**MongoDB Text Search**

1. **Creating a Text Index**

* **Indexing a Single Field**
* **Indexing Multiple Fields (Compound Indexing)**
* **Indexing the Entire Document (Wildcard Indexing)**

1. **Advanced Searching**

* **Phrase Search**
* **Negation Search**

1. **Looking Behind the Scenes**
2. **Weighted Text Search**
3. **Partitioning Text Indexes**

## **Data Preparation**

We are going to store the documents of the following structure:

**{**

**"subject":"Joe owns a dog",**

**"content":"Dogs are man's best friend",**

**"likes": 60,**

**"year":2015,**

**"language":"english"**

**}**

**Let insert some example documents:**

db.messages.insert({"subject":"Joe owns a dog", "content":"Dogs are man's best friend", "likes": 60, "year":2015, "language":"english"})

db.messages.insert({"subject":"Dogs eat cats and dog eats pigeons too", "content":"Cats are not evil", "likes": 30, "year":2015, "language":"english"})

db.messages.insert({"subject":"Cats eat rats", "content":"Rats do not cook food", "likes": 55, "year":2014, "language":"english"})

db.messages.insert({"subject":"Rats eat Joe", "content":"Joe ate a rat", "likes": 75, "year":2014, "language":"english"})

# **Creating a Text Index**

A text index is created quite similar to how we create a regular index, except that it specifies the text keyword instead of specifying an ascending/descending order.

## **Indexing a Single Field**

Create a text index on the subject field of our document using the following query:

**db.messages.createIndex({"subject":"text"})**

**Test of Text Index to search dogs keyword**

**db.messages.find({$text: {$search: "dogs"}})**

To get some statistics about how relevant the resultant documents are

**{ $meta: "textScore" } =** provides information on the processing of the $text operator

**sort  =** sort the documents by their textScore using the sort command

**Note:** A higher textScore indicates a more relevant match.

**db.messages.find({$text: {$search: "dogs",$caseSensitive: false}}, {score: {$meta: "textScore"}}).sort({score:{$meta:"textScore"}})**

**Notes: We are searching dogs rather than dog then why in our search both are coming**

## **Indexing Multiple Fields (Compound Indexing)**

**db.messages.createIndex({"subject":"text","content":"text"})**

**Note : one text index per collection**

**db.messages.dropIndex("subject\_text")**

**db.messages.createIndex({"subject":"text","content":"text"})**

**Now search in all document with given keywords**

**db.messages.find({$text:{$search:"cat"}},{score:{$meta:"textScore"}}).sort({score:{$meta:"textScore"}})**

## **Indexing the Entire Document (Wildcard Indexing)**

There are situation when you want to search in all document rather than explicitly defined index fields so in that case we use this type of indexes

**db.messages.createIndex({"$\*\*":"text"})**

**db.messages.dropIndex("subject\_text\_content\_text")**

**db.messages.insert({"subject":"Birds can cook", "content":"Birds donot eat rats","likes":12, "year":2013, location: "Chicago", "language":"english"})**

**db.messages.find({$text:{$search:"chicago"}},{score:{$meta:"textScore"}}).sort({score:{$meta:"textScore"}})**

**Notes: Wild Index can be slow when we have large dataset**

1. **Advanced Searching**

## **2.1) Phrase Search**

You can search for phrases like **“smart birds who love cooking”**using text indexes. By default, the phrase search makes an **OR** search on all the specified keywords, i.e. it will look for documents which contains either the keywords **smart**, **bird**, **love** or **cook**.

**db.messages.find({$text:{$search:"smart birds who cook"}},{score:{$meta:"textScore"}}).sort({score:{$meta:"textScore"}})**

We can get exact word by and operator as “” “”

**db.messages.find({$text: {$search: "\"cook food\""}}, {score: {$meta: "textScore"}}).sort({score:{$meta:"textScore"}})**

## **2.2) Negation Search**

Prefixing a search keyword with – (minus sign) excludes all the documents that contain the negated term. For example, try searching for any document which contains the keyword rat but does not contain birdsusing the following query:

**db.messages.find({$text: {$search: "rat -birds"}}, {score: {$meta: "textScore"}}).sort({score:{$meta:"textScore"}})**

# **Behind Scene How Thing Works**

We are putting query with or,and and negated operator

**db.messages.find({$text: {$search: "dogs who cats dont eat ate rats \"dogs eat\" -friends"}}, {score: {$meta: "textScore"}}).sort({score:{$meta:"textScore"}}).explain(true)**

# **Weighted Text Search**

When we have indexes on more than one field in our document, most of the times one field will be more important (i.e. more weight) than the other. For example, when you are searching across a blog, the title of the blog should be of highest weight, followed by the blog content.

The default weight for every indexed field is 1. To assign relative weights for the indexed fields, you can include the weights option while using the create Index command. modify our indexes to include weights; with the subject field having a weight of 3 against the content field having a weight of 1.

**db.messages.dropIndex("subject\_text\_content\_text")**

**db.messages.createIndex( {"$\*\*": "text"}, {"weights": { subject: 3, content:1 }} )**

# **Partitioning Text Indexes**

we can create a compound index that specifies an ascending/descending index key on yearfollowed by a text index on the subject field. By doing this, we are doing two important things:

We are logically partitioning the entire collection data into sets separated by year.

This would limit the text search to scan only those documents which fall under a specific year (or call it set).

**db.messages.createIndex( { "year":1, "subject": "text"} )**

Now search by partition key and text index at the same time

**db.messages.find({year: 2015, $text: {$search: "cats"}}, {score: {$meta: "textScore"}}).sort({score:{$meta:"textScore"}})**