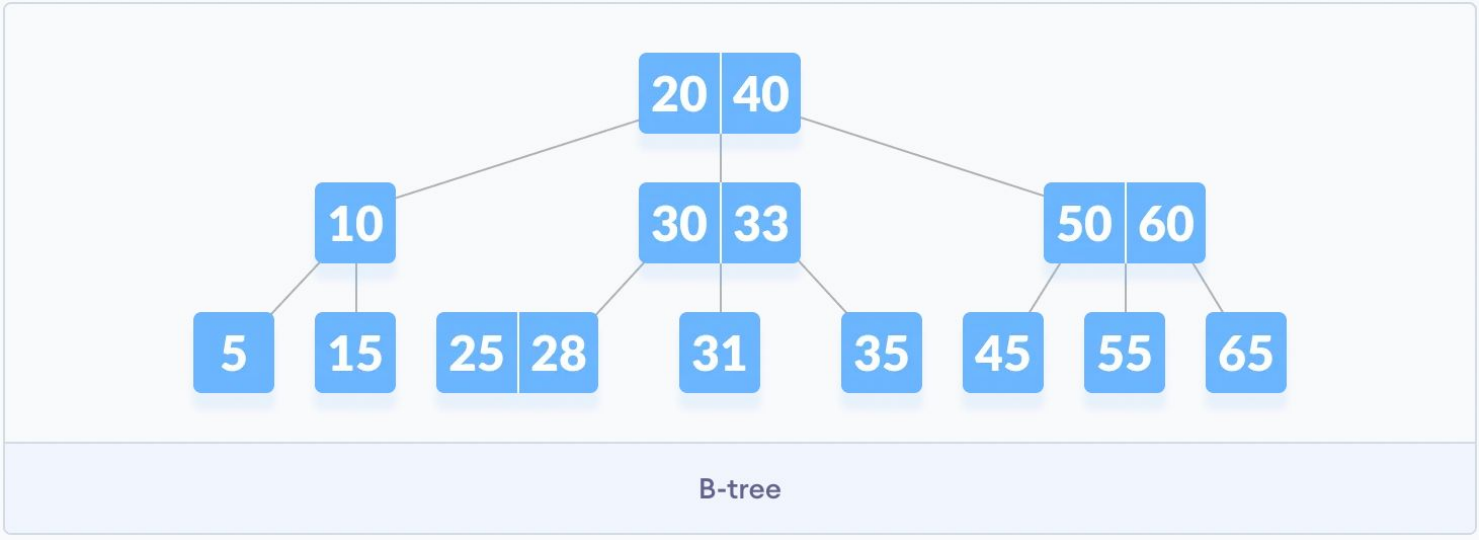
MongoDB Index 101

2022-06-29

Brown and friends

What is index?

Data Structure



B-tree is a special type of **self-balancing search tree** in which each node can contain more than one key and can have more than two children.

https://www.mongodb.com/docs/manual/indexes/#ref-b-tree-id2

B-Tree vs Binary Tree

B-tree and Binary Tree Data Structure Comparison

Both structures operate in the average in O(log n) time. Note that in the worst case, B-tree, at O(log n), is faster than Binary Search Tree at O(n).

	Average	Worst Case
Space	O(*n*)	O(*n*)
Search	O(log *n*)	O(log *n*)
Insert	O(log *n*)	O(log *n*)
Delete	O(log *n*)	O(log *n*)

B-tree is optimized for systems that read and write large blocks of data. It is commonly used in databases and file systems.

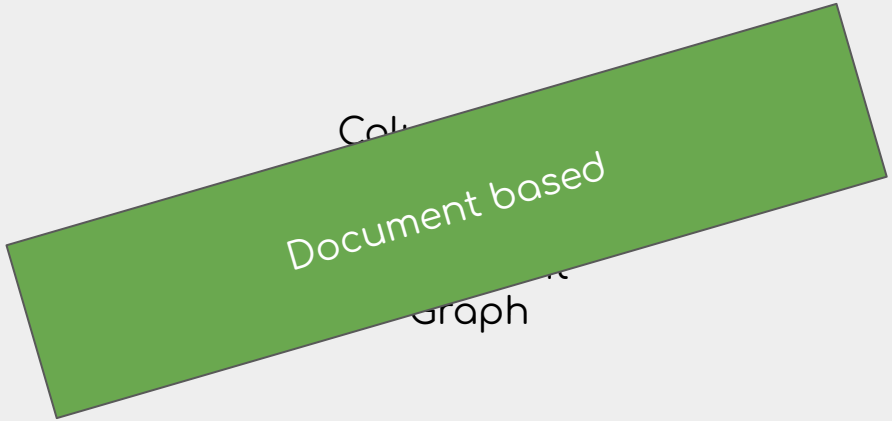
	Average	Worst Case
Space	O(*n*)	O(*n*)
Search	O(log *n*)	O(*n*)
Insert	O(log *n*)	O(*n*)
Delete	O(log *n*)	O(*n*)

The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient.

MongoDB Database Type

Columns, Row
Columnar
Key-Value
Document
Graph

MongoDB Database Type



Demo Collection


```
{ 7 items
  "id" : int 1000
  "first_name" : string "Geneva"
  "last_name" : string "Kopacek"
  "email" : string "gkopacekrr@squarespace.com"
  "gender" : string "Agender"
  "language" : string "Montenegrin"
  "is_active" : bool false
}
```


Database: test
Collection: user
Documents: 1,000k

Default Index

```
{
 "_id" : ObjectId("62bbdd5a6978e39c78e3bf87"),
 "id" : 1,
 "first_name" : "Nikolas",
 "last_name" : "Godleman",
 "email" : "ngodleman0@webeden.co.uk",
 "gender" : "Male",
 "language" : "Fijian",
 "is_active" : false
}
```

```
getCollectionIndexInfo("users","_id_");
```

Key	Value
(1)	{ name : "_id_" } (6 fields)
v	2
key	{ _id : 1 }
name	_id_
indexSize	65,536 (64.0KB)
usage stats	Array[1]
0	{ name : "_id_", host : "4a334d3d6114:27017" } (5 fields)
name	_id_
key	{ _id : 1 }
host	4a334d3d6114:27017
accesses	{ ops : 2, since : ISODate("2022-06-29T11:54:35.922+07:00") }
spec	{ v : 2, key : { _id : 1 }, name : "_id_" }
index details	{ type : "file" } (15 fields)
metadata	{ formatVersion : 8 }
creationString	access_pattern_hint=none,allocation_size=4KB,app_metadata=(f
type	file
uri	statistics:table:index-3--4937500220967527413
LSM	{12 fields}
block-manager	{10 fields}
btree	{22 fields}
cache	{60 fields}
cache_walk	{21 fields}
checkpoint-cleanup	{ "pages added for eviction" : 0, "pages removed" : 0, "pages skip
compression	{13 fields}
cursor	{34 fields}
reconciliation	{36 fields}
session	{ } (4 fields)
transaction	{13 fields}

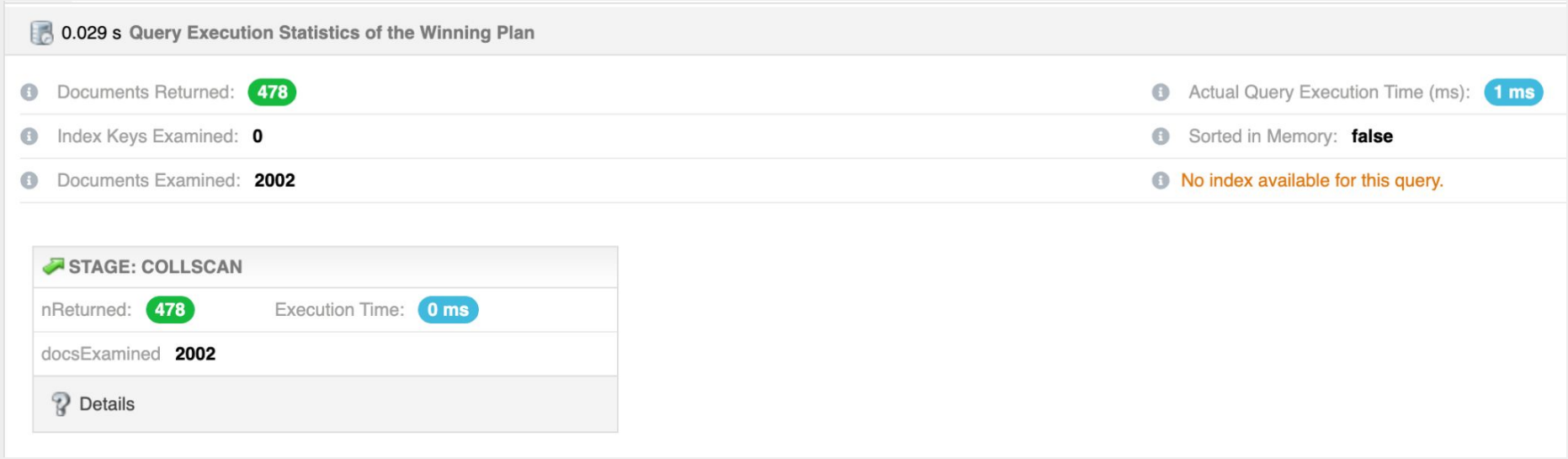
Query with no Index

```
db.users.find({"is_active": false})
```

slow query
expensive operation
long command operation
collection scan

Explain();

```
db.users.find({"is_active": true}).explain(true)
```



https://www.mongodb.com/docs/manual/reference/command/explain/#explain

Simple Index explanation

Consider an index { a : 1 }
Just only “a” field in index

```
{ _id: ObjectId(), a: 1, b: "ab" }
{ _id: ObjectId(), a: 1, b: "cd" }
{ _id: ObjectId(), a: 1, b: "ef" }
{ _id: ObjectId(), a: 2, b: "jk" }
{ _id: ObjectId(), a: 2, b: "lm" }
{ _id: ObjectId(), a: 2, b: "no" }
{ _id: ObjectId(), a: 3, b: "pq" }
{ _id: ObjectId(), a: 3, b: "rs" }
{ _id: ObjectId(), a: 3, b: "tv" }
```

If you query for { a: 2, b: "no" } **MongoDB must scan 3 documents** in the collection to return the one matching result.

Design the Index

Know your application (read/write heavy)
Know you query

Example
An API for order services

1. Find by order_id
2. Find by order status
3. Find by user_id
4. ...

Production Practices

High Cardinality

The index cardinality refers to **how many possible values there are for a field**. The field sex only has two possible values. It has a very low cardinality. Other fields such as names, usernames, phone numbers, emails, etc. will have a **more unique value for every document in the collection, which is considered high cardinality.**

Compound Index

Try to create indexes on high-cardinality keys or put high-cardinality keys first in the compound index.

Selectivity

```
{_id:ObjectId(),name:"John",sex:"male"}
{_id:ObjectId(),name:"Rich",sex:"male"}
{_id:ObjectId(),name:"Mose",sex:"male"}
{_id:ObjectId(),name:"Sami",sex:"male"}
{_id:ObjectId(),name:"Cari",sex:"female"}
{_id:ObjectId(),name:"Mary",sex:"female"}
```

Case#1
If your index is **{name:1}**, If you run the query

{ name: "John", sex: "male"}

You will have to **scan 1 document**. Because you allowed MongoDB to be selective.

Case#2
If your index is **{sex:1}**, If you run the query

{sex: "male", name: "John"}

You will have to **scan 4 documents**.

Take a Break with Book

Index Cardinality

Cardinality refers to how many distinct values there are for a field in a collection. Some fields, such as `"gender"` or `"newsletter opt-out"`, might only have two possible values, which is considered a very low cardinality. Others, such as `"username"` or `"email"`, might have a unique value for every document in the collection, which is high cardinality. Still others fall somewhere in between, such as `"age"` or `"zip code"`.

In general, the greater the cardinality of a field, the more helpful an index on that field can be. This is because the index can quickly narrow the search space to a much smaller result set. For a low cardinality field, an index generally cannot eliminate as many possible matches.

For example, suppose we had an index on `"gender"` and were looking for women named Susan. We could only narrow down the result space by approximately 50% before referring to individual documents to look up `"name"`. Conversely, if we indexed by `"name"`, we could immediately narrow down our result set to the tiny fraction of users named Susan and then we could refer to those documents to check the gender.

As a rule of thumb, try to create indexes on high-cardinality keys or at least put high-cardinality keys first in compound indexes (before low-cardinality keys).

Using explain() and hint()

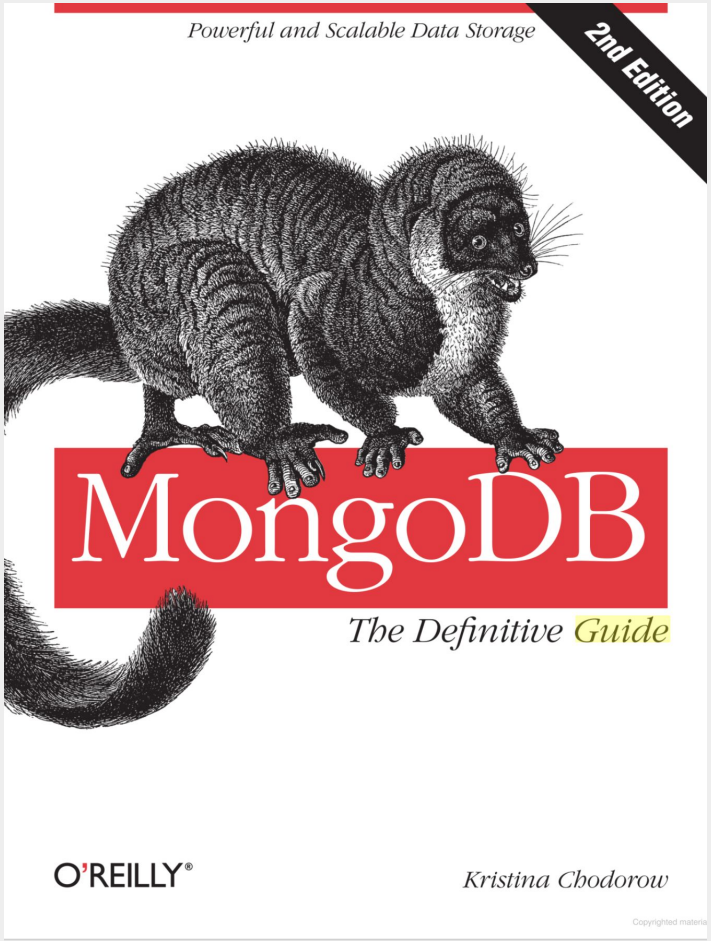
As you have seen above, `explain()` gives you lots of information about your queries. It is one of the most important diagnostic tools there is for slow queries. You can find out which indexes are being used and how by looking at a query's *explain*. For any query, you can add a call to `explain()` at the end (the way you would add a `sort()` or `limit()`, but `explain()` must be the last call).

There are two types of `explain()` output that you'll see most commonly: indexed and non-indexed queries. Special index types may create slightly different query plans, but most fields should be similar. Also, sharding returns a conglomerate of `explain()`s (as covered in Chapter 13), as it runs the query on multiple servers.

The most basic type of `explain()` is on a query that doesn't use an index. You can tell that a query doesn't use an index because it uses a `"BasicCursor"`. Conversely, most queries that use an index use a `"BtreeCursor"` (some special types of indexes, such as geospatial indexes, use their own type of cursor).

The output to an `explain()` on a query that uses an index varies, but in the simplest case, it looks something like this:

```
> db.users.find({"age" : 42}).explain()
{
    "cursor" : "BtreeCursor age_1_username_1",
    "isMultiKey" : false,
```

Planner & Stage

Planner

queryPlanner = winning plan

rejected plans = mongo optimizer reject the plan

executionStats = show execution stats

allPlansExecution = show all plan

Stage

- COLLSCAN for a collection scan
- IXSCAN for scanning index keys
- FETCH for retrieving documents
- SHARD_MERGE for merging results from shards
- SHARDING_FILTER for filtering out orphan documents from shards

Usecase#1


```
{
    "_id" : ObjectId("62bc2eca6978e39c78e3c757"),
    "id" : 1000,
    "first_name" : "Geneva",
    "last_name" : "Kopacek",
    "email" : "gkopacekrr@squarespace.com",
    "gender" : "Agender",
    "language" : "Montenegrin",
    "is_active" : false
},
{
    "_id" : ObjectId("62bc2eca6978e39c78e3c756"),
    "id" : 999,
    "first_name" : "Samaria",
    "last_name" : "Tumility",
    "email" : "stumilityrq@indiegogo.com",
    "gender" : "Female",
    "language" : "Kannada",
    "is_active" : true
},
```


```
db.getSiblingDB("demo")
db.users.createIndex({language: 1, is_active: 1});
```

```
1  use "demo"
2
3  db.users.find({"language": "Japanese"}).explain(true)
```

0.066 s **Query Execution Statistics of the Winning Plan**

Documents Returned: 14	Actual Query Execution Time (ms): 11 ms
Index Keys Examined: 14	Sorted in Memory: false
Documents Examined: 14	Query used the following index: REGULAR language_1_is_active_1

STAGE: FETCH
nReturned: 14 Execution Time: 7 ms 100%
Details

STAGE: IXSCAN
nReturned: 14 Execution Time: 0 ms
indexName REGULAR language_1_is_active_1
Details

Usecase#2

```
{
    "_id" : ObjectId("62bc2eca6978e39c78e3c757"),
    "id" : 1000,
    "first_name" : "Geneva",
    "last_name" : "Kopacek",
    "email" : "gkopacekrr@squarespace.com",
    "gender" : "Agender",
    "language" : "Montenegrin",
    "is_active" : false
},
{
    "_id" : ObjectId("62bc2eca6978e39c78e3c756"),
    "id" : 999,
    "first_name" : "Samaria",
    "last_name" : "Tumility",
    "email" : "stumilityrq@indiegogo.com",
    "gender" : "Female",
    "language" : "Kannada",
    "is_active" : true
},
```

```
db.getSiblingDB("demo")
db.users.createIndex({language: 1, is_active: 1});
```

```
1  use "demo"
2
3  db.users.find({"language": "Japanese", "is_active": true}).explain(true)
```

0.035 s **Query Execution Statistics of the Winning Plan**

Documents Returned: 6	Actual Query Execution Time (ms): 2 ms
Index Keys Examined: 6	Sorted in Memory: false
Documents Examined: 6	Query used the following index: REGULAR language_1_is_active_1

STAGE: FETCH

nReturned: 6 Execution Time: 0 ms

Details

STAGE: IXSCAN

nReturned: 6 Execution Time: 0 ms

indexName REGULAR language_1_is_active_1

Details

Usecase#3


```
{
    "_id" : ObjectId("62bc2eca6978e39c78e3c757"),
    "id" : 1000,
    "first_name" : "Geneva",
    "last_name" : "Kopacek",
    "email" : "gkopacekrr@squarespace.com",
    "gender" : "Agender",
    "language" : "Montenegrin",
    "is_active" : false
},
{
    "_id" : ObjectId("62bc2eca6978e39c78e3c756"),
    "id" : 999,
    "first_name" : "Samaria",
    "last_name" : "Tumility",
    "email" : "stumilityrq@indiegogo.com",
    "gender" : "Female",
    "language" : "Kannada",
    "is_active" : true
},
```


```
db.getSiblingDB("demo")
db.users.createIndex({language: 1, is_active: 1});
```

```
1  use "demo"
2
3  db.users.find({"is_active": true}).explain(true)
```

0.032 s **Query Execution Statistics of the Winning Plan**

Documents Returned:	478	Actual Query Execution Time (ms):	1 ms
Index Keys Examined:	0	Sorted in Memory:	false
Documents Examined:	1001	No index available for this query.	

STAGE: COLLSCAN

nReturned: 478 Execution Time: 0 ms

docsExamined 1001

Details