BASE DE DATOS 2
TP3

Alumnos:

Ailen Panigo
Paula Vaccaro
Jerónimo Marchetti

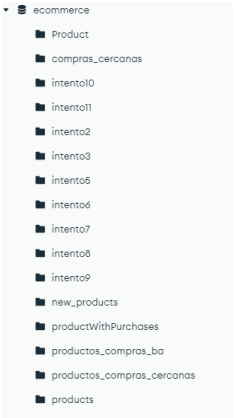
Ayudante:

Natalia Ortu

Índice:

Prólogo:

Este trabajo fue realizado utilizando mongo shell (mongosh), y corroborando los resultados mediante MongoDB Compass.



Estas son todas las colecciones resultantes. No todas se exponen en el trabajo ya que por ejemplo, las nombre “intento*N*” fueron, en su mayoría, de prueba y testeo para corroborar funcionalidades que se necesitan para la resolución de los puntos de la parte 4.

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: este concepto existe también en MongoDB.
- Tabla / Relación: este concepto no existe en MongoDB. La alternativa son las 'Colecciones' y la diferencia con las 'Tablas' es que son grupos de documentos en lugar de grupos de registros.
- Fila / Tupla: este concepto no existe en MongoDB. La alternativa es 'Documento'.
- Columna: este concepto no existe en MongoDB. La alternativa son los 'Campos' que son pares clave/valor.
- Clave primaria: en MongoDB se reemplaza el concepto de 'clave primaria' por un campo "_id" (identificador único) para cada documento. Garantiza unicidad dentro de la colección, y si no se proporciona, se generará uno automáticamente.
- Clave foránea: este concepto no existe en MongoDB, pero se puede lograr referenciar otros objetos de manera manual mediante la adición de un campo que contenga un objectID referenciando a otro objeto. O sea, el valor del campo "_id" del otro objeto. A esto se lo llama referencia. Otra forma es incorporar toda la información que se relacione en un único documento, aunque causaría repetición de la información.

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

En las bases de datos relacionales, cuando se realiza alguna operación CRUD, toda la tabla se bloquea, lo que impide el acceso a otros usuarios que quieran realizar operaciones en la misma tabla. Sin embargo, en MongoDB, solo se bloquea el documento específico que se va a modificar, lo que significa que otros usuarios aún pueden acceder y modificar otros documentos simultáneamente. Esto hace que MongoDB sea compatible con el modelo ACID a nivel de documento.

A partir de la versión 4.0 de MongoDB, se introdujeron transacciones ACID a nivel multi-documento. Esto significa que ahora es posible realizar transacciones que involucren múltiples documentos y garantizar la consistencia e integridad de los datos en esas transacciones. Sin embargo, se estima que alrededor del 90% de los modelos de datos que utilizan el enfoque de documentos en MongoDB no necesitarán transacciones, ya que el sistema proporciona un buen nivel de aislamiento y coherencia en las operaciones a nivel de documento.

Aunque la mayoría de los casos pueden ser manejados sin transacciones, existen situaciones en las que las transacciones multi-documento o multi-colección son la mejor opción. Las transacciones en MongoDB funcionan de manera similar a las transacciones en

otras bases de datos. Para utilizar transacciones, primero se abre una sesión para utilizarla para ejecutar operaciones CRUD entre documentos, colecciones e incluso shards (fragmentos de datos distribuidos).

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

- Single Field: se aplican a un solo campo de una colección. Para declararlo se usa un comando similar a este: *db.users.ensureIndex({ "user_id" : -1 })*. El número indica que queremos que el índice se ordene de forma descendente. Si quisiéramos un orden ascendente, el parámetro debe ser un 1.

- Compound index: el índice se generará sobre varios campos. *db.users.ensureIndex({ "user_name" : 1, "age":-1 })* El índice que se generará con la instrucción anterior, agrupará los datos primero por el campo user_name y luego por el campo age. Es decir, se generaría algo así:

"Antonio", 35
"Antonio", 18
"María", 56
"María", 30
"María", 21
"Pedro", 19
"Unai", 34
"Unai", 27

Podemos usar los índices compuestos para consultar uno o varios de los campos, sin ser 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. Este índice compuesto ordenará el campo user_name de la misma manera que si creamos un índice simple. Lo que no podemos hacer es buscar sólo sobre el campo “age”. Para ese caso tendríamos que crear un índice específico. Si el índice compuesto tuviera tres campos, podríamos utilizarlo al consultar sobre el primer campo, sobre el primer y segundo campo, o sobre los tres campos.

- Multikey index: se aplican a campos que tienen información contenida en arreglos. se crea una entrada para cada uno de los elementos del arreglo.

- Geospatial index: La indexación geoespacial de MongoDB permite ejecutar de manera eficiente consultas espaciales en una colección que contiene formas y puntos geoespaciales.

- Text index: MongoDB proporciona índices de texto para admitir consultas de búsqueda de texto sobre el contenido de un string. Los índices de texto pueden incluir cualquier campo cuyo valor sea un string o una matriz de strings. Una colección solo puede tener un índice de búsqueda de texto, pero ese índice puede cubrir varios campos.

- Hashed index: Los índices hash mantienen entradas con hash de los valores del campo indexado. Los índices hash admiten la fragmentación mediante claves de fragmentación hash. La fragmentación basada en hash utiliza un índice hash de un campo

como clave de partición para dividir los datos en su clúster fragmentado. El uso de una clave fragmentada con hash para fragmentar una colección da como resultado una distribución más uniforme de los datos. Los índices hash utilizan una función hash para calcular el hash del valor del campo de índice. La función hash colapsa los documentos incrustados y calcula el hash para el valor completo, pero no admite índices de varias claves (es decir, matrices). Específicamente, crear un índice hash en un campo que contiene una matriz o intentar insertar una matriz en un campo indexado hash devuelve un error.

4. ¿Existen claves foráneas en MongoDB?

Este concepto no existe en MongoDB y sus alternativas fueron explicadas en la primera pregunta.

Se puede lograr referenciar otros objetos de manera manual mediante la adición de un campo que contenga un objectID referenciando a otro objeto. O sea, el valor del campo "_id" del otro objeto. A esto se lo llama referencia. Otra forma es incorporar toda la información que se relacione en un único documento, aunque causaría repetición de la información.

Parte 2: Primeros pasos con MongoDB

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:'Yerba Paysandu', price:600}

```
use ecommerce
db.products.insertOne({name:'Yerba Paysandu', price:600})
db.products.find().pretty() /*Devuelve, además de los datos insertados,
el campo _id que se genera automáticamente*/
```

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?

Uno de los registros que nos devolvió la consulta es este:

```
{
  _id: ObjectId("6475199fe44092591aad6b71"),
  name: 'Yerba Paysandu',
  price: 600
}
```

La diferencia es que se agregó el atributo _id, que como se explicó en el punto 1, funciona como identificador único para cada documento.

6.

```
db.products.insertMany([
      {name:'Fideos Marolio t', price:450, tags: ['tallarines', '500']},
      {name:'Fideos Marolio ftcni', price:630, tags: ['fetuccini', '500']},
      {name:'Fideos Oneway', price:450},
      {name:'Fideos Familia Unita lanzamiento', price:800, tags:
['tallarines', '500', 'al huevo']}
])
```

Realice la consulta correspondiente para obtener los productos:

- de $750 o menos

db.products.find({price:{$lte:750}})

```
{
  _id: ObjectId("647510e13d486900cdade251"),
  name: 'Yerba Paysandu',
  price: 600
}
{
  _id: ObjectId("647511583d486900cdade252"),
  name: 'Fideos Marolio t',
  price: 450,
  tags: [
    'tallarines',
    '500'
  ]
}
{
  _id: ObjectId("647511db3d486900cdade253"),
  name: 'Fideos Marolio ftcni',
  price: 630,
  tags: [
    'fetuccini',
    '500'
  ]
}
{
  _id: ObjectId("647511db3d486900cdade254"),
  name: 'Fideos OneWay',
  price: 450
}
```

- que tengan la etiqueta (tag) “tallarines”
db.products.find({tags:'tallarines'})


```
{
  _id: ObjectId("647511583d486900cdade252"),
  name: 'Fideos Marolio t',
  price: 450,
  tags: [
    'tallarines',
    '500'
  ]
}
{
  _id: ObjectId("647511db3d486900cdade255"),
  name: 'Fideos Familia Unita Lanzamiento',
  price: 800,
  tags: [
    'tallarines',
    '500',
    'al huevo'
  ]
}
```


- que no tengan etiquetas (es decir, que el atributo esté ausente)
db.products.find({tags:{$exists:false}})


```
{
  _id: ObjectId("647510e13d486900cdade251"),
  name: 'Yerba Paysandu',
  price: 600
}
{
  _id: ObjectId("647511db3d486900cdade254"),
  name: 'Fideos OneWay',
  price: 450
}
```


- que incluyan la palabra ‘Marolio’ en su nombre
db.products.find({name:{$regex:"Marolio"}})


```
{
  _id: ObjectId("647511583d486900cdade252"),
  name: 'Fideos Marolio t',
  price: 450,
  tags: [
    'tallarines',
    '500'
  ]
}
{
  _id: ObjectId("647511db3d486900cdade253"),
  name: 'Fideos Marolio ftcni',
  price: 630,
  tags: [
    'fetuccini',
    '500'
  ]
}
```


- con la palabra ‘Marolio’ en su nombre y con un precio mayor de $500
db.products.find({name:{$regex:"Marolio"}, price:{$gt:500}})


```
{
  _id: ObjectId("647511db3d486900cdade253"),
  name: 'Fideos Marolio ftcni',
  price: 630,
  tags: [
    'fetuccini',
    '500'
  ]
}
```


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

db.products.find({name:{$regex:"Marolio"}, price:{$gt:500}}, {name:1, _id:0})


```
{
  name: 'Fideos Marolio ftcni'
}
```

7. Actualice la ’Fideos Marolio t’ cambiándole el precio a $530
db.products.updateOne({ name: 'Fideos Marolio t' },{ $set: { price: 350} })

8. Cree el array de etiquetas (tags) para la “Fideos OneWay”.
db.products.updateOne({ name: 'Fideos Oneway' },{ $set: { tags: [] } })

9. Agregue “tirabuzon” a las etiquetas de la “Fideos OneWay”.
db.products.updateOne({ name: 'Fideos Oneway' },{ $push: { tags: { $each: ['tirabuzon’] } } })

10. Incremente en un 10% los precios de todos los tallarines.
db.products.updateMany({ tags: 'tallarines' },{ $mul: { price: 1.1 } })

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’).

```
db.products.deleteMany({}) //borro todos los elementos de la colección
load('C:/Users/jerom/Desktop/TP3 BBDD2 2023 - Material/generator.js') //cargo script
```

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

ecommerce> db.purchases.getIndexes()
obtengo el resultado: [{ v: 2, key: { _id: 1 }, name: '_id_' }]

Significa que hay un índice definido en la colección "purchases".
"v: 2" indica la versión del índice. En este caso, el valor "2" significa que se está utilizando el formato de índice de MongoDB versión 2.
"key: { _id: 1 }" especifica las claves de índice utilizadas. En este caso, el índice está definido en el campo "_id" con un valor ascendente de "1". El índice en el campo "_id" es un índice predeterminado y único que se crea automáticamente para cada documento en MongoDB.
"name: 'id'" es el nombre asignado al índice. En este caso, el nombre del índice es "id" debido a que está basado en el campo "_id".

12. 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 ver la cantidad de documentos examinados y el tiempo en milisegundos de la consulta. Cree un índice adecuado para el campo productName, y realice la misma consulta que realizó con anterioridad. Compare los datos obtenidos.

ecommerce> db.purchases.find({ productName: /11/ }).explain("executionStats")

Obtengo como resultado, entre otros valores:

```
 executionStats: {
    executionSuccess: true,
    nReturned: 2045,
    executionTimeMillis: 39,
    totalKeysExamined: 0,
    totalDocsExamined: 44793,
….
}
```

Marcado en amarillo se encuentra el campo que indica los documentos examinados, que fueron 44793 (docsExamined). También el que indica los milisegundos tardados, que fueron 36 (executionTimeMilis). El campo **totalKeysExamined es 0** ya que al no haber ningún índice definido aún , no se utilizan claves para la búsqueda por índices.

Crearemos un índice ascendente para el campo productName.
ecommerce> db.purchases.createIndex({ productName: 1 })
Volveremos a realizar la consulta y vemos que se redujeron los milisegundos de ejecución:

```
executionStats: {
        executionSuccess: true,
        nReturned: 2045,
        executionTimeMillis: 66,
        totalKeysExamined: 44793,
        totalDocsExamined: 2045
…
}
```

Vemos que la consulta tarda más, ya que al ser una búsqueda por un valor específico dentro de un string, de igual manera debe chequear que cada string contenga ese valor dentro, sumado a que debe utilizar el índice (se examinaron las keys de todos los documentos, totalKeysExamined con índice, coincide con el valor totalDocsExamined sin índice).
Creando un índice de texto de la siguiente manera, de igual modo se tardan más millis que sin índice:
ecommerce> db.purchases.createIndex({ productName: “text” })

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.

Dentro de mongo shell asigno a una variable el valor del archivo
var geo =
{"type":"MultiPolygon","coordinates":[[[[-58.46305847167969,-34.53456089748654],[-58.49979400634765,-34.54983198845187],[-58.532066345214844,-34.614561581608186],[-58.528633117675774,-34.6538270014492],[-58.48674774169922,-34.68742794931483],[-58.479881286621094,-34.68206400648744],[-58.46855163574218,-34.65297974261105],[-58.465118408203125,-34.64733112904415],[-58.4585952758789,-34.63998735602951],[-58.45344543457032,-34.63603274732642],[-58.447265625,-34.63575026806082],[-58.438339233398445,-34.63038297923296],[-58.38100433349609,-34.62162507826766],[-58.38237762451171,-34.59251960889388],[-58.37894439697265,-34.5843230246475],[-58.46305847167969,-34.53456089748654]]]]}

Creo índice de tipo geoespacial “2dsphere”

```
db.purchases.createIndex({ location: "2dsphere" })
```

Se hará en el campo location de la colección.

Algunos de los locales que se encuentran son los siguientes:


```
[
  {
    _id:
ObjectId("64751ac1e44092591aaeeb55"),
    productName: 'Producto 19354',
    totalCost: 500,
    location: {
      type: 'Point',
      coordinates: [ -58.41733561432913,
-34.56319403425656 ]
    }
  },
  {
    _id:
ObjectId("64751adde44092591aaf36cc"),
    productName: 'Producto 41224',
    totalCost: 300,
    location: {
      type: 'Point',
      coordinates: [ -58.41659860723172,
-34.563551184316466 ]
    }
  },
  {
    _id:
ObjectId("64751ac0e44092591aaee986"),
    productName: 'Producto 18858',
    totalCost: 1300,
    location: {
      type: 'Point',
      coordinates: [ -58.41560302161189,
-34.56272093185842 ]
    }
  },
  {
    _id: ObjectId("64751ad3e44092591aaf1cfe"),
    productName: 'Producto 33816',
    totalCost: 400,
    location: {
      type: 'Point',
      coordinates: [ -58.41464026001376,
-34.56424918432742 ]
    }
  },
  {
    _id:
ObjectId("64751ad3e44092591aaf1bbf"),
    productName: 'Producto 33500',
    totalCost: 500,
    location: {
      type: 'Point',
      coordinates: [ -58.41488970442061,
-34.56438181511269 ]
    }
  },
  {
    _id: ObjectId("64751ac8e44092591aaeff87"),
    productName: 'Producto 25391',
    totalCost: 600,
    location: {
      type: 'Point',
      coordinates: [ -58.41634721048599,
-34.56492065991616 ]
    }
  },
  {
    _id:
ObjectId("64751abee44092591aaee384"),
    productName: 'Producto 17258',
    totalCost: 1200,
    location: {
      type: 'Point',
      coordinates: [ -58.41629547654806,
-34.56427574034025 ]
    }
  },
  {
    _id:
ObjectId("64751aade44092591aaeb982"),
    productName: 'Producto 5307',
    totalCost: 1200,
    location: {
      type: 'Point',
      coordinates: [ -58.41684162073031,
-34.56461830542508 ]
    }
  },
  {
    _id: ObjectId("64751acfe44092591aaf1080"),
    productName: 'Producto 30350',
    totalCost: 1200,
    location: {
      type: 'Point',
      coordinates: [ -58.41664792533956,
-34.565611397978664 ]
    }
  },
  {
    _id:
ObjectId("64751ae1e44092591aaf41de"),
    productName: 'Producto 44357',
    totalCost: 1400,
    location: {
      type: 'Point',
      coordinates: [ -58.41661338429255,
-34.56550312398322 ]
    }
  },
  {
    _id:
ObjectId("64751ab8e44092591aaed39d"),
    productName: 'Producto 12737',
    totalCost: 1900,
    location: {
      type: 'Point',
      coordinates: [ -58.417069335999386,
-34.56696540501953 ]
    }
  },
  {
```

```
  _id:
ObjectId("64751aade44092591aaeb933"),
  productName: 'Producto 5188',
  totalCost: 900,
  location: {
   type: 'Point',
   coordinates: [ -58.41642032569339,
-34.566908362249514 ]
  }
 },
 {
  _id:
ObjectId("64751ab3e44092591aaec549"),
  productName: 'Producto 8633',
  totalCost: 400,
  location: {
   type: 'Point',
   coordinates: [ -58.41456766563133,
-34.566548420570776 ]
  }
 },
 {
  _id:
ObjectId("64751aafe44092591aaebd24"),
  productName: 'Producto 6386',
  totalCost: 1300,
  location: {
   type: 'Point',
   coordinates: [ -58.41588337579875,
-34.56537538361285 ]
  }
 },
 {
```

```
  _id:
ObjectId("64751ad1e44092591aaf16d8"),
  productName: 'Producto 32136',
  totalCost: 400,
  location: {
   type: 'Point',
   coordinates: [ -58.41333696016694,
-34.56419278563425 ]
  }
 },
 {
  _id:
ObjectId("64751adee44092591aaf3954"),
  productName: 'Producto 41912',
  totalCost: 900,
  location: {
   type: 'Point',
   coordinates: [ -58.40011030934854,
-34.582081255003956 ]
  }
 },
 {
  _id:
ObjectId("64751abae44092591aaed915"),
  productName: 'Producto 14332',
  totalCost: 1400,
  location: {
   type: 'Point',
   coordinates: [ -58.40145919017694,
-34.58202747893189 ]
  }
 },
 {
```

```
  _id:
ObjectId("64751ab8e44092591aaed31e"),
  productName: 'Producto 12585',
  totalCost: 900,
  location: {
   type: 'Point',
   coordinates: [ -58.40118280843138,
-34.58081674541617 ]
  }
 },
 {
  _id:
ObjectId("64751aafe44092591aaebca3"),
  productName: 'Producto 6208',
  totalCost: 1400,
  location: {
   type: 'Point',
   coordinates: [ -58.401375566793924,
-34.580761548477234 ]
  }
 },
 {
  _id:
ObjectId("64751aa5e44092591aaeadcb"),
  productName: 'Producto 1850',
  totalCost: 1300,
  location: {
   type: 'Point',
   coordinates: [ -58.4010665450443,
-34.581544813366634 ]
  }
 }
]
```


Ahora utilizaremos .explain("executionStats") para ver el tiempo de ejecución con y sin índice.

ecommerce > db.purchases.find({location: {$geoIntersects: {$geometry:geo}}}).explain("executionStats")

Con índice:

```
            executionStats: {
  executionSuccess: true,
  nReturned: 9732,
  executionTimeMillis: 103,
  totalKeysExamined: 12326,
  totalDocsExamined: 12308,
  ….. etcétera
}
```

Ahora eliminaremos el índice

ecommerce> db.purchases.dropIndex("location_2dsphere")

Volvemos a correr la consulta y obtenemos los siguientes stats:

```
executionStats: {
  executionSuccess: true,
  nReturned: 9732,
  executionTimeMillis: 152,
  totalKeysExamined: 0,
  totalDocsExamined: 44793,
  …. etcétera
}
```

Vemos que hay gran diferencia en cuánto al total de documentos examinados, ya que con el índice se examinan 12308, mientras que sin el 44794. Es decir, se revisan casi 4 veces más documentos sin el índice. También hay diferencia en la totalKeys, al no haber

índice no se examinan.

La cantidad de documentos examinados afecta al tiempo de ejecución, ya que sín índice tarda 152 milisegundos, mientras que con índice tarda 103.

Parte 4.
Aggregation framework.

14. Usando el framework de agregación, obtenga 5 productos aleatorios de la colección.

Para obtener 5 productos aleatorios de la colección utilizando el framework de agregación, haremos la siguiente consulta:

ecommerce> db.products.aggregate([{ $sample: { size: 5 } }])

Que nos devuelve el siguiente resultado:

```
[
 {
  _id: ObjectId("647519b4e44092591aada364"),
  name: 'Producto 14324',
  price: 335,
  tags: [ 'oferta', 'sello nutricional', 'precios cuidados' ]
 },
 {
  _id: ObjectId("647519abe44092591aad8abe"),
  name: 'Producto 8014',
  price: 259,
  tags: []
 },
 {
  _id: ObjectId("647519b5e44092591aada630"),
  name: 'Producto 15040',
  price: 270,
  tags: [ 'precios cuidados', 'oferta' ]
 },
 {
  _id: ObjectId("64751a5be44092591aae118c"),
  name: 'Producto 11758',
  price: 304,
  tags: [ 'oferta', 'sello nutricional', '2x1' ]
 },
 {
  _id: ObjectId("647519a8e44092591aad8344"),
  name: 'Producto 6100',
  price: 112,
  tags: []
 }
]
```

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

Para obtener las compras que se hayan entregado a 1 km o menos del centro de la ciudad de Buenos Aires ([latitud: -34.5968, longitud: -58.4586]) utilizando el framework de agregación en MongoDB y guardar los resultados en una nueva colección hacemos la siguiente consulta:

```
db.purchases.aggregate([
 {
  $geoNear: {
   near: {
    type: "Point",
    coordinates: [-58.4586, -34.5968]
   },
   distanceField: "distance",
   maxDistance: 1000,
   spherical: true
  }
 },
 {
  $out: "compras_ba"
 }])
```

Esto guardará los documentos en la colección “compras_ba”. Podemos obtener estos con la siguiente consulta

db.compras_ba.find()

Que nos devuelve mucha cantidad de registros que no copiaremos debido a que ocuparían muchas páginas, pero uno de ellos es:


```
[{
   _id: ObjectId("64751abee44092591aaee211"),
   productName: 'Producto 16820',
   totalCost: 1800,
   location: {
     type: 'Point',
     coordinates: [ -58.45846533231763, -34.596344130541475 ]
   },
   distance: 52.22570452826737
 } …]
```


16. Obtenga una colección de los productos incluidos en las compras obtenidas en la consulta 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.

```
db.products.aggregate([
{$lookup:
        {from:"compras_ba",
        localField:"name",
        foreignField:"productName", as: "array"}
        },
 {$match:
        {"array": {$ne:[]}}},
        {$project: {name: 1, price:1, tags: 1}
        },
{$out: "intento11"}
])
```

17. Obtenga una nueva colección de productos, cuyos nombres incluyan el string “111”. En cada documento (cada producto) se debe agregar un atributo “purchases” que consista en un array con todas las “compras” en donde se incluye.

```
db.products.aggregate(
[
{$match:{name:/111/}}
,{$lookup:
        {from:"purchases",
        localField:"name",
        foreignField:"productName",
        as:"purchases"}
 } ,
 {$project:
        {name:1,
        price:1,
         purchases:"$purchases"}
},
 {$out: "intento9"}
])
```