Indexes in MongoDB

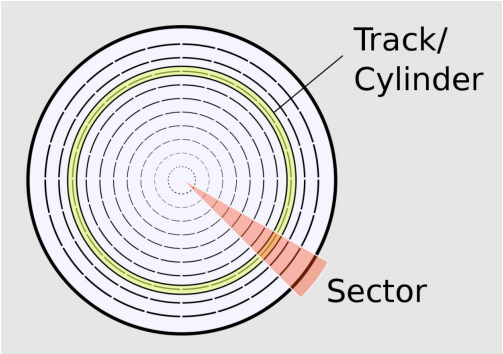
Relevel
by Unacademy

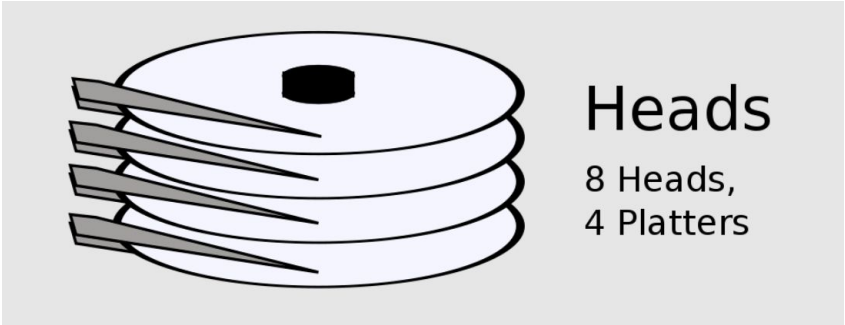
Concepts

- What are Indexes
- How database indexes work ?
- Indexes in MongoDB
- Types of Indexes in MongoDB
  - Single Field Index
  - Compound Index
  - Multi-key Index
  - Text Index

How database stores data?

- Database stores the data on secondary storage device like HDD or SSD
- A hard disk comprises of concentric circles known as tracks & sections called sectors
- For fetching the data from a HDD is made up of following two operations
  - Rotation
  - Disk seek
- Time taken to read the data is sum of Rotational delay and disk seek time





What is a database index?

- Special data structure that speeds up database read operations
- Works similar to index page in a book
- Stores reference to the documents in a database collection

Why are database indexes needed?

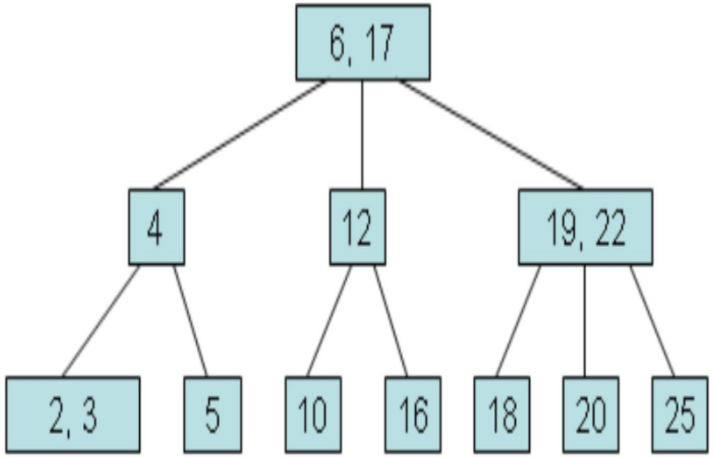
- Collection scan
  - Naive way to fetch any document is to fetch all the documents and check if the document satisfies the criteria
  - Collection scan is time taking since all the documents are read
- Analogy
  - Searching all the occurrences of a word in a book Vs searching the occurrences using an Index
- Using an index, collection scan can be skipped

How a database index works?

- Collection scan
  - Since it iterates on all the documents, if there are N documents, the time complexity is O(N)
- Index scan
  - Index is created on a particular field of a document. For eg:- EmployeeId in Employees document
  - MongoDB internally uses B-tree data structure for the index.
- B-tree
  - It's a n-array tree. It can have at most N children and N-1 keys
  - All leaf nodes are at the same level
  - Root Node must have at least 2 children
  - Every node can contain at least N/2 children except the root & the leaf nodes
- The keys in B-tree nodes act as separation values and divide the tree into left sub-tree which contains elements less than the left key.
- The right subtree contains elements that are greater than the right key
- Middle subtree has all the elements whose value lies between left & right key

B-tree

- Search starts from the root node.
- For eg:- Let's search 20. 20 is greater than 17, so we will search the rightmost sub-tree
- We check the next node containing 19, 22. 20 falls between 19 and 22, so we continue our search in the middle sub-tree
- Finally, we find 20 as the leaf node.
- Search for any element requires 3 comparisons as the height of the tree is 3



When should we avoid Indexes?

- Table has only few records
- Write-heavy databases that require frequent batch updates or insert operations
- Indexes shouldn’t be used on fields that are frequently changed
- Avoid using indexes of the fields that have large number of NULL values

Types of Indexes in MongoDB

- Single Field Index
- Compound Index
- Multikey Index
- Text Index

Insert data in User DB

- Insert 100k records
- Find the details of a random user with id - 53591

```
Atlas atlas-9kygzn-shard-0 [primary] Twitter> db.Users.find({userId: 53591})
[
  {
    _id: ObjectId("62246d7609c3e2f8681f935f"),
    userId: 53591,
    username: 'user_53591',
    age: 26,
    createdAt: ISODate("2022-03-06T08:14:46.551Z"),
    modifiedAt: ISODate("2022-03-06T08:14:46.551Z"),
    countOfFriends: 247,
    numberOfPosts: 1582
  }
]
```

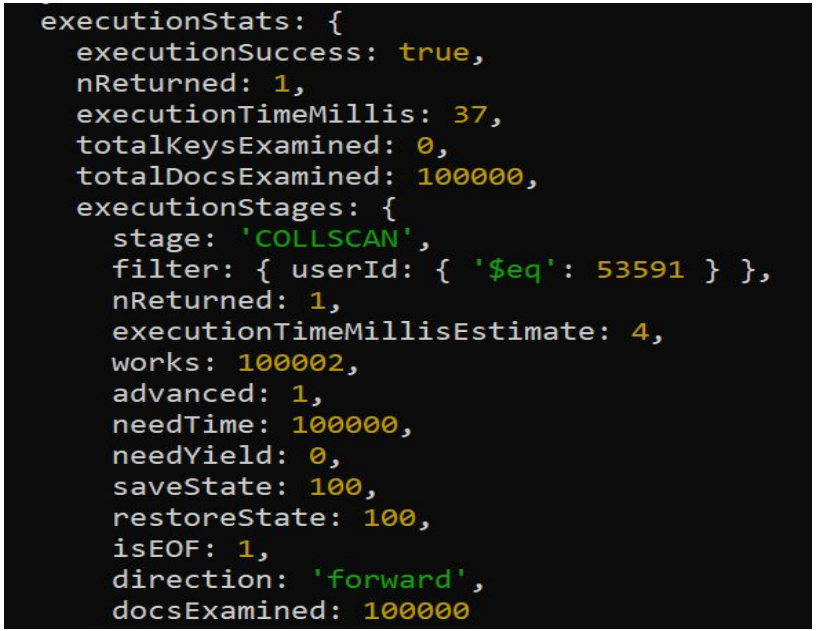
Explain

- Explain method gives an in depth overview of the execution of any query
- We will use executionStats mode to find time taken, records read, etc

```
Atlas atlas-9kygzn-shard-0 [primary] Twitter> db.Users.find({userId: 53591}).explain("executionStats")
```

Query on a non-indexed field

- If an index is not indexed, then the read query performs a collection scan.
- It reads all the documents in the collection
- The read query is inefficient



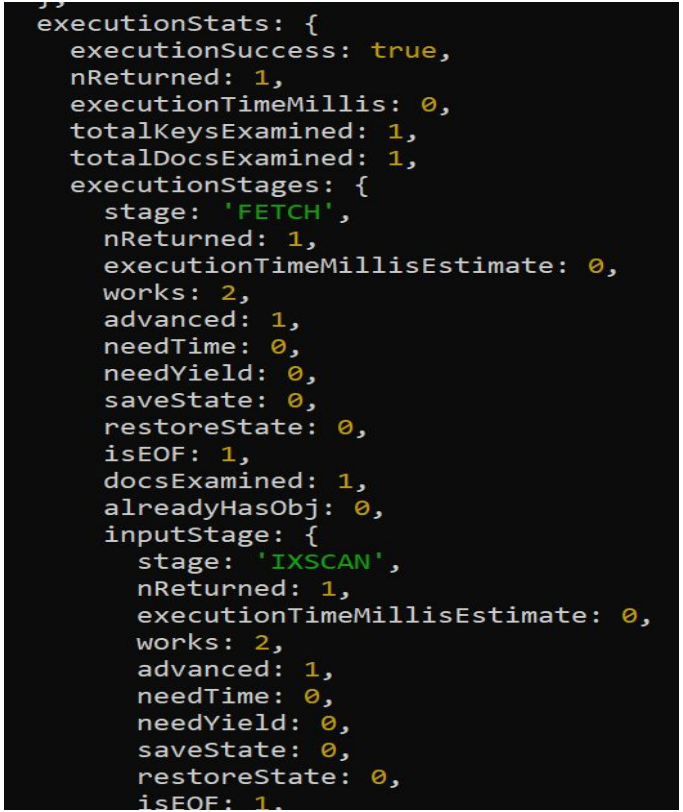
Single Field Index

- Index exists on a single field
- Multiple single indexes can be created using a different index for each field
- A single field index can be created on a field embedded inside another field
- Additionally, we can create a single field index on an embedded document

```
Atlas atlas-9kygzn-shard-0 [primary] Twitter> db.Users.createIndex({userId: 1})
userId_1
```

Single Field Index performance

- Only requires 1 document search.
- Fetches a single document
- Efficient and faster than a non-indexed lookup



Types of Query Patterns

- Equality query
  - db.Users.find({age : 90})
- Range query
  - db.Users.find({countOfFriends: {“$gt”: 100, “$lt”: 300}})
- Multi-value query with sorted result
  - db.Users.find({countOfFriends: {“$gt”: 100, “$lt”: 300}}).sort({age: 1})

Compound Index

- Compound Index allows us to create an index on multiple fields of a document
- Each field in the compound index can have a different sorting order
- Initial ordering is determined using the sorting order of the first field. In case two documents have the same first field, they will be sorted using the order of the second field
- Use cases

  - Find the number of users with age > 80 and count of posts > 200
  - Find the users < 18 yrs having more than 100 friends

Compound Index Examples

- Find the number of users with age > 18 and count of posts > 100
- Query - **db.Users.find({ numberOfPosts: { "$gt": 100 }, age: { "$gt": 18 } }).sort({ numberOfPosts: -1 })**

Performance of Compound Index


```
executionStats: {
  executionSuccess: true,
  nReturned: 76893,
  executionTimeMillis: 146,
  totalKeysExamined: 78792,
  totalDocsExamined: 76893,
  executionStages: {
    stage: 'FETCH',
    nReturned: 76893,
    executionTimeMillisEstimate: 36,
    works: 78792,
    advanced: 76893,
    needTime: 1898,
    needYield: 0,
    saveState: 79,
    restoreState: 79,
    isEOF: 1,
    docsExamined: 76893,
    alreadyHasObj: 0,
    inputStage: {
      stage: 'IXSCAN',
      nReturned: 76893,
      executionTimeMillisEstimate: 6,
      works: 78792,
      advanced: 76893,
      needTime: 1898,
      needYield: 0,
      saveState: 79,
      restoreState: 79,
      isEOF: 1,
      keyPattern: { numberOfPosts: -1, age: 1 },
      indexName: 'numberOfPosts_-1_age_1',
      isMultiKey: false,
      multiKeyPaths: { numberOfPosts: [], age: [] },
      isUnique: false,
      isSparse: false,
      isPartial: false,
      indexVersion: 2,
      direction: 'forward',
      indexBounds: { numberOfPosts: [ '[inf.0, 100)' ], age: [ '(18, inf.0]' ] },
      keysExamined: 78792,
      seeks: 1899,
      dupsTested: 0,
      dupsDropped: 0
    }
```

Multikey Index

- MongoDB allows users to create indexes on array fields of a document
- The database automatically detects if the field is an array and creates a multikey index
- The index can consist of array fields with scalar values or fields with nested documents

```
Atlas atlas-9kygzn-shard-0 [primary] Twitter> db.Tweets.createIndex({"hashTag": 1})
hashTag_1
```

```
Atlas atlas-9kygzn-shard-0 [primary] Twitter> db.Tweets.find({hashTag: "war"})
```

Text Index

- To perform searches on fields that have strings or list of strings, we use Text index
- We can specify the weight of each field constituting the index. The weight of the field indicates its importance relative to the other fields
- The score of each field is calculated by calculating the sum of product of weights and the number of matches
- A database collection can only contain one text index although different fields can be included in the text index

```
Atlas atlas-9kygzn-shard-0 [primary] Twitter> db.Tweets.createIndex({tweet:"text", hashTag: "text"})
tweet_text_hashTag_text
```

Text Score

- A score is calculated for each document depending on the occurrences of the search term, number of matches found, & weights assigned to the individual fields
- A field with higher weight will have more impact on the search results than the one with lesser weight

```
_id: ObjectId("6224b17b7370210800f45946")
userId: 1
username: "user_1"
tweet: "The war has entered its second day"
› hashTag: Array
created… : 2022-03-06T13:04:59.118+00:00
modifie… : 2022-03-06T13:04:59.118+00:00
score: 2.35
```

Summary

	Single Field Index	Compound Index	Multikey Index	Text Index
Use case	Create an index on one field to speed up read operation. Multiple indexes can be created using different single fields.	Create index on multiple fields of a collection. Improves performance of queries that query two or more fields	Useful to index a field that contains an array value. Supports efficient queries against array fields.	Useful to search occurrences of a word in a collection. Uses score to return relevant results.
Limitation	Query doesn't perform efficient lookup if it contains two or more fields in the search criteria	MongoDB doesn't support creation of compound index on more than 32 fields	For Compound multi-key index, at most one indexed field can be an array	Only one text index can be created on a collection

Thank You!