Universidad Nacional de La Plata

Facultad de Informática

Análisis de performance en diferentes bases de datos

Autores

Federico Di Claudio, Federico Mozzon

Palabras clave

NoSQL, SQL, DataBase, Performance, MySQL, Mongodb, Elasticsearch, Neo4J

Resumen

En el siguiente trabajo se presenta un análisis de tiempo de ejecución de distintos motores de bases de datos, tanto relacionales como no relacionales. Fueron elegidos ***MySQL***[1], ***Neo4J***[2], ***MongoDB***[3] y ***Elastic***[4]. Se realizará un CRUD (Create, Read, Update, Delete) de una entidad en cada uno y se evaluará con cuál se obtiene mejores desempeños, a su vez se contemplarán distintos escenarios para cada uno de los motores haciendo pruebas con una tabla con diez mil, cien mil y un millón de registros. Al final, se redactarán las conclusiones obtenidas de las pruebas.

Motivación

Durante la cursada se vieron distintos motores de bases de datos, tanto relacionales como no relaciones, y su conversión desde un modelo de base de datos a objetos y viceversa. También se mencionaron distintas formas de modelar los datos a persistir en una base de datos por ejemplo Mongo y Elastic utilizan un formato basado en documentos binarios con objetos embebidos, Neo4J utiliza grafos, componiendose de nodos y relaciones, y MySQL, posiblemente el más conocido de los tres, usa un modelo relacional basado en tablas.
Dado que sólo se vieron en profundidad dos de los motores mencionados previamente, éste trabajo surge con la intención de desarrollar los siguientes conceptos:

- Conocer las particularidades que posee un tipo de modelo basado en grafos.
- Interactuar con tecnologías mencionadas pero no abordadas como son Elastic y Neo4J.
- Distinguir las diferentes filosofías de mapeo que sugieren las distintas bases de datos.
- Analizar cuál de ellas proporciona un mejor rendimiento ante operaciones básicas y en distintas circunstancias, analizando las ventajas de cada una y teniendo en cuenta su filosofía de mapeo.

---

[1] https://www.mysql.com/
[2] https://neo4j.com/
[3] https://www.mongodb.com/
[4] https://www.elastic.co/es/products/elasticsearch

Desarrollo

El repositorio junto con las instrucciones sobre cómo levantar cada una de las pruebas se encuentra en el siguiente repositorio: Prueba de persistencia

Para llevar a cabo las distintas pruebas se desarrolló un conjunto de aplicaciones, utilizando el lenguaje ***Java***[5] y el framework ***Spring Data***[6]. También se utilizó contenedores ***Docker***[7], manejados mediante un ***Docker-Compose***, que nos permite correr uno por cada motor de base de datos independientemente.

El framework ***Spring Data*** nos permite de manera sencilla realizar operaciones sobre un motor de base de datos y posee adaptaciones para múltiples motores, entre ellos se utilizó las correspondientes bases de datos utilizadas: Neo4J, Elasticsearch, Mongo y MySQL. Ofrece realizar el mapeo mediante anotaciones de Java o mediante archivos ***XML***[8], siendo la primera la utilizada para el correspondiente mapeo del modelo utilizado.
Se desarrollaron cuatro aplicaciones independientes, una para cada motor de base de datos utilizado, con el fin de mantener la independencia entre las pruebas. Todas las aplicaciones tienen el mismo modelo el cual es parte del modelo empleado en la aplicación Bithub realizada durante la cursada, tomándose sólo las clases User y Commit y la relación entre estas. Todas las aplicaciones utilizan una misma estructura que se compone además del modelo de repositorios desarrollados según la implementación de Spring Data, un servicio[9] que se encarga de realizar las operaciones sobre los repositorios[10] y un conjunto de test que utilizan los servicios y toman los tiempos de cada operación.
La configuración requerida por Spring Data para funcionar se realizó mediante las *application propperties* de Java en el caso de Neo4J, MySQL y Elasticsearch y utilizando clases de Java en Mongo. Estas configuraciones tienen como objetivo conectar la aplicación con el motor de base de datos que corre sobre un docker, configurando los puertos, bases de datos, y su autenticación.

Con el objetivo de hacer de hacer más sencilla la prueba se escribió un docker-compose, que orquesta las configuraciones de cada contenedor los cuales manipulan los motores de bases de datos. Las imágenes utilizadas son en todos los casos imágenes oficiales disponibles en Docker Hub. Además se agregó ***Kibana***[11] para manejar los datos de Elasticsearch. No es el objetivo ejecutar a la vez todos los docker, para mantener lo mayor posible la independencia de las pruebas.
Para todos los motores de base de datos a excepción de Elastic, en el cual esto no es posible, se proporciona archivos ***JSON***[12] o ***CSV***[13], que contiene registros aleatorios listos para ser insertados y realizar las pruebas sobre estos.

---

[5] https://www.java.com/
[6] https://spring.io/projects/spring-data
[7] https://www.docker.com/
[8] https://developer.mozilla.org/es/docs/Web/XML/Introducci%C3%B3n_a_XML
[9] https://martinfowler.com/eaaCatalog/serviceLayer.html
[10] https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design
[11] https://www.elastic.co/es/products/kibana
[12] https://www.json.org/json-en.html
[13] https://en.wikipedia.org/wiki/Comma-separated_values

Pruebas

Las pruebas son las mismas para todos los motores y consisten en:

- Una inserción de una entidad usuario el cual posee un nombre, un email y un identificador el cual es generado de manera automática por cada uno de los motores
- Una actualización de una entidad usuario, al cual se le a cambiado su propiedad email. El usuario a editar ya fue cargado en la memoria con anterioridad, por lo que el tiempo requerido para tal operación es sólo la modificación de su campo y su nueva persistencia.
- El borrado de una entidad mediante su identificador.
- La búsqueda de una entidad User en particular mediante su propiedad email. Para agilizar los tiempos de búsqueda fue creado un índice mediante tal campo en cada BD.
- Obtener un listado de los commits de un usuario en particular mediante el email de este. El objetivo de esta prueba es ver cómo se comporta cada base de datos cuando tiene que recuperar objetos de más de un tipo relacionados.

Cada prueba se realizará sobre una misma base de datos en tres ocasiones que difieren en la cantidad de registros User y Commits que se tienen insertados la base de datos: diez mil, cien mil y un millón. El equipo utilizado se trata de una Lenovo X1 Carbon con un I7 de 5ta generación que dispone de 8GB de RAM, 256 de SSD en el disco.
Las pruebas consistieron en obtener el horario del sistema en el cual la operación comienza a ejecutarse y se vuelve a tomar una vez finalizada la operación, finalmente se realiza la diferencia entre ambos números, una consideración a tener en cuenta es que si bien son milésimas de segundo las que se agregan, esto le agrega tiempo al valor devuelto finalmente

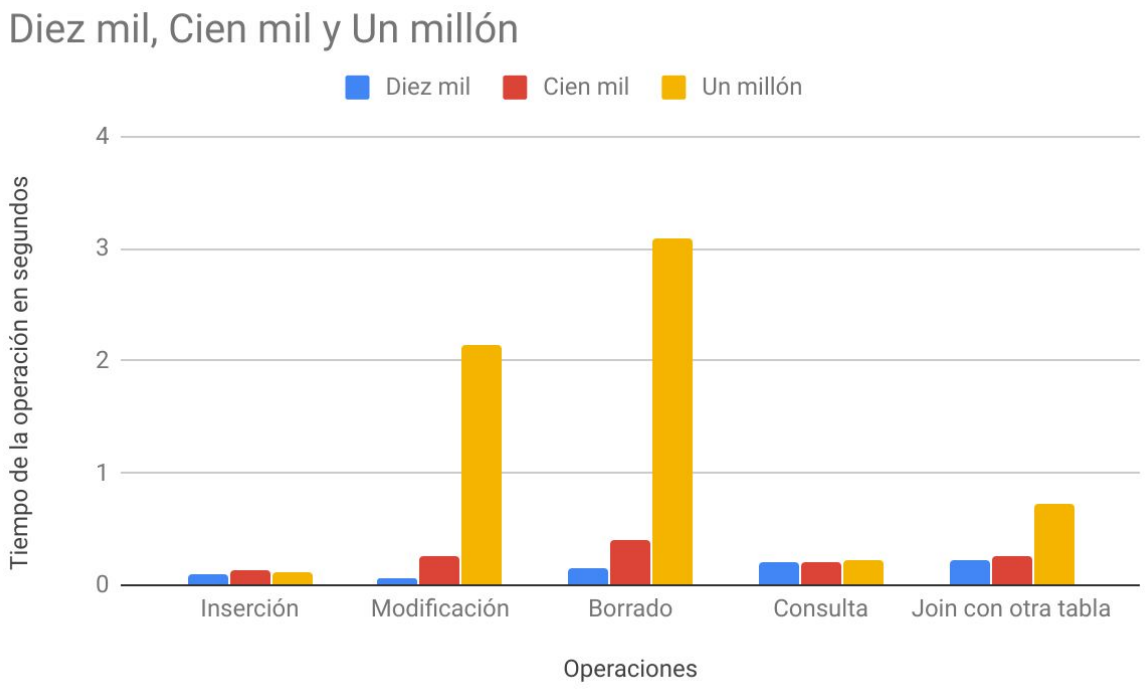
A continuación para cada una de las bases de datos se explica cómo implementa el manejo de datos, como fue realizado el mapeo, como fueron realizadas la pruebas y los resultados obtenidos de las pruebas.
En el Readme del proyecto en Github, se encuentra explicado cómo se hizo la inserción de los datos aleatorios en cada base de datos y los links para descargarlos.

MySQL

MySQL es el motor de base de datos más conocido y el de mayor antigüedad de entre los seleccionados, de tipo relacional, que guarda la información en tablas relacionadas mediante claves foráneas.
El mapeo consiste en tablas separadas, para User y Commit, y en mantener el identificador del usuario autor del commit en el registro correspondiente al commit. Esto requiere que para la última prueba, la que involucra commits, se realice un producto entre ambas tablas.
La inserción de los los registros dados para realizar las pruebas requieren tiempo. Aproximadamente la inserción del millón nos requirió un poco más de tres horas.
Entre los registros cargados, existen algunos que fueron generados manualmente, esto para que asegurarnos de que sean únicos y están en todos los archivos a cargar. El usuario con el mail “sancheztueardasgary@hotmail.com” se sabe que no tiene commits, por lo que va a ser utilizado para las pruebas que no los involucren, y el usuario con el mail "kennethmyers@yahoo.com" tiene un commit a recuperar.

Se representa mediante tabla y gráfico los resultados obtenidos al realizar las distintas pruebas, representados en segundos:

	Inserción	Modificación	Borrado	Consulta	Join con otra tabla
Diez mil	0,092807072	0,070103153	0,145559153	0,199134184	0,21519847
Cien mil	0,139320864	0,257713287	0,404655347	0,205647344	0,25006714
Un millón	0,120574418	2,147421204	3,08971555	0,230372479	0,733308856

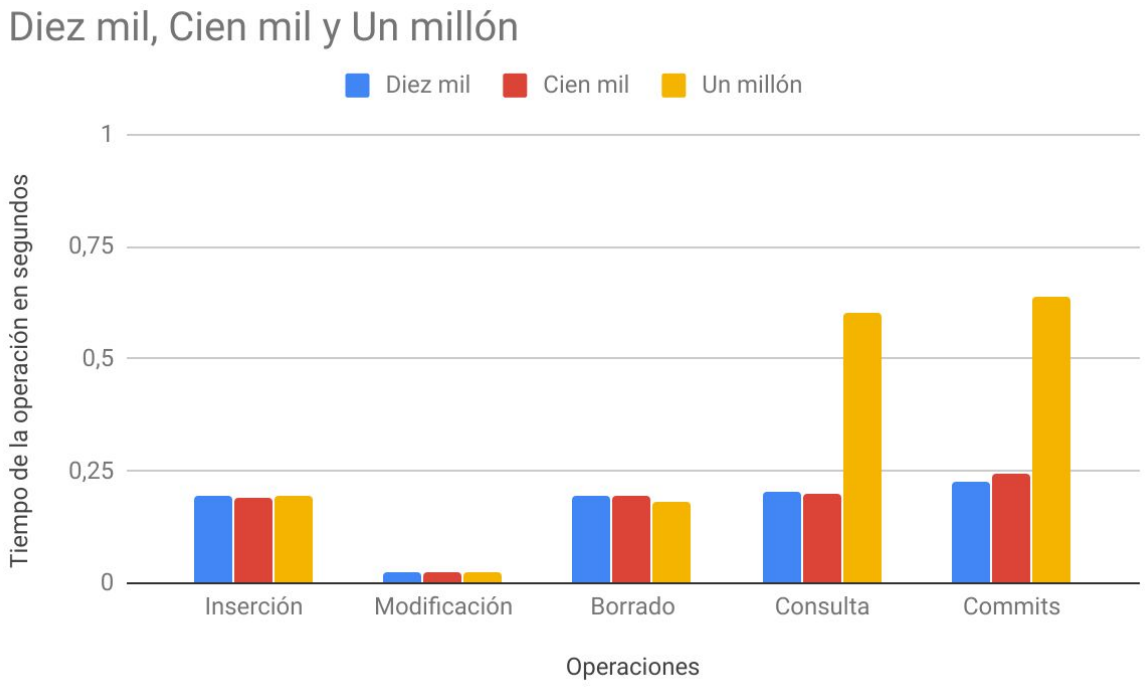


Mongodb

MongoDB es un sistema de base de datos basado en documentos, a diferencia de MySQL donde cada entidad es almacenada en una fila de una tabla, aquí se almacenan en texto en formato BSON (especificación similar a JSON). Los documentos son almacenados en colecciones, aunque esta no obliga a los documentos a declarar propiedades específicas.
Prioriza la performance sobre la consistencia utilizando objetos embebidos. Esto si bien puede traer problemas de consistencia por que podría generarnos replicación de los datos, promete mejores tiempos en las consultas.No existe el concepto de clave foránea, aunque puede simularse mediante DBRef.
El mapeo de nuestra aplicación se propuso respetar dicha filosofía de MongoDB utilizando objetos embebidos, los commits se guardan en el mismo documento que el usuario, no existiendo una colección aparte.
Entre los registros que fueron dados existe uno con mail "earnest.klocko@hotmail.com", disponible en todos los archivos a importar, se sugiere el uso de este para realizar las pruebas.

Se representa mediante tabla y gráfico los resultados obtenidos al realizar las distintas pruebas, representados en segundos:

	Inserción	Modificación	Borrado	Consulta	Commits
Diez mil	0,194359192	0,024378965	0,195123038	0,203110525	0,227258208
Cien mil	0,189756913	0,023118349	0,194333247	0,200549747	0,245506103
Un millón	0,194733162	0,024781402	0,180030557	0,602131868	0,637546714

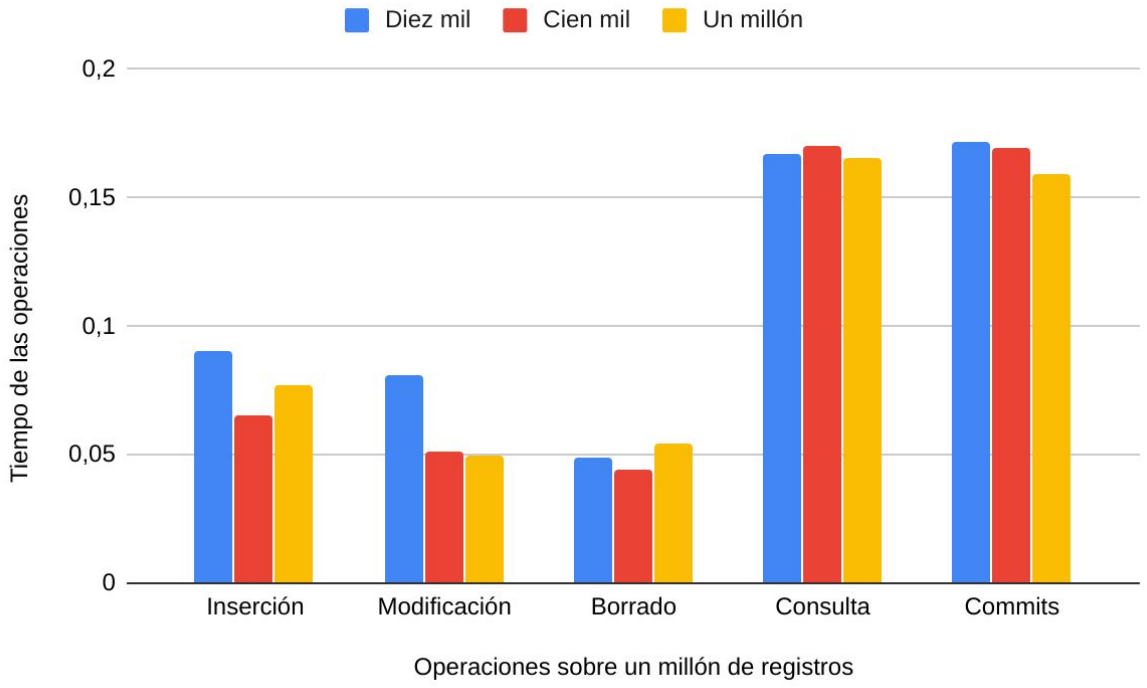


Elasticsearch

Elasticsearch, al igual que MongoDB, se basa en documentos, estos de tipo JSON, los cuales son almacenados en índices. Esta implementado como si de una APIRest se tratara, por lo que la forma de acceder y modificar los datos de un índice es mediante peticiones HTTP. Es de código abierto bajo licencia apache.
De forma similar a MongoDB, prioriza la performance sobre la consistencia en los datos, utilizando por defecto objetos embebidos.
El mapeo de nuestra aplicación se realizó con el objetivo de respetar dicha filosofía, por lo que los commits son guardados en una lista dentro del documento creado para representar un usuario.
Para facilitar las pruebas, el generador que se provee, siempre crea dos usuarios no aleatorios: uno con el mail "fedemozzon@gmail.com", el cual no posee commits y tiene como objetivo realizar sobre él las pruebas que no lo requieran; y otro usuario con el mail "fede@gmail.com" que posee dos commits, para las pruebas que sí necesitan de commits.
Cabe aclarar que a diferencia de las anteriores bases de datos en este caso no es necesario crear un índice para mejorar los tiempos de búsqueda de un usuario por su mail, estos se crean automáticamente en Elasticsearch.

Se representa mediante tabla y gráfico los resultados obtenidos al realizar las distintas pruebas, representados en segundos:

	Inserción	Modificación	Borrado	Consulta	Commits
Diez mil	0,090248026	0,080794044	0,048861275	0,167244001	0,171796498
Cien mil	0,065371162	0,050989994	0,044091403	0,170221356	0,16967595
Un millón	0,077226452	0,049423706	0,054049233	0,165744649	0,159083108

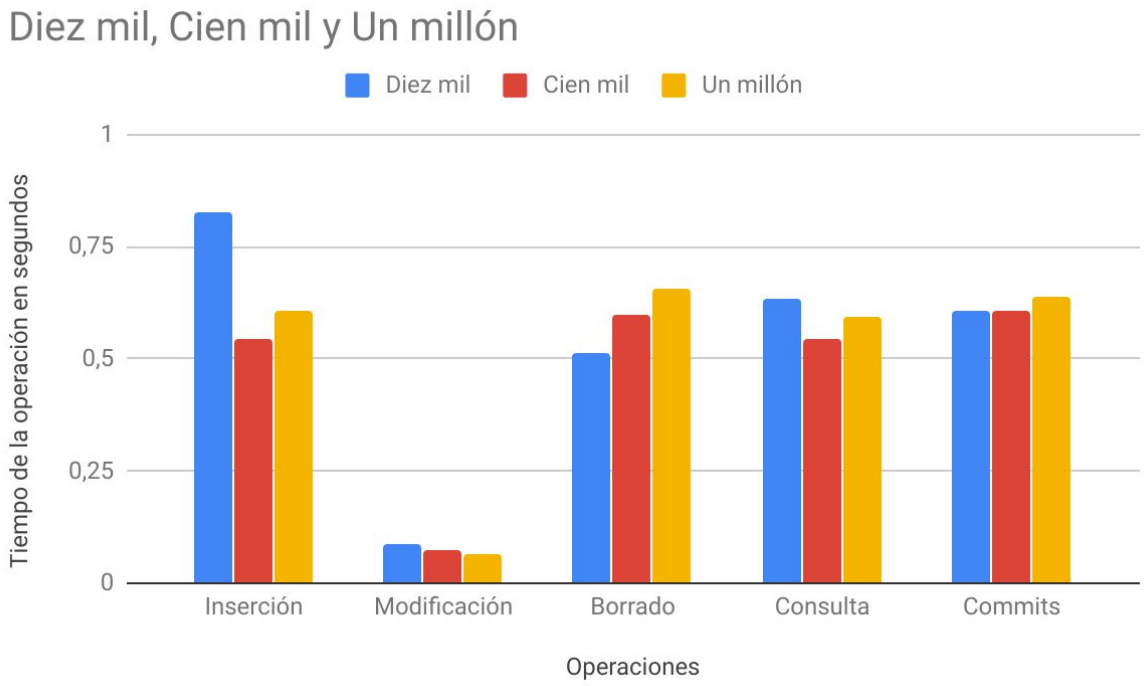


Neo4J

Esta base de datos, también no relacional, se basa en grafos, cada entidad guardada es representada por un nodo del mismo y las relaciones entre tales entidades mediante un aristas. Cada nodo y arista posee además de sus propiedades o campos de tipo clave valor, una etiqueta o label que especifica el tipo de entidad o relación que se trata.
El mapeo en nuestra aplicación, crea nodos con la etiqueta user o commit según corresponda, y cada usuario se conecta mediante relaciones con etiqueta commits a sus respectivos commits, mientras que tales commits también conocen a su autor.Para simplificar la prueba, en todos los paquetes de archivos a importar se encuentra un usuario con el mail "sancheztueardasgary@hotmail.com" el cual se propone utilizarlo para las consultas más básicas que no involucran commits y otro usuario con mail "kennethmyers@yahoo.com" que si posee un listado de commits a recuperar.

Se representa mediante tabla y gráfico los resultados obtenidos al realizar las distintas pruebas, representados en segundos:

	Inserción	Modificación	Borrado	Consulta	Commits
Diez mil	0,826157504	0,085369619	0,512970141	0,635050117	0,608733805
Cien mil	0,54569713	0,072732147	0,596626707	0,545041807	0,608372491
Un millón	0,607303451	0,062747698	0,654200427	0,591397171	0,637552382



Conclusiones

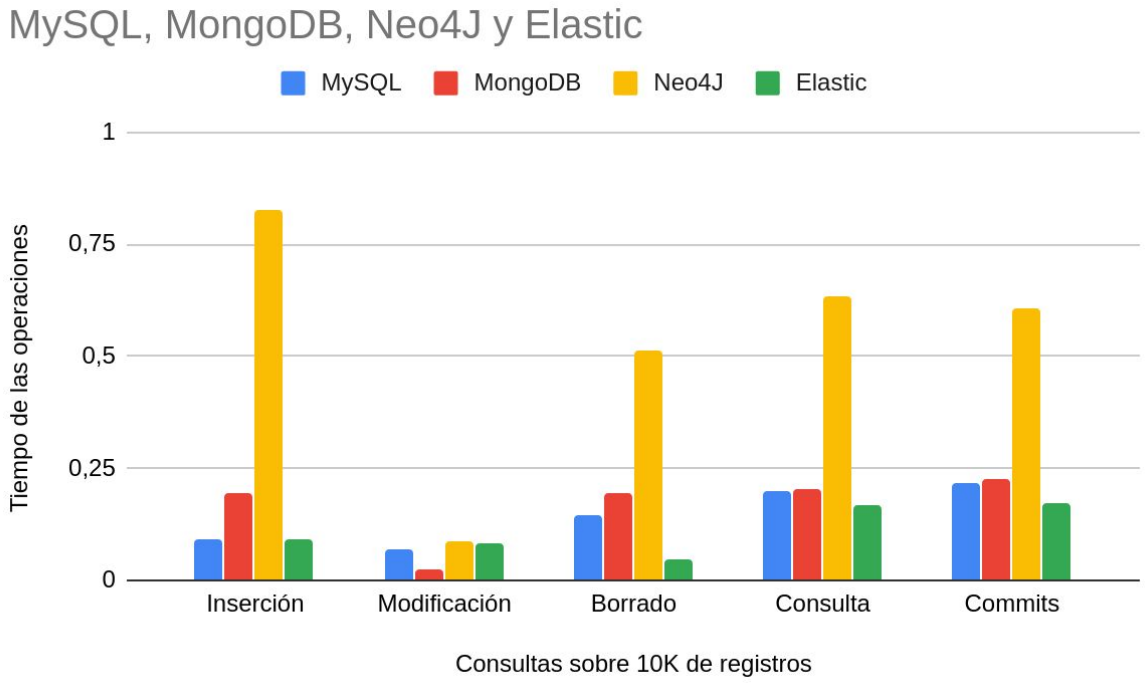
Se redactan a continuación una serie de conclusiones y sensaciones obtenidas a lo largo del desarrollo de la prueba.

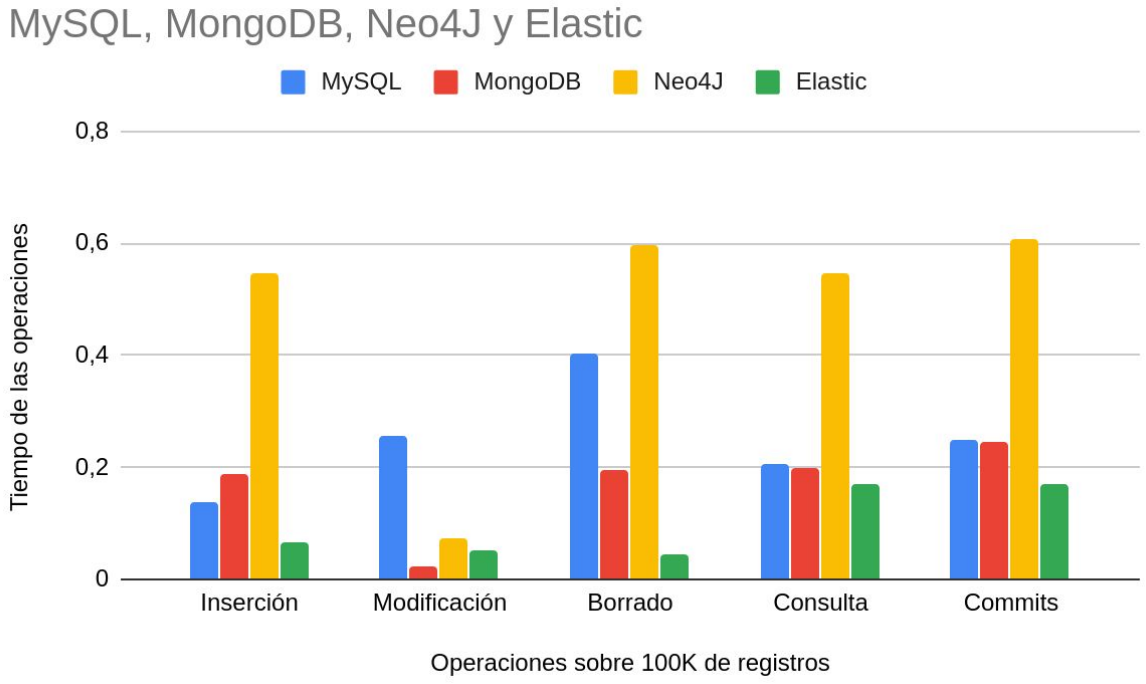
En cuanto al desarrollo el armado de las consultas y demás cosas referidas a la interacción con la base de datos, utilizar el framework Spring Data simplifica en gran medida esto, aunque su configuración y puesta en funcionamiento para cada base de datos requiere tiempo, nos encontramos con muchos errores en esta etapa y no siempre fue sencillo solucionarlo. Existió una gran diferencia en este aspecto con Neo4J, su implementación fue mucho más sencilla que con las otras bases de datos.

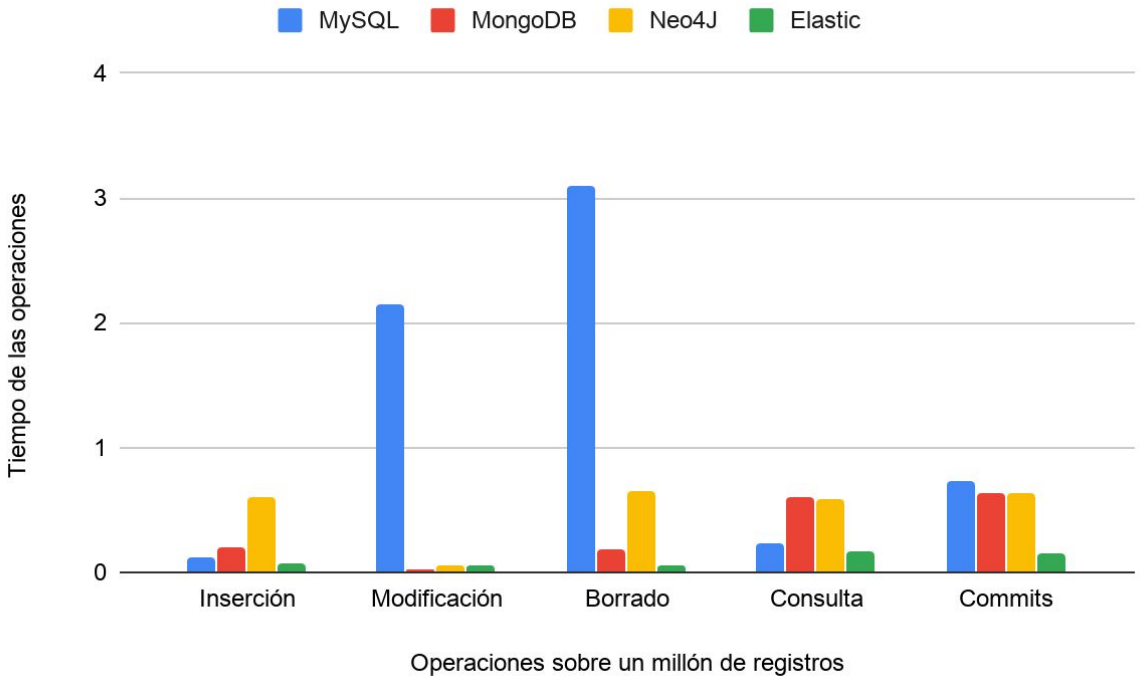
La inserción de datos en grandes cantidades para realizar las pruebas fue rápida tanto en Mongo como en Neo4J, no solo por los tiempos requeridos, en tan solo segundos se insertaron un millón de entidades, sino por el formato que soportan para ser representados, JSON y CSV respectivamente. Si bien MySQL cumple con este último aspecto, pudiendo insertar desde un archivo CSV, los tiempos de inserción fueron muy altos, más de tres horas para un millón de registros, que se contrasta mucho comparado con los segundos de las anteriormente mencionadas. En cuanto a Elasticsearch, el no poder ser posible exportar e importar los datos a

archivos con facilidad (tiene que definir un propio formato llamado bulk) la inserción de datos aleatorios se debe hacer en el momento y los tiempos de carga no cercanos a los obtenidos con MySQL.

Los siguientes gráficos muestran, para dada una cierta cantidad de datos, los tiempos que requirió cada base de datos para realizar una operación:







En la inserción de una entidad destaca Elasticsearch, con tiempos muy superiores a las otras bases de datos. En cuanto el contexto dependiendo de la cantidad de registros, ninguna base de datos perdió rendimiento en la inserción a medida que estos crecían, de hecho, algunas como Elasticsearch o Neo4J, lo mejoraron. De esta forma podemos concluir que la cantidad de registros ya existen en la base de datos no es un dato de importancia en cuanto al rendimiento de la inserción. El peor rendimiento lo obtuvimos con Neo4J, con gran diferencia sobre el resto.

En la modificación podemos concluir, que al igual que con la inserción, la cantidad de entidades ya existentes en la base de datos no modifica sus tiempos, a excepción de MySQL donde sí se ve una pérdida de rendimiento muy importante, sobre todo con un millón de registros, además de que con menos registros queda por detrás de las otras bases de datos. El que mejores tiempos obtuvo fue MongoDB, si bien Elasticsearch también está basado en documentos, creemos que tiene un tiempo levemente superior por la mantención de los índices que este genera y mantiene automáticamente.

En el borrado Elasticsearch obtiene mejores tiempos por amplia diferencia, que se agranda si la cantidad de registros aumenta, sobre todo porque también aumenta el tiempo en la otras bases de datos. Tanto en Mongo como en Neo4J, aunque en este último los tiempos de borrado sean más altos, estos se mantienen estables con respecto a la cantidad de registros, lo que nos parece una clara ventaja ante MySQL donde se incrementan en gran medida.

Los tiempos de búsqueda de un usuario por un mail fueron parejos en las distintas bases de datos a excepción de Neo4J que tuvo un tiempo mucho mayor con pocos registros, aunque se igualó luego en mas cantidad. Nos sorprendió el rendimiento de búsqueda de MySQL, bastante mejor que otras, aunque aumenta muchísimo cuando tiene que hacer el producto para obtener los commits de un usuario, las otras bases de datos en esta última mantienen mantienen sus tiempos, principalmente por mantener los objetos embebidos o utilizar relaciones. La que mejor rendimiento obtuvo tanto para la consulta sencilla como para obtener los commits fue Elasticsearch, esto es más notable cuando se consulta sobre un millón de commits, donde el tiempo requerido es menor a la mitad que Mongo o Neo4J.

Como conclusión final de Neo4J podemos decir que si bien los tiempos fueron muy superiores en comparación con poca cantidad de registros, estos se mantuvieron estables a medida que la cantidad de entidades guardadas aumentaba, lo que resultó en un buen tiempo, incluso prácticamente a Mongo en la consulta por Commits con un millón de entidades y teniendo en cuenta que Mongo guarda los commits embebidos en el mismo documento, mientras que Neo4J utiliza relaciones.

Mongo obtiene buenos tiempos en casi todas las circunstancias y al mantener los commits embebidos dentro del usuario, los tiempos de recuperación de estos son los mismos que una consulta solo sobre usuarios. Como contra a medida que la cantidad de documentos aumenta los tiempos también lo hacen, aunque no tanto como MySQL.

MySQL tiene buenos tiempos con pocos registros pero cuando estos aumentan a 100k ya se puede notar una diferencia, mas si se trata de un millon, no tan clara en la inserción, pero si en las demás operaciones. Como los datos, al recuperar los commits, se encuentran en dos tablas, realizar el producto entre estas requiere un tiempo más que los objetos embebidos o las relaciones de Neo4J.

Finalmente, Elasticsearch es la que ha mostrado mejores rendimientos en todos las operaciones probadas y bajo cualquier circunstancia, sobre todo saca mucha ventaja a medida que la cantidad de entidades guardadas es mayor, por que no decrece su rendimiento mientras que si lo hace en su competencia.