15 Indexación: Estrategias

The best indexes for your application must take a number of factors into account, including the kinds of queries you expect, the ratio of reads to writes, and the amount of free memory on your system.

When developing your indexing strategy you should have a deep understanding of your application's queries. Before you build indexes, map out the types of queries you will run so that you can build indexes that reference those fields. Indexes come with a performance cost, but are more than worth the cost for frequent queries on large data sets. Consider the relative frequency of each query in the application and whether the query justifies an index.

The best overall strategy for designing indexes is to profile a variety of index configurations with data sets similar to the ones you'll be running in production to see which configurations perform best. Inspect the current indexes created for your collections to ensure they are supporting your current and planned queries. If an index is no longer used, drop the index.

Generally, MongoDB only uses one index to fulfill most queries. However, each clause of an $or query may use a different index, and in addition, MongoDB can use an intersection of multiple indexes.

Create Indexes to Support Your Queries

Create a Single-Key Index if All Queries Use the Same, Single Key

If you only ever query on a single key in a given collection, then you need to create just one single-key index for that collection.

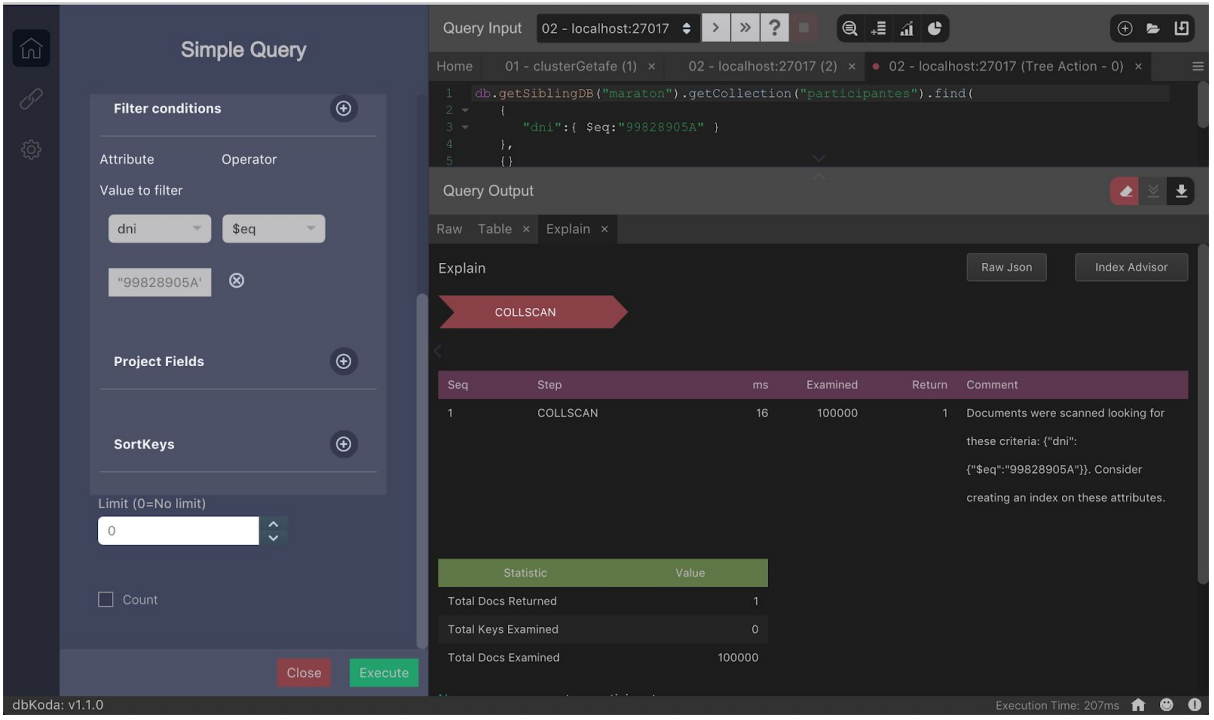
Práctica

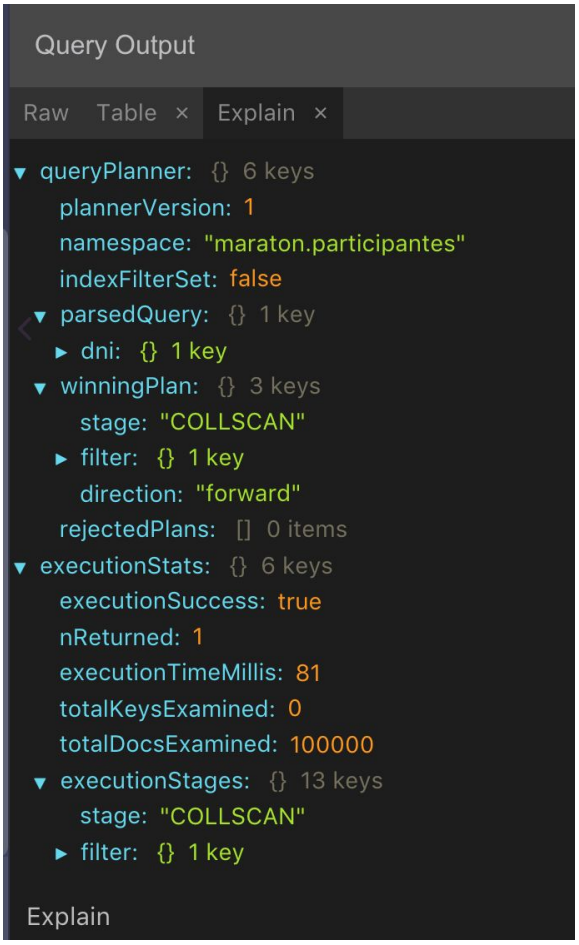
> use maraton

```
let nombres = ['Laura','Juan','Fernando','María'];
let apellidos = ['Fernández','González','Pérez','López'];
let letras = ['A','B','P','C','X','D'];
let participantes = []
for (i=0; i < 100000; i++) {
   participantes.push({
       _id: i,
       nombre: nombres[Math.floor(Math.random()*nombres.length)],
       apellido1: apellidos[Math.floor(Math.random()*apellidos.length)],
       apellido2: apellidos[Math.floor(Math.random()*apellidos.length)],
       edad: Math.floor(Math.random()*100),
       dni: Math.floor(Math.random()* 100000000 ) +
letras[Math.floor(Math.random()*letras.length)]
   })
}

db.participantes.insert(participantes);
```

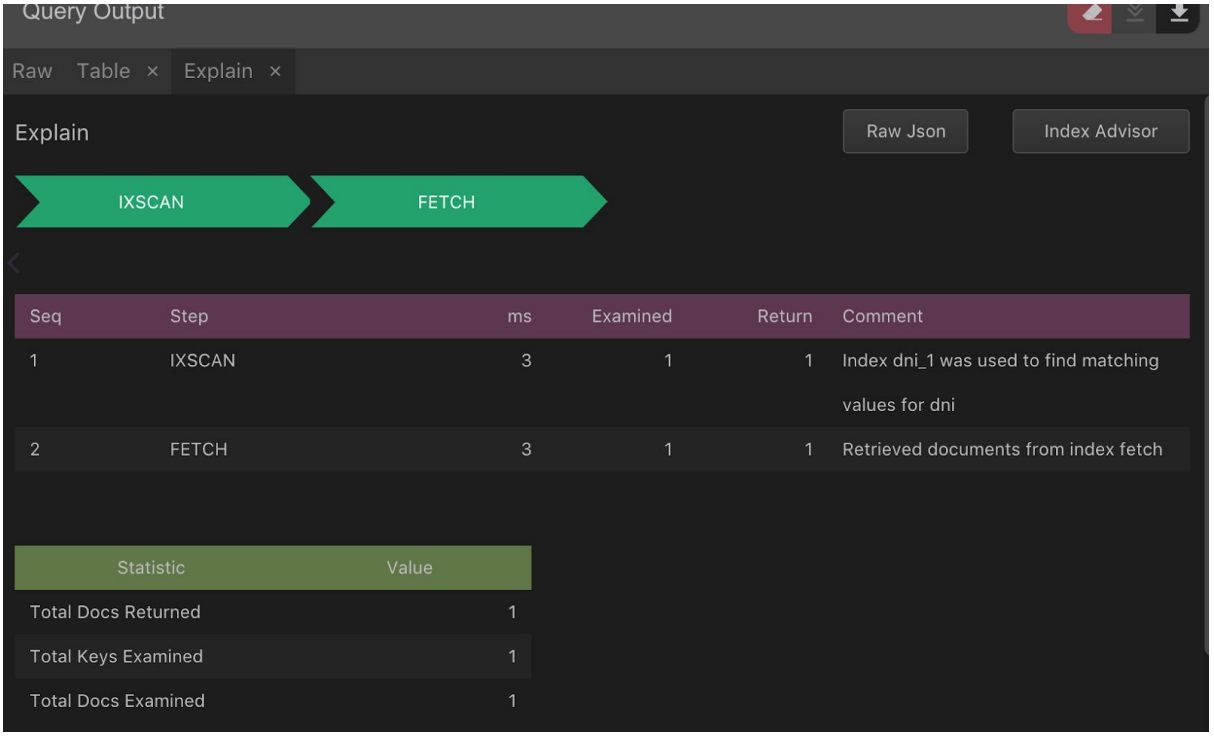
Consulta por DNI sin índice:

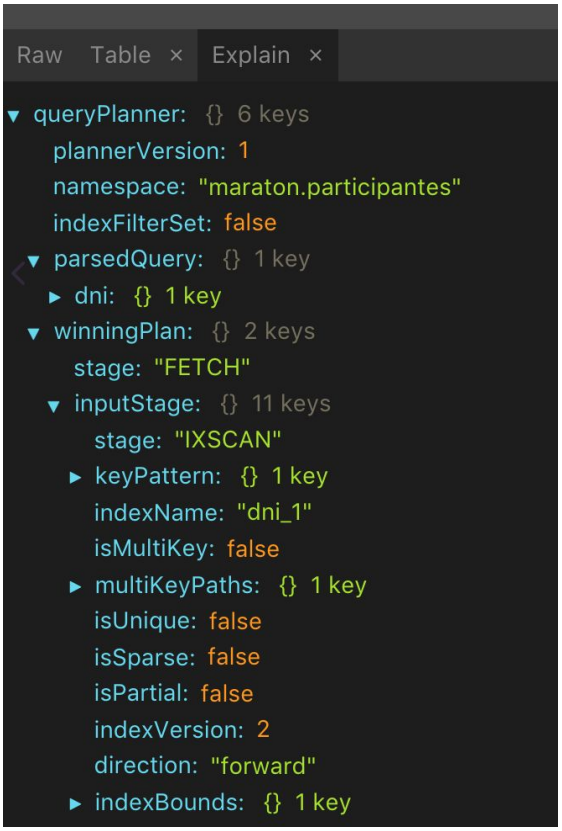




Ahora creamos un índice

```
> db.participantes.createIndex({dni: 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```





Create Compound Indexes to Support Several Different Queries

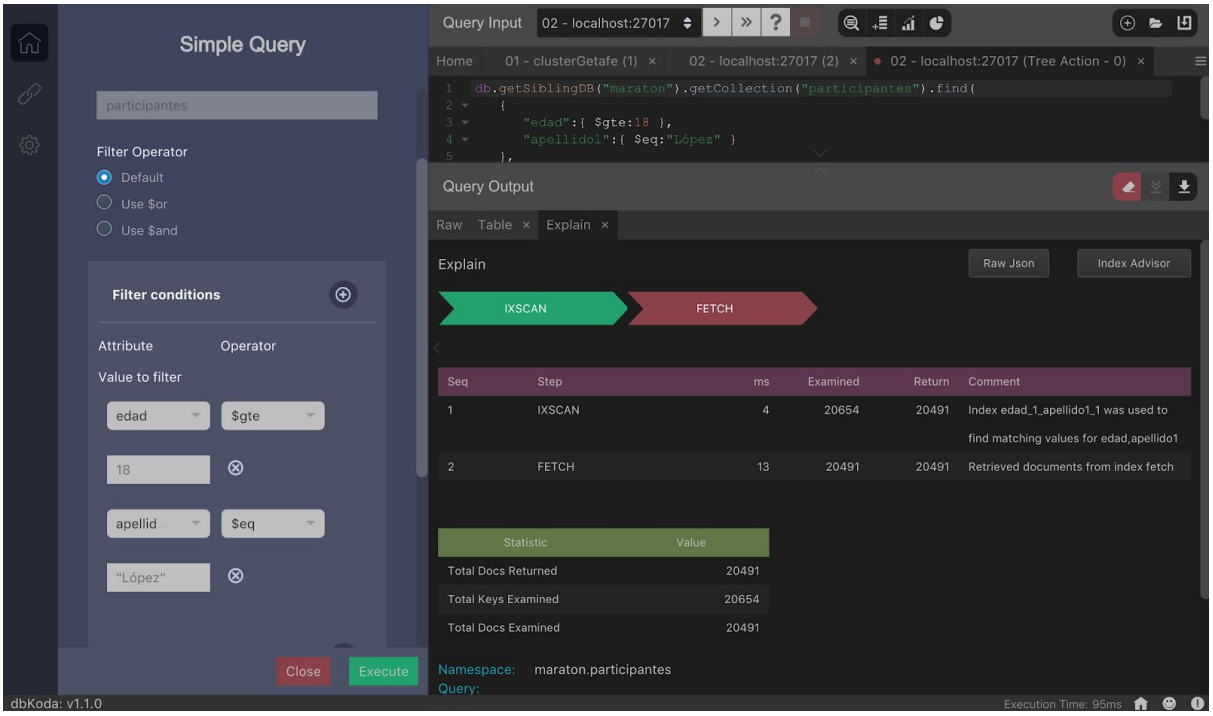
If you sometimes query on only one key and at other times query on that key combined with a second key, then creating a compound index is more efficient than creating a single-key index. MongoDB will use the compound index for both queries.

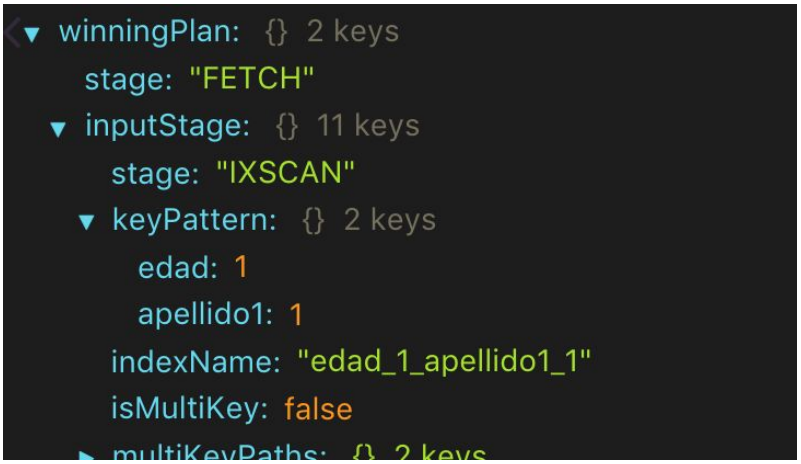
Práctica

Si busco por apellido1 y edad

```
> db.participantes.createIndex({edad: 1, apellido1: 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Y ejecutamos una consulta:

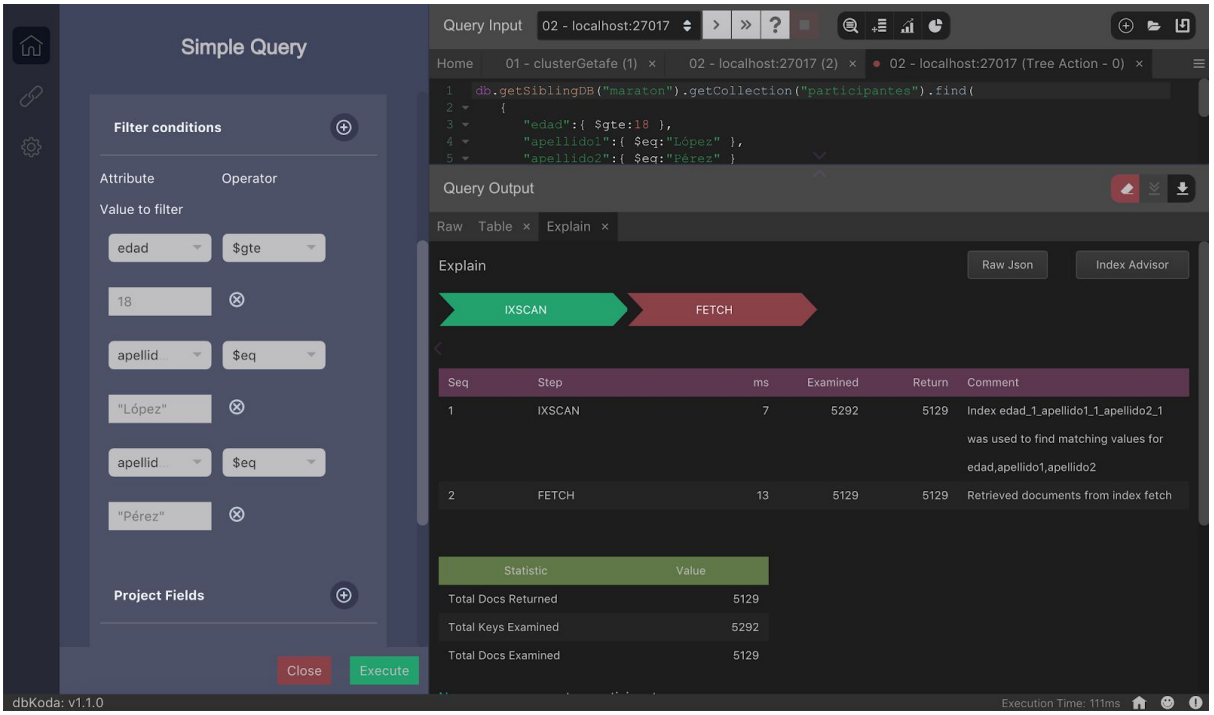




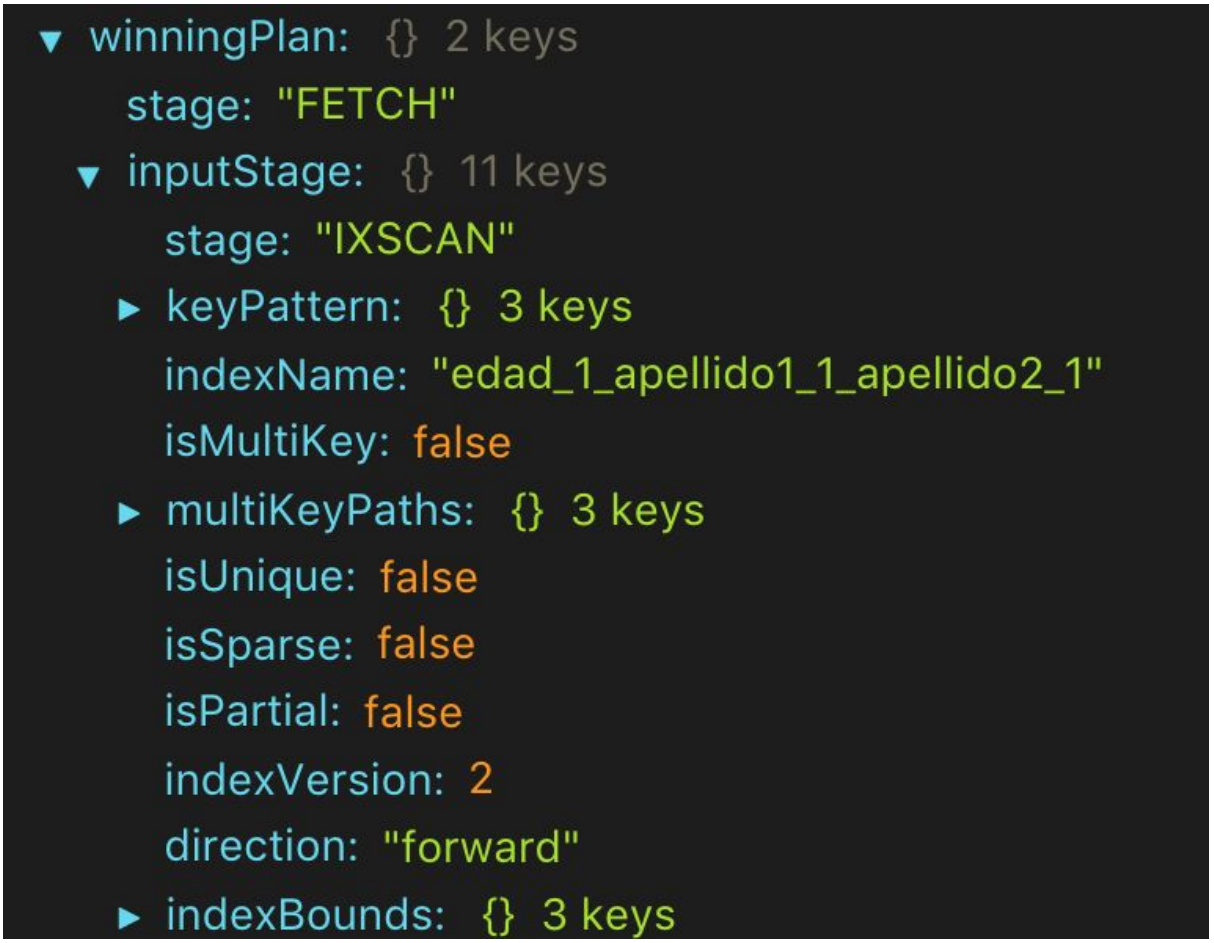
Con tres campos:

```
> db.participantes.createIndex({edad: 1, apellido1: 1, apellido2: 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

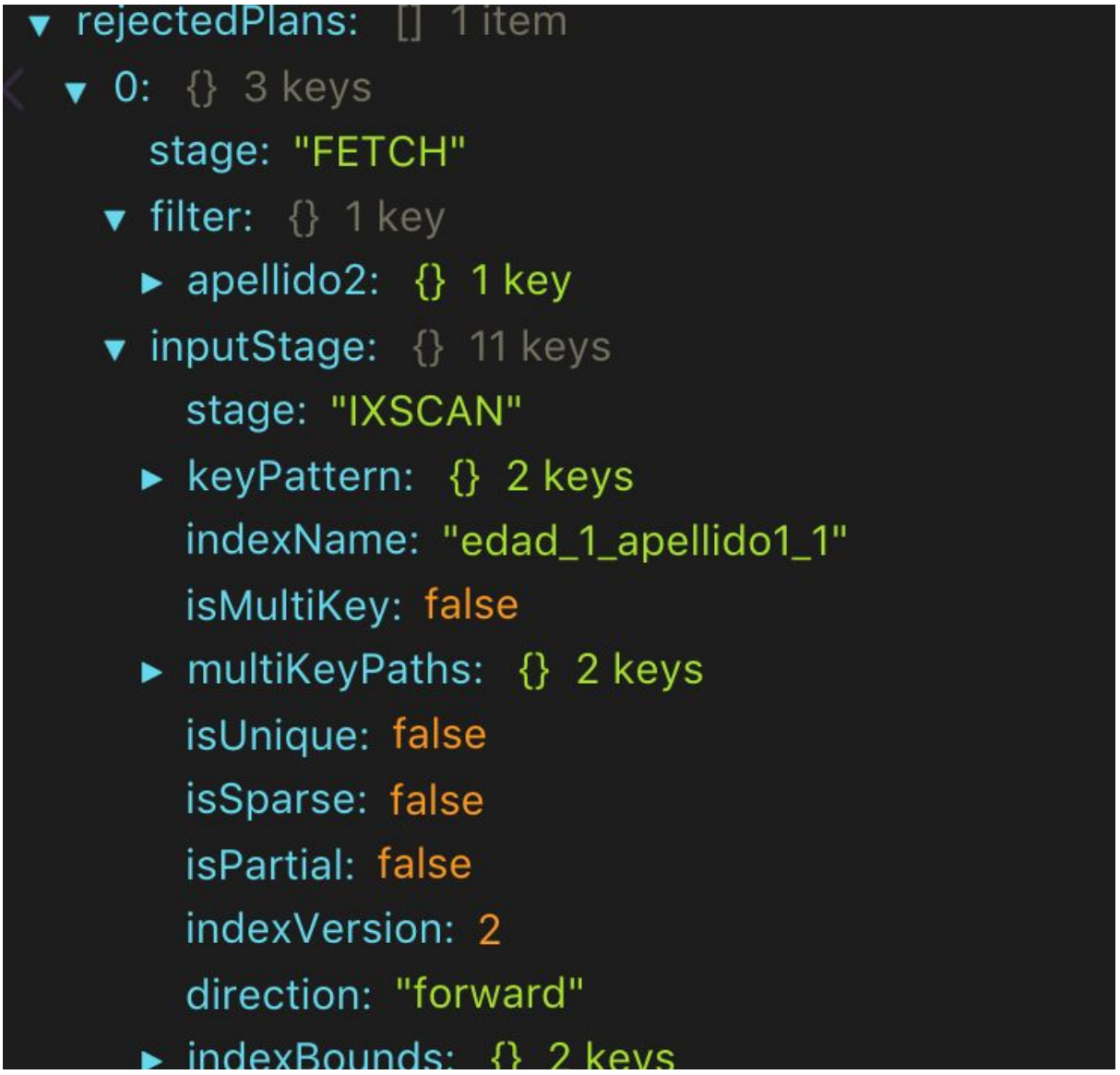
Y creamos una consulta:



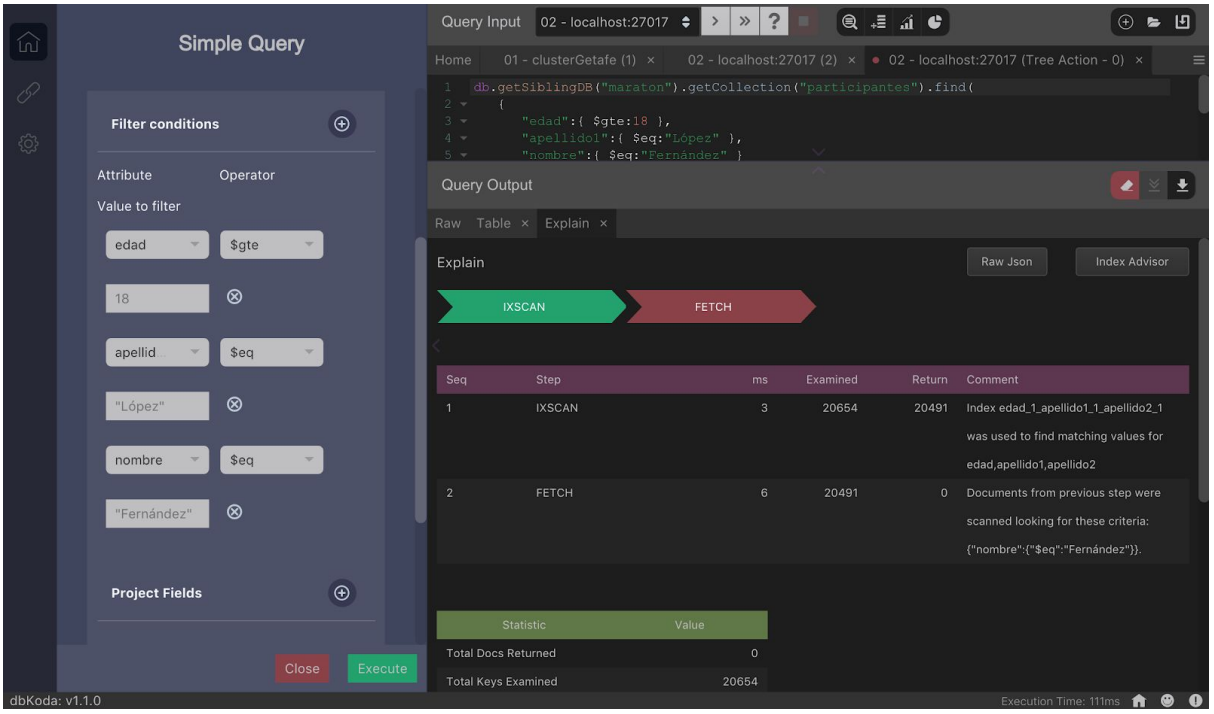
Plan ganador:



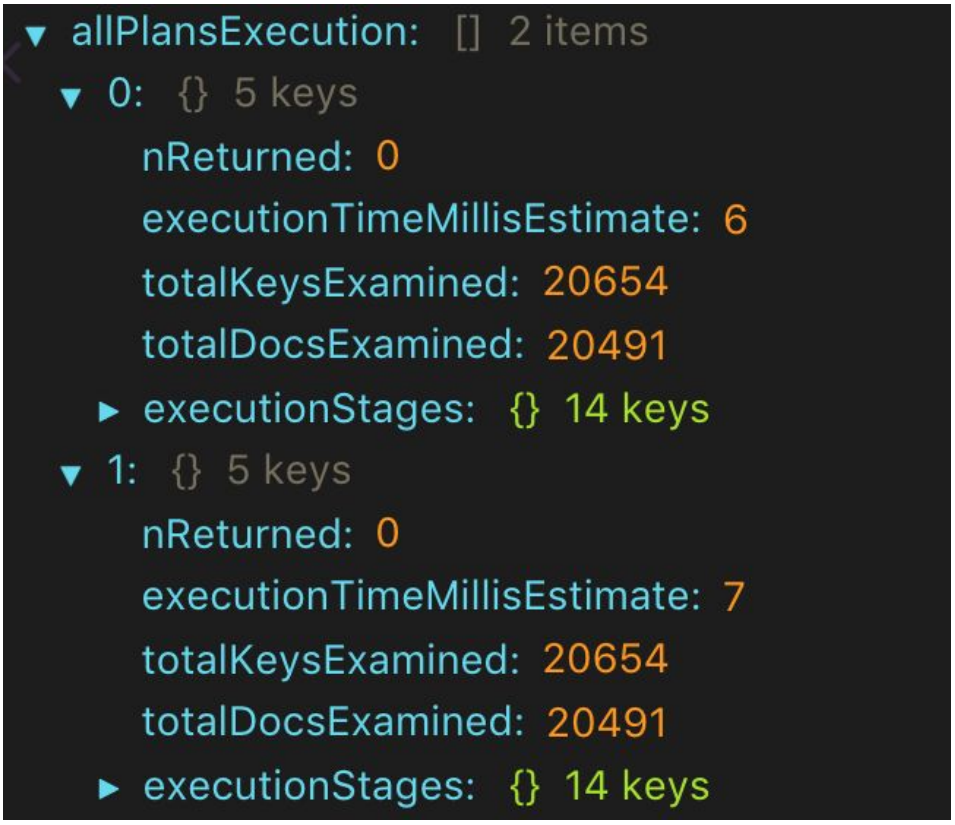
Plan desechado:



¿Qué ocurre si interviene un tercer campo diferente al utilizado en el último índice?



En principio también utiliza el de 3 campos. En allPlansExecution se pueden ver los datos:



Index Use and Collation

To use an index for string comparisons, an operation must also specify the same collation. That is, an index with a collation cannot support an operation that performs string comparisons on the indexed fields if the operation specifies a different collation.

Creamos un índice con colación sobre el campo nombre:

```
> db.participantes.createIndex({nombre: 1},{collation:{locale: "es", strength: 1}})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 3,
        "numIndexesAfter" : 4,
        "ok" : 1
}
```

La creación del índice solo es a los efectos de que este pueda ser utilizado por la consulta con colación.

Si pasamos la consulta sin colación, ni usa la colación:

```
> db.participantes.find({nombre: "juan"})
Vacío
```

Ni usa el índice:

```
> db.participantes.find({nombre: "juan"}).explain()
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "maraton.participantes",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "nombre" : {
                                "$eq" : "juan"
                        }
                },
                "queryHash" : "F53262C1",
                "planCacheKey" : "C1628CCF",
                "winningPlan" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "nombre" : {
                                        "$eq" : "juan"
                                }
```

```
                        },
                        "direction" : "forward"
                }
...
```

En cambio con la colación en este ejemplo encuentra minúsculas:

```
> db.participantes.find({nombre: "juan"}).collation({locale: "es", strength: 1})
{ "_id" : 0, "nombre" : "Juan", "apellido1" : "González", "apellido2" : "López", "edad" : 7,
"dni" : "13593548A" }
{ "_id" : 1, "nombre" : "Juan", "apellido1" : "López", "apellido2" : "Pérez", "edad" : 4,
"dni" : "21030172C" }
{ "_id" : 5, "nombre" : "Juan", "apellido1" : "González", "apellido2" : "López", "edad" : 16,
"dni" : "72284457P" }
```

Y utiliza el índice:

```
> db.participantes.find({nombre: "juan"}).collation({locale: "es", strength:
1}).explain()
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "maraton.participantes",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "nombre" : {
                                "$eq" : "juan"
                        }
                },
                "collation" : {
                        "locale" : "es",
                        "caseLevel" : false,
                        "caseFirst" : "off",
                        "strength" : 1,
                        "numericOrdering" : false,
                        "alternate" : "non-ignorable",
                        "maxVariable" : "punct",
                        "normalization" : false,
                        "backwards" : false,
                        "version" : "57.1"
                },
                "queryHash" : "A98837A4",
                "planCacheKey" : "8BBECEBD",
                "winningPlan" : {
                        "stage" : "FETCH",
```

```
                        "inputStage" : {
                                "stage" : "IXSCAN",
```

Con una colación diferente ya no usa el índice:

```
> db.participantes.find({nombre: "juan"}).collation({locale: "en", strength:
1}).explain()
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "maraton.participantes",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "nombre" : {
                                "$eq" : "juan"
                        }
                },
                "collation" : {
                        "locale" : "en",
                        "caseLevel" : false,
                        "caseFirst" : "off",
                        "strength" : 1,
                        "numericOrdering" : false,
                        "alternate" : "non-ignorable",
                        "maxVariable" : "punct",
                        "normalization" : false,
                        "backwards" : false,
                        "version" : "57.1"
                },
                "queryHash" : "36F91AFC",
                "planCacheKey" : "F875A2AD",
                "winningPlan" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "nombre" : {
                                        "$eq" : "juan"
                                }
                        },
                        "direction" : "forward"
                },
                "rejectedPlans" : [ ]
        },
...
```

Si la colección tiene una colación por defecto se usará el índice siempre que la colación de la consulta sea la misma de la colación.

```
> db.createCollection("fuuu", {collation: {locale: "es", strength: 1}})
{ "ok" : 1 }
> db.fuuu.createIndex({nombre: 1},{collation:{locale: "es", strength: 1}})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
> db.fuuu.insert([{nombre: "Tomás"},{nombre: "Carlos"}])

> db.fuuu.find({nombre: "tomas"})
{ "_id" : ObjectId("5e386e28a1491be19abf6c5f"), "nombre" : "Tomás" }
> db.fuuu.find({nombre: "tomas"}).explain()
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "maraton.fuuu",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "nombre" : {
                                "$eq" : "tomas"
                        }
                },
                "collation" : {
                        "locale" : "es",
                        "caseLevel" : false,
                        "caseFirst" : "off",
                        "strength" : 1,
                        "numericOrdering" : false,
                        "alternate" : "non-ignorable",
                        "maxVariable" : "punct",
                        "normalization" : false,
                        "backwards" : false,
                        "version" : "57.1"
                },
                "queryHash" : "A98837A4",
                "planCacheKey" : "8BBECEBD",
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
```

Use Indexes to Sort Query Results

In MongoDB, sort operations can obtain the sort order by retrieving documents based on the ordering in an index. If the query planner cannot obtain the sort order from an index, it will sort the results in memory.

Sort operations that use an index often have better performance than those that do not use an index. In addition, sort operations that do not use an index will abort when they use 32 megabytes of memory.

NOTE. As a result of changes to sorting behavior on array fields in MongoDB 3.6, when sorting on an array indexed with a multikey index the query plan includes a blocking SORT stage. The new sorting behavior may negatively impact performance.

In a blocking SORT, all input must be consumed by the sort step before it can produce output. In a non-blocking, or indexed sort, the sort step scans the index to produce results in the requested order.

Sort with a Single Field Index

If an ascending or a descending index is on a single field, the sort operation on the field can be in either direction.

Práctica

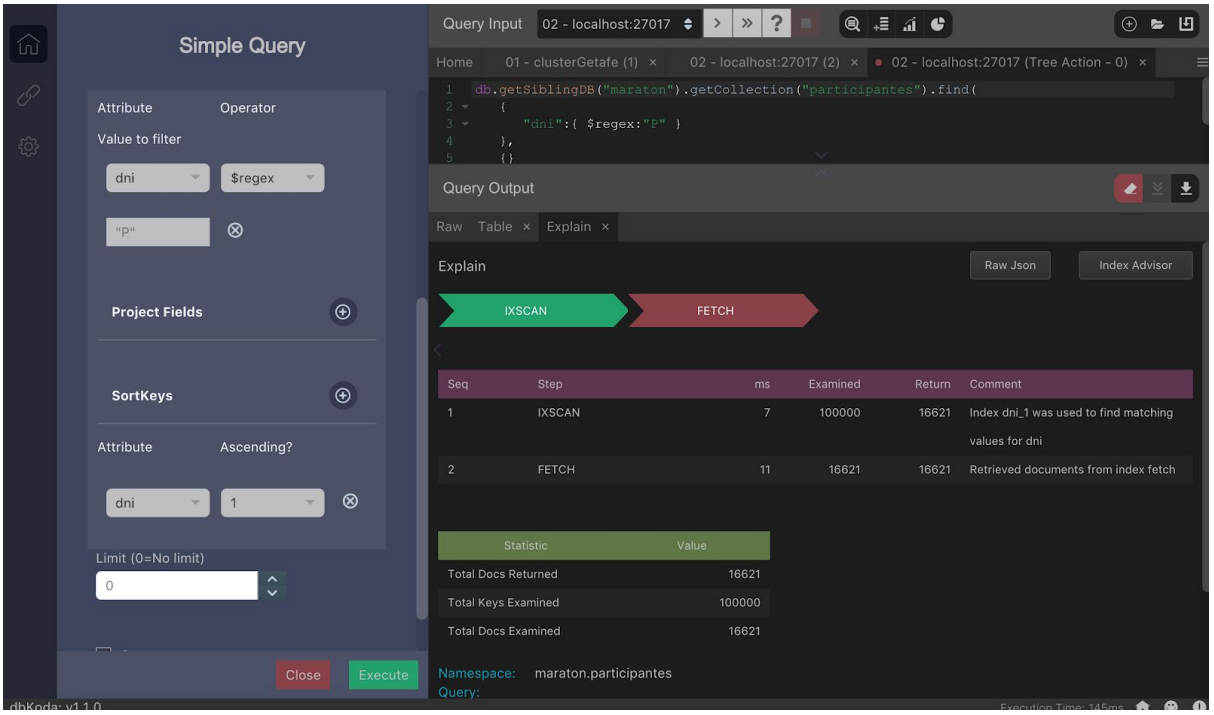
Reinicializamos índices para observar mejor:

```
> db.participantes.dropIndexes()
{
        "nIndexesWas" : 4,
        "msg" : "non-_id indexes dropped for collection",
        "ok" : 1
}
```

Sobre un índice simple, la ordenación se puede realizar en cualquier sentido:

```
> db.participantes.createIndex({dni: 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Comprobamos:



Sort on Multiple Fields

You can specify a sort on all the keys of the index or on a subset; however, the sort keys must be listed in the same order as they appear in the index. For example, an index key pattern { a: 1, b: 1 } can support a sort on { a: 1, b: 1 } but not on { b: 1, a: 1 }.

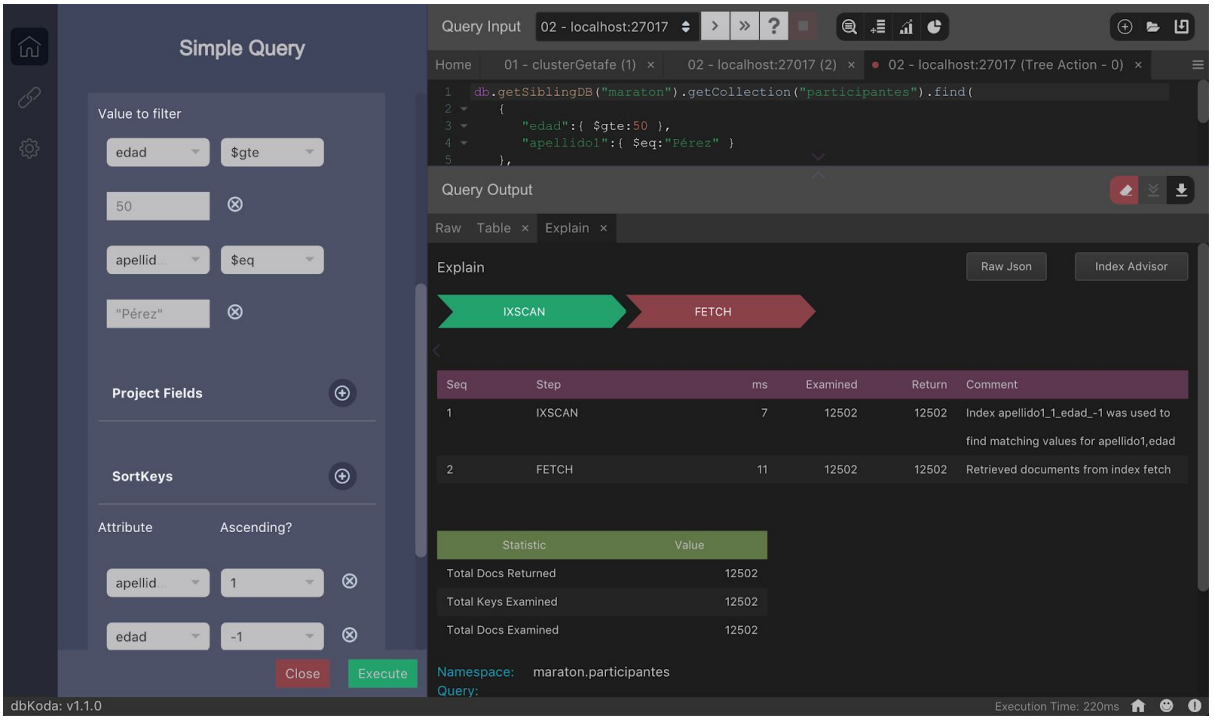
For a query to use a compound index for a sort, the specified sort direction for all keys in the cursor.sort() document must match the index key pattern or match the inverse of the index key pattern. For example, an index key pattern { a: 1, b: -1 } can support a sort on { a: 1, b: -1 } and { a: -1, b: 1 } but not on { a: -1, b: -1 } or {a: 1, b: 1}.

Práctica

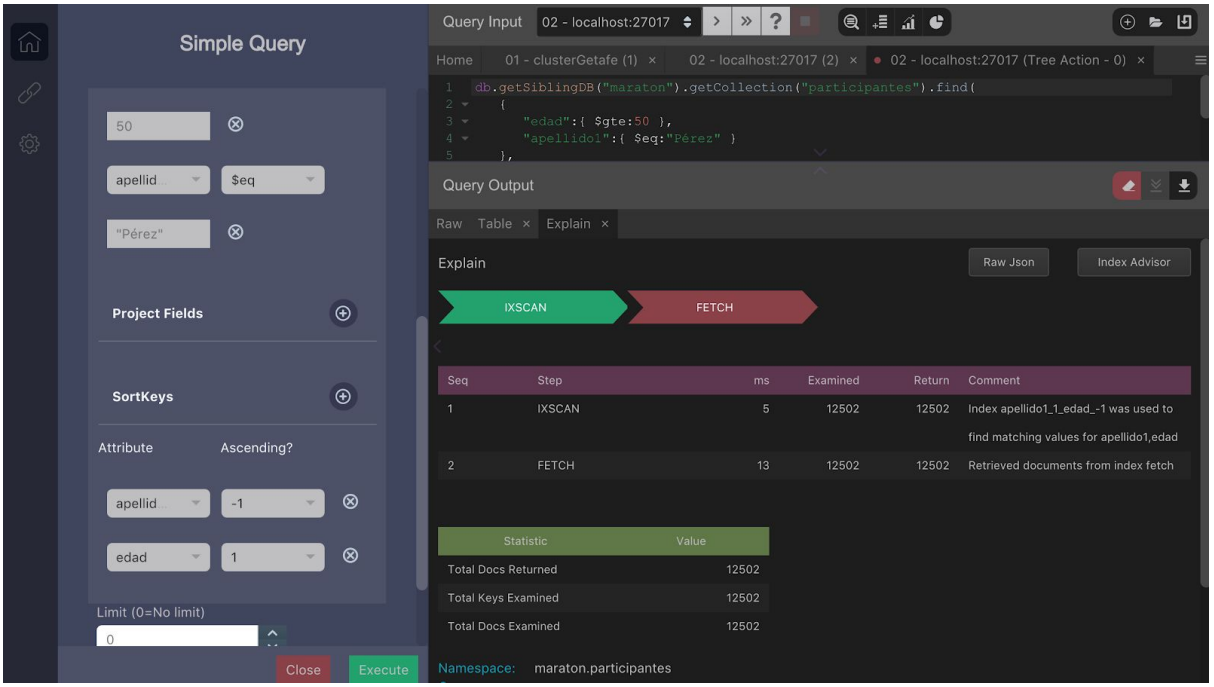
```
> db.participantes.createIndex({apellido1: 1, edad: -1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Comprobamos que la siguientes consultas utilizan el índice para ordenar y no tienen que usar la ordenación en memoria:

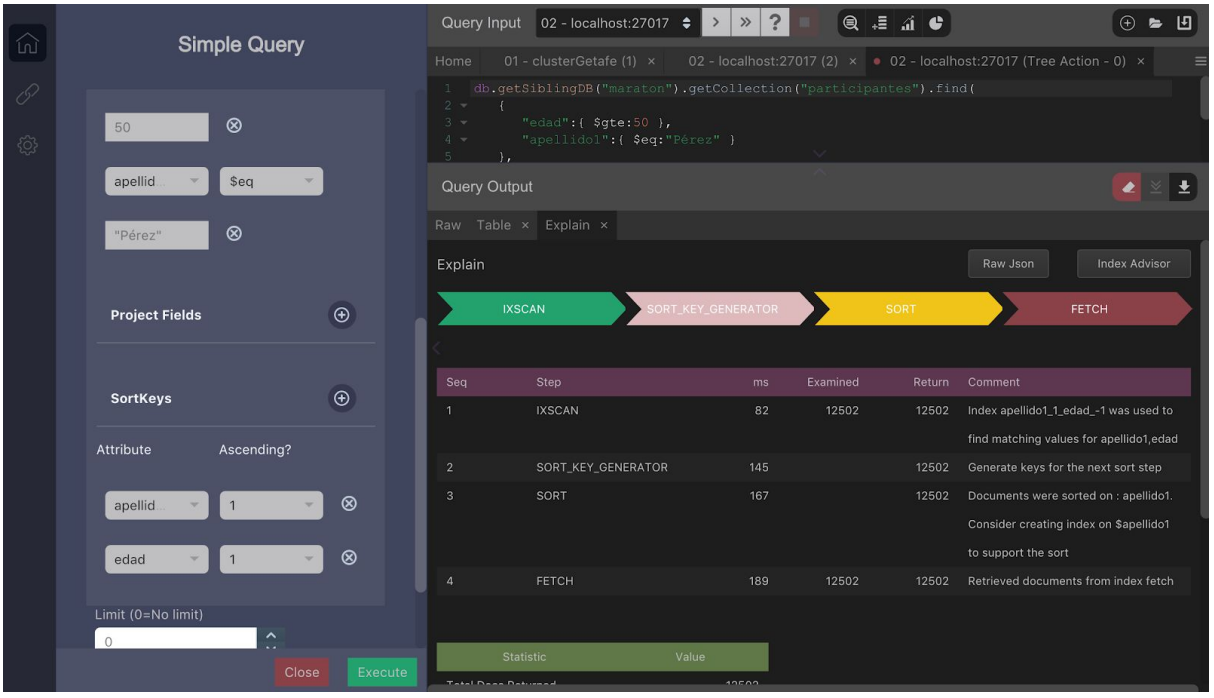
En el orden del índice



Y en el inverso:



En cambio si no todos los campos se ordenan con el orden inverso del índice, el índice se sigue usando pero es necesario ordenar en memoria:



Si la ordenación del set de datos supera 32 megas se interrumpe la operación.

Sort and Index Prefix

If the sort keys correspond to the index keys or an index prefix, MongoDB can use the index to sort the query results. A prefix of a compound index is a subset that consists of one or more keys at the start of the index key pattern.

Práctica

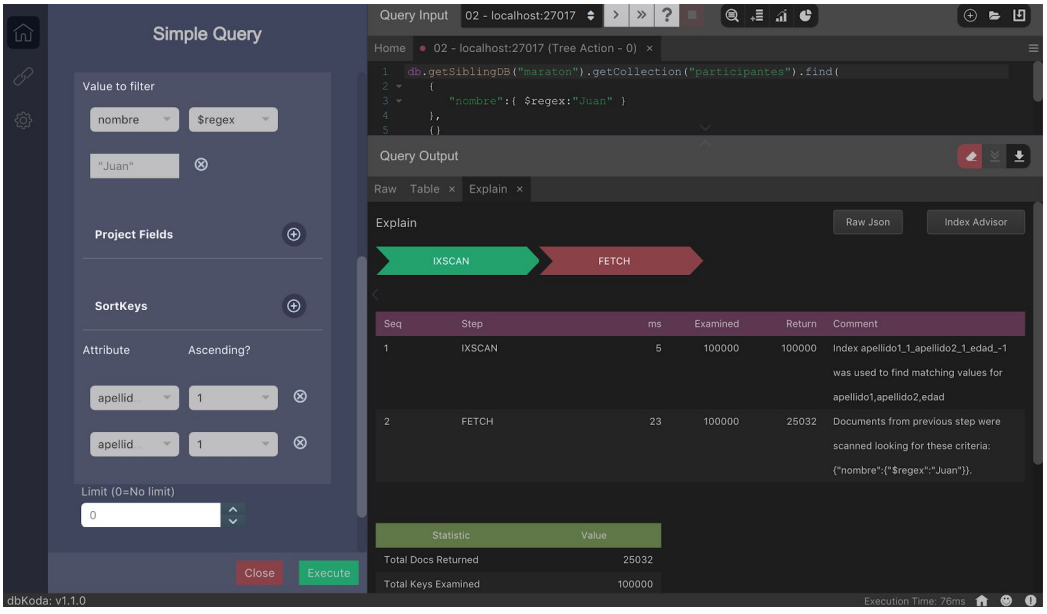
```
> db.participantes.createIndex({apellido1: 1, apellido2: 1, edad: -1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
```

El índice podrá ser utilizado con los siguientes prefijos en el método sort():

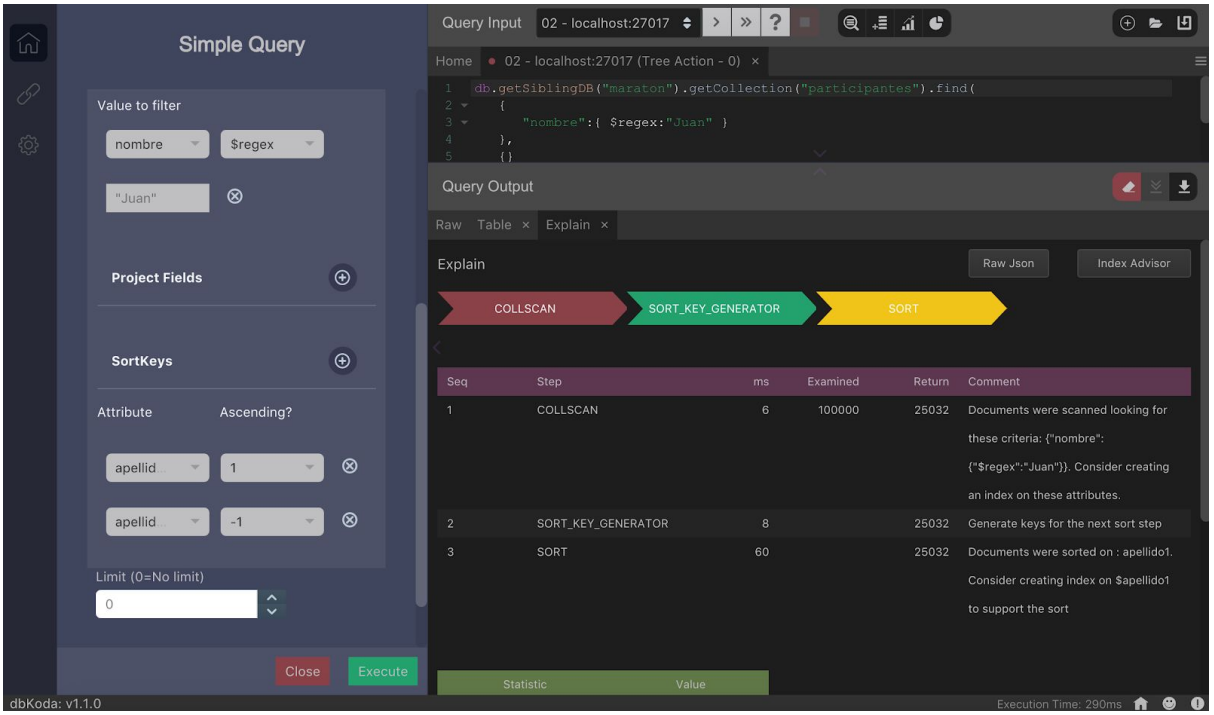
{apellido1: 1}
{apellido1: -1}
{apellido1: 1, apellido2: 1}
{apellido1: -1, apellido2: -1}
{apellido1: 1, apellido2: 1, edad: -1}
{apellido1: -1, apellido2: -1, edad: 1}

Particularidades:

Aunque la consulta tenga un campo no contenido en el índice, usará el índice si la ordenación contiene el prefijo, primero para seleccionar todos los documentos en el orden de la consulta y posteriormente en el fetch para recorrerlos y encontrar las coincidencias. Por ejemplo:

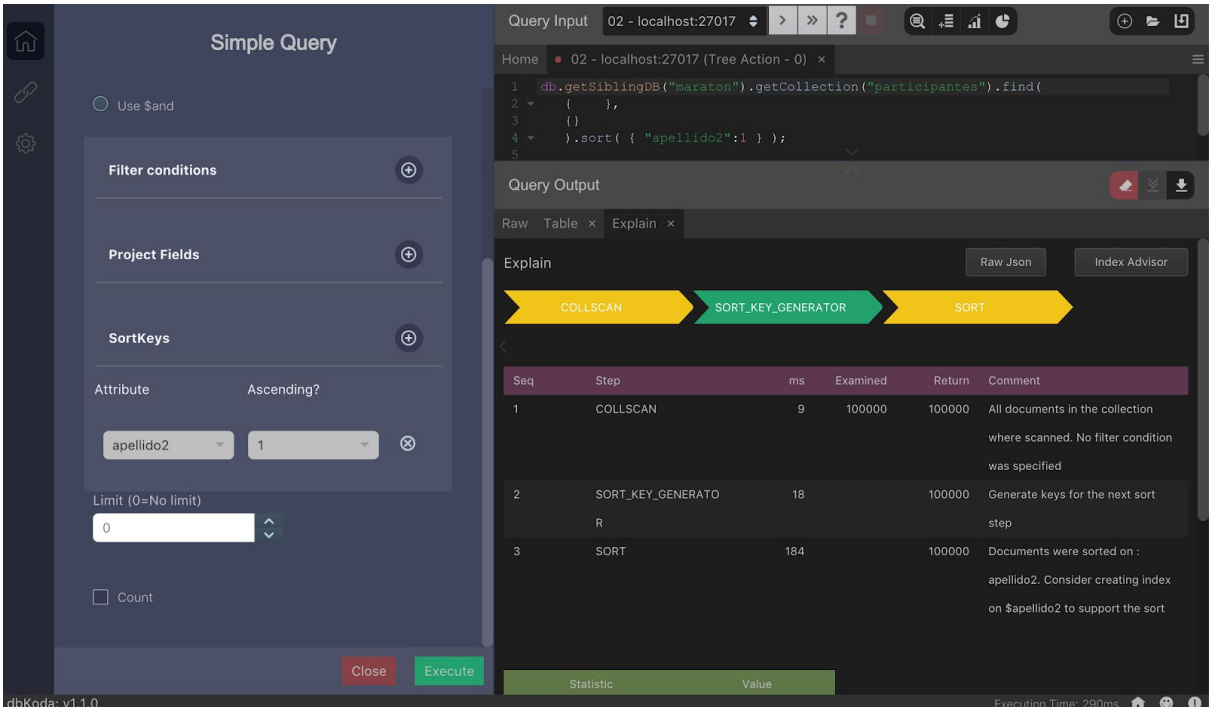


Si no coinciden ni los campos ni el prefijo del índice en la ordenación, se crea un scan de toda la colección y se ordena en memoria. Por ejemplo:

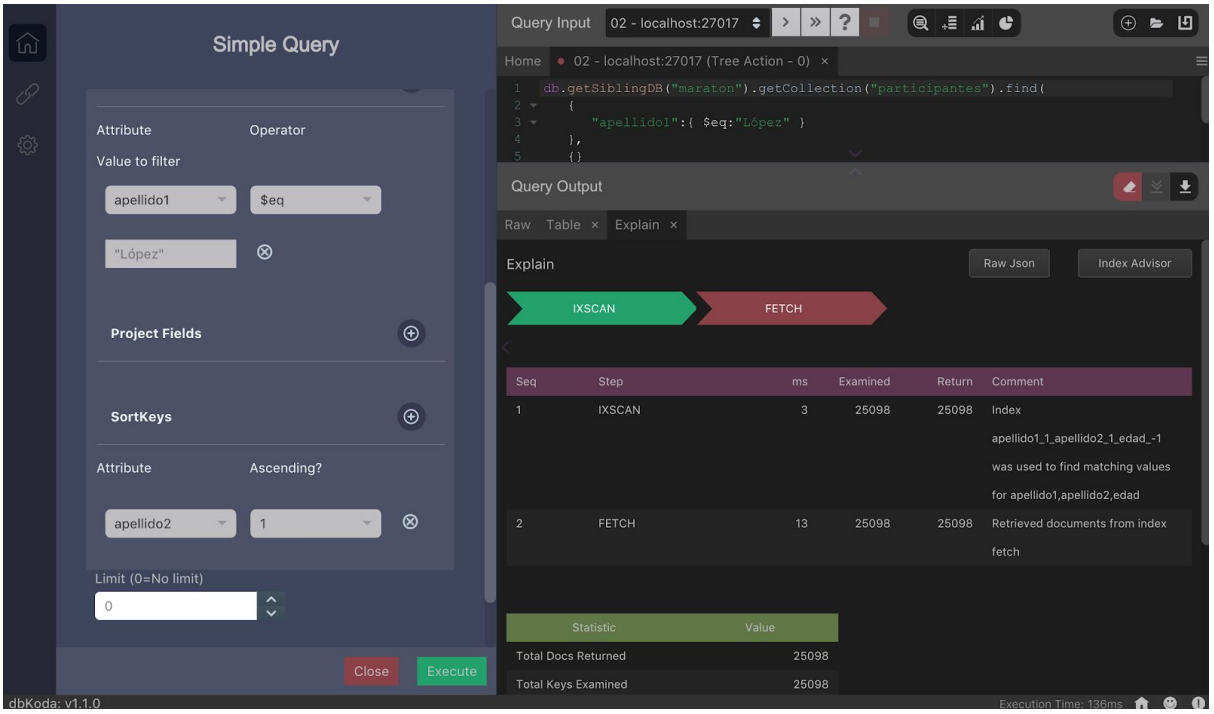


Una última particularidad es que si en la consulta existe un campo con consulta de igualdad que pueda complementar el prefijo en el sort, podrá utilizar el índice para no tener que ordenar en memoria.

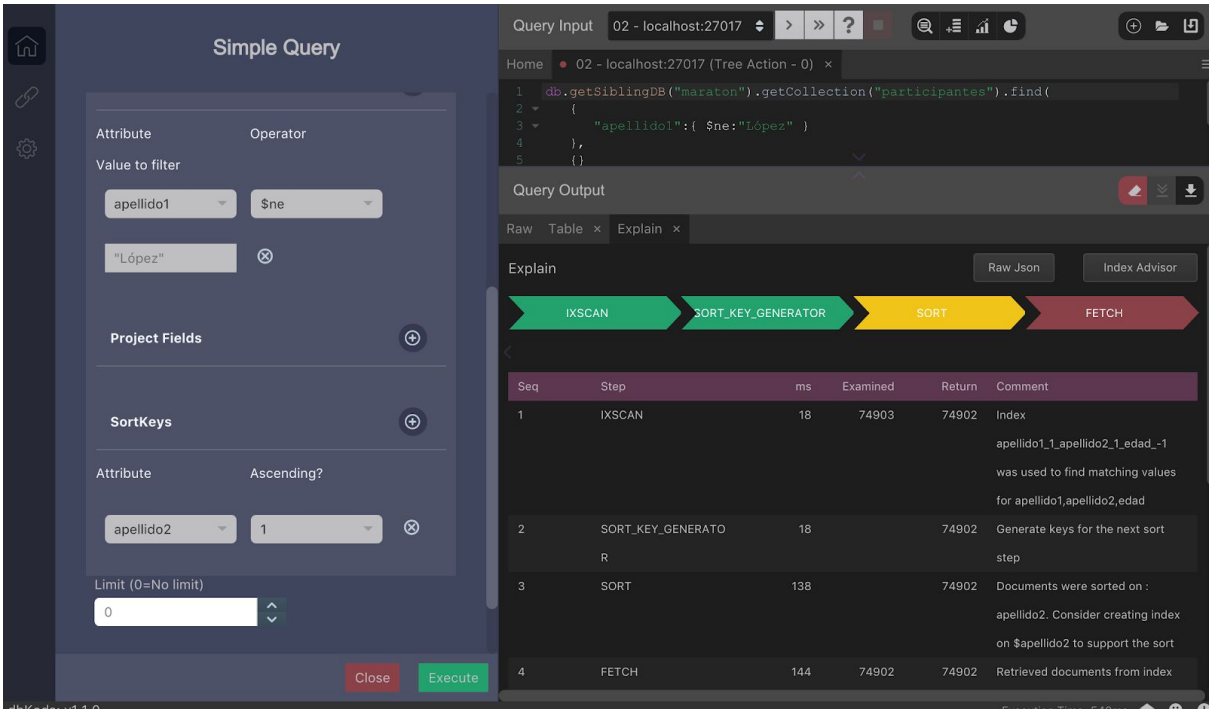
Por ejemplo esta consulta no usa el índice:



Si le añadimos en la consulta un campo de igualdad que se pueda añadir a los campos de la ordenación para completar el prefijo, la consulta usará el indice y no ordenará en memoria.



Siempre y cuando sea de igualdad:



Ensure Indexes Fit in RAM

For the fastest processing, ensure that your indexes fit entirely in RAM so that the system can avoid reading the index from disk.

To check the size of your indexes, use the db.collection.totalIndexSize() helper, which returns data in bytes.

Práctica

> db.participantes.totalIndexSize()

Indexes that Hold Only Recent Values in RAM

Indexes do not have to fit entirely into RAM in all cases. If the value of the indexed field increments with every insert, and most queries select recently added documents; then MongoDB only needs to keep the parts of the index that hold the most recent or “right-most” values in RAM. This allows for efficient index use for read and write operations and minimize the amount of RAM required to support the index.

Create Queries that Ensure Selectivity

Selectivity is the ability of a query to narrow results using the index. Effective indexes are more selective and allow MongoDB to use the index for a larger portion of the work associated with fulfilling the query.

To ensure selectivity, write queries that limit the number of possible documents with the indexed field. Write queries that are appropriately selective relative to your indexed data.

Index intersection

MongoDB can use the intersection of multiple indexes to fulfill queries. In general, each index intersection involves two indexes; however, MongoDB can employ multiple/nested index intersections to resolve a query.

To determine if MongoDB used index intersection, run explain(); the results of explain() will include either an AND_SORTED stage or an AND_HASH stage.

Covered querys

The MongoDB covered query is one which uses an index and does not have to examine any documents. An index will cover a query if it satisfies the following conditions:

- All fields in a query are part of an index.
- All fields returned in the results are of the same index.

Práctica

```
> db.participantes.find({apellido1: "López"},{apellido1: 1, createdAt: 1, _id: 0}).explain()
{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "maraton.participantes",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "apellido1" : {
                                "$eq" : "López"
                        }
                },
                "winningPlan" : {
                        "stage" : "PROJECTION",
                        "transformBy" : {
                                "apellido1" : 1,
                                "createdAt" : 1,
                                "_id" : 0
                        },
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "apellido1" : 1,
                                        "createdAt" : -1
                                },
                                "indexName" : "apellido1_1_createdAt_-1",
...
}
```