lenguaje CQL, muy similar al SQL, lo que ahorra algo de tiempo en aprendizaje [15,16].

- **Kundera:** Es una biblioteca compatible con JPA para varios almacenes de datos NoSQL, entre ellos está Cassandra. Ayuda mucho con el modelado NoSQL de nuestra base de datos con técnicas que optimizan, como la incorporación de relaciones, pero hay que tener cuidado porque si no se entiende bien el modelado de la base de datos de Cassandra puede hacer que se convierta en una base de datos RDBMS, y ese no es uno de los objetivos. Un punto positivo es que administra las transacciones y la persistencia de almacenamiento de los datos en varios nodos. Al utilizar JPA la documentación de su utilización es amplia y tiene buena disponibilidad [17].

Una vez visto dos de las opciones viables para utilizar en este proyecto, hay que elegir una para comenzar con la realización de BIBLIOTEQ. En este caso me parece más conveniente utilizar DataStax, pues es un Cliente orientado solo a Cassandra, y se va a poder aprender más sobre el propio funcionamiento de esta base de datos que si usamos Kundera. Aparte del inconveniente que resulta perder algo de aprendizaje sobre el funcionamiento interno de la base de datos al usar JPA, se tiene el inconveniente de poder acabar convirtiendo nuestra base de datos en una RDBMS. Por estas razones se escoge DataStax.

Ahora, el siguiente paso es definir un modelo de datos para nuestra base de datos, las familias de columnas que van a tener y las columnas dentro de la familia, así como sus claves para poder acceder a la información.

KEYSPACE

Es lo primero que se va a definir junto con sus atributos principales. En nuestro caso se va a nombrar “biblioteca”.

El factor de replicación en nuestro caso será de uno, ya que es el número de copias que existen de nuestros datos, y al utilizar Cassandra con un único nodo no se necesita almacenar más copias de los datos.

Como estrategia de replicación se escoge **“SimpleStrategy”,** esta es la estrategia utilizada para un anillo de nodos. En este caso los datos se colocarán en un único nodo.

El resultado final para la creación de del keyspace será el siguiente:

```
CREATE KEYSPACE IF NOT EXISTS biblioteca
WITH REPLICATION ={
'class':'SimpleStrategy',
'replication_factor':1};
```

Ilustración 41: Creación de un espacio de claves en Cassandra

FAMILIA DE COLUMNAS

Una vez definido y creado el keyspace, toca conformar las familias de columnas con sus campos y primary keys respectivas. Siguiendo el negocio de la aplicación comentado anteriormente, las tablas resultantes serían las siguientes:

USUARIO	
nombre	varchar
apellidos	varchar
nombreUser	varchar (Primary Key)
cargo	varchar
NIF	varchar (Primary key)
password	varchar

Tabla 4: Modelo de datos que siguen los usuarios en Cassandra

```
CREATE TABLE Usuario(
    nombre varchar,
    apellidos varchar,
    cargo varchar,
    nif varchar,
    nombreUser varchar,
    password varchar,
    PRIMARY KEY(nif,nombreUser));
```

Ilustración 42: Creación de la familia de columnas para usuarios en Cassandra

Para definir las publicaciones hay que tener en cuenta cómo se va a realizar el almacenamiento de los autores, de las imágenes de las portadas y cómo se va a hacer la distinción entre libros, revistas o proyectos.

Para almacenar los autores, Cassandra va a permitir definir tipos de datos propios, así que se definirá el tipo de dato Autor con su nombre y sus apellidos, y se guardarán como una lista de autores.

Para la portada, Cassandra no permite almacenar ficheros de más de 10 MB y esto es un problema. Por lo tanto, para no almacenar un archivo en varias columnas particionándolo, se va a utilizar el sistema de ficheros del servidor y de esta manera se ganará en rendimiento.

Por último hay que tratar la distinción entre los tres tipos de publicaciones. Para ello hay varias opciones:

- Crear tres familias de columnas distintas y asociarlas con las publicaciones con un id.
- Generar dentro de la familia de columnas de publicaciones tres super columnas que contengan a su vez las columnas necesarias con los campos de un libro, otra

super columna con las columnas necesarias para definir los campos de una revista y lo mismo para los proyectos.

- Hacer lo mismo que se realiza con los autores, se crean tres columnas más dentro de la familia de columnas llamada “publicaciones”, y se definen tres tipos de datos, uno que se llame “libro”, otro que se llame “revista” y otro “proyecto”; y que cada columna almacene un tipo de dato de los definidos.

Utilizar super columnas no está dentro de las buenas prácticas del lenguaje CQL, que es el que se está utilizando por lo tanto esa opción queda descartada.

Para este modelo se va a elegir la tercera opción comentada, es decir, utilizar tipos definidos. Esta es la opción escogida porque Cassandra da la opción de dejar cualquier campo de un registro como “NULL”, por tanto, no se consumirían recursos dejando como nula la columna “revista” y “proyecto” en caso de dar de alta un libro. También se escoge porque es la opción con mejor rendimiento puesto que para almacenar o buscar un libro, una revista o un proyecto solo se necesitaría una ‘Query’ y, en caso de escoger la primera opción comentada, habría que realizar mínimo dos. La primera opción es menos viable además porque se quiere diferenciar este esquema de uno relacional y aprovechar las ventajas que proporciona Cassandra, y si se hace separando los datos en varias familias de columnas y utilizando las keys para relacionar éstas, no se diferenciaría prácticamente de un modelo relacional, por tanto no se cumpliría uno de los objetivos principales del proyecto.

```
CREATE TYPE autor(
    nombre_autor varchar,
    apellidos_autor varchar
);
```

Ilustración 43: Ejemplo de creación de un UDT en Cassandra

PUBLICACION	
id_publicacion	varchar (Primary key)
Isbn	varchar (Primary key)
Titulo	varchar (Primary key)
autores	List<frozen<autor>>
fecha_publicacion	date
Portada	varchar
Materia	varchar
num_ejemplares	int
url	string
Libro	UDT frozen<libro>
Revista	UDT frozen<revista>
Proyecto	UDT frozen<proyecto>

Tabla 5: Modelo de datos que siguen las publicaciones en Cassandra

```
CREATE TABLE publicaciones(
    id_publicacion varchar,
    isbn varchar,
    titulo varchar,
    autores list<frozen<autor>>,
    fecha_publicacion date,
    portada varchar,
    materia varchar,
    num_ejemplares int,
    url varchar,
    libro frozen<libro>,
    revista frozen<revista>,
    proyecto frozen<proyecto>,
    PRIMARY KEY(id_publicacion,isbn,titulo)
);
```

Ilustración 44: Creación de la familia de columnas para publicaciones en Cassandra

Tras ya tener definido los usuarios y las publicaciones solo queda definir las reservas. Cabe destacar que en Cassandra no hay relaciones, por tanto, hay que definir a nivel de aplicación si se quiere eliminar un usuario o una publicación no puede tener asignada una reserva.

RESERVAS	
Id_reserva	varchar (Primary key)
Id_publicacion	varchar
nombre_usuario	varchar
NIF	varchar
titulo(publicación)	varchar
fecha_reserva	date
fecha_devolucion	date
finalizada	boolean

Tabla 6: Modelo de datos que siguen las reservas en Cassandra

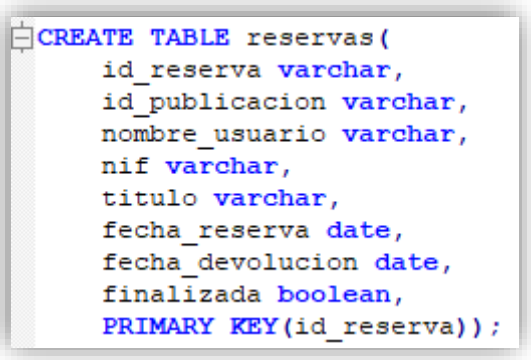


Ilustración 45: Creación de la familia de columnas para reservas en Cassandra

La asignación de primary keys en esta tabla es importante, ya que es conveniente poder buscar reservas en el sistema a partir del id_publicacion, el nombre de usuario o su NIF para que no se pueda borrar un usuario o publicación si tiene asignada una reserva en el sistema, pero en caso de que estos campos sean primary key, un usuario no podría reservar varias publicaciones debido a que no puede haber claves repetidas, ni una publicación se podría reservar tantas veces como números de ejemplares haya. Por tanto, solo se va a tener como primary key el id de la reserva.

Tras crear las familias de columnas se comienza con la adaptación de BIBLIOTEQ a Cassandra.

Antes de nada, hay que importarse una serie de librerías al proyecto para que se pueda usar el cliente de conexión a Apache Cassandra.

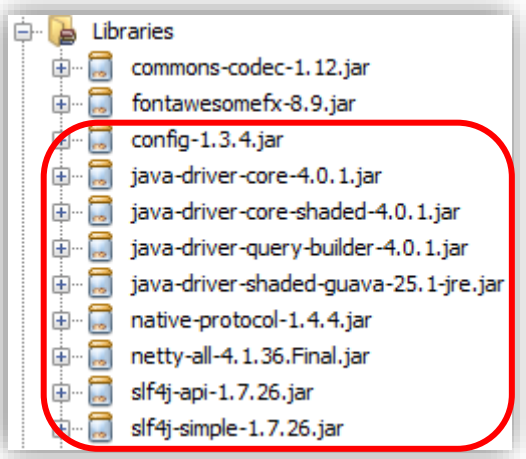


Ilustración 46: Librerías necesarias para el uso del driver de Cassandra en Java

Para una mejor organización se ha creado un servicio a base de datos, de esta manera cuando se quiera hacer alguna operación con Cassandra simplemente hay que llamar a los métodos del servicio.

Para conectarse a Cassandra hay que dejar escuchando conexiones al servidor y abrir una sesión CQL.

```
public class CassandraService {

 private CqlSession session;

 public CassandraService(){

 }

 public void connect(){
     session = CqlSession.builder().build();
 }
 public void connectToBiblioteca(){
     session = CqlSession.builder().withKeyspace(CqlIdentifier.fromCql("biblioteca")).build();
 }
 public void closeSession(){
     session.close();
 }
```

Ilustración 47: Creación de un servicio que se conecta al espacio de claves creado previamente y un método que cierra sesión con Cassandra

En el método “connect()” se abre una sesión con Apache Cassandra, pero no se especifica ningún espacio de claves al que conectarse. Sin embargo, si se fija uno en el método “connectToBiblioteca()”, se está conectando al espacio de claves definido previamente en el modelo.

Es importante que cuando se haga una operación de escritura se cierre sesión y se abra otra nueva, pues si no, los cambios en la base de datos no quedarán reflejados.

Hay varias formas de realizar consultas a nuestra base de datos a través de Java con este controlador, a continuación se va a mostrar ejemplos de dos de estas formas que son las más usadas por la comunidad:

```
public ResultSet buscarUsuarioPorNombreUser(String nombreUser){
    Select select =
            selectFrom("Usuario")
                    .all()
                    .whereColumn("nombreUser").isEqualTo(literal(nombreUser))
                    .allowFiltering();
    ResultSet rs = session.execute(select.asCql());
    return rs;
}
```

Ilustración 48: Ejemplo de consulta realizada con Query Builder

```
public ResultSet buscarUsuarioByNombreUser(String nombreUser){
    ResultSet rs = session.execute("SELECT * FROM usuario WHERE nombreuser=\'"
            +nombreUser+"\' ALLOW FILTERING;");
    return rs;
}
```

Ilustración 49: Ejemplo de consulta realizada a través de una declaración simple

El segundo ejemplo sería una consulta del lenguaje CQL que es enviada a Cassandra. Es el procedimiento conocido como declaración simple construida directamente desde una cadena de caracteres. Se suelen usar para consultas que no se suelen utilizar muchas veces en la aplicación.

El primer ejemplo es la misma consulta realizada con la utilidad que proporciona Datastax llamada Query Builder. Para el desarrollo del proyecto se han usado ambas metodologías a la hora de realizar diversas consultas a nuestra base de datos.

Hay dos grandes motivos por los que usar Query Builder, el primero es la seguridad, de esta manera se pueden evitar las inyecciones CQL, y el segundo es que permite realizar consultas de forma dinámica, algo que es muy útil para las búsquedas de publicaciones por varios campos. Sin embargo, el rendimiento se ve afectado considerablemente, por esta razón es por la que se ha decidido implementar consultas con ambos métodos.

El lenguaje CQL es un lenguaje muy similar al SQL, está muy apoyado por la comunidad por esta razón, por tanto, se van a incluir ejemplos de cómo se podrían realizar distintas operaciones con Query Builder.

A parte de las declaraciones simples, con esta utilidad se pueden realizar dos tipos de declaraciones más:

- Declaraciones preparadas: Se usa cuando hay que realizar una consulta varias veces. El funcionamiento es simple, se prepara la consulta solo una vez, esta consulta se guardará en la caché y la próxima vez que se ejecute la consulta el controlador solo enviará el campo por el que busca la consulta, de esta manera se ahorrará mucho más en rendimiento. También pueden ser utilizadas para construir consultas que sean demasiado complejas para realizarlas con una declaración simple.
- Declaraciones por lotes: Sirve para ejecutar varias consultas pero de manera atómica.

Para el desarrollo de BIBLIOTEQ se han usado declaraciones preparadas y declaraciones simples siempre fijándose en mejorar el rendimiento.

La mayoría de las consultas de declaración simple se han utilizado con fines de validación, como, por ejemplo, que no se pueda dar de alta un usuario si hay otro con el mismo nombre de usuario o NIF, dado que son las que mejor rendimiento proporcionan si se van a ejecutar pocas veces.

En las imágenes vistas anteriormente se ha visto que las consultas se ejecutan con el método “execute” de la sesión. La ejecución de estas consultas devuelve un “ResultSet” que es un iterable de filas, hay varias formas de recorrer un “ResultSet”, la más correcta sería utilizando un bucle.

```
for (Row row: rs){
    //procesar la fila
}
```

Ilustración 50: Forma para recuperar los registros que devuelve una consulta

Una fila equivale a un registro de la familia de columnas consultada. Sí solo se quiere el primer registro o fila de los resultados obtenidos habría que hacer lo siguiente:

```
rs.one();
```

Ilustración 51: Método para obtener el primer registro de todos los que devuelve una consulta

Con Query Builder no solo se pueden realizar operaciones de lectura, también se pueden ver ejemplos de escrituras en la base de datos:

```java
public void insertUsuario(Usuario usuario){
    Insert insert =
            insertInto("usuario")
            .value("\"nif\"", literal(usuario.getNif()))
            .value("\"nombreuser\"", literal(usuario.getNombreUsuario()))
            .value("\"nombre\"",literal(usuario.getNombre()))
            .value("\"apellidos\"", literal(usuario.getApellidos()))
            .value("\"cargo\"", literal(usuario.getCargo()))
            .value("\"password\"", literal(usuario.getPassword()));
    session.execute(insert.asCql());

}

public void borrarUsuario(String usuario,String nif){
    Delete delete =
            deleteFrom("usuario")
            .whereColumn("nombreuser")
            .isEqualTo(literal(usuario))
            .whereColumn("nif").isEqualTo(literal(nif)).ifExists();
    session.execute(delete.asCql());
}
```

Ilustración 52: Ejemplo de cómo se inserta y elimina un registro usando Query Builder

```java
public void updateReserva(String id_reserva,LocalDate fecha_devolucion){
    Update update =
            update("reservas")
            .setColumn("\"finalizada\"", literal(true))
            .setColumn("\"fecha_devolucion\"", literal(fecha_devolucion))
            .whereColumn("\"id_reserva\"").isEqualTo(literal(id_reserva));

   session.execute(update.asCql());
}
```

Ilustración 53: Ejemplo de actualización de un registro usando Query Builder

Para tratar los valores definidos (UDT), hay que acceder a los metadatos del espacio de claves y poder crear un “UdtValue” con los campos que se requieren, por ejemplo, para el UDT que se denominó anteriormente como libro se haría de la siguiente manera:

```java
//Conseguir el objeto definido libro.
Optional<UserDefinedType> udt_libro = session.getMetadata().getKeyspace("biblioteca")
        .get().getUserDefinedType("libro");
UdtValue libroValue = udt_libro.get().newValue()
        .setInt("\"edicion\"", libro.getEdicion())
        .setString("\"editorial\"", libro.getEditorial())
        .setString("\"localidad\"", libro.getLocalidad());
```

Ilustración 54: Tratamiento de un UDT en Java

Para poder buscar por columnas que no sean Primary Key hay que crearse índices secundarios, si se quiere usar operaciones típicas provenientes de SQL, como "LIKE", se necesitará un índice secundario SASI.

```
CREATE CUSTOM INDEX publicaciones_materia_indx ON publicaciones (materia)
USING 'org.apache.cassandra.index.sasi.SASIIndex'
WITH OPTIONS = {'mode': 'CONTAINS',
'analyzer_class': 'org.apache.cassandra.index.sasi.analyzer.NonTokenizingAnalyzer',
'case_sensitive': 'false'};
```

Ilustración 55: Creación de un índice secundario tipo SASI en Cassandra

Ya se ha hablado anteriormente del uso de índices y sus recomendaciones de uso, por lo que en este apartado no se va a contemplar las conclusiones sacadas.

Este ejemplo de índice SASI visto en la imagen anterior, permitiría usar el comando "LIKE" sobre la columna **"materia"** de publicaciones.

6. Comparación de las API

Para la comparación de las API de MongoDB y Cassandra, se van a plantear cuatro escenarios:

- Escenario 1: Comprobar el tiempo de acceso a la aplicación a través del inicio de sesión.
- Escenario 2: Tiempo en dar de alta una publicación en el sistema.
- Escenario 3: Tiempo en realizar una búsqueda de una publicación.
- Escenario 4: Tiempo en reservar esa publicación y mostrarte por pantalla las indicaciones a seguir.

Para realizar la comprobación se van a utilizar ambas aplicaciones de BIBLIOTEQ y se van a comprobar los tiempos de ejecución de los accesos a base de datos en todos los escenarios, con el fin de sacar en claro cuál es la base de datos que realiza operaciones de la manera más rápida y eficaz a través del Cliente Java.

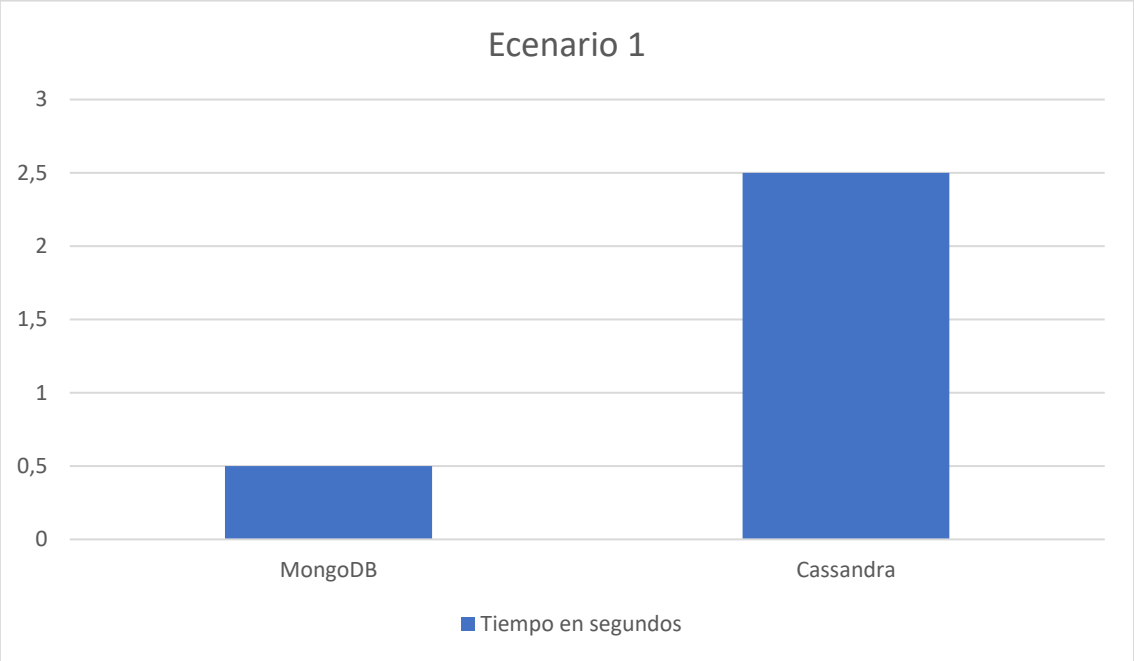


Ilustración 56: Comparación de tiempos en el inicio de sesión de BIBLIOTEQ MongoDB y BIBLIOTEQ Cassandra

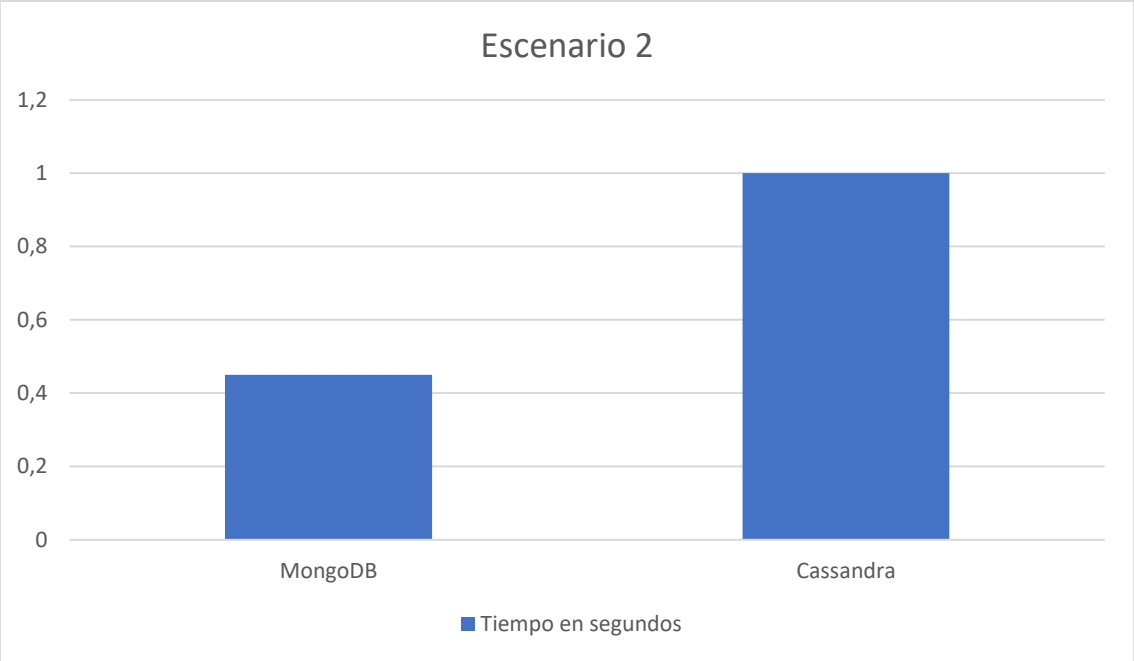


Ilustración 57: Comparación de tiempos en el alta de publicaciones de BIBLIOTEQ MongoDB y BIBLIOTEQ Cassandra

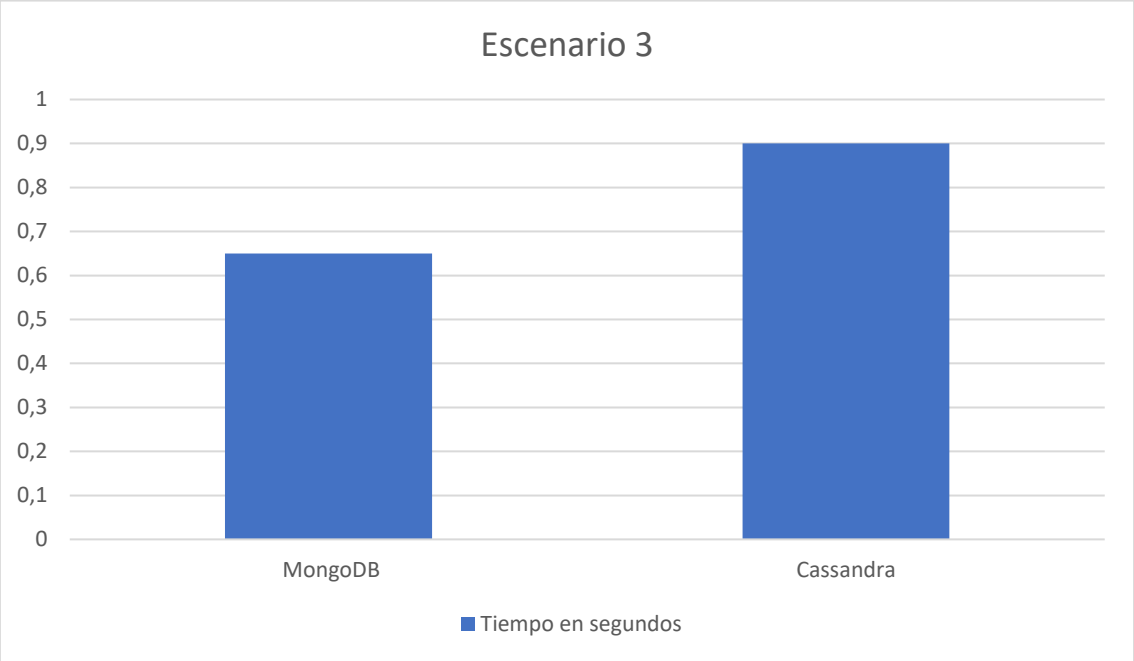


Ilustración 58: Comparación de tiempos en la búsqueda de publicaciones de BIBLIOTEQ MongoDB y BIBLIOTEQ Cassandra

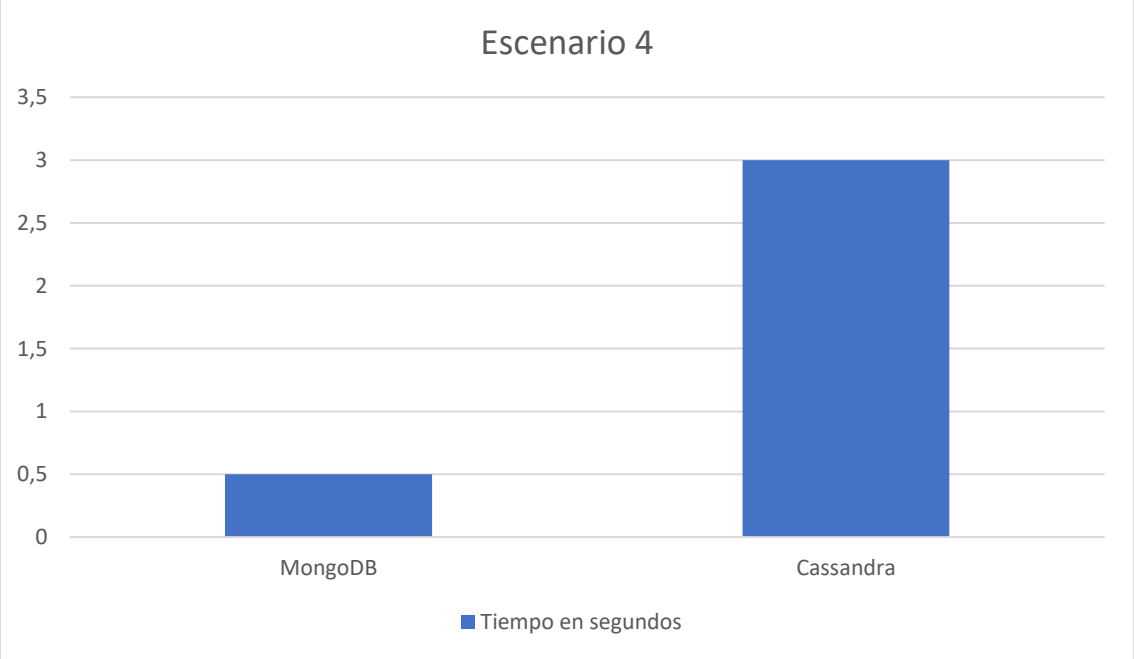


Ilustración 59: Comparación de tiempos en la reserva de publicaciones de BIBLIOTEQ MongoDB y BIBLIOTEQ Cassandra

Como se puede observar en los gráficos, en todos los escenarios de pruebas MongoDB accede de manera más rápida a la base de datos y realiza las operaciones a ejecutar en una cantidad de tiempo inferior que con el Cliente Java de Apache Cassandra. A excepción del escenario 3, que los tiempos de acceso a base de datos y realización de operaciones es parecido aunque inferior en MongoDB, en el resto de los escenarios la

API de MongoDB es mucho más rápida que la de Apache Cassandra que llega a alcanzar o incluso a superar el segundo.

Por tanto, si se desea que los tiempos de respuesta en la aplicación sean lo más cortos posibles habría que escoger la utilización de MongoDB con la API de Java que proporciona.

7. PRESUPUESTO

7.1. Estudio de mercado de MongoDB

Antes de comenzar a realizar el presupuesto, se va a realizar un breve estudio de mercado con las diversas opciones presupuestarias, de esta manera el escoger una opción u otra se podrá basar en aspectos cualitativos y no solo en el precio.

7.1.1. MongoDB Atlas

MongoDB tiene diversos partners, por lo que más que un base de datos es una empresa, bastante fiable.

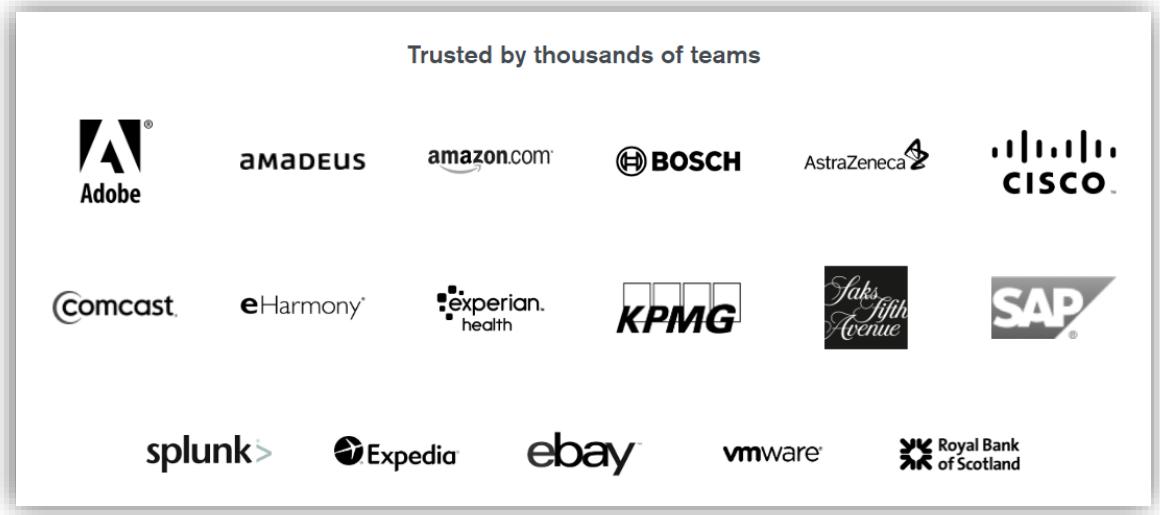


Ilustración 60: Empresas que han contratado de proveedor a MongoDB

En resumen, no solo proporciona el servicio de base de datos, en su página web se puede encontrar una gran cantidad de herramientas complementarias para gestionar los datos que se almacenan.

También MongoDB Atlas da la posibilidad de contratar el servicio de hospedaje Cloud en AWS, Azure o GCP (Google Cloud Plataform).

Se podría contratar este servicio de hospedaje independientemente de MongoDB, pero Atlas no solo proporciona el hospedaje, también proporciona acceso a una interfaz en la que se puede gestionar varios de los recursos:

- En la sección de seguridad se puede gestionar los accesos a nuestra base de datos, añadir usuarios con diversos permisos, ver con qué IP está estableciendo la conexión al clúster, y las conexiones que están activas en tiempo real. También

da la posibilidad de visualizar una serie de métricas de cuántas conexiones se han producido durante ciertas horas del día y en cuál de los nodos se han producido.

- Tiene integrado un sistema de alertas, las alertas tendrán que ser creadas de cero en base a las necesidades, un ejemplo podría ser que cuando un nodo del clúster se cae, avise para levantar otro. Las alertas se pueden enviar por SMS o por correo electrónico.
- Proporciona un acceso directo a nuestra base de datos en la que se podrá visualizar los documentos que tiene y realizar consultas sobre esta, lo que evita un coste adicional de contratar otra GUI. Incluye métricas de las operaciones que se realizan en las bases de datos de nuestro clúster.
- Se podrá visualizar un log en tiempo real.

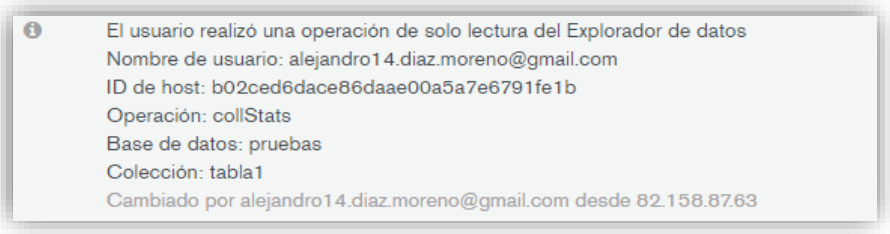


Ilustración 61: Ejemplo de log en MongoDB Atlas

- Permite integración de otras herramientas MongoDB. Como Charts que permite crear un Dashboard para añadir gráficas sobre los datos que se almacenan.
- Incluye de forma gratuita una tarifa de soporte con acceso de lunes a viernes en el que se pueden realizar consultas en un chat. Da la opción de ampliar esta tarifa por 49$ al mes, que incluye acceso los siete días de la semana a cualquier hora, un soporte completo con asesoramiento sobre cualquier tema, respuesta en un máximo de 8 horas y el chat que está también en el plan gratuito.

También existe la posibilidad de contratar dos planes para la automatización de copias de seguridad, se diferencian en que una de las dos es instantánea y se puede realizar de nodo a nodo, es decir, sirve para distribuir la base de datos en varios nodos y poder asegurar la consistencia de los datos de la manera más efectiva posible. Sin embargo, el otro plan se basa en snapshots que se realizan cuando se necesite, por ejemplo 1 vez al día.

En Atlas viene incluida la última versión de MongoDB Community Server, en la que no hay que realizar ninguna instalación ni configuración del clúster.

7.1.2. Servidor On Premise

En caso de que se que opte por la opción de un servidor On-Premise en vez de usar la nube, se tiene una elección de dos opciones:

- MongoDB Community Server que incluye la última versión de la base de datos no relacional, y te proporcionan un enlace en la que se podrá monitorizar el clúster y la base de datos.

- MongoDB Enterprise Server que también incluye la última versión de la base de datos y una gran cantidad de utilidades en las que destacan una GUI para visualización de los datos, un motor de almacenamiento en memoria, la posibilidad de encriptar los datos y auditorías para la seguridad de los datos que se almacenan.

Los únicos inconvenientes son que se pierde la posibilidad de escalado, la instalación del servicio MongoDB en el servidor que se contrate corre a cargo del cliente, a diferencia de los servicios Cloud, hay que hacer un mantenimiento tanto del clúster como del servidor y los costes de servidor pueden aumentar si el cliente no dispone del espacio físico para su alojamiento. La empresa proveedora del servidor y de su mantenimiento seleccionada es DELL, debido a que permite configurar el servidor a nuestro gusto, su servicio de mantenimiento y soporte es muy completo, y es una empresa con un posicionamiento de mercado muy bueno que trabaja con muchos partners del sector IT.

7.1.3. DocumentDB

Amazon DocumentDB es un servicio de base de datos de documentos ágil, escalable, de alta disponibilidad y completamente administrado, que admite cargas de trabajo de MongoDB.

Es decir, a pesar de ser una base de datos que no es MongoDB funciona como tal. Normalmente se utiliza para realizar migraciones de MongoDB de On Premise al Cloud. Para utilizarla no hay que cambiar nada en nuestra aplicación, ya que se sincroniza con el cliente MongoDB utilizado para BIBLIOTEQ.

Ofrece una serie de utilidades y características que respaldan su elección como una buena opción de mercado:

- Para comenzar su uso simplemente hay que acceder a la consola de administración de AWS e iniciar el clúster, las instancias ya están preconfiguradas con todos los valores que se fijaron previamente en la elección del tipo de instancia. Por lo tanto, la administración y configuración del clúster son automáticos, lo que supone ahorrarse estos costes.
- Viene incluida la herramienta Amazon CloudWatch para monitorizar el clúster y sus instancias, así como poder ver una serie de métricas como conexiones activas, uso de memoria, uso de CPU, etc.
- Se puede indicar si se quiere que se instalen nuevos parches para mantener actualizada la base de datos, y en caso de instalarlos se puede seleccionar cuándo, de esta forma no afectará a la disponibilidad de la aplicación.
- El escalado en almacenamiento es automático, cuando se detecta que queda poco espacio se aumentan 10 GB en el almacenamiento. Este escalado va de 10 en 10 GB hasta llegar a 64 TB, sin embargo, para el escalado de recursos, como puede ser la memoria, hay que acceder a la consola de administración de AWS y

configurar los valores que se desean, también se pueden añadir o eliminar instancias en el clúster.

- Administra las réplicas de manera automática en hasta quince instancias de nuestro clúster. Esto hace no preocuparse ni por la replicación ni por la fragmentación de nuestro clúster. La arquitectura de esta base de datos es idéntica a la de MongoDB.
- Incluye una serie de recursos para controlar la seguridad del entorno. Se puede realizar un aislamiento del clúster en una red virtual privada, configurar un firewall y controlar los accesos a la red en la que se sitúa el clúster. También proporciona la posibilidad de gestionar los privilegios de cada usuario que se conecta al clúster y cifrar el contenido de la base de datos.
- El clúster está monitorizando automáticamente para encontrar errores y repararlos. En caso de que se detecte un error, la instancia y lo procesos que se estaban ejecutando se reinician, el reinicio se lleva a cabo en 30 segundos aproximadamente y durante este se aísla la caché para que siga existiendo al finalizar. Si no hay manera de recuperar la instancia, o no se tiene la instancia caída aprovisionada, se crea automáticamente una nueva.
- Por último, se incluye una serie de instantáneas del clúster, estas instantáneas son copias de seguridad del clúster. Se pueden utilizar para reestablecer un clúster en un punto anterior al actual o, incluso, se puede crear un nuevo clúster a partir de éstas.

A pesar de que DocumentDB es una base de datos distinta a MongoDB es una muy buena oportunidad de mercado. Está totalmente gestionada y administrada por Amazon por lo tanto se ahorra en costes y en tiempo para su puesta en marcha.

7.2. Matriz presupuestaria para la implementación de BIBLIOTEQ con MongoDB

Una vez analizadas todas las posibilidades para implementar BIBLIOTEQ se va a realizar una matriz presupuestaria que las compare económicamente.

Dentro de estos presupuestos se incluirán los gastos de mano de obra de BIBLIOTEQ, estos costes equivalen a 7 € la hora, cuarenta horas semanales durante dos meses, incluyendo fases de prueba. Tiene un coste añadido de adaptabilidad para un servidor JBOSS, necesario para su uso online, este coste supone un 15 % del precio del desarrollo, por lo que el precio final de BIBLIOTEQ sería de 2.576 €.

Los costes de mantenimiento y soporte de BIBLIOTEQ no están incluidos en la línea base, sino que funcionan por bolsa de horas, de manera que cada hora empleada en resolución de incidencias o mejoras en la aplicación se cobrará a 8 € la hora. Este servicio dependerá del contrato, ya que no se tiene porque adquirir o renovar en los siguientes años.

Aparte de la adquisición de BIBLIOTEQ, también se está adquiriendo todo el entorno, lo que supone implantar un Servidor JBOSS para alojar la aplicación online, el servidor JBOSS cuesta 25 € el mes, que da como resultado un coste anual de 300 €.

	MongoDB Atlas (Cloud)	On Premise	DocumentDB (Cloud)
BIBLIOTEQ APP	2.576 €	2.576 €	2.576 €
Servidor JBOSS	300 € / año	300 € / año	300 € / año
Hosting de la BBDD	6.095 € / año	1.700 €	4.981 € / año
Configuración del entorno	-	1.700 €	-
Mantenimiento y soporte	512 € / año	1. 063 €	-
Mantenimiento y soporte de la APP	Bolsa de horas de 8 €/h	Bolsa de horas de 8 €/h	Bolsa de horas de 8 €/h
Base de datos MongoDB	-	-	-
Gestión de las copias de seguridad	120 €	-	12 €
Formación	-	-	-
TOTAL	9.603 €	7.339 €	7.869 €

Tabla 7: Matriz presupuestaria para implantar BIBLIOTEQ con MongoDB

Para la opción de **MongoDB Atlas** se tiene un conjunto de réplicas de tres nodos con 16 GB de memoria y el clúster tendrá 80 GB de almacenamiento interno. Si se quiere fragmentar el clúster para escalar horizontalmente habría que añadir otro conjunto de réplicas, primeramente con un conjunto es suficiente para comenzar. Las instancias o nodos del conjunto son instancias de AWS tipo M40. De estos 80 GB de almacenamiento de datos, se harán copias de seguridad. También hay que contar con que se realizarán aproximadamente 5 millones de operaciones E/S al mes, se tendrá una transferencia entrante de datos al mes de 1 GB (4 KB cada reserva aproximadamente) y una trasferencia de datos saliente de 12 GB al mes.

- El conjunto de réplicas de tres nodos se calcula que estará activo durante 18 horas todos los días del año, lo que tiene un precio de 16,7 €, lo que daría un precio anual de 6.095 €.
- Las operaciones E/S son gratuitas.
- La transferencia de datos tiene el mismo coste ya sean entrantes o salientes, que será de 0,0803 € el GB de datos. Si se ha calculado aproximadamente 13 GB de transferencia de datos por mes, el precio mensual será de 1,04 € el mes, lo que da un valor anual de 12 €.

Para esta solución se tendría un precio total de 6.107 € el año, sin contar la gestión de copias de seguridad que está en otro apartado de la matriz, este coste ascenderá a 120 € el año. Viene incluido un mantenimiento del clúster muy básico, por ello se va a contratar un plan ampliado de 43 € el mes, lo que supone un total de 512 € al año.

Si se elige que el hosting sea **On Premise,** se perderá esa opción de escalado, sería un servidor de 32 GB de memoria RAM, 1 TB de espacio de almacenamiento y 4 cores, y un solo nodo, es decir, cuando hay que realizar mantenimiento de la BBDD o del servidor se pararía por completo el entorno de producción. El precio de este servidor es de 1.700 €. El espacio físico no está incluido, pero es un servidor con dimensiones de un ordenador de torre por lo que se puede alojar con facilidad.

Para la configuración hay que tener en cuenta que se tiene que realizar una instalación completa del servidor con su SSOO y los puertos que se van a utilizar. Aparte hay que instalarle MongoDB Community Server y configurar el clúster para poder conectarse con BIBLIOTEQ, y crear una máquina virtual con acceso al servidor para poder gestionarlo. DELL, la empresa proveedora del servidor, da una opción de instalar el hardware in situ y de configurar el servidor de manera remota por 1.000 €. Luego habría que contratar otro servicio para instalar MongoDB que podría incrementar el coste en 700 €, haciendo un total de 1.700 € por la configuración completa del entorno.

El mantenimiento del servidor también se puede contratar con DELL, en este caso se ha seleccionado el servicio ProSupport Plus for Enterprise, el mantenimiento tiene 3 años de duración y vienen incluido herramientas de monitorización del servidor, asistencia para arreglar problemas de software, asistencia para arreglar problemas de hardware y, si se necesita una pieza nueva para el servidor, se tendrá al siguiente día laborable, y disponibilidad de asistencia técnica las 24 horas del día durante todos los días de la semana. El coste de mantenimiento para los tres años es de 1.063 €.

Por último, el precio de la gestión de las copias de seguridad va incluido tanto en el mantenimiento como en la configuración del clúster.

En el caso de **DocumentDB** de AWS se tienen 3 instancias, una principal sobre la que realizarán las operaciones de lectura y escritura, y dos réplicas sobre las que se repartirán operaciones de lectura, todo esto formará un conjunto de réplicas. Estas instancias tendrán 16 GB de memoria y dos CPU, el clúster podrá almacenar 50 GB de datos, con posibilidad de escalar. De estos 50 GB de almacenamiento de datos se harán copias de seguridad. También hay que contar con que se realizarán aproximadamente 5 millones de operaciones E/S al mes, y se tendrá una transferencia entrante de datos al mes de 1 GB (4 KB cada reserva aproximadamente) y una trasferencia de datos saliente de 12 GB al mes.

- Tres instancias activas 18 horas al día durante los 365 días del año sería un total de 1.624 € cada instancia. Supondría un total de 4.872 € al año, que es lo que se refleja en la matriz presupuestaria.
- El precio de almacenar 50 GB al mes es de 4,5 € el mes, lo que supone un coste anual de 54 €. Si se quiere escalar el almacenamiento tiene que ser de 10 GB en 10 por lo tanto el coste de escalado es de 0,9 € el mes. El coste de escalado no se incluye en la matriz presupuestaria, pero es bueno saber que si ese mes ha

habido un aumento de 10 GB en el almacenamiento del clúster, se pagará 0,9 € más ese mes y los siguientes.

- El costo mensual de 5 millones de operaciones es de 4,5 € el mes, un total de 54 € el año.
- Las transferencias de datos entrantes son gratuitas, las salientes tendrán un coste de 1 € el mes.

El precio total para el hosting de la base de datos será de 4.981 €. El coste de gestión de las copias de seguridad se ha incluido en otro apartado dentro de la matriz presupuestaria, para DocumentDB ascenderá a la cifra de 12 € el año. El mantenimiento y soporte del clúster y la configuración vienen incluidos con la contratación de esta solución. Si se quiere escalar horizontalmente se podrán añadir más instancias, es mejor añadirlas de tres en tres para formar conjuntos de réplicas.

La formación para todos los presupuestos es gratuita, ya que dentro del precio de mantenimiento de los entornos se incluye formación por parte de estas empresas.

También con la adquisición de BIBLIOTEQ viene incluido manual de usuario de BIBLIOTEQ. Cada desarrollo o mejora que entre dentro del plan de mantenimiento deberá tener un análisis previo y se entregará con documentación, con el fin de que nunca se pierda conocimiento de la aplicación.

Hay que tener en cuenta que se han realizado estimaciones para obtener el cálculo de este presupuesto. Sobre todo, en los servicios Cloud, no se sabe nunca exactamente si se va a necesitar escalar vertical u horizontalmente, si en un mes se va a tener un número de operaciones E/S mayor o menor que el mes anterior, o también si la transferencia de datos puede variar mensualmente, por esta razón en estas implementaciones se paga bajo demanda y este presupuesto es realizado en función de unos parámetros preestablecidos que se intentan asemejar lo máximo posible a las cifras una vez se implementara dicho entorno. Por lo tanto, el coste en estos presupuestos puede aumentar o disminuir.

7.3. Estudio de mercado de Apache Cassandra

Al igual que con MongoDB, se va a hacer un estudio de mercado con diversas opciones en las que se pueden implementar un entorno con Apache Cassandra. Se va a plantear una solución On Premise y otra Cloud.

7.3.1. On Premise

Para su implementación On Premise en un único nodo se elige la misma opción que con MongoDB, es decir, como empresa proveedora del servidor y su mantenimiento se escoge a DELL.

7.3.2. Distribución DataStax de Apache Cassandra en Microsoft Azure

Para entornos de producción actualmente la solución más viable es la utilización de la distribución de DataStax de Apache Cassandra. Aparte del propio servicio de base de datos, proporciona también el mantenimiento y configuración del clúster.

Se podría usar Apache Cassandra sin ser distribuida por ninguna empresa porque es de código libre, pero con DataStax se asegura de que el entorno que se va a utilizar está listo para la producción, minimizando los riesgos ya que se somete a una serie de pruebas y controles de calidad, aparte este entorno es administrado por esta empresa.

La implementación se va a realizar con Microsoft Azure, ya que es la única que ofrece una plantilla de implementación que es gratuita, pero a cambio todos los componentes contratados tienen que ser de esta empresa.

En resumen, los servicios que van a proporcionar estas dos empresas son los siguientes:

- Para la puesta en marcha de la solución simplemente hay que seguir una serie de pasos muy simples dentro del portal de Microsoft Azure y se montará automáticamente toda la infraestructura. Dará la posibilidad de seleccionar los recursos que se quieren como, por ejemplo, la cantidad de memoria de los nodos, el almacenamiento interno, etc. Dependiendo de lo seleccionado se tendrá un precio u otro.
- Desde el portal de Azure se tiene la monitorización de todos los componentes que se han contratado para poder gestionar errores o caídas en algunos de los nodos, aparte también se pueden gestionar la configuración, ésta es modificable.
- Mantenimiento, configuración del clúster y soporte técnico ante dudas.
- La última versión de Apache Cassandra y los drivers para conectar Apache Cassandra con la aplicación.
- Al ser un servicio Cloud la posibilidad de escalado siempre está presente, simplemente habría que cambiar la configuración de los componentes para un escalado vertical o añadir más nodos en el clúster para un escalado horizontal.
- Visualización de métricas y de gráficas de los datos almacenados en el clúster.

7.3.3. Apache Cassandra en Amazon EC2

Se pueden encontrar guías de implementación de Apache Cassandra para AWS, pero en esas mismas guías se recomienda usar la propia base de Amazon llamada DynamoDB ya que es totalmente administrada.

Si se elige esta opción sin seguir las recomendaciones, la única empresa que administra los clústeres de Cassandra es DataStax, que trabaja con Azure, por lo tanto, se tendrían que administrar por los usuarios el clúster y los costes ascenderían considerablemente.

7.4. Matriz presupuestaria para la implementación de BIBLIOTEQ con Cassandra

Antes de establecer los presupuestos para ambas implementaciones, hay que destacar que el valor de BIBLIOTEQ es el mismo que el que se ha desarrollado en los presupuestos anteriores, al igual que el servidor JBOSS y el mantenimiento de la aplicación. El resto de los conceptos varían al ser contratados por otras empresas.

También cabe destacar que el entorno On Premise es exactamente igual, lo único que habrá que ajustar el valor para el mantenimiento y el soporte distribuido por la empresa DataStax Enterprise.

	On Premise	**Microsoft Azure (Cloud)**
BIBLIOTEQ APP	2.576 €	2.576 €
Servidor JBOSS	300 € / año	300 € / año
Hosting de la BBDD	1.700 €	6.570 € / año
Configuración del entorno	1.000 €	-
Mantenimiento y soporte	1. 851,4 €	1.661,6 €
Mantenimiento y soporte de la APP	Bolsa de horas de 8 €/h	Bolsa de horas de 8 €/h
Base de datos Apache Cassandra	-	-
Gestión de las copias de seguridad	-	21 €
Formación	-	-
TOTAL	7.427,4 €	11,128 €

Tabla 8: Matriz presupuestaria para implantar BIBLIOTEQ con Apache Cassandra

El coste de DataStax Enterprise para su soporte 24 horas al día los 365 días del año es de 0,03 € el nodo por hora de uso. La opción **On Premise** tiene un solo nodo, el servidor estará activo durante 24 horas los 365 días del año, lo que supone un precio total 262,8 € al año. Como el precio del mantenimiento de DELL está calculado sobre 3 años el valor que hay que sumarle es de 788,4 €. Este soporte es del clúster de la base de datos, dentro de este precio se incluye la ayuda para la configuración del clúster en el servidor, por tanto, este precio hay que descontárselo a la configuración del entorno.

El precio resultante para el concepto configuración del entorno es de 1.000 € y para el mantenimiento y soporte es de 1.851,4 €.

Para montar la arquitectura en **Microsoft Azure** se van a contratar 8 máquinas virtuales escalables para que formen el clúster en forma de anillo. Esas máquinas se pagan por uso, y se estima que estarán en funcionamiento 18 horas los 365 días del año. Si cuesta cada una 0,125 € por hora de uso el precio anual de una de las máquinas es de 821,25 €. Como se tienen que contratar 8 tendremos un precio anual final de 6.570 €. Estas

máquinas tienen la posibilidad de escalar verticalmente y también se pueden añadir nuevos nodos al anillo. Cada máquina se compone de 2 CPU, 16 GB de memoria RAM y 50 GB de almacenamiento. Todos estos nodos estarán contenidos en una red virtual.

La red virtual es gratuita ya que no se superan los 100 GB de intercambio de datos entrantes ni salientes mensualmente. Incluye un número ilimitado de operaciones E/S.

El precio de las instantáneas para generar las copias de seguridad del clúster es de 21 € el año. Cada instantánea tendrá una copia completa del clúster.

El mantenimiento y soporte de la infraestructura Cloud lo ofrece también Azure. Hay dos opciones de mantenimiento posibles:

- Estándar: Es mejor en caso de tener una dependencia crítica empresarial mínima de Azure. Su precio es de 85 €.
- Professional Direct: Se suele elegir cuando el uso de Azure es crítico para el negocio, su precio es de 843,3.

La principal diferencia es que con la Professional se gestionan las migraciones y la seguridad, y con la estándar no.

El mantenimiento que ofrece DataStax sobre el clúster es el mismo comentado para la opción de On Premise, solo que se tendrán 8 nodos en el anillo y no uno, por lo que el precio es de 1.578,8 € al año. Como se van a combinar estos dos mantenimientos, se escoge el estándar para el mantenimiento de la infraestructura montada. El precio final del mantenimiento es de 1.661,6 €. El precio del mantenimiento de Azure se paga una vez y dura hasta que se dejen de utilizar los componentes contratados. Por tanto, el precio del primer año es de 1.661,6 €, posteriormente se reducirá a 1.578,8 €.

La configuración del entorno viene incluida en el plan de mantenimiento y soporte que ofrecen las dos empresas, ya que este entorno es administrado por ellos.

7.5. Conclusiones presupuestarias

Hay una gran diferencia entre la implantación de Apache Cassandra y MongoDB, se trata del respaldo empresarial, en MongoDB es mucho mayor, pues no solo es una solución sino es una empresa que tiene de partners a AWS, Microsoft Azure o Google Cloud Plataform para facilitar la implantación y administración de cualquiera de las soluciones que ofrece.

En cambio para Apache Cassandra este respaldo se ha intentado conseguir mediante la empresa DataSatax Enterprise. El problema es que para esta empresa no es el único producto que distribuye, puesto que tiene su propia solución para bases de datos no relacionales en la que facilitan su implementación en plataformas como AWS, Microsoft Azure o Google Cloud Plataform, mientras que para implantar Apache Cassandra solo se tienen facilidades si se utiliza Azure.

En conclusión, se puede decir que MongoDB tiene todas sus soluciones más centralizadas y permite acoger varios precios de implementación, se ha escogido AWS porque bajo las mismas condiciones es el más barato. Sin embargo, para implantar Apache Cassandra solo se puede elegir los precios de DataStax Enterprise y Microsoft Azure si se quiere tener una facilidad de implementación y que tanto la solución como la infraestructura, sea totalmente administrada por estas empresas.

8. Conclusión del proyecto

Al principio del proyecto se marcó el objetivo de saber responder a la pregunta **"¿qué base de datos es mejor?"**, después de una introducción a las bases de datos NoSQL y sus diferencias con las RDBMS se ha sacado la conclusión de que no hay una respuesta a esta pregunta, todo depende de las prioridades que se tengan y de su uso concreto.

Cumpliendo el teorema de CAP, siempre va a haber un tipo de base de datos que aporte algo diferente que otra de otro tipo distinto.

Si se priorizan las relaciones entre los datos, y sobre todo asegurar el aislamiento, la atomicidad, la consistencia y la durabilidad en las transacciones, y una disponibilidad constante de la base datos sacrificando que esta no sea escalable horizontalmente, se deberá utilizar una base de datos relacional.

Por el contrario, si las relaciones entre los datos no son prioritarias y lo que se quiere es realizar consultas de gran cantidad de datos de la manera más rápida posible, o tener la posibilidad de un esquema flexible para una aplicación orientada a objetos y que nuestra arquitectura permita una escalabilidad tanto vertical como horizontal, la mejor opción es implementar una base de datos NoSQL.

Después de realizar comparaciones a nivel técnico, comparaciones de rendimiento en aplicaciones y comparaciones presupuestarias de implantación de MongoDB y Apache Cassandra, la conclusión es la misma que la anterior, no se podría decir cuál de las dos bases de datos NoSQL más utilizadas en la actualidad es mejor.

La base en sus diferencias está en su arquitectura y en cómo se modelan los datos.

MongoDB proporciona un esquema flexible y un nivel de consistencia muy alto, sacrificando la disponibilidad ante caídas de los nodos durante los periodos de decisión del nuevo nodo primario.

Apache Cassandra proporciona una mayor rapidez de lectura de grandes cantidades de datos a través de las claves primarias de los registros y asegura estar disponible siempre, debido a que si cae un nodo del anillo el resto está funcionando y todos son considerados como nodos primarios. Pero sacrifica la consistencia de sus datos, puesto que cada nodo no contiene todo el conjunto de datos que posee el clúster, sino la parte que le ha sido asignada y otra parte que se le replica de otros nodos en base al factor de replicación, aparte, su esquema aunque se puede modificar, no proporciona una gran flexibilidad.

Esas serían las conclusiones más generales, cada solución es mejor para unas utilidades determinadas.

Lo que sí se puede determinar es cuál base de datos es mejor para aplicarse a BIBLIOTEQ.

En base a todo el estudio realizado de ambas aplicaciones la mejor es MongoDB. Los motivos son los siguientes:

- La flexibilidad que aportan los documentos a la hora de almacenar datos, sin duda es la opción más cómoda para una aplicación orientada a objetos, el modelo se adapta a los datos y no al revés. Con Apache Cassandra se tienen que definir un modelo e intentar que los datos se adapten en mayor medida a este modelo, a pesar de haberlo conseguido es algo más costoso de implementar.
- En MongoDB se permiten distintas opciones para relacionar documentos, sin embargo, en Apache Cassandra no se permiten las relaciones, por lo que la relación usuario-reserva y publicación-reserva es difícil de administrar.
- Las transacciones están mejor definidas, proporciona los típicos métodos para confirmar transacciones o abortarlas, y poder volver a un principio reintentando la transacción en caso de que se produzcan errores.
- Se pueden hacer búsquedas por cualquier campo de un documento, en Apache Cassandra también existe la posibilidad de hacerlo, pero se reduce algo el rendimiento y si el sistema se vuelve demasiado grande pone en riesgo la arquitectura con muchos índices secundarios.
- La utilización de índices aporta un incremento del rendimiento en las consultas de los datos, aparte se tiene distintos tipos índices que se pueden aplicar a cada campo de un documento.
- Según las pruebas de rendimiento de los controladores parecen más optimizados en MongoDB que en Apache Cassandra. Si se ven las comparaciones de los tiempos de ejecución, se puede ver la notoria diferencia de uno con otro.

Una vez explicados los motivos de la elección de MongoDB hay que elegir una de las tres opciones dadas en los presupuestos. Sin duda la opción de implementarlo On Premise en mi opinión debería de estar descartada, toda la base de la arquitectura de MongoDB está hecha para que sea implementado en la nube y dé la posibilidad de escalar horizontalmente y consumir el número justo de recursos.

Para concluir, se ha visto un caso real de una solución en la que es bueno el uso de MongoDB, BIBLIOTEQ. Otras soluciones para las que se utiliza esta base de datos son para el análisis de datos en redes sociales o en aplicaciones móviles los cuales su desarrollo es orientado a objectos.

Pero que Cassandra no sea la mejor opción utilizarla en BIBLIOTEQ no significa que no tenga casos de uso en los que puede ser igual o mejor MongoDB:

- Actualmente Cassandra es utilizada por Facebook para la implementación de su bandeja de entrada.
- Es muy utilizada en análisis de datos de redes sociales o incluso en recolectar y manejar grandes cantidades de datos provenientes de sensores.
- Búsquedas en catálogos de tiendas online e implementación de carritos de compra. Empresas como eBay la utilizan.
- Rastrear y monitorizar actividades en tiempo real.

9. Referencias Bibliográficas

[1] A. Castro, J. S. González y M. Callejas, "Utilidad y funcionamiento de las bases de datos NoSQL", *Facultad de Ingeniería*, vol. 21, no. 33, pp. 21-32, diciembre 2012.

[2] D. MacCreary y A. Kelly. *Making Sense of NoSQL: A guide for managers and the rest of us.* Manning Publications, 2013.

[3] R. Fernández. (2017, May 31). NoSQL: clasificación de las bases de datos según el teorema CAP. [Online]. Available: https://www.genbeta.com/desarrollo/nosql-clasificacion-de-las-bases-de-datos-segun-el-teorema-cap

[4] R. Herranz, "Bases de datos NoSQL: Arquitectura y ejemplos de aplicación". Proyecto fin de carrera, Universidad Carlos III. Madrid, 2014.

[5] D. Sullivan. *NoSQL for mere mortals,* Addison-Wesley Profesionals, 2015.

[6] Javier de PANDORAFMS. (2015, Nov 18). NoSQL vs SQL; principales diferencias y cuándo elegir cada una de ellas. [Online]. Available: https://pandorafms.com/blog/es/nosql-vs-sql-diferencias-y-cuando-elegir-cada-una/

[7] Slashmobility. (2016, Dic 13). NoSQL vs SQL: ¿Cuál elegir? [Online]. Available: https://slashmobility.com/blog/2016/12/nosql-vs-sql-cual-elegir/

[8] Blog acens. (2014, Feb 28). Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar. [Online]. Available: https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf

[9] Tecnologías información. (2018). Bases de datos documental. [Online]. Available: https://www.tecnologias-informacion.com/documentosbd.html

[10] J. J. López, "Cassandra NoSQL". Proyecto fin de ciclo G.S. 2018.

[11] B. Reded Tinoco. (2018, Feb 08). Configuración de la arquitectura base de Apache Cassandra. [Online]. Available: https://www.ibm.com/developerworks/ssa/library/ba-set-up-apache-cassandra-architecture/index.html

[12] MongoDB, Inc. (2019, May 31). Manual de MongoDB 4.0 versión online. [Online]. Available: https://docs.mongodb.com/manual/

[13] Baeldung. (2018, Oct 26). Una guía para Cassandra con Java. [Online]. Available: https://www.baeldung.com/cassandra-with-java

[14] DataStax, Inc. (2019, Jun 11). CQL for the DataStax Distribution of Apache Cassandra, DataStax documentation. [Online]. Available: https://docs.datastax.com/en/ddaccql/doc/

[15] C. Kalantzis. (2016, Feb 1). Astyanax – Retirnado a un viejo amigo. [Online]. Available: https://medium.com/netflix-techblog/astyanax-retiring-an-old-friend-6cca1de9ac4

[16] DataStax, Inc. (2019, Abr 09). Controlador Java Datastax para Apache Cassandra. [Online] Available: https://docs.datastax.com/en/developer/java-driver/3.6/#datastax-java-driver-for-apache-cassandra

[17] Karthikprasad13 (2016, Jul 05). Kundera wiki. [Online] Available: https://github.com/Impetus/Kundera/wiki

[18] Comunicación Future Space, (2017, Dic 01). Future Bites. [Online] Available: https://bites.futurespace.es/teorema-cap-2/

[19] Blog AWS. ¿Qué es una base de datos clave-valor? [Online] Available: https://aws.amazon.com/es/nosql/key-value/

ANEXO: Manual de usuario de BIBLIOTEQ

Inicio de sesión

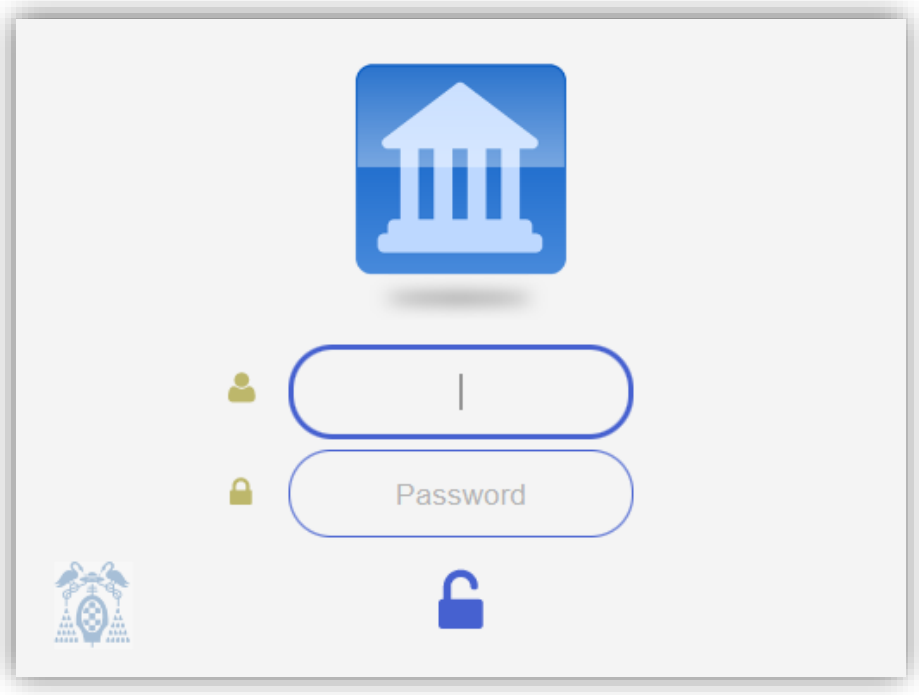


Ilustración 62: Pantalla de inicio de sesión en BIBLIOTEQ

El primer paso en la aplicación después de haberse dado de alta un usuario con su contraseña es iniciar sesión. A partir de aquí se dan dos casuísticas distintas:

- Que el usuario que inicia sesión sea “Administrador”.
- Que el usuario que inicia sesión sea “Alumno”.

Cargo administrador

Ilustración 63: Pantalla de un administrador en BIBLIOTEQ

En esta imagen se pueden ver las cinco acciones principales que puede hacer un administrador:

Dar de alta a un usuario

Para dar de alta un usuario, se tiene que seleccionar el cargo que se desea que tenga y rellenar sus datos, una vez hecho eso se pulsa sobre el icono de color azul y éste se dará de alta de manera automática en el sistema.

Ilustración 64: Pantalla de alta de usuarios en BIBLIOTEQ

Dar de baja a un usuario

Para dar de baja a un usuario primero hay que buscarlo en el sistema.

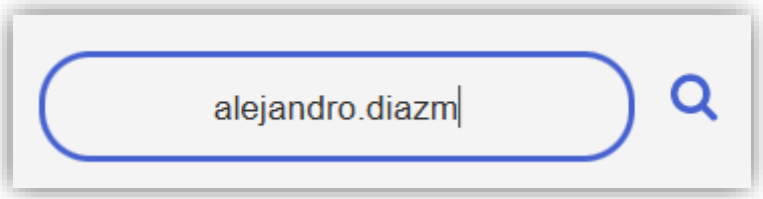


Ilustración 65: Barra de búsqueda de usuarios en BIBLIOTEQ

Tras escribir el nombre de usuario simplemente se pulsa sobre la lupa y la aplicación te redirige a otra pantalla donde muestra los datos principales del usuario:

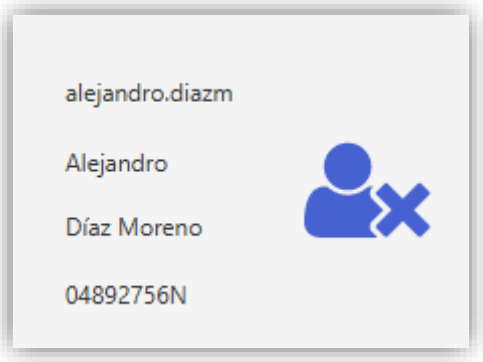


Ilustración 66: Pantalla de bajas de usuarios en BIBLIOTEQ

Para eliminar el usuario solo hay que pulsar sobre el icono y se borrará, a no ser que tenga asignada una reserva no finalizada, una vez borrado aparecerá el siguiente mensaje de confirmación

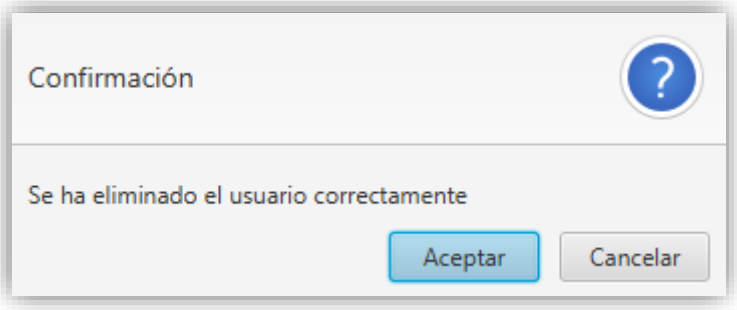


Ilustración 67: Confirmación de baja de un usuario en BIBLIOTEQ

Dar de alta una publicación

Las publicaciones que se pueden dar de alta son de tres tipos: libros, revistas y proyectos.

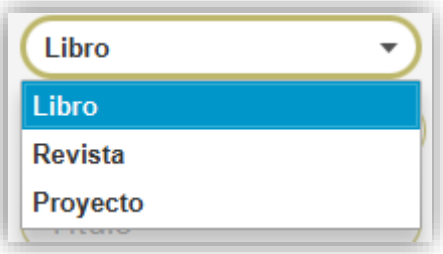


Ilustración 68: Selección de tipos de publicación

Tras seleccionar el tipo de publicación, se pasa a rellenar los datos de la pantalla, pero hay que tener en cuenta cómo añadir los autores y cómo añadir la portada.

Para añadir los autores habrá que escribir su nombre y sus apellidos en la zona de la pantalla que indica autores y pulsar sobre el icono:

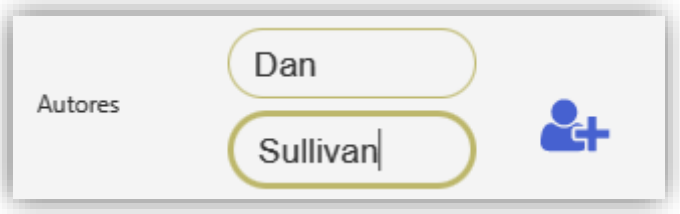


Ilustración 69: Asignar un autor a una publicación

Una vez se rellenan los campos y se añade el autor, estos campos se vacían para que en el caso de que haya varios autores puedan seguir añadiéndose sin ningún problema.

Para añadir la portada habrá que pulsar sobre el botón: 

Tras pulsar este botón se abrirá en el equipo una pantalla de nuestro gestor de archivos en el que se tendrá que buscar una imagen de la portada y añadirla

La pantalla resultante de alta de publicaciones una vez rellenados todos los datos, sería la siguiente:

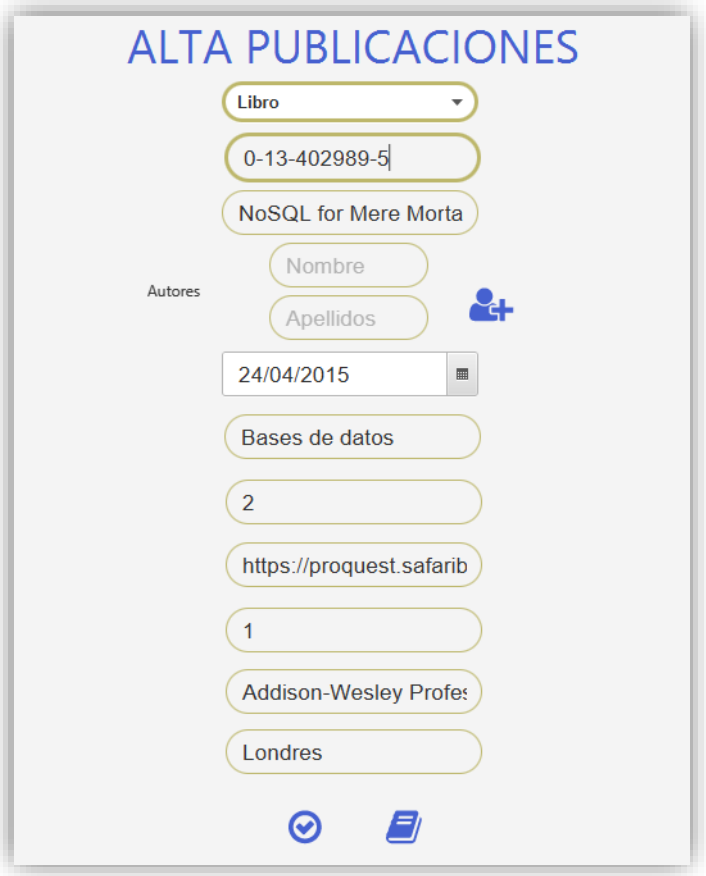


Ilustración 70: Pantalla de alta de publicaciones en BIBLIOTEQ

Para finalizar, una vez se ha añadido la portada, los autores y completado todos los campos, se pulsa en el botón: 

Dar de baja una publicación

El funcionamiento es el mismo que el visto previamente con los usuarios, se busca una publicación por su título y aparece una pantalla con información acerca de la publicación.

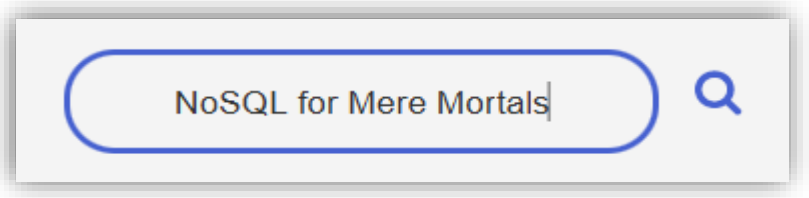


Ilustración 71: Barra de búsqueda de publicaciones por título

Para eliminarla solo habría que pulsar el icono azul con forma de cruz.

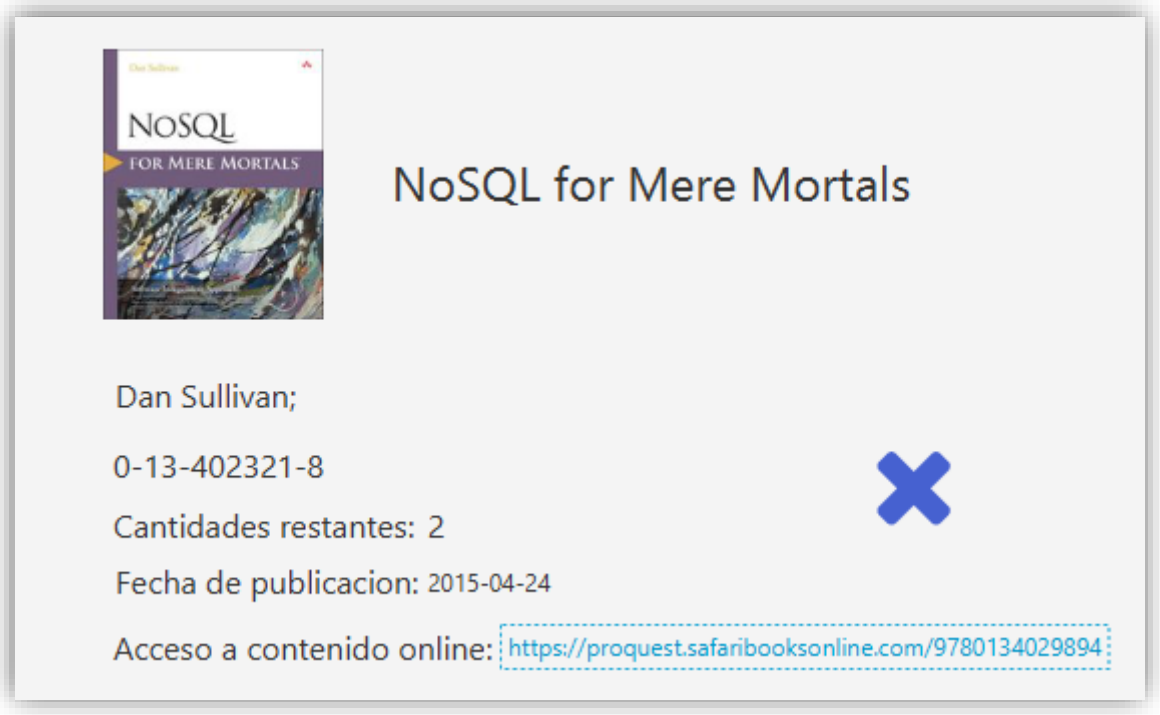


Ilustración 72: Pantalla de bajas de publicaciones en BIBLIOTEQ

Gestión de reservas

La gestión de las reservas se basa en gestionar las devoluciones de las publicaciones. Primeramente, hay que buscar la reserva en el sistema, para la aplicación en Cassandra se busca a través del id de la reserva que se emite en la factura de ésta. Pero en MongoDB se busca a través del NIF y del título de la publicación.

Ilustración 73: Barra de búsqueda de reservas con Cassandra

Buscador de reservas en MongoDB:

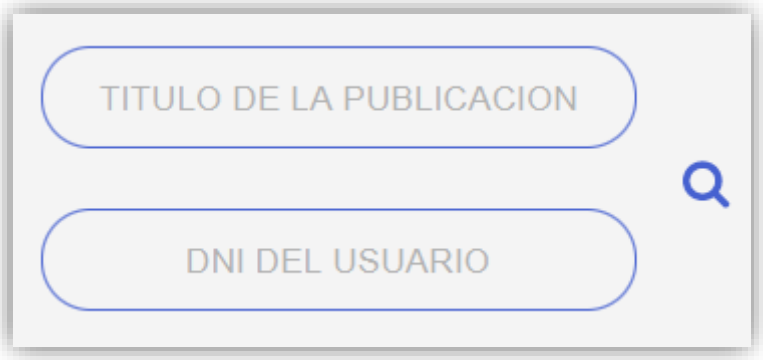


Ilustración 74: Barra de búsqueda de reservas con MongoDB

Una vez se ha encontrado el resultado de la reserva, se puede finalizar pulsando sobre el icono y automáticamente se cambia el valor de la fecha de devolución prevista por la real, que es cuando se ha devuelto la publicación en la biblioteca, también se cambia el estado de la reserva a finalizada. Como la reserva se finaliza tras la devolución en persona del libro, el sistema automáticamente le suma uno a la cantidad de ejemplares de la publicación y así poder ser reservada de nuevo.

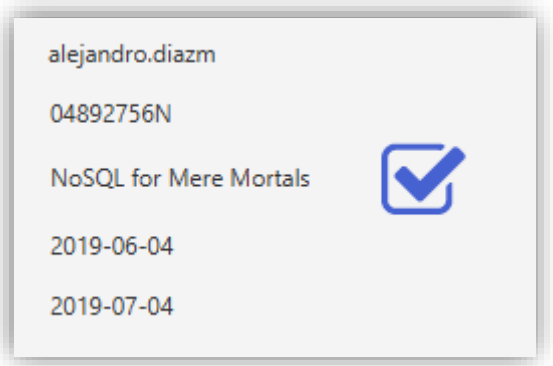


Ilustración 75: Pantalla de finalización de reservas en BIBLIOTEQ

Cargo alumno

Un usuario con cargo alumno puede realizar las siguientes acciones: buscar publicaciones, reservar publicaciones y acceder a la publicación online siempre y cuando la Universidad la tuviera disponible.

Ilustración 76: Pantalla de búsqueda de publicaciones para alumnos en BIBLIOTEQ

Como se puede ver en la imagen, un usuario puede rellenar varios campos a la vez, para que sus búsquedas sean más exactas.

Un ejemplo para la publicación dada de alta en pasos anteriores:

Ilustración 77: Ejemplo de búsqueda en BIBLIOTEQ

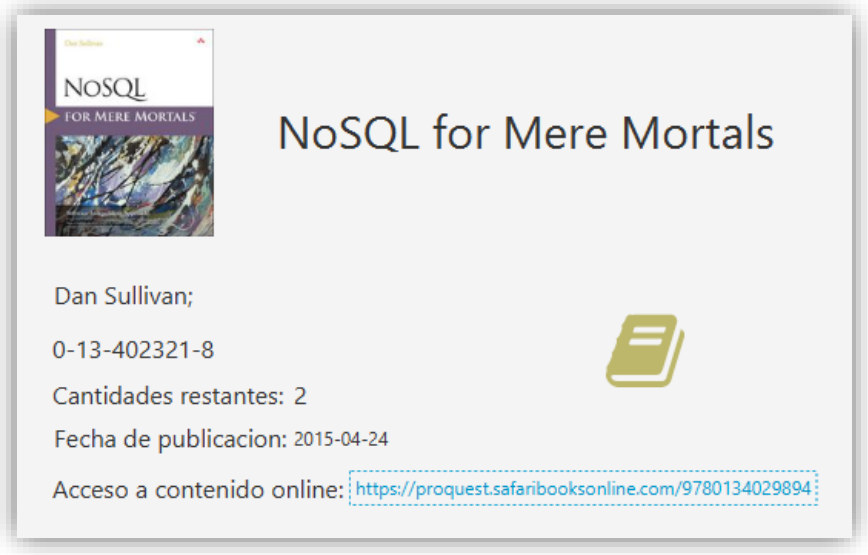


Ilustración 78: Resultado de una búsqueda en BIBLIOTEQ

Para reservar la publicación hay pulsar sobre el libro dorado y aparecerá el siguiente mensaje.

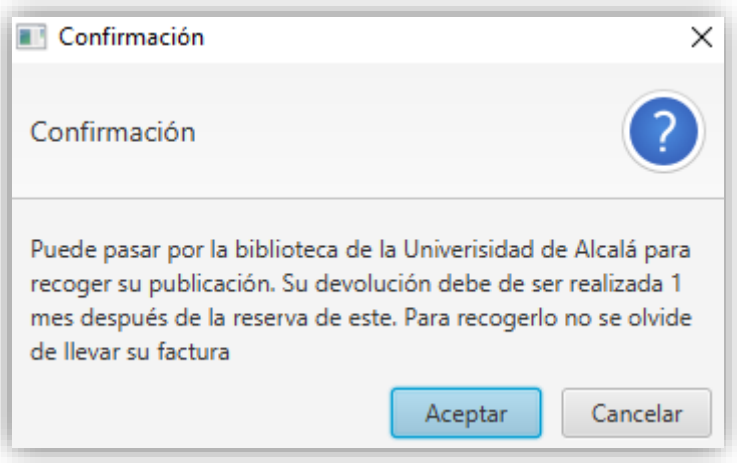


Ilustración 79: Mensaje de confirmación de una reserva en BIBLIOTEQ

Una vez reservado también se descarga una factura, si BIBLIOTEQ está implementada con Cassandra tendrá un id de reserva.

```
Este documento deberá ser presentado en el momento de recoger o devolver la publicación reservada
DATOS DE LA RESERVA:
ID DE LA RSERVA: reservas_7
NIF: 04892756N
NOMBRE DE USUARIO: alejandro.diazm
TITULO DE LA PUBLICACION: NoSQL for Mere Mortals
FECHA DE LA RESERVA: 2019-06-04
FECHA DE DEVOLUCION: 2019-07-04
(LA PUBLICACION PUEDE SER DEVUELTA CON ANTERIORIDAD PERO NUNCA DESPUES DE HABERSE CUMPLIDO EL PLAZO O HABRA UNA PENALIZACION)
```

Ilustración 80: Ejemplo de factura en BIBLIOTEQ

Para acceder a la publicación en versión digital solo hay que pulsar en el hipervínculo y se abrirá una ventana en el navegador siempre y cuando se esté dentro de la red de la universidad.

Validaciones de la aplicación

En este apartado se van a exponer las validaciones que tiene el sistema para que no se produzcan inconsistencias.

- Si en la pantalla de inicio de sesión se deja alguna de las credenciales en blanco.
- Por otro lado, si se intenta autentificar con un usuario o contraseña incorrectos.
- Cuando se intenta dar de alta un usuario y coincide el nombre de usuario o el NIF, o cuando se intenta dar de alta una misma publicación varias veces.
- Al intentar buscar un usuario o publicación que no existe para darle de baja aparecerá también un mensaje de error. También se mostrará al buscar una publicación para reservar o verla en formato digital.
- Tampoco se pueden gestionar reservas ya finalizadas.
- En el caso de que se intente borrar un usuario o publicación y tengan reservas pendientes de finalizar tampoco se pueden borrar.
- Por último, cuando se intenta reservar una publicación que tiene cero números de ejemplares disponibles aparecerá lo siguiente.

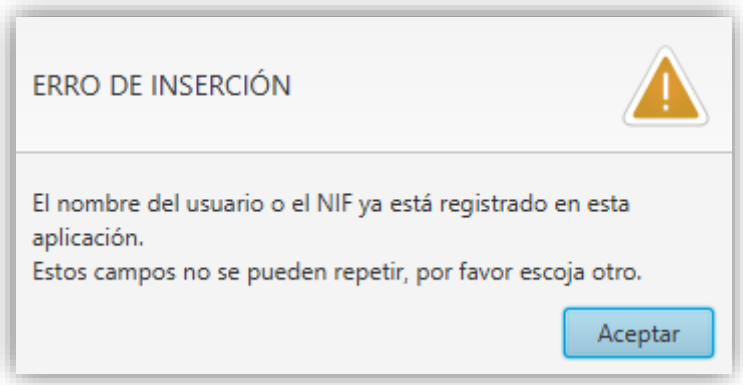


Ilustración 81: Ejemplo de validación en BIBLIOTEQ

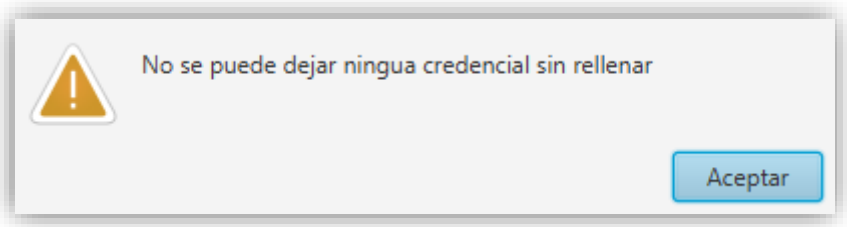


Ilustración 82: Ejemplo de validación en BIBLIOTEQ

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá