Explorando el uso de índices en mongoDB (Acme-Explorer)

$indexStats (aggregation)

Primero necesitaremos, si no lo tenemos aún, crear un usuario administrador para poder hacer uso del operador $indexStats.

Podemos crear un nuevo usuario administrador con el shell de mongo:

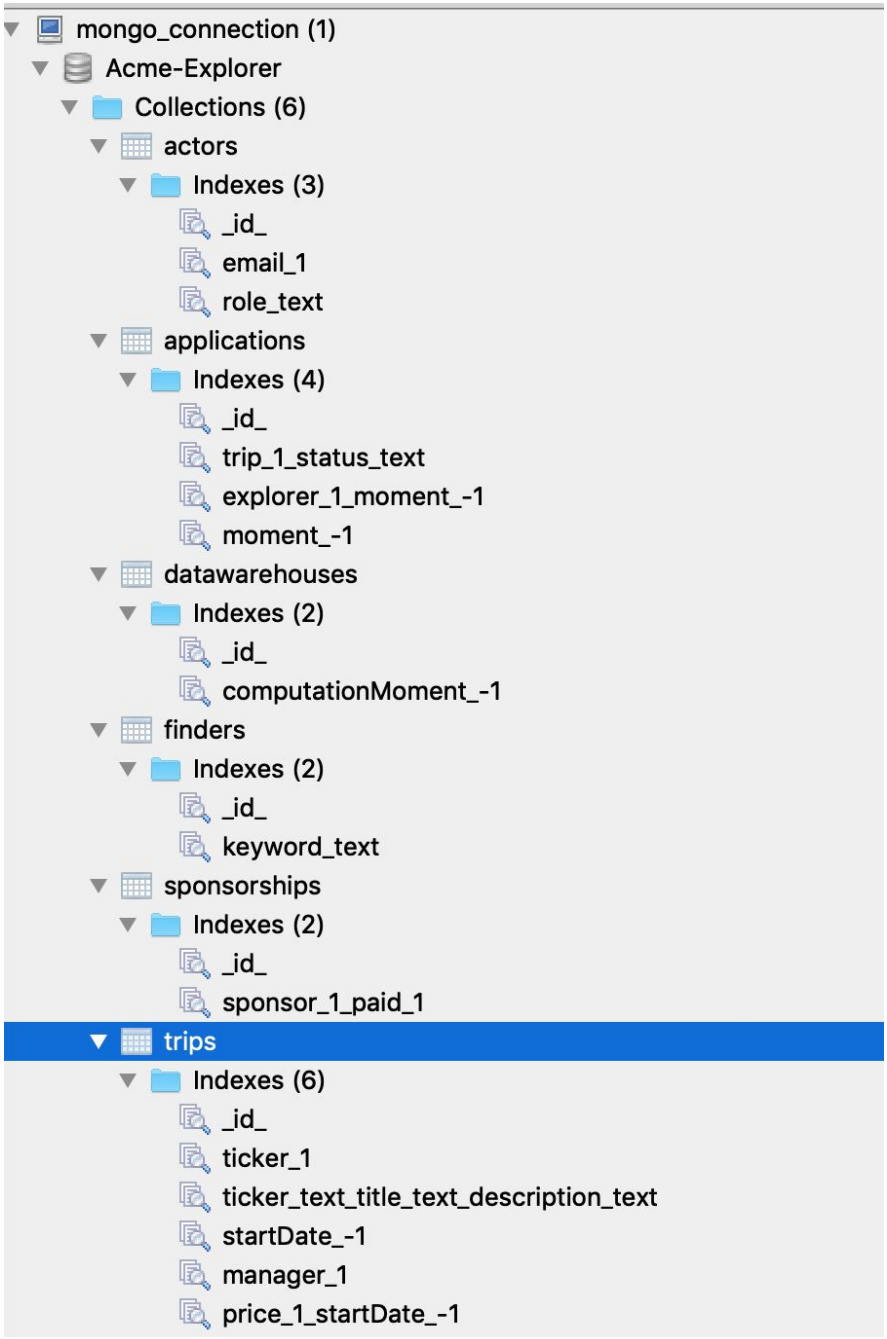
```
use Acme-Explorer
db.createUser( { user: “myAdmin”,
                 pwd: “myAdminPassword”,
                 roles: [ “dbAdmin”] })
```

Después nos conectamos a la base de datos haciendo uso de este nuevo usuario.

Lo ideal sería que nuestra aplicación hubiera estado ya en parte asociada al front-end para que el uso fuera más representativo de lo que cabría esperar una vez se desplegará.

En este caso simularemos el uso de una serie de colecciones y métodos de la API para luego proceder a obtener las estadísticas de los índices asociados.

Nuestra base de datos tiene los siguientes índices (por colección):



Haremos pruebas con los métodos de búsqueda principales de todas las colecciones y posteriormente haremos uso de db.coleccion.aggregate([{ $indexStats: { } }]) con cada colección para echar un vistazo al uso del índice obtenido.

Este comando devuelve a su vez un documento con los siguientes campos:

Output Field	Description
name	Index name
key	Index key specification
host	The hostname and port of the mongod process
accesses.ops	The number of operations that used the index
accesses.since	The time from which MongoDB started gathering the index usage statistics

Fuente: MongoDB Documentation

A nosotros lo que más nos interesa es el nombre del índice y el número de operaciones que lo usaron.

Tabla 1: Pruebas y Resumen de Estadísticas por Colección

Actors	**Se ejecutan pruebas con:** getAllActors getActorById getAllActorsByRole **Index Key | accesses.ops** **role_text = 0** **email_1 = 0** **_id_ = 5** **Observaciones:** Muy extraño que no se use el índice role_text al hacer varias llamadas a getAllActorsByRol, con diversos roles. Se debería revisar la implementación de este índice.
Trips	**Se ejecutan pruebas con:** getAllTrips getTripsSearch

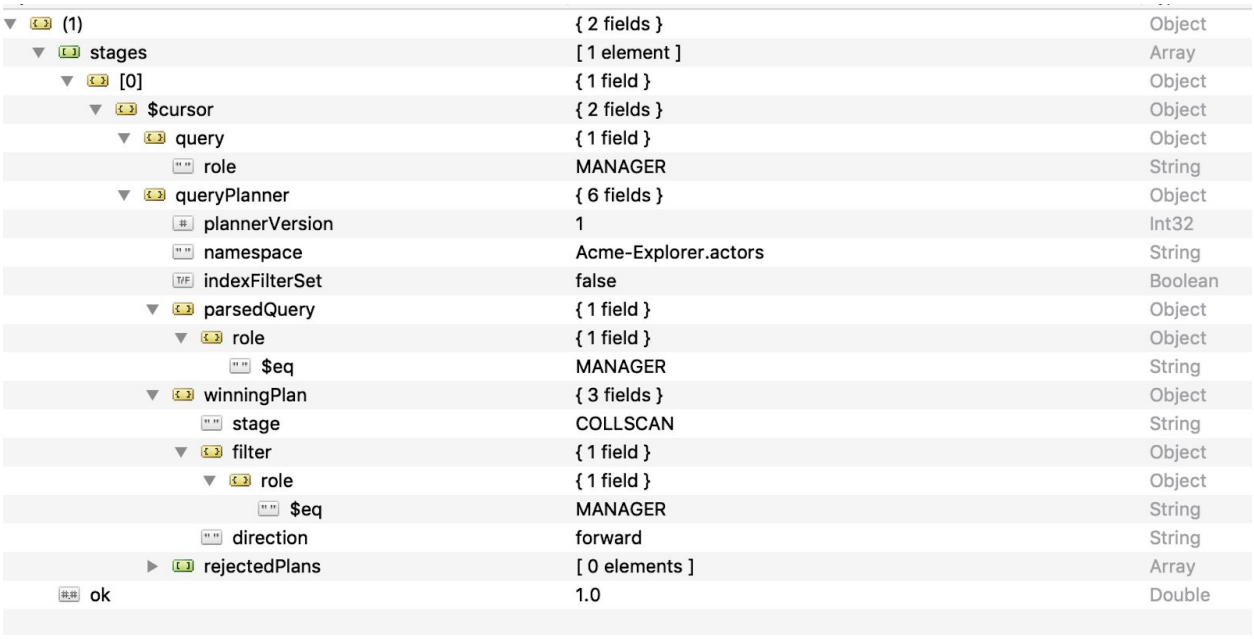
	putTripByID putAddStage **Index Key | accesses.ops** **ticker_1 = 0** **manager_1 = 0** **price_1_startDate_-1=0** **startDate_-1 = 0** **_id_ = 14** **Ticker_text_title_text_description_text = 7** **Observaciones:** Faltan llamadas que se beneficien de algunos de los índices. El índice de campos textuales es muy útil para la llamada getTripsSearch.
Finders	**Se ejecutan pruebas con:** getAllFinders putFinderByID **Index Key | accesses.ops** **keyword_text = 0** **_id_ = 1** **Observaciones:** Faltan llamadas que se utilicen el índice keyword_text.

explain()

Con el operador explain podemos hacernos una idea sobre qué estrategia ha decidido mongodb utilizar al ejecutar una query, que índice, etc.

Por ejemplo, investiguemos qué ocurre cuando buscamos a un actor por rol.

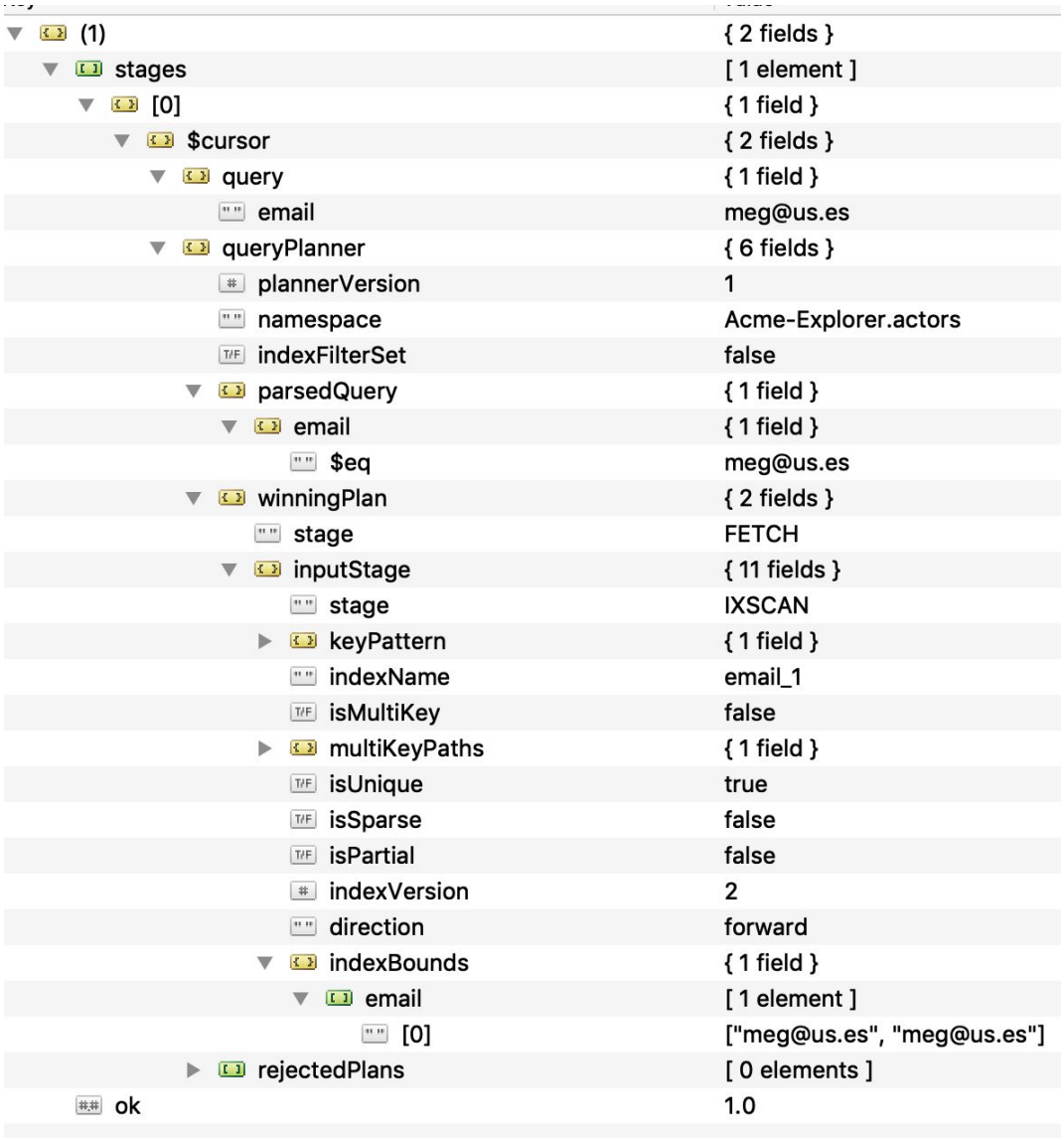
db.actors.explain("executionStats").aggregate([
{$match:{role:"MANAGER"}}
]);



(1)	{ 2 fields }	Object
stages	[1 element]	Array
[0]	{ 1 field }	Object
$cursor	{ 2 fields }	Object
query	{ 1 field }	Object
role	MANAGER	String
queryPlanner	{ 6 fields }	Object
plannerVersion	1	Int32
namespace	Acme-Explorer.actors	String
indexFilterSet	false	Boolean
parsedQuery	{ 1 field }	Object
role	{ 1 field }	Object
$eq	MANAGER	String
winningPlan	{ 3 fields }	Object
stage	COLLSCAN	String
filter	{ 1 field }	Object
role	{ 1 field }	Object
$eq	MANAGER	String
direction	forward	String
rejectedPlans	[0 elements]	Array
ok	1.0	Double

No hace uso del índice existente. Sin embargo, parece que el índice por email si está en siendo utilizado. Lo podemos ver realizando una búsqueda por email:

db.actors.explain('executionStats').aggregate([
{$match:{email:"meg@us.es"}}
]);



Change Events

Para hacer un uso en tiempo real de nuestra base de datos tenemos que crearla en modo replica.

Fuente:
https://blog.usejournal.com/using-mongodb-as-realtime-db-with-nodejs-c6f52c266750

Hacemos lo siguiente:

Cerramos todas las instancias existentes de mongodb.
Luego ejecutamos:

$ mongod --port 27017 --replSet rs0

Esto inicializa mongodb en modo replica con nombre rs0, lo cual es necesario para que funcione change streams.

Ahora antes de crear la base de datos debemos inicializar la réplica, en otra ventana de comandos lanzamos mongo y luego en el shell de mongo escribiremos:

rs.initiate()

Luego creamos una nueva base de datos, podemos usar el comando:

use Acme-Explorer

Después tenemos que conectarnos a nuestra BD, la uri de la base de datos se modificará, deberá incluir un nuevo parámetro:

?replicaSet=rs0

Con estos cambios ya podremos hacer uso de watchers en nuestras colecciones. Es conveniente realizar programáticamente el setup del set de réplica, podríamos crear una función que llamar al inicio, antes de conectarse a la base de datos:

Esto nos lo podemos saltar si ya hemos inicializado la réplica a mano, pero yo utilizo esta opción en local, en producción no podría utilizarse:

```
// Boilerplate to start a new replica set. You can skip this if you already
// have a replica set running locally or in MongoDB Atlas.
async function setupReplicaSet() {
  const bind_ip = 'localhost';
  // Starts a 3-node replica set on ports 31000, 31001, 31002, replica set
  // name is "rs0".
  const replSet = new ReplSet('mongod', [
    { options: { port: 31000, dbpath: `${__dirname}/data/db/31000`, bind_ip } },
```

```
    { options: { port: 31001, dbpath: `${__dirname}/data/db/31001`, bind_ip } },
    { options: { port: 31002, dbpath: `${__dirname}/data/db/31002`, bind_ip } }
  ], { replSet: 'rs0' });

  // Initialize the replica set
  await replSet.purge();
  await replSet.start();
  console.log(new Date(), 'Replica set started...');
}
```

Fuente: http://thecodebarbarian.com/stock-price-notifications-with-mongoose-and-mongodb-change-streams

(Para usar esta opción es necesario descargar un paquete de npm con el módulo mongodb-topology-manager)

Tener en cuenta que ahora la URI de conexión a nuestra DB cambiaría por:

const uri = 'mongodb://localhost:31000,localhost:31001,localhost:31002/' +
'Acme-Explorer?replicaSet=rs0';

Si una de nuestras colecciones tiene un watcher activo, cada vez que se produzca un cambio en la colección podremos ejecutar código definido en un callback.