Diseño de Bases de Datos II

Grupo: Chimichanga

Integrantes:

Emanuel Perez Gianolini

Martín Gustavo Reyes

Luis María Sarmiento Esquembre

Facundo Jorge Guarnier

Introducción

Chimichanga es un espacio para que poetas profesionales y aspirantes a poetas tengan un lugar donde compartir su trabajo y contar con el feedback de sus compañeros. Aquí usuarios de todo el mundo pueden subir a la aplicación sus poemas, ver los poemas subidos por el resto de los usuarios, comentar y puntuarlos.

Un poema puede pertenecer a uno de cinco géneros posibles: romántico, comedia, terror, ciencia ficción o educativo. Además de su género, nos interesa conocer otros datos del poema como por ejemplo: título, contenido, autor, fecha de publicación, ubicación desde donde se publicó el poema, promedio del puntaje de sus comentarios, y cantidad de visualizaciones.

En cuanto a los usuarios los datos que se necesitan de cada uno son: nombre, mail, contraseña, promedio, cantidad de poemas publicados y cantidad de comentarios hechos, donde un usuario puede comentar una sola vez un poema en particular.

Al momento de ver los poemas publicados en Chimichanga, el usuario debe poder filtrar los poemas en base al género y poder ordenarlos por: mejor promedio, peor promedio, más visualizaciones, publicados más recientemente o poemas escritos desde Latinoamérica.

Interfaz

En base a estos requerimientos se diseñó la aplicación de la siguiente manera:

Usuario

Poemas

Poema

Filtro

Modelado

Una vez realizado el diseño de la interfaz gráfica de Chimichanga procedemos a crear el modelado de datos. Para ello se plantearon cuatro colecciones con sus respectivos campos.

Usuario

```
{
        _id: ObjectId,
        nombre: String,
        mail: String,
        contraseña: String,
        promedio: int,
        poemasPublicados: int,
        comentariosHechos: int,
}
```

Poemas

```
{
        _id: ObjectId,
        titulo: String,
        contenido: String,
        autor: String,
        autor_id: ObjectId,
        fecha: Date,
        ubicacion: String,
        promedio: int,
        cant_visualizaciones: int,
        tipo: String
}
```

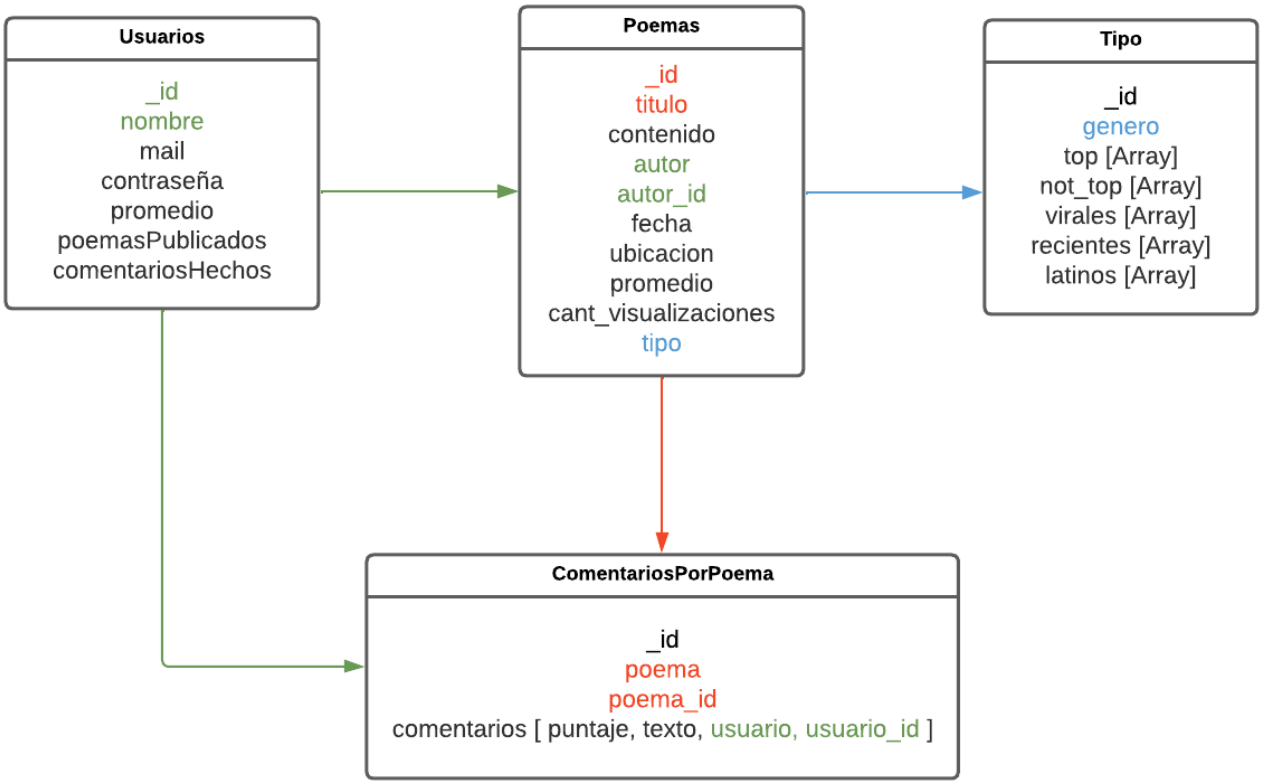
ComentariosPorPoema

```
{
        _id: ObjectId,
        poema: String,
        poema_id: ObjectId,
        comentarios:   [{
                            puntaje: int,
                            texto: String,
                            usuario: String,
                            usuario_id: ObjectId
                        }]
}
```

Tipo

```
{
        _id: ObjectId,
        genero: String,
        top:    [{
                        titulo: String,
                        promedio: int
                }],
        not_top: [{
                        titulo: String,
                        promedio: int
                }],
        virales: [{
                        titulo: String,
                        cant_visualizaciones: int
                }],
        recientes: [{
                        titulo: String,
                        fecha: Date
                }],
        latinos:    [{
                        titulo: String,
                        ubicacion: String
                }]
}
```

A continuación mostramos una representación gráfica de las relaciones entre las colecciones:



Scripts

Una vez definidas las colecciones, utilizamos los scripts que se muestran a continuación para poblar la Base de Datos de la aplicación con 10000 usuarios y 50000 poemas.

Crear Base de Datos de la aplicación (*chimichanga)*

```
use("chimichanga");
```

Crear y poblar la colección *Usuarios*

```
const cant_usuarios = 10000
db.createCollection("Usuarios")
var bulk = db.Usuarios.initializeUnorderedBulkOp();
for (var i = 0; i < cant_usuarios; i++){
   bulk.insert({
            "nombre" : "Usuario"+i,
            "mail": "Usuario"+i+"@email.com",
            "contraseña": "123456789",
            "promedio": Math.floor(Math.random() * 10) + 1,
            "poemasPublicados" : "-",
            "comentariosHechos" : "-"
        }
   )
}
bulk.execute()
```

Ejemplo de documento de la colección *Usuarios*:

```
_id: ObjectId('6371358a3fbab509060a2e24')
nombre: "Usuario1"
mail: "Usuario1@email.com"
contraseña: "123456789"
promedio: 9
poemasPublicados: "-"
comentariosHechos: "-"
```

Crear y poblar la colección *Poemas*

```
const cant_poemas = 50000
db.createCollection("Poemas")
ubicacion = ["Italia", "Francia", "España", "Estados Unidos", "Peru","Argentina", "Chile",
"Australia", "Sudafrica", "China","Alemania","Portugal", "Rusia", "Camerun", "India", "Brasil",
"Qatar", "Marruecos", "Japon"]
const tipos = ["romantico", "comedia", "terror", "ciencia ficcion", "educativo"]
const usuarios = db.Usuarios.find()
var bulk = db.Poemas.initializeUnorderedBulkOp();
for (var i = 0; i < cant_poemas; i++) {
   var dia = Math.floor(Math.random() * 27) + 1
   if (mes < 10) {
       mes = "0" + String(mes)
   }
   var mes = Math.floor(Math.random() * 11) + 1
   if (mes < 10) {
       mes = "0" + String(mes)
       }
   var anio = Math.floor(Math.random() * (7)) + 2015
   nombre_random = "Usuario" + Math.floor(Math.random() * cant_usuarios)
   autor_id =  db.Usuarios.find({nombre:nombre_random}).next()._id
   bulk.insert({
           "titulo":"titulo "+i,
           "contenido":"contenido "+i,
           "autor": nombre_random,
           "autor_id" : autor_id ,
           "fecha": new Date(anio, mes , dia),
           "ubicacion": ubicacion[Math.floor(Math.random() * ubicacion.length)],
           "promedio": Math.floor(Math.random() * 10) + 1,
           "cant_visualizaciones": Math.floor(Math.random() * cant_usuarios),
           "tipo": tipos[Math.floor(Math.random() * 5)],
       })
   if (i % 700 == 0) {
       bulk.execute()
       var bulk = db.Poemas.initializeUnorderedBulkOp();
   }
}
bulk.execute()
```

Ejemplo de documento de la colección *Poemas*:

```
_id: ObjectId('636face72e43904e1084864d')
titulo: "titulo 0"
contenido: "contenido 0"
autor: "Usuario2080"
autor_id: ObjectId('636fa9d42e43904e108450f6')
fecha: 2016-09-11T03:00:00.000+00:00
ubicacion: "Sudafrica"
promedio: 10
cant_visualizaciones: 2092
tipo: "comedia"
```

Crear y poblar la colección *ComentariosPorPoema*

```
db.createCollection("ComentariosPorPoema");
function comentar(usuarios) {
    comentarios = ["Horrible", "Malisimo", "Muy Malo", "Malo", "No me gusto", "Regular", "Me
gusto", "Bueno", "Buenisimo", "Perfecto"]
   cant_comentarios = Math.floor(Math.random() * 50);
   list_comentarios = []
   list_usuarios = []
   for (var i = 0; i < cant_usuarios; i++) {
       list_usuarios.push("Usuario"+i)
   }
   if (Math.floor(Math.random() * 2) == 1 ) {
       for (var i = 0; i < cant_comentarios; i++) {
           indice_list_usuarios = Math.floor(Math.random() * list_usuarios.length)
           usuario_id = usuarios[indice_list_usuarios]._id
           usuario = usuarios[indice_list_usuarios].nombre
           list_usuarios.splice(indice_list_usuarios,1)
           puntaje = Math.floor(Math.random() * 10) +
           dict_comentario = {};
           dict_comentario["puntaje"] = puntaje;
           dict_comentario["texto"] = comentarios[puntaje-1];
           dict_comentario["usuario"] = usuario;
           dict_comentario["usuario_id"] = usuario_id;
           list_comentarios.push(dict_comentario);
       }
   }
   return list_comentarios;
}

var poemas = db.Poemas.find()
var usuarios = db.Usuarios.find().toArray()
var bulk = db.ComentariosPorPoema.initializeUnorderedBulkOp();
for (var i = 0; i < cant_poemas; i++) {
   poema = poemas.next()
   poema_titulo = poema.titulo
   poema_id = poema._id
   bulk.insert({
               "poema": poema_titulo,
               "poema_id": poema_id,
               "comentarios": comentar(usuarios),
           })
   if (i % 700 == 0) {
       bulk.execute()
       var bulk = db.ComentariosPorPoema.initializeUnorderedBulkOp();
   }
}
bulk.execute()
```

Ejemplo de documento de la colección *ComentariosPorPoema*:

Crear y poblar la colección *Tipo*

```
function rellenar_top(genero, orden) {
   var tops10 = (db.Poemas.aggregate([
               {$match : { tipo : genero }},
               {$sort: {promedio:orden}},
               {$project:{ titulo : 1 , promedio : 1 , _id: 0 }}
           ]).pretty()).toArray()
   return tops10;
}
function rellenar_virales(genero) {
   var virales = (db.Poemas.aggregate([
               {$match : { tipo : genero }},
               {$sort: {cant_visualizaciones:-1}},
               {$project:{ titulo : 1 , cant_visualizaciones: 1 , _id: 0 }}
           ]
       ).pretty()).toArray()
   return virales;
}
function rellenar_recientes(genero) {
   var recientes = (db.Poemas.aggregate([
               {$match : { tipo : genero }},
               {$sort: {fecha:-1}},
               {$project:{ titulo : 1 , fecha : 1, _id: 0 }}
           ]).pretty()).toArray()
   return recientes;
}
function rellenar_latinos(genero) {
   var latinos = (db.Poemas.aggregate([
               {$match : { tipo : genero }},
               {$match : { ubicacion : { $in:["Argentina", "Chile", "Peru", "Brasil"] }}},
               {$sort: {ubicacion:1}},
               {$project:{ titulo : 1 , ubicacion : 1, _id: 0 }}
           ]).pretty()).toArray()
   return latinos;
}
db.createCollection("Tipo")
const generos = ["romantico", "comedia", "terror", "ciencia ficcion", "educativo"]
for (var i = 0; i < generos.length; i++) {
   db.Tipo.insert(
       {
           "genero": generos[i],
           "top": rellenar_top(generos[i], -1),
           "not_top": rellenar_top(generos[i], 1),
           "virales": rellenar_virales(generos[i]),
           "recientes": rellenar_recientes(generos[i]),
           "latinos":rellenar_latinos(generos[i]),
       }
   )
}
```

Ejemplo de documento de la colección *Tipo*:

Ejemplo de documento del Array *top* de la colección *Tipo*:

Una vez creadas y pobladas las cuatro colecciones de *chimichanga* hay que actualizar los campos *poemasPublicados* y *comentariosHechos* de la colección *Usuarios* con sus respectivos valores. Para ello utilizamos los siguientes scripts:

Actualizar el campo *poemasPublicados* de la colección *Tipo*

```
const usuarios = db.Usuarios.find({}).toArray()
const poemas = db.Poemas.find()
dict = {}
for (var i=0; i< usuarios.length; i++) {
   dict[usuarios[i]._id] = 0
}
while (poemas.hasNext()) {
   autor = poemas.next().autor_id
   dict[autor] = dict[autor] + 1
}
var bulk = db.Usuarios.initializeUnorderedBulkOp();
for (var i=0; i< usuarios.length; i++) {
   bulk.find({_id: usuarios[i]._id}).update( {$set:{poemasPublicados: dict[usuarios[i]._id]}})
   if (i % 700 == 0) {
       bulk.execute()
       var bulk = db.Usuarios.initializeUnorderedBulkOp();
   }
}
bulk.execute()
```

Actualizar el campo *comentariosHechos* de la colección *Tipo*:

```
const usuarios = db.Usuarios.find({}).toArray()
const comentariosPorPoema = db.ComentariosPorPoema.find()
dict = {}
for (var i=0; i< usuarios.length; i++) {
   dict[usuarios[i]._id] = 0
}
while (comentariosPorPoema.hasNext()) {
   arraycomentarios = comentariosPorPoema.next().comentarios
   for (var i=0; i< arraycomentarios.length; i++ ) {
       usuario = arraycomentarios[i].usuario_id
       dict[usuario] = dict[usuario] + 1
   }
}
var bulk = db.Usuarios.initializeUnorderedBulkOp();
for (var i=0; i< usuarios.length; i++) {
   bulk.find({_id: usuarios[i]._id}).update({$set:{comentariosHechos: dict[usuarios[i]._id]}})
   if (i % 700 == 0) {
       bulk.execute()
       var bulk = db.Usuarios.initializeUnorderedBulkOp();
   }
}
bulk.execute()
```

Ejemplo de documento de la colección *Usuarios* actualizada:

```
_id: ObjectId('636fa9d42e43904e108448d7')
nombre: "Usuario1"
mail: "Usuario1@email.com"
contraseña: "123456789"
promedio: 6
poemasPublicados: 5
comentariosHechos: 60
```

Finalmente la base de datos *chimichanga* quedaría de esta manera:

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
ComentariosPorPoema	50000	56.58MB	1.16KB	13.31MB	1	1.72MB	1.72MB
Poemas	50000	10.52MB	221B	4.05MB	1	1.69MB	1.69MB
Tipo	5	10.67MB	2.13MB	2.83MB	1	24KB	24KB
Usuarios	10000	1.56MB	164B	864KB	1	912KB	912KB

Consultas

A continuación obtendremos información de interés de las colecciones a través de las siguientes consultas:

Poemas de ciencia ficción con promedio mayor o igual a 9

Consulta

```
db.Poemas.aggregate([
            {$match : {promedio:{$gte:9}, tipo: "ciencia ficcion"}},
            {$project:{ _id: 0 , titulo: 1 , autor_nombre: 1, promedio: 1 }},
            {$sort: {promedio:-1}},
])
```

OutPut

```
[
  { titulo: 'titulo 43', promedio: 10 },
  { titulo: 'titulo 57', promedio: 10 },
  { titulo: 'titulo 99', promedio: 10 },
  { titulo: 'titulo 102', promedio: 10 },
  { titulo: 'titulo 110', promedio: 10 },
  { titulo: 'titulo 129', promedio: 10 },
  { titulo: 'titulo 172', promedio: 10 },
  { titulo: 'titulo 213', promedio: 10 },
  { titulo: 'titulo 241', promedio: 10 },
  { titulo: 'titulo 321', promedio: 10 },
  { titulo: 'titulo 323', promedio: 10 },
  { titulo: 'titulo 329', promedio: 10 },
  { titulo: 'titulo 409', promedio: 10 },
  { titulo: 'titulo 424', promedio: 10 },
  { titulo: 'titulo 481', promedio: 10 },
  { titulo: 'titulo 506', promedio: 10 },
  { titulo: 'titulo 554', promedio: 10 },
  { titulo: 'titulo 623', promedio: 10 },
  { titulo: 'titulo 629', promedio: 10 },
  { titulo: 'titulo 664', promedio: 10 }
]
Type "it" for more
```

10 primeros poemas publicados en el 2021

Consulta

```
db.Poemas.aggregate([
            {$match: {fecha: {"$gte": ISODate("2021-01-01T00:00:00Z")}}},
            {$project:{ _id: 0 , titulo:1, fecha: 1}},
            {$sort: {fecha: -1}},
            {$limit : 10 },
])
```

OutPut

```
{ "titulo" : "titulo 7108", "fecha" : ISODate("2021-12-27T03:00:00Z") }
{ "titulo" : "titulo 13563", "fecha" : ISODate("2021-12-27T03:00:00Z") }
{ "titulo" : "titulo 18211", "fecha" : ISODate("2021-12-27T03:00:00Z") }
{ "titulo" : "titulo 1224", "fecha" : ISODate("2021-12-27T03:00:00Z") }
{ "titulo" : "titulo 9346", "fecha" : ISODate("2021-12-27T03:00:00Z") }
{ "titulo" : "titulo 14027", "fecha" : ISODate("2021-12-27T03:00:00Z") }
{ "titulo" : "titulo 24849", "fecha" : ISODate("2021-12-27T03:00:00Z") }
{ "titulo" : "titulo 19805", "fecha" : ISODate("2021-12-27T03:00:00Z") }
{ "titulo" : "titulo 13125", "fecha" : ISODate("2021-12-27T03:00:00Z") }
{ "titulo" : "titulo 3008", "fecha" : ISODate("2021-12-27T03:00:00Z") }
```

Poemas educativos publicados en Europa

Consulta

```
db.Poemas.aggregate([
            {$match : { tipo : "educativo" }},
            {$match :{ubicacion :{ $in:["Italia","Francia","España","Alemania", "Portugal"]}}},
            {$sort: {titulo: 1}},
            {$project:{ titulo : 1 , ubicacion : 1, _id: 0 }}
])
```

OutPut

```
[
  { titulo: 'titulo 10026', ubicacion: 'Italia' },
  { titulo: 'titulo 10034', ubicacion: 'Portugal' },
  { titulo: 'titulo 10045', ubicacion: 'España' },
  { titulo: 'titulo 10053', ubicacion: 'Italia' },
  { titulo: 'titulo 10097', ubicacion: 'España' },
  { titulo: 'titulo 10114', ubicacion: 'España' },
  { titulo: 'titulo 10133', ubicacion: 'Portugal' },
  { titulo: 'titulo 10168', ubicacion: 'Alemania' },
  { titulo: 'titulo 10202', ubicacion: 'España' },
  { titulo: 'titulo 10220', ubicacion: 'Portugal' },
  { titulo: 'titulo 10271', ubicacion: 'Francia' },
  { titulo: 'titulo 10281', ubicacion: 'Italia' },
  { titulo: 'titulo 10288', ubicacion: 'Francia' },
  { titulo: 'titulo 10294', ubicacion: 'España' },
  { titulo: 'titulo 10300', ubicacion: 'España' },
  { titulo: 'titulo 10319', ubicacion: 'Portugal' },
  { titulo: 'titulo 10325', ubicacion: 'Alemania' },
  { titulo: 'titulo 10335', ubicacion: 'Francia' },
  { titulo: 'titulo 10369', ubicacion: 'Alemania' },
  { titulo: 'titulo 10387', ubicacion: 'Portugal' }
]
Type "it" for more
```

País donde fue escrito el poema más viral

Consulta

```
db.Poemas.aggregate([
            {$sort: {cant_visualizaciones:-1, cant_comentarios: 1}},
            {$project:{ titulo : 1 , ubicacion: 1 , cant_visualizaciones: 1, _id: 0 }},
            {$limit : 1 },
 ]).pretty()
```

OutPut

```
[
  {
    titulo: 'titulo 3239',
    ubicacion: 'Japon',
    cant_visualizaciones: 9999
  }
]
```

Cantidad de comentarios tiene el poema “titulo 25000”

Consulta

```
db.ComentariosPorPoema.aggregate([
            {$match : { poema : "titulo 25000" }},
            {$project: {poema: 1   , count : { $size:"$comentarios" }, _id: 0 }}
])
```

OutPut

```
[ { poema: 'titulo 25000', count: 49 } ]
```

Mail del autor del poema con más visualizaciones de EEUU

Consulta

```
db.Poemas.aggregate([
            {$lookup:{
                from: "Usuarios",
                localField:"autor_id",
                foreignField:"_id",
                as: "usuario",
                }
            },
            {$match: {ubicacion: "Estados Unidos"}},
            {$project:{ _id: 0, titulo:1 , "usuario.nombre": 1,
                "usuario.mail": 1, cant_visualizaciones: 1}},
            {$sort: {cant_visualizaciones: -1}},
            {$limit : 1},
])
```

OutPut

```
[
  {
    titulo: 'titulo 890',
    cant_visualizaciones: 9990,
    usuario: [ { nombre: 'Usuario7000', mail: 'Usuario7000@email.com' } ]
  }
]
```

Ver los 10 poemas mas polémicos de cualquier género, es decir con la mayor cantidad de comentarios

Consulta

```
db.ComentariosPorPoema.aggregate([
            {$lookup:{
                from: "Poemas",
                localField:"poema_id",
                foreignField:"_id",
                as: "poemas"}},
                {$project:{ _id: 0 , poema: 1 , "poemas.autor":1,"poemas.promedio":1,cantidad:
{$size: "$comentarios"} }},
            {$sort: {cantidad: -1}},
            {$limit : 10 },
        ])
```

OutPut


```
[
  {
    poema: 'titulo 45',
    poemas: [ { autor: 'Usuario3277', promedio: 10 } ],
    cantidad: 49
  },
  {
    poema: 'titulo 363',
    poemas: [ { autor: 'Usuario9381', promedio: 2 } ],
    cantidad: 49
  },
  {
    poema: 'titulo 642',
    poemas: [ { autor: 'Usuario6597', promedio: 2 } ],
    cantidad: 49
  },
  {
    poema: 'titulo 137',
    poemas: [ { autor: 'Usuario9764', promedio: 10 } ],
    cantidad: 49
  },
  {
    poema: 'titulo 300',
    poemas: [ { autor: 'Usuario3343', promedio: 10 } ],
    cantidad: 49
  },
  {
    poema: 'titulo 388',
    poemas: [ { autor: 'Usuario9358', promedio: 10 } ],
    cantidad: 49
  },
  {
    poema: 'titulo 676',
    poemas: [ { autor: 'Usuario2330', promedio: 10 } ],
    cantidad: 49
  },
  {
    poema: 'titulo 660',
    poemas: [ { autor: 'Usuario3739', promedio: 2 } ],
    cantidad: 49
  },
  {
    poema: 'titulo 519',
    poemas: [ { autor: 'Usuario8184', promedio: 10 } ],
    cantidad: 49
  },
  {
    poema: 'titulo 0',
    poemas: [ { autor: 'Usuario2080', promedio: 10 } ],
    cantidad: 49
  }
]
```

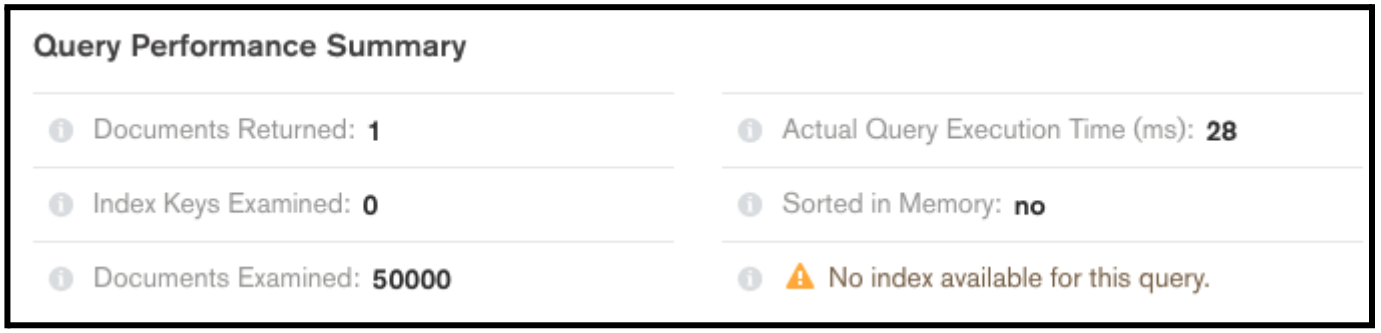
Índices

Mediante la utilización de índices en distintos campos de las colecciones, es posible mejorar considerablemente el tiempo de respuesta en las operaciones de lectura, es decir al momento de realizar consultas. Para ello, detallaremos a continuación, alguno de los resultados obtenidos antes de indexar y luego de hacerlo.

Consulta

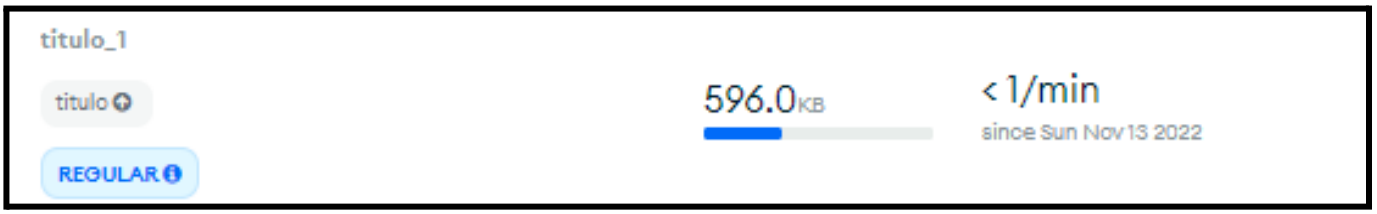
```
db.Poemas.find(
            {titulo:"titulo 49300"}
      ).explain("executionStats").executionStats
```

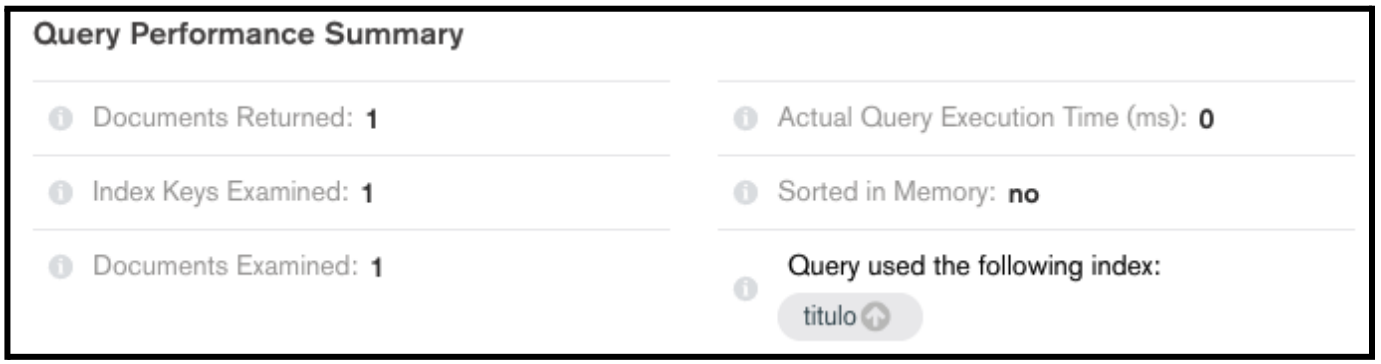
Sin Indexar



Indexado

```
db.Poemas.createIndex({titulo: 1})
```



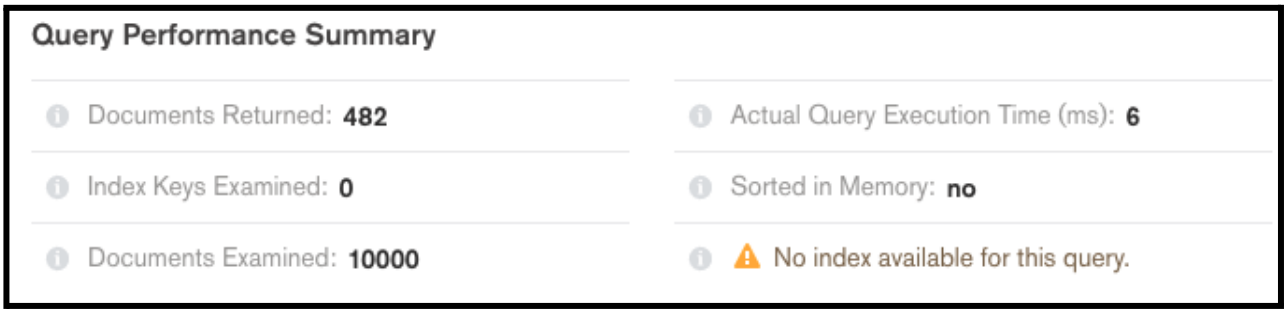


Se puede observar una inmensa mejoría en el tiempo de ejecución de la consulta, llevando el tiempo inicial de 28 ms a 0 ms. Esto es así ya que pasa de recorrer 50000 documentos a ir directamente a dicho documento para encontrar el poema con “título 49300”.

Consulta

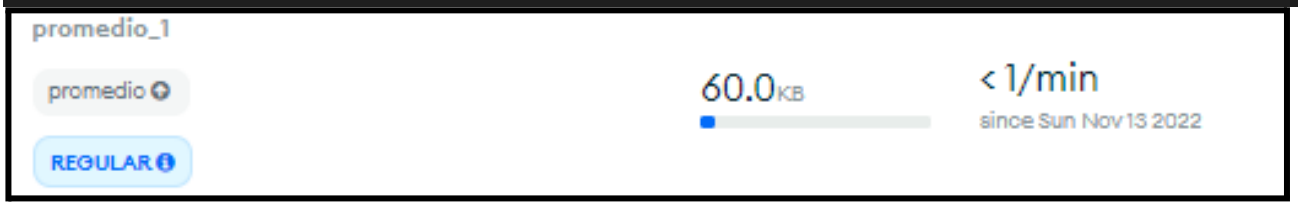
```
db.Usuarios.find({promedio:{$in:[1,10]},
poemasPublicados:{$gt:6}}).explain("executionStats").executionStats
```

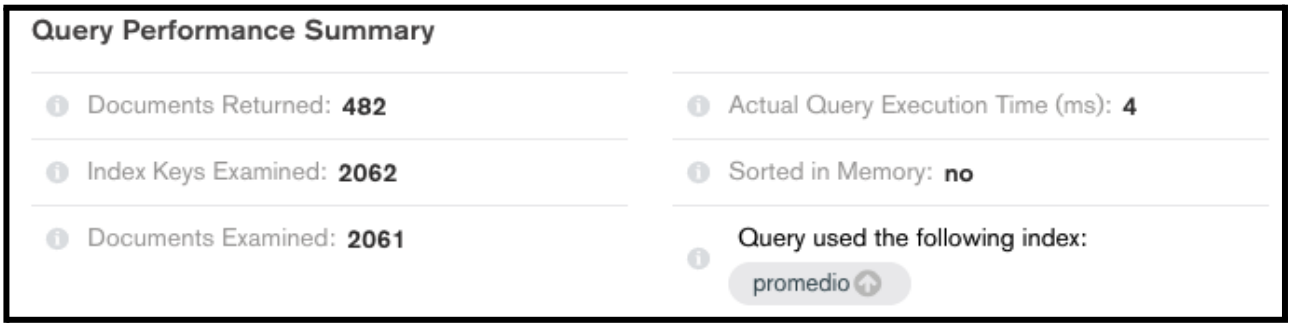
Sin Indexar



Indexado simple

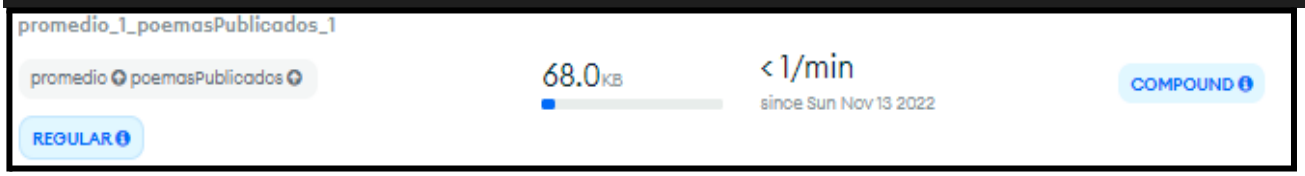
```
db.Usuarios.createIndex({promedio: 1})
```

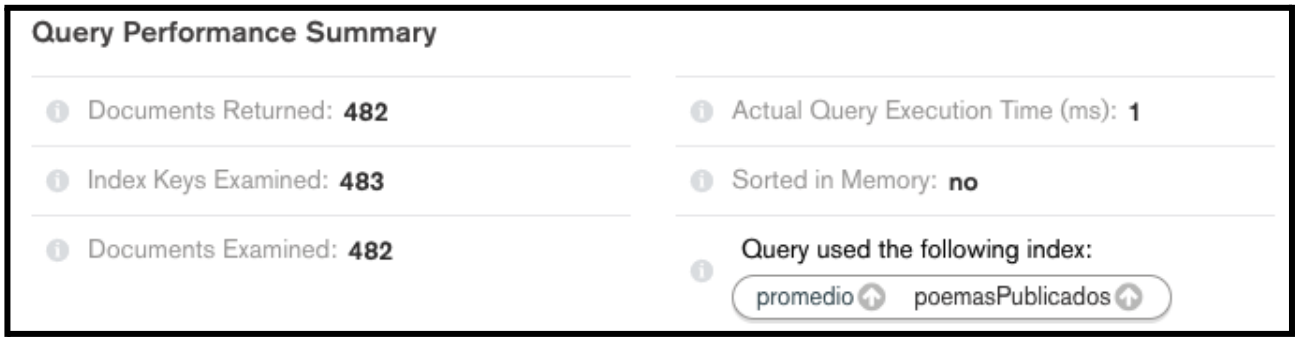




Indexado compuesto

```
db.Usuarios.createIndex({promedio: 1, poemasPublicados: 1})
```



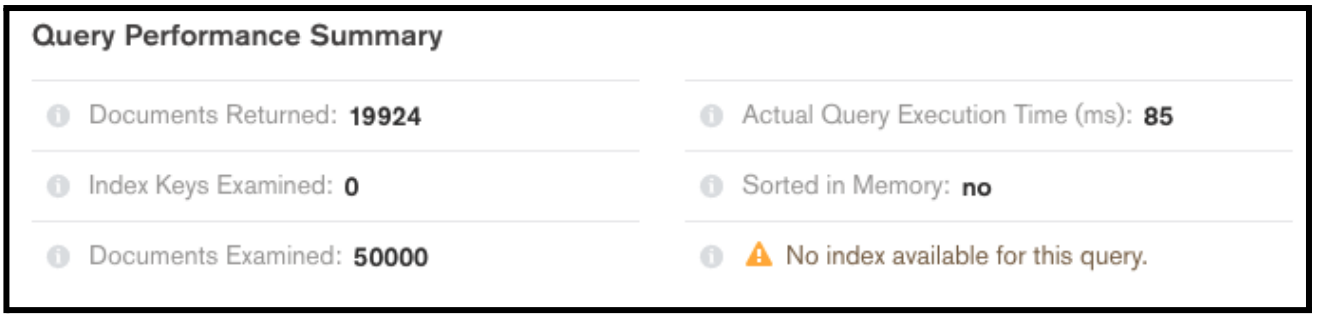


En este caso podemos observar que utilizando un índice compuesto, logramos una velocidad aún mayor que la obtenida con un índice simple.

Consulta

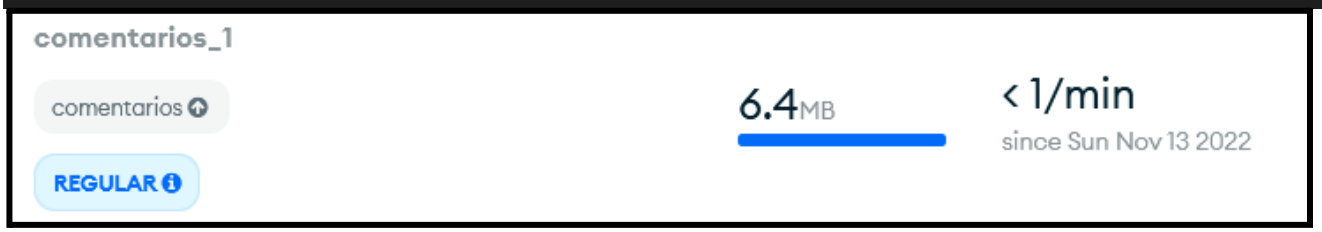
```
db.ComentariosPorPoema.find({"comentarios.puntaje":10}).explain("executionStats").executionStats
```

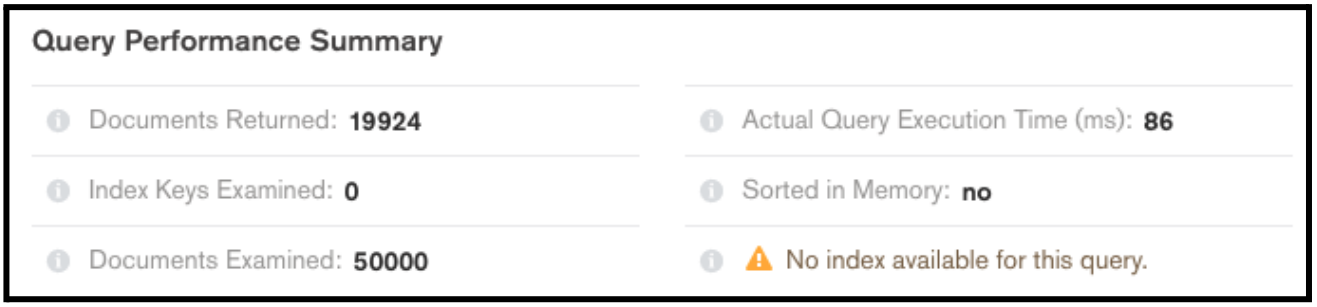
Sin Indexar



Indexado Multikey (al array “comentarios”) NO Funciona

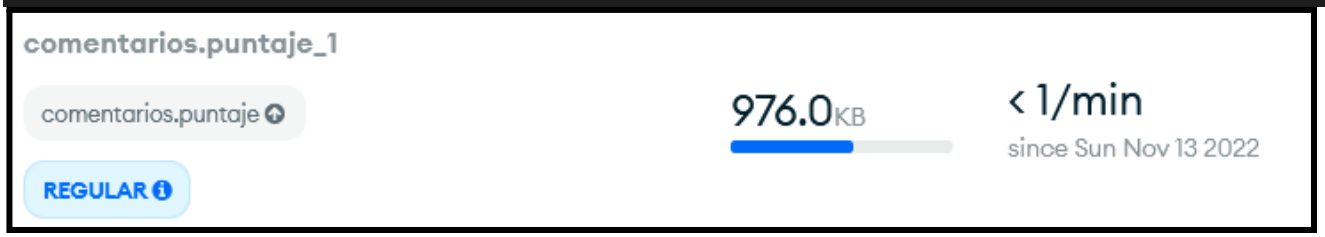
```
db.ComentariosPorPoema.createIndex({comentarios: 1})
```

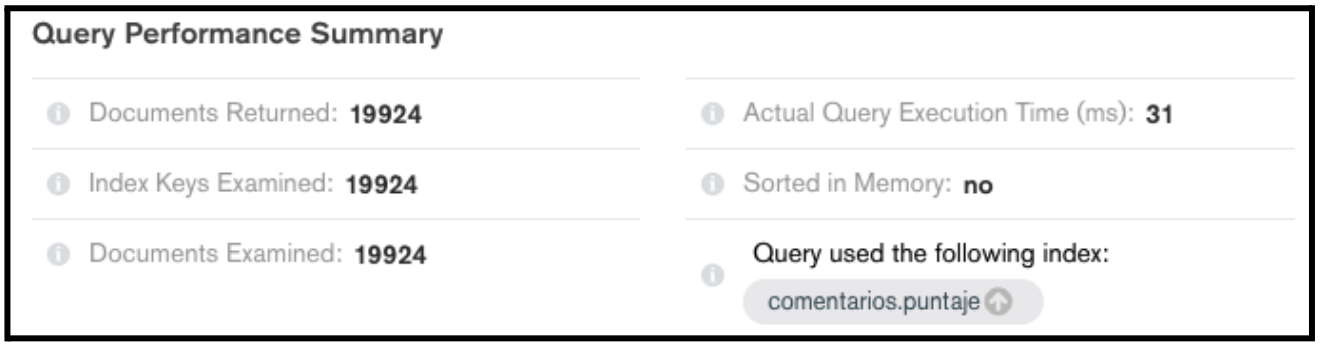




Indexado Multikey (al campo “puntaje” del array “comentarios”)

```
db.ComentariosPorPoema.createIndex({"comentarios.puntaje": 1})
```



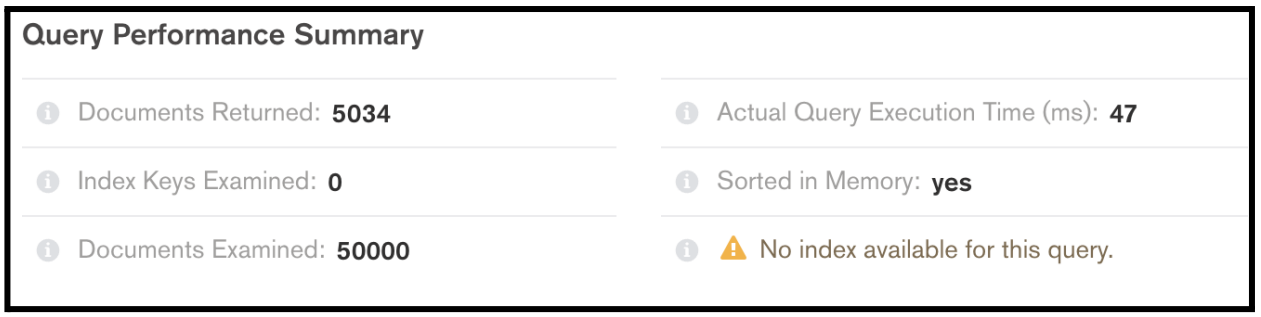


Como podemos ver, al implementar índices de tipo Multikey en un campo de tipo Array, debemos especificar el campo específico de dicho Array al que queremos hacer referencia. Caso contrario, si solo indicamos en array en general, el índice será creado pero no funcionará (además ocupa mucho más espacio de almacenamiento).

Consulta

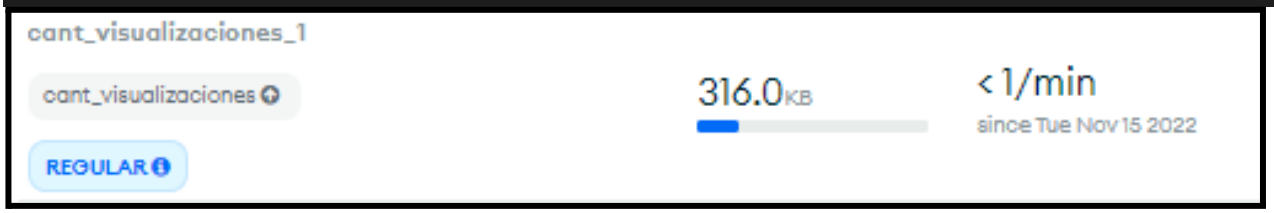
```
db.Poemas.find({cant_visualizaciones: {$gt:9000 }}).sort({cant_visualizaciones:1})
```

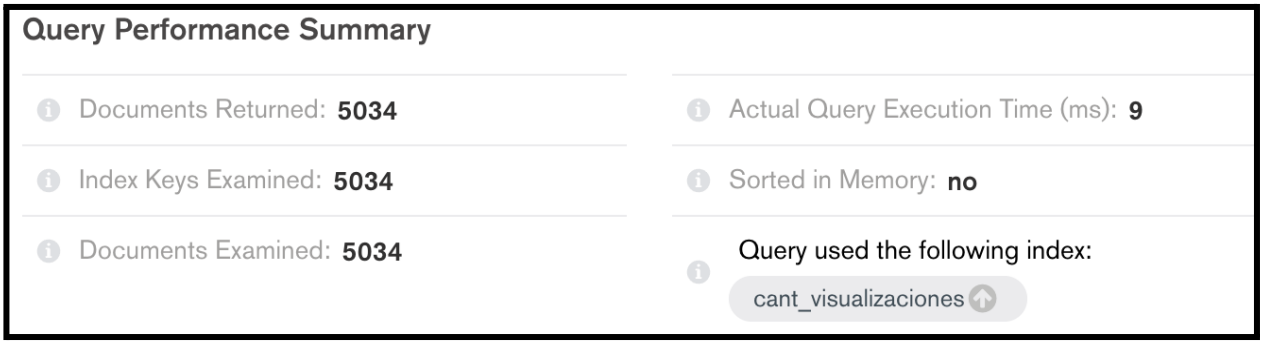
Sin Indexar



Indexado creciente

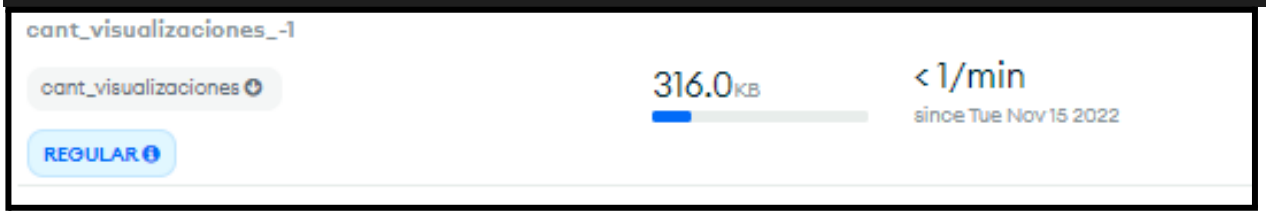
```
db.Poemas.createIndex({cant_visualizaciones: 1})
```

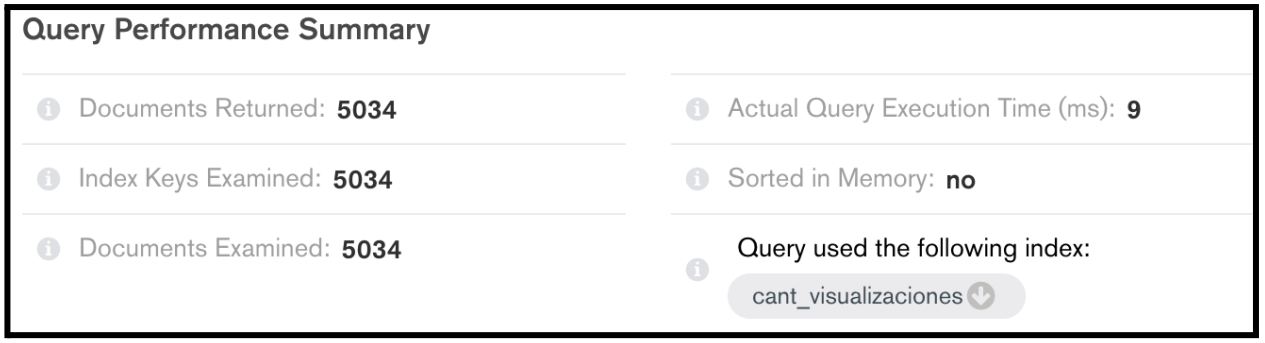




Indexado decreciente

```
db.Poemas.createIndex({cant_visualizaciones: -1})
```



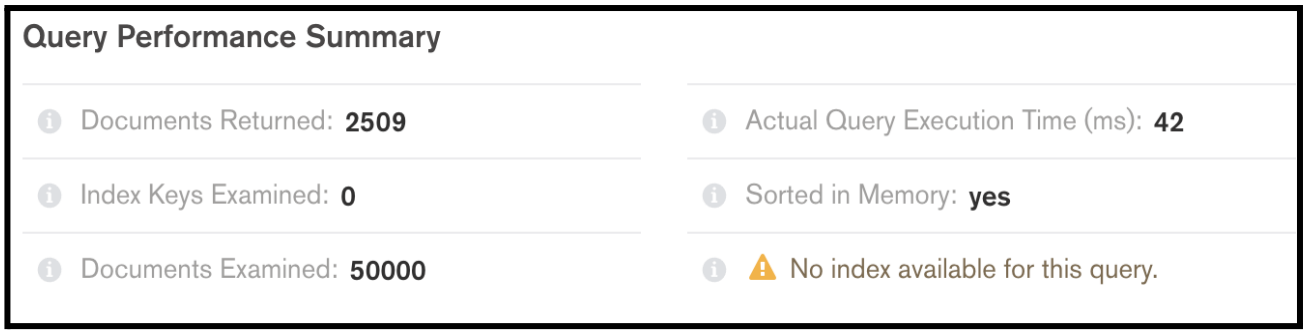


En este caso, vemos una mejoría al usar índices, pero es indiferente modificar el tipo de índice, es decir, que sea ascendente o descendente. No implica cambios en el tiempo de ejecución para casos de índice simple.

Consulta

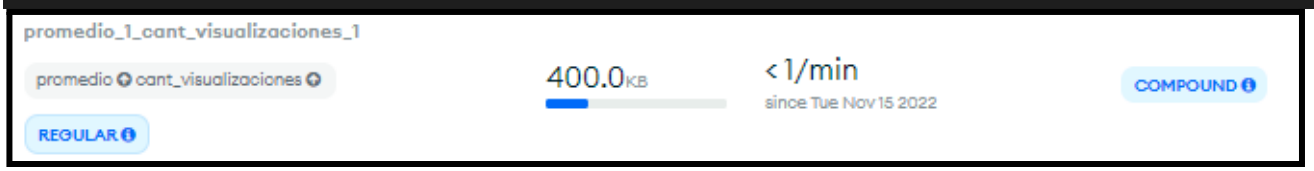
```
db.Poemas.find({promedio:10,cant_visualizaciones:{$gt:5000}}).sort({promedio:1})
```

Sin Indexar



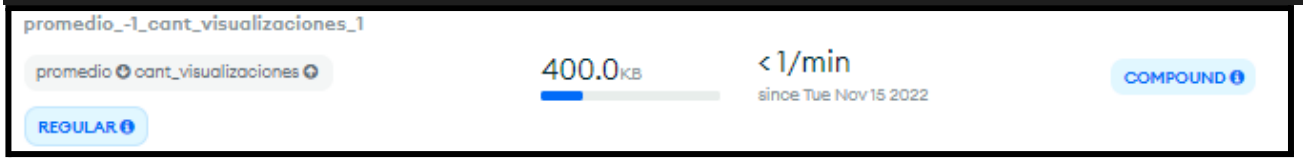
Indexado compuesto creciente

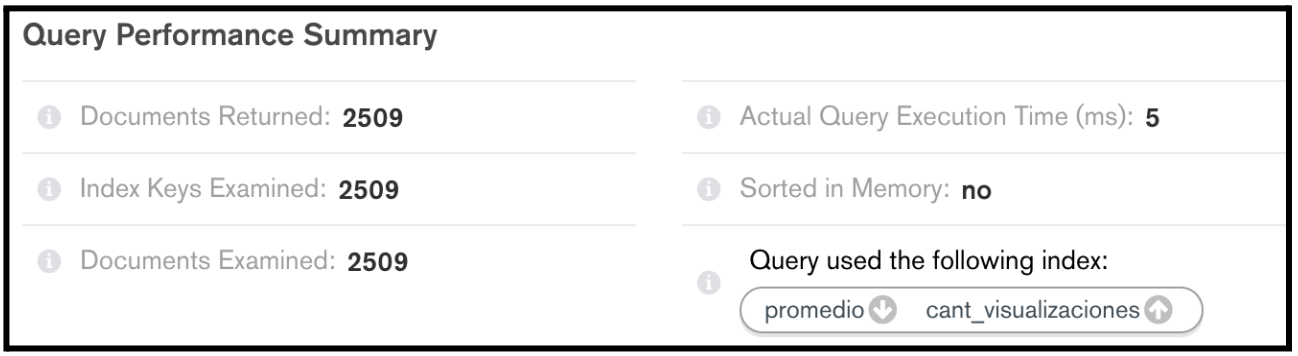
```
db.Poemas.createIndex({promedio:1 , cant_visualizaciones: 1})
```



Indexado compuesto (1er campo decreciente - 2do campo creciente)

```
db.Poemas.createIndex({promedio: -1, cant_visualizaciones: 1})
```





Podemos observar que para esta consulta en específico, mediante la utilización de un índice compuesto, pudimos reducir en gran medida el tiempo de respuesta. Y aún más implementando un índice decreciente. Por lo que es importante tener en cuenta estas herramientas a la hora de decidir qué tipo de índices vamos a crear para nuestro sistema

Backup

El sistema de Backup implementado de MongoDB Atlas no está disponible para los clusters gratis M0. Por lo que recomiendan utilizar Mongodump para realizar las copias de seguridad y Mongorestore para restaurar dichos datos. Mongoexport, Mongodump y Mongorestore son herramientas de terceros, ajenas a los desarrolladores de MongoDB, de línea de comandos

Además se recomienda evitar usar mongoimport y mongoexport para copias de seguridad de producción de instancias completas. Ya que no conservan de forma fiable algunos tipos de datos de BSON, porque JSON solo puede representar un subconjunto de los tipos compatibles con BSON. Una de las diferencias importantes es que mongodump es más rápido que mongoexport para fines de copia de seguridad. Mongodump almacena datos como binario, mientras que mongoexport almacena datos como JSON o CSV.

Por ello, decidimos utilizar mongodump y mongorestore como se describe en los métodos de copia de seguridad de MongoDB para este tipo de funcionalidad.

Gracias a mongodump podemos realizar una descarga masiva de las bases de datos alojadas en nuestro servidor. Esta aplicación genera copias binarias de datos utilizando el formato BSON.

El comando para respaldar la base de datos chimichanga, provista por AMW (Amazon Web Services) y ubicada en los servidores de San Pablo es:

```
mongodump --uri
mongodb+srv://Chimichanga:Chimichanga4985net@clusterchimichanga.vtb19bm.mongodb.net/chimichanga
```

OutPut

Observaciones

Trabajo Colaborativo

Mongo ofrece **gran utilidad para trabajar de manera colaborativa online**, lo que nos permitió trabajar todos juntos, a la vez, sobre un mismo archivo. **PlayGround** (live share de VS Code) y **Mongodb Atlas** (**limitación de 500 MB en su versión gratuita**, por lo que tuvimos que limitar la cantidad de comentarios que tiene un poema y el tamaño de otras colecciones a la hora de poblarlas).

Estas herramientas, nos permitieron evitar distintos problemas, como por ejemplo: Que cada miembro del grupo trabajará sobre una versión desactualizada del script utilizado para poblar la base de datos. Contar con una base de datos en la computadora de cada miembro del grupo o depender de un solo miembro que se encargue de "Hostear" la Base de Datos en su máquina

Poblado de la Base de Datos

Para el proceso de poblar las diferentes colecciones, al comienzo, utilizamos un bucle de **"inserts comunes"**. Pero, **mediante este método el proceso era muy tardado** para ciertas colecciones como "ComentariosPorPoema". Luego, investigando sobre las funcionalidades que ofrece MongoDB, comenzamos a experimentar con las operaciones de tipo "**bulk.insert**". Este método, consta de 2 partes, primero debemos, mediante un bucle, **crear una lista de tareas** de la base de datos y una vez creada, MongoDB ejecuta en paralelo, haciendo uso de varios núcleos a la vez, cada una de estas tareas. De esta manera, reducimos considerablemente el tiempo de ejecución del poblado de cada colección

Algo a tener en cuenta, es que el método "bulk", crea la lista de tareas en un archivo, cuyo límite es de 16 MB por lo que, en algunos casos, se debe realizar la operación "bulk.execute" cada cierta cantidad de de tareas. Para ahorrar problemas con el límite de estos archivos. En nuestra experiencia para la colección "ComentariosPorPoema", la cual es de las más complejas, ejecutar la lista cada 700 tareas resultó ser un punto adecuado. Si lo hacíamos cada 1000 tareas en ocasiones, ocurría que esos 1000 poemas tenían muchos comentarios cada uno y se terminaba superando el límite de archivo, generando distintos problemas.

Estructura de la operación:

Primero debemos crear una variable "bulk" de tipo OrderedBulkOp (para ejecución secuencial) o UnorderedBulkOp (para ejecución en paralelo). Especificando la colección a la que pertenece.

```
var bulk = db.ComentariosPorPoema.initializeUnorderedBulkOp();
```

Luego, dentro de un bucle, debemos insertar la tarea en la lista mediante:

```
bulk.insert(...estructura del documento...)
```

Una vez terminada la creación de la lista de tareas, debemos ejecutarlas mediante:

```
bulk.execute()
```

Para Consultas de mayor complejidad

Para hacer consultas más complejas, que las posibles con “find”, MongoDB ofrece las operaciones de tipo “Aggregation”. Estas operaciones cuentan con distintos métodos según lo que se desee consultar. Algunos ejemplos útiles que implementamos en nuestra base de datos son:

- **lookup** (realiza un “left outer join” a una colección en la misma base de datos, para filtrar en documentos desde la colección llamada)
- **match** (toma los documentos que cumplan la condición especificada)
- **sort** (ordena de manera ascendente o descendente, según el campo especificado)
- **project** (especifica qué campos se deben mostrar o no, con 1 y 0 respectivamente)
- **limit** (Limita la cantidad de documentos que se devuelven en la consulta)

Se debe tener en cuenta que importa el orden en el que se coloquen cada uno de estos campos, ya que se ejecutan secuencialmente (además, algunos pueden ponerse más de una vez en la misma operación). Por ejemplo: se debe poner primero el método “match” y luego, el método “limit”, porque si no, se tomarán solamente los primeros 10 documentos de la colección (si se hace limit:10) y luego verificará la condición sobre esos 10 documentos, en vez de hacerlo para toda la colección y luego mostrar los 10 primeros.