**Bases de Datos 2 2021 -TP3**

**Bases de Datos NoSQL / Práctica con MongoDB**

**entrega: 31/5**

**Parte 1: Bases de Datos NoSQL y Relacionales**

▶︎ Si bien las BBDD NoSQL tienen diferencias fundamentales con los sistemas de BBDD

Relacionales o RDBMS, algunos conceptos comunes se pueden relacionar. Responda las

siguientes preguntas, considerando MongoDB en particular como Base de Datos NoSQL.

**1. ¿Cuáles de los siguientes conceptos de RDBMS existen en MongoDB? En caso de no existir, ¿hay alguna alternativa? ¿Cuál es?**

• Base de Datos: Existe en Mongodb.

• Tabla / Relación: Existe en Mongodb. Por convención, una tabla en Mongo, es denominada ‘collection’ (coleccion)

• Fila / Tupla: Existe en Mongodb, aunque con el nombre de ‘document’ (documento)

• Columna: No existe este concepto como tal. En Mongo, se lo puede encontrar con el nombre de ‘field’, el cual contendrá un par nombre-valor (o clave-valor) en un documento.

**2. MongoDB tiene soporte para transacciones, pero no es igual que el de los RDBMS. ¿Cuál es el alcance de una transacción en MongoDB?**

Cuando se habla de transacciones dentro de una base de datos, se hace referencia a la capacidad de poder realizar operaciones entre múltiples tablas, colecciones o registros y que se garantice que si una de estas operaciones falla, el efecto de todas las operaciones es revertido. Esto garantiza la consistencia en los datos y es la manera en que las bases de datos, mantienen la integridad de los datos cuando se realizan operaciones entre múltiples registros.

En el caso de MongoDB, previo a la versión 4.0, se garantizaba transacciones ACID *(Atomicity, Consistency, Isolation, Durability)* a nivel de documento. Esto quiere decir que, las operaciones realizadas en los subdocumentos dentro de un documento cumplían con la condición de que si un error ocurría, la base de datos se encargaba de hacer rollback al estado anterior al inicio de la operación.

Sin embargo existen casos de uso en los cuales, es absolutamente necesario contar con transaccionalidad entre documentos. Cuando las aplicaciones se encontraban empleando MongoDB, lo que debían hacer (previo a la versión 4.0) era implementar la lógica transaccional dentro del código, lo que generaba aumentaba la complejidad y en algunos casos, la latencia de la ejecución de las operaciones.

Debido a esto, en su versión 4.0, MongoDB implementa las transacciones ACID entre documentos y hace una interfaz agradable para implementar esta funcionalidad dentro de aplicaciones existentes.

**3. Para acelerar las consultas, MongoDB tiene soporte para índices. ¿Qué tipos de índices soporta?**

Los índices en MongoDB se generan en forma de Árbol-B o B-Tree. Es decir, que los datos se guardan en forma de árbol, pero manteniendo los nodos balanceados. Esto incrementa la velocidad a la hora de buscar y también a la hora de devolver resultados ya ordenados.

Los tipos de índices que soporta MongoDB, son:

* Índices simples o de un solo campo: Estos índices se aplican a un solo campo de la colección. Para declarar un índice de este tipo, se debe utilizar el siguiente comando: *db.users.ensureIndex( { "user\_id" : 1 } )*

Este tipo de índices, facilita el ordenamiento de los documentos.

* Índices compuestos: permite crear índices sobre varios campos. Para declarar un índice compuesto, se debe ejecutar el siguiente comando:

*db.users.ensureIndex( { "user\_name" : 1, "age":-1 } )*

El índice que se genera con la instrucción anterior, agrupa los datos primero por el campo user\_name y luego por el campo age. Es decir, se obtendría la siguiente salida:

"Antonio", 35

"Antonio", 18

"María", 56

"María", 30

"María", 21

"Pedro", 19

"Unai", 34

"Unai", 27

Lo interesante de los índices compuestos, es que permite ser utilizados para consultar uno o varios de los campos, sin que sea necesario incluirlos todos. En el ejemplo anterior, el índice se puede utilizar siempre que se hagan consultas sobre user\_name o sobre user\_name y age.

* Índices multikey (multillave): Estos índices, son aplicados a campos que tienen información contenida en arreglos. Al usar estos índices, MongoDB crea una entrada para cada uno de los elementos del arreglo. Si se indexa un campo que contiene un valor de matriz, MongoDB crea entradas de índice independientes para cada elemento de la matriz. Estos índices de varias claves permiten que las consultas seleccionen documentos que contienen matrices, haciendo coincidir el elemento o elementos de las matrices.
* Índices geoespaciales: Para utilizar este tipo de índices, MongoDB proporciona dos índices especiales: índices 2d, que usan geometría plana al devolver resultados e índices 2dsphere que usan geometría esférica para devolver resultados.
* Índices de texto:MongoDB proporciona un texttipo de índice que admite la búsqueda de un string en una colección. Estos índices de texto no almacenan palabras vacías específicas del idioma (por ejemplo, "el", "a", "o") y derivan las palabras en una colección para almacenar solo palabras raíz.
* Índices hash: MongoDB proporciona un tipo de índice hash , que indexa el hash del valor de un campo. Estos índices tienen una distribución de valores más aleatoria a lo largo de su rango, solo admiten coincidencias de igualdad y no pueden admitir consultas basadas en rangos.

**4. ¿Existen claves foráneas en MongoDB?**

No, no existen llaves foráneas en MongoDB. Para relacionar dos documentos, existen dos variantes:

* **Manual References:** En este caso, se guarda el campo \_id de un documento como referencia en otro documento. En este modelo la aplicación debe ejecutar una segunda consulta para devolver los datos relacionados.
* **DBRefs:** Son referencias de un documento a otro utilizando el valor del campo del primer documento \_id, nombre de la colección (y opcionalmente el nombre de base de datos). Con esto los DBRefs permiten vincular documentos de varias colecciones para linkearse en una sola colección. Los DBRefs proporcionan en esencia una semántica común para la representación de los vínculos entre documentos. Los DBRefs también requieren consultas adicionales para devolver los documentos de referencia. Por otro lado, la mayoría de los drivers ofrecen métodos de utilidad para hacer la query de la DBRef de forma automática.

**Parte 2: Primeros pasos con MongoDB**

▶︎ Descargue la última versión de MongoDB desde el sitio oficial. Ingrese al cliente de línea de comando para realizar los siguientes ejercicios.

**5. Cree una nueva base de datos llamada ecommerce, y una colección llamada products. En esa colección inserte un nuevo documento (un producto) con los siguientes atributos:**

**{name:’Caldera Caldaia Duo’, price:140000}**

**recupere la información del producto usando el comando db.products.find() (puede**

**agregar la función .pretty() al final de la expresión para ver los datos indentados). Notará que no se encuentran exactamente los atributos que insertó. ¿Cuál es la diferencia?**

▶︎ Una característica fundamental de MongoDB y otras bases NoSQL es que los documentos no tienen una estructura definida, como puede ser una tabla en un RDBMS. En una misma colección pueden convivir documentos con diferentes atributos, e incluso atributos de múltiples valores y documentos embebidos.

Se inicia el cliente de mongo:

>> sudo service mongod start

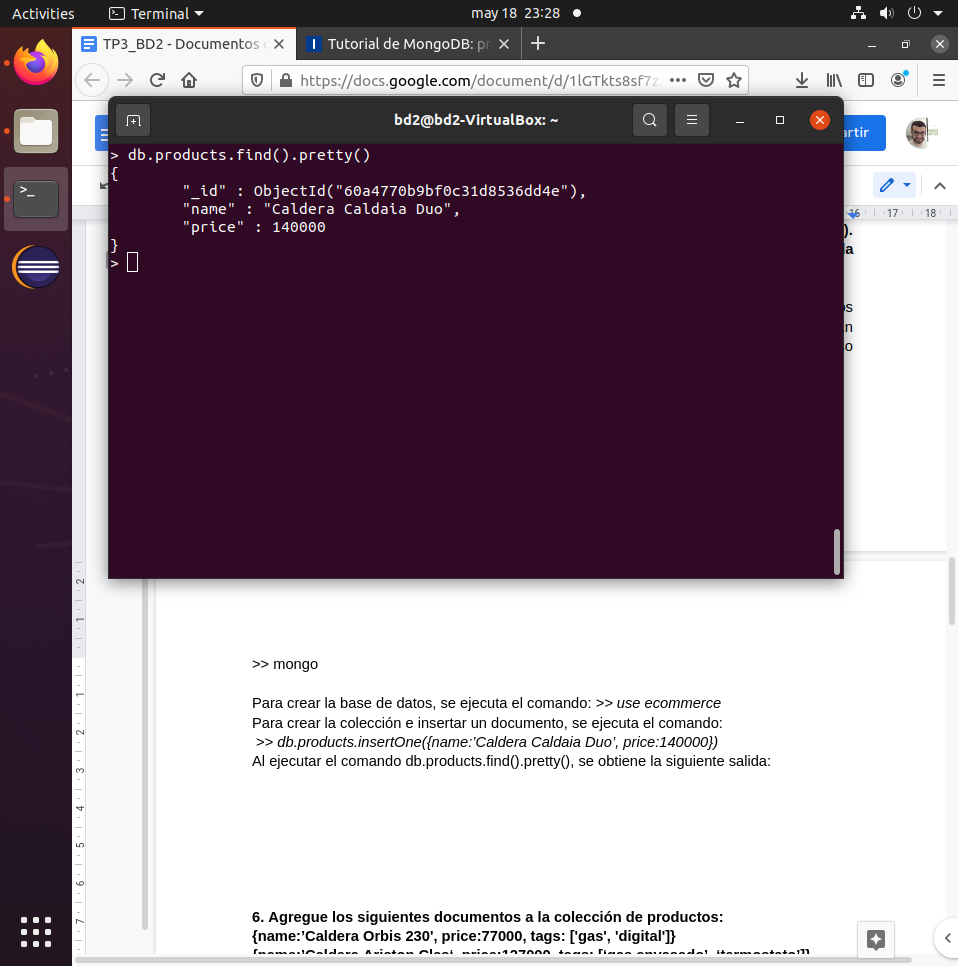
>> mongo

Para crear la base de datos, se ejecuta el comando:

*>> use ecommerce*

Para crear la colección e insertar un documento, se ejecuta el comando:  
 >> *db.products.insertOne({name:’Caldera Caldaia Duo’, price:140000})*

Al ejecutar el comando db.products.find().pretty(), se obtiene la siguiente salida:

Además de los atributos insertados, MongoDB agrega un identificador al documento ingresado. El identificador se reconoce con el nombre de “\_id”, junto con un hash asociado.

**6. Agregue los siguientes documentos a la colección de productos:**

**{name:’Caldera Orbis 230', price:77000, tags: ['gas', 'digital']}**

**{name:’Caldera Ariston Clas‘, price:127000, tags: [‘gas envasado’, ‘termostato’]}**

**{name:’Caldera Caldaia S30‘, price:133000}**

**{name:’Caldera Mural Orbis 225cto’, price:100000, tags: ['gas', ‘digital’,**

**‘termostato’]}**

**Y busque los productos:**

**- de $100.000 o menos**

**- que tengan la etiqueta (tag) “digital”**

**- que no tengan etiquetas (es decir, que el atributo esté ausente)**

**- que incluyan la palabra ‘Orbis’ en su nombre**

**- con la palabra ‘Orbis’ en su nombre y menores de $100.000**

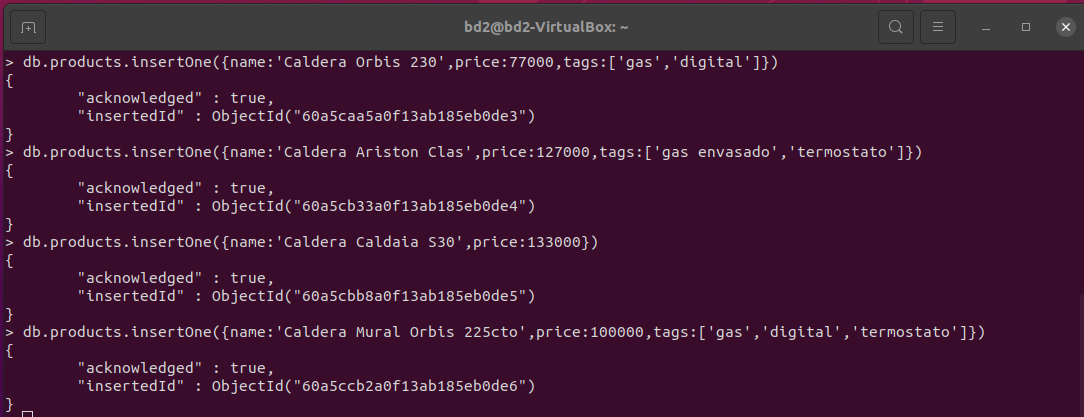
**vuelva a realizar la última consulta pero proyecte sólo el nombre del producto en los resultados, omitiendo incluso el atributo \_id de la proyección.**

Para crear los documentos indicados, se ejecutaron los siguientes comandos:

>> *db.products.insertOne({name:’Caldera Orbis 230’, price:77000, tags:*['gas', 'digital']*})*

*>> db.products.insertOne({name:’Caldera Ariston Clas’, price:127000, tags:['gas envasado', 'termostato']})*

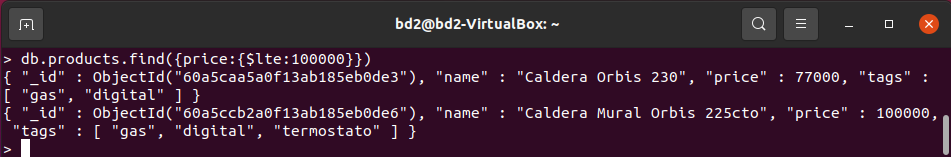
*>> db.products.insertOne({name:’Caldera Caldaia S30’, price:133000})  
>>db.products.insertOne({name:’Caldera Mural Orbis 225cto’, price:100000, tags:['gas’,’digital’,'termostato']})*

**

**Consultas sobre la colección de ‘products’:**

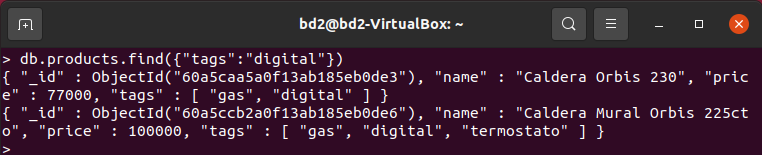
* Buscar aquellos productos de $100.000 o menos:

*>> db.products.find({price:{$lte:100000}})*



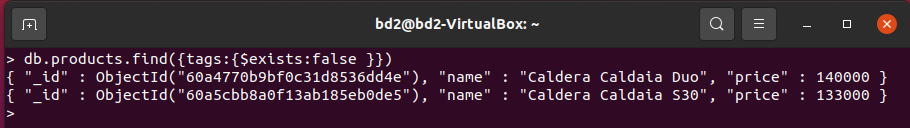
* Buscar aquellos productos que tengan la etiqueta (tag) “digital”

*>> db.products.find({"tags":"digital"})*

**

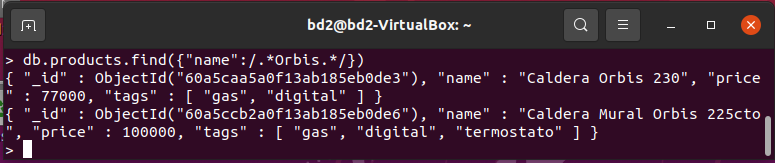
* Buscar aquellos productos que no tengan etiquetas (es decir, que el atributo esté ausente)

*>>db.products.find({tags:{$exists:false }})*

**

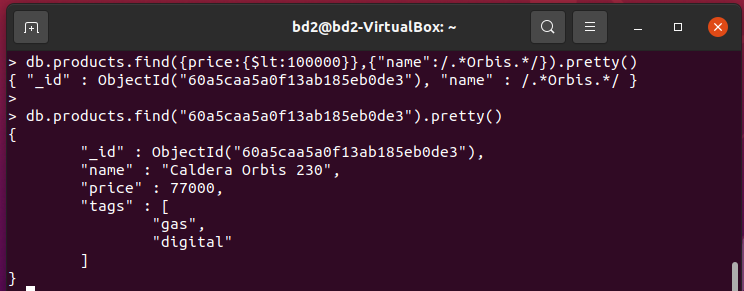
* Buscar aquellos productos que incluyan la palabra ‘Orbis’ en su nombre

*>> db.products.find({ "name": /.\*Orbis.\*/})*

**

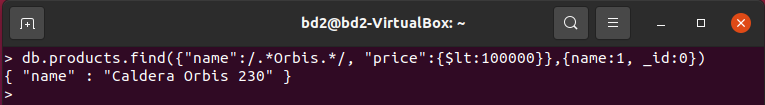
* Buscar aquellos productos con la palabra ‘Orbis’ en su nombre y menores de $100.000

*>> db.products.find({price:{$lt:100000}},{"name":/.\*Orbis.\*/})*

**

* Buscar aquellos productos con la palabra ‘Orbis’ en su nombre y menores de $100.000, proyectando sólo el nombre del producto en los resultados, omitiendo incluso el atributo \_id de la proyección.

*>> db.products.find({"name":/.\*Orbis.\*/, "price":{$lt:100000}},{name:1, \_id:0})*



▶︎ En MongoDB hay diferentes maneras de realizar actualizaciones, de acuerdo a las

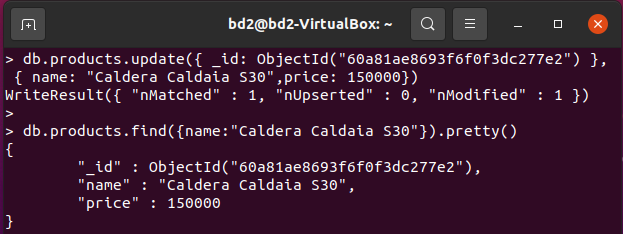
necesidades del esquema flexible de documentos.

* 7. Actualice la “Caldera Caldaia S30” cambiándole el precio a $150.000.

Se actualiza el documento:  
 *db.products.update({ \_id: ObjectId("60a81ae8693f6f0f3dc277e2") }, { name:   
 "Caldera Caldaia S30",price: 150000})*

Se chequea que el mismo se haya actualizado:

*db.products.find({name:"Caldera Caldaia S30"}).pretty()*

****

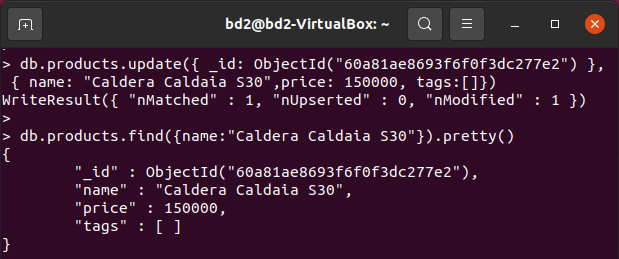
* 8. Cree el array de etiquetas (tags) para la “Caldera Caldaia S30”.

Se actualiza el documento:

*db.products.update({ \_id: ObjectId("60a81ae8693f6f0f3dc277e2") }, { name:   
 "Caldera Caldaia S30",price: 150000, tags:[]})*

Se chequea que el mismo se haya actualizado:

*db.products.find({name:"Caldera Caldaia S30"}).pretty()*



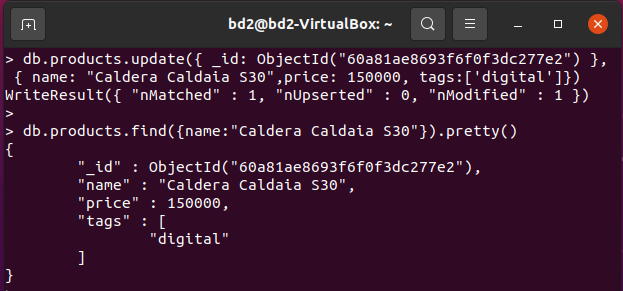
* 9. Agregue “digital” a las etiquetas de la “Caldera Caldaia S30”.

Se actualiza el documento:

*db.products.update({ \_id: ObjectId("60a81ae8693f6f0f3dc277e2") }, { name:   
 "Caldera Caldaia S30",price: 150000, tags:['digital']})*

Se chequea que el mismo se haya actualizado:

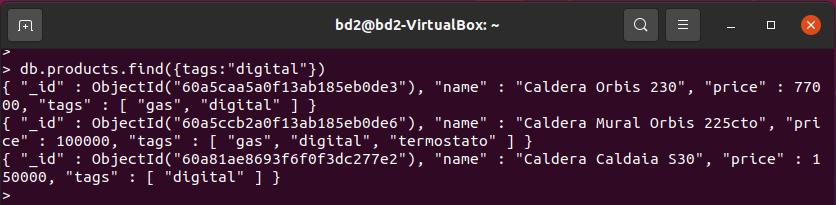
*db.products.find({name:"Caldera Caldaia S30"}).pretty()*

****

* 10. Incremente en un 10% los precios de todas las calderas digitales.

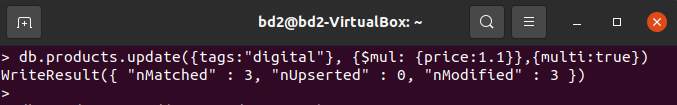
Para validar el correcto incremento en los precios, se busca los productos en donde   
 en el tag sea ‘digital’. Para esto, se ejecuta la siguiente consulta:

*db.products.find({tags:"digital"})*



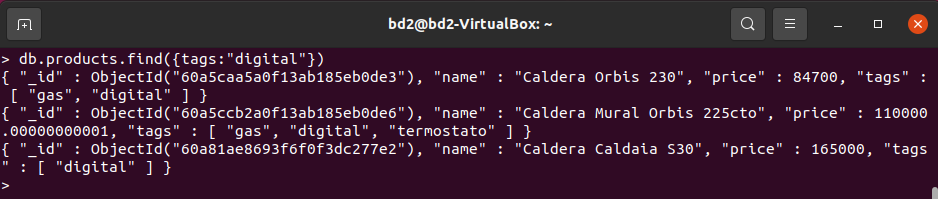
Luego, se ejecuta la actualización correspondiente, por sobre el precio de dichos   
 productos.

*db.products.update({tags:"digital"}, {$mul: {price:1.1}},{multi:true})*



Por último, se vuelve a ejecutar la query inicial, para validar que los precios hayan   
 variado.

*db.products.find({tags:"digital"})*



**Parte 3: Índices**

▶︎ Elimine todos los productos de la colección. Guarde en un archivo llamado ‘generador.js’ el siguiente código JavaScript y ejecútelo con: load(<ruta del archivo ‘generador.js’>). Si

utiliza un cliente que lo permita (ej. Robo3T), se puede ejecutar directamente en el espacio de consultas.

11. Busque en la colección de compras (purchases) si existe algún índice definido.

12. Cree un índice para el campo productName. Busque los las compras que tengan en el nombre

del producto el string “11” y utilice el método explain("executionStats") al final de la

consulta, para comparar la cantidad de documentos examinados y el tiempo en

milisegundos de la consulta con y sin índice.

13. Busque las compras enviadas dentro de la ciudad de Buenos Aires. Para esto, puede definir una variable en la terminal y asignarle como valor el polígono del archivo provisto

caba.geojson (copiando y pegando directamente). Cree un índice geoespacial de tipo

2dsphere para el campo location de la colección purchases y, de la misma forma que en el

punto 12, compare la performance de la consulta con y sin dicho índice.

for (var i = 1; i <= 50000; i++) {

var randomTags = ['envio express', 'oferta', 'cuotas', 'verificado'].sort( function()

{ return 0.5 - Math.random() } ).slice(1, Math.floor(Math.random() \* 5));

var randomPrice = Math.ceil(110000+(Math.random() \* 300000 - 100000));

db.products.insert({

name:'Producto '+i,

price:randomPrice,

tags: randomTags

});

}

for (var i = 1; i <= 50000; i++) {

if (Math.random() > 0.7) {

var randomPurchases = Math.ceil(Math.random() \* 5);

for (var r = 1; r <= randomPurchases; r++){

var randomLong = -34.56 - (Math.random() \* .23);

var randomLat = -58.4 - (Math.random() \* .22);

var shippingCost = 200+Math.ceil(Math.random()\*20) \* 100;

db.purchases.insert({

productName:’Producto '+i,

shippingCost: shippingCost,

location: {type: "Point",coordinates: [randomLat, randomLong]}

});

}

}

}

**Parte 4: Aggregation Framework**

▶︎MongoDB cuenta con un Aggregation Framework que brinda la posibilidad de hacer analítica

en tiempo real del estilo OLAP (Online Analytical Processing), de forma similar a otros

productos específicos como Hadoop o MapReduce. En los siguientes ejercicios se verán

algunos ejemplos de su aplicabilidad.

14. Obtenga 5 productos aleatorios de la colección.

15. Usando el framework de agregación, obtenga las compras que se hayan enviado a 1km (omenos) del centro de la ciudad de Buenos Aires ([-58.4586,-34.5968]) y guárdelas en una

nueva colección.

16. Obtenga una colección de los productos que fueron enviados en las compras del punto

anterior. Note que sólo es posible ligarlas por el nombre del producto.

▶︎ Si la consulta se empieza a tornar difícil de leer, se pueden ir guardando los agregadores en variables, que no son más que objetos en formato JSON.

17. Usando la colección del punto anterior, obtenga una nueva en la que agrega a cada producto un atributo purchases que consista en un array con todas las compras de cada producto.

18. Obtenga el promedio de costo de envío pagado para cada producto del punto anterior.

**Referencias:**

<https://docs.mongodb.com/manual/>

<https://www.genbeta.com/desarrollo/una-introduccion-a-mongodb>

<https://docs.mongodb.com/manual/indexes/>

<https://www.genbeta.com/desarrollo/mongodb-creacion-y-utilizacion-de-indices>

<https://www.codigofuente.org/introduccion-a-mongodb/>

<https://platzi.com/contributions/introduccion-al-uso-de-indices-en-mongodb/>

<http://gpd.sip.ucm.es/rafa/docencia/nosql/indices.html>

<https://charlascylon.com/2013-08-01-tutorial-mongodb-%C3%ADndices>

<https://unpocodejava.com/2014/07/07/referencias-en-mongodb/>

<https://platzi.com/tutoriales/1533-mongodb/3757-transacciones-en-mongodb-2/>