Lecture Notes for Lecture 12 of CS 5200 (Database Management System) for the Summer 1, 2019 session at the Northeastern University Silicon Valley Campus.

MongoDB Indexing and Text Searching

Philip Gust,
Clinical Instructor
Department of Computer Science

Material contained in this presentation is based in part on and uses content from recommended readings for the course.

Miscellaneous Items

Getting Connection in ApacheDB Stored Procedure/Function

- To get the current connection within an ApacheDB stored procedure or function through the DriverManager class:

```
Connection conn = DriverManager.getConnection("jdbc:default:connection");
```

Miscellaneous Items

Regular Expression Queries in MongoDB in Java

- There are three ways to query in Java: filters, patterns, and basic objects
- Here is using filters:

```
Document query = new Document("equipment","gloves");

//whatever pattern you need. But you do not need the "/" delimiters
String pattern = ".*" + query.getString("equipment") + ".*";

//find(regex("field name", "pattern", "options"));
collection.find(regex("equipment", pattern, "i"));
```

source: https://stackoverflow.com/questions/32714333/java-mongodb-regex-query

Miscellaneous Items

Regular Expression Queries in MongoDB in Java

- There are three ways to query in Java: filters, patterns, and basic objects
- Here is using patterns and filters:

```
Pattern regex = Pattern.compile("ind", Pattern.CASE_INSENSITIVE);
Bson filter = Filters.eq("name", regex);
```

source: https://stackoverflow.com/questions/32714333/java-mongodb-regex-query

Miscellaneous Items

Regular Expression Queries in MongoDB in Java

- There are three ways to query in Java: filters, patterns, and basic objects
- Here is using basic objects

```
Document regexQuery = new Document();
regexQuery.append("$regex", ".*" + Pattern.quote(searchTerm) + ".*");
BasicDBObject criteria = new BasicDBObject("name", regexQuery);
DBCursor cursor = collection.find(criteria);
```

source: https://stackoverflow.com/questions/32714333/java-mongodb-regex-query

Miscellaneous Items

Project Information

- Deliverables and Audience
- Presentations
- Final Due Date

Lecture 11 Review

- In this lecture we looked at techniques for data modeling in MongoDB, including document structure, atomicity of write operations, document growth, and data use and performance.
- Next, we looked at how to perform queries on documents in both the Mongo shell and the Java APIs, and how to control the order and which fields are returned.
- Finally, we discuss several options for graphical development tools to develop MongoDB databases and applications.

Today’s Topics

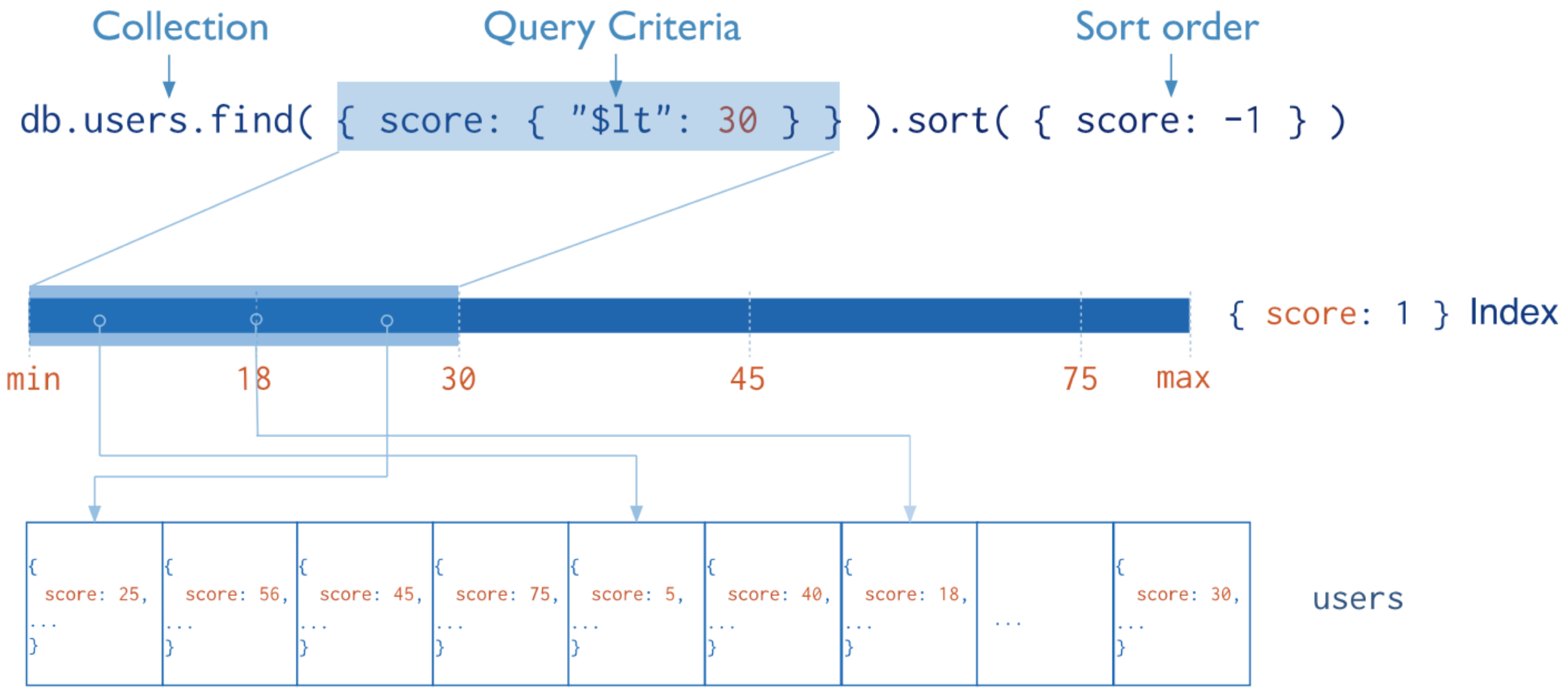
- In this lecture we will see how to define and use indexes to improve the performance of queries against a MongoDB databases
- MongoDB supports several kinds of indexes, and we will look at each of these in turn and learn how to use them
- We will also learn how to build indexes for a database, and to use index properties
- Finally , we will look in more detail at the special capabilities for indexing text in documents, searching for indexed text, and specifying weightings in order to determine relevancy of results.

MongoDB Indexing and Text Searching

- An index allows queries in MongoDB to be executed more efficiently.
- Without indexes, MongoDB must scan fields of every document in a collection to match query statements. With indexes, MongoDB can use them to limit the number of documents it must examine.
- An index is a special data structure that stores a small portion of the collection’s data set in a way that is easy to traverse
- The index stores the value of a specific field or set of fields , ordered by the value of the field. The ordering of entries supports efficient equality matches, and range-based query operations.
- MongoDB can also return sorted results using the ordering of the index.

MongoDB Indexing and Text Searching

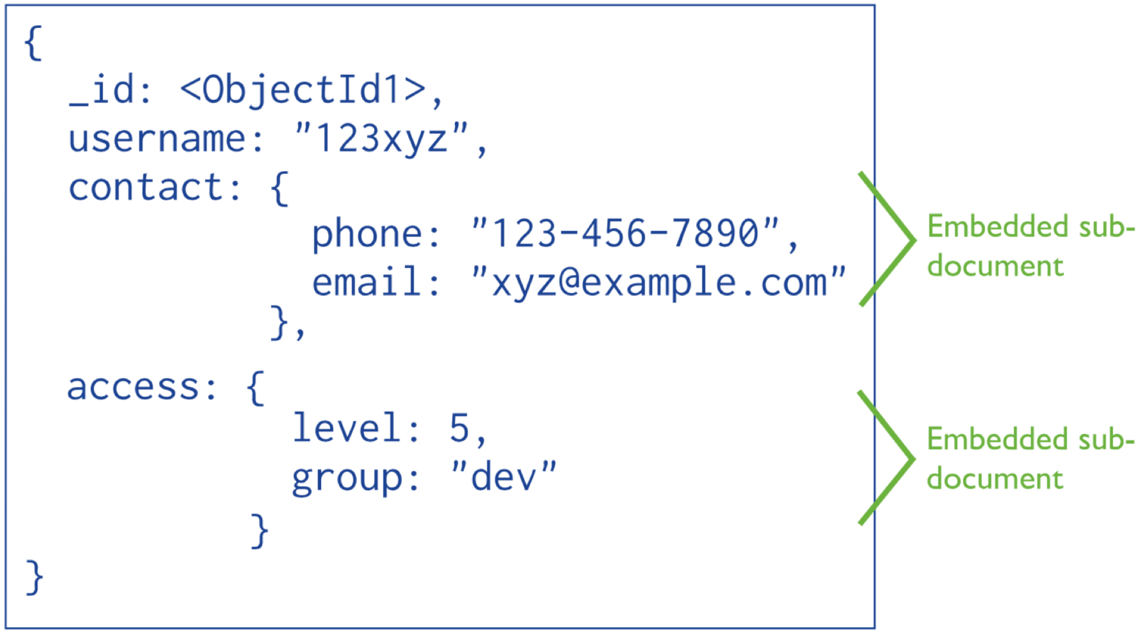
- This illustrates a query that selects and orders the matching documents using an index.
- MongoDB defines indexes at the collection level and supports indexes on any field or sub-field of documents in a collection.



MongoDB Indexing and Text Searching

Default _id Index

- MongoDB creates a unique index on the _id field during the creation of a collection.
- The _id index prevents clients from inserting two documents with the same value for the _id field. You cannot drop this index on the _id field.



```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
           }
}
```

MongoDB Indexing and Text Searching

Creating an Index

- To create an index from MongoDB shell, use *db.collection.createIndex()*

  db.collection.createIndex(*<key and index type specification>*, *<options>* **)**
- The db.collection.createIndex() method only creates an index if an index of the same specification does not already exist.
- MongoDB indexes use a B-tree data structure to store indexes.

MongoDB Indexing and Text Searching

Creating an Index from Java

```
import com.mongodb.client.ListIndexesIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import org.bson.Document;

public class CreatingIndex {
  public static void main( String args[] ) {
    // Creating a Mongo client
    MongoClient mongo = new MongoClient( "localhost" , 27017 );
    System.out.println("Connected to the database successfully");

    // Accessing the database
    MongoDatabase database = mongo.getDatabase("myDb");
```

MongoDB Indexing and Text Searching

Creating an Index from Java

```
    // Retrieving a collection
    MongoCollection<Document> collection = database.getCollection("sampleCollection");
    System.out.println("Collection sampleCollection selected successfully");

    //use 1 for ascending index , -1 for descending index
    BasicDBObject index = new BasicDBObject("description", 1);

    collection.createIndex(index);
    System.out.println("Index created successfully ");

    mongo.close();
  }
}
```

MongoDB Indexing and Text Searching

Listing an Index from Java

```
import com.mongodb.client.ListIndexesIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.BasicDBObject;
import com.mongodb.MongoClient;
import org.bson.Document;

public class CreatingIndex {
  public static void main( String args[] ) {
    // Creating a Mongo client
    MongoClient mongo = new MongoClient( "localhost" , 27017 );
    System.out.println("Connected to the database successfully");

    // Accessing the database
    MongoDatabase database = mongo.getDatabase("myDb");
```

MongoDB Indexing and Text Searching

Listing an Index from Java

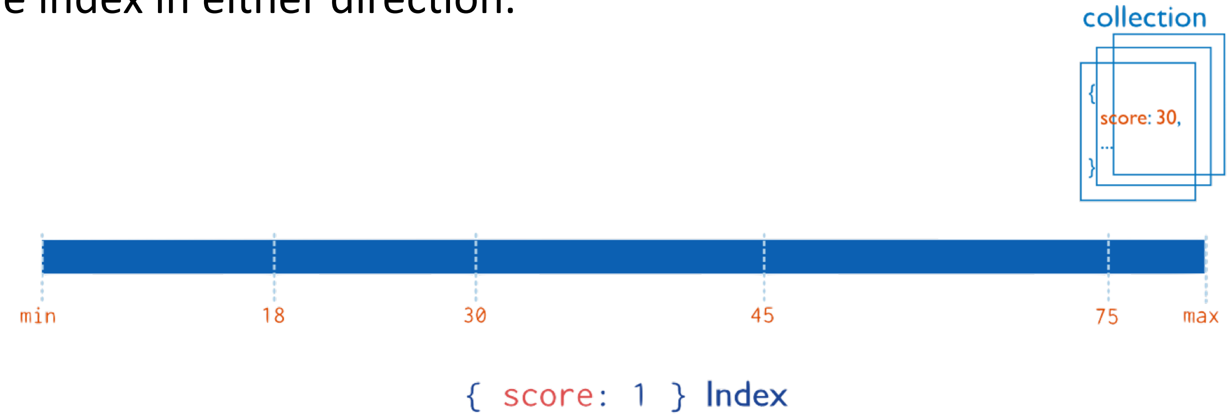
```
        // Retrieving a collection
        MongoCollection<Document> collection = database.getCollection("sampleCollection");
        System.out.println("Collection sampleCollection selected successfully");

        // Retrieving a collection
        MongoCollection<Document> collection = database.getCollection("sampleCollection");
        System.out.println("Collection sampleCollection selected");

        System.out.println("Indexes:");
        ListIndexesIterable<BasicDBObject> indexes =
                        collection.listIndexes(BasicDBObject.class);
        for (BasicDBObject index : indexes) {
            System.out.printf("index is : %s\n", index.toString());
        }

        mongo.close();
    }
}
```

MongoDB Indexing and Text Searching

Index Types

MongoDB provides a number of different index types to support specific types of data and queries

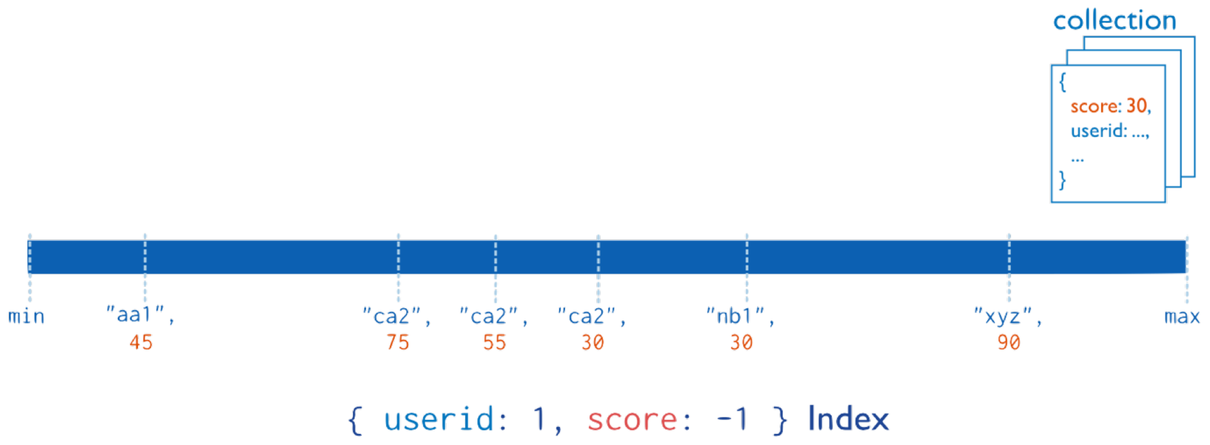
- **Single Field:**
  - In addition to the MongoDB-defined _id index, MongoDB supports the creation of user-defined ascending/descending indexes on a single field of a document.
  - For a single-field index and sort operations, the sort order (i.e. ascending or descending) of the index key does not matter because MongoDB can traverse the index in either direction.



MongoDB Indexing and Text Searching

Index Types

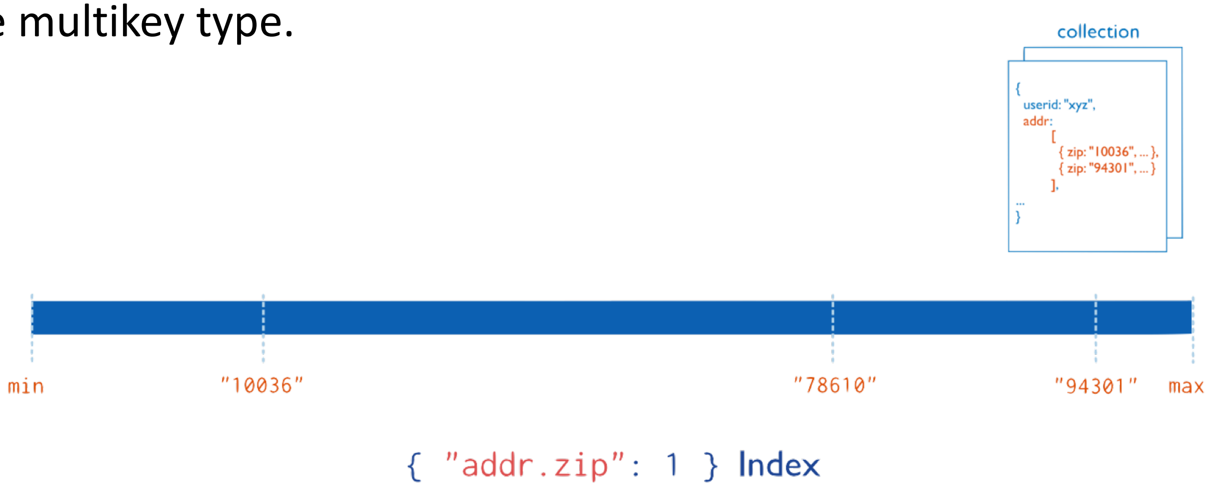
- **Compound Index**
  - MongoDB also supports user-defined indexes on multiple fields, i.e. compound indexes.
  - The order of fields listed in a compound index has significance. For instance, if a compound index consists of { **userid:** 1, **score:** -1 }, the index sorts first by **userid** and then, within each **userid** value, sorts by **score**.
  - For compound indexes and sort operations, the sort order (i.e. ascending or descending) of the index keys can determine whether the index can support a sort operation.



MongoDB Indexing and Text Searching

Index Types

- **Multikey Index**
  - MongoDB uses multikey indexes to index the content stored in arrays. If a field is indexed that holds an array value, MongoDB creates separate index entries for every element of the array.
  - These multikey indexes allow queries to select documents that contain arrays by matching on element or elements of the arrays.
  - MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value; you do not need to explicitly specify the multikey type.



```
{
  userid: "xyz",
  addr:
      [
        { zip: "10036", ... },
        { zip: "94301", ... }
      ],
...
}
```

MongoDB Indexing and Text Searching

Index Types

- **Geospatial Index**
  - To support efficient queries of geospatial coordinate data, MongoDB provides two special indexes: 2d indexes that uses planar geometry when returning results and 2dsphere indexes that use spherical geometry to return results.
- **Text Indexes**
  - MongoDB provides a text index type that supports searching for string content in a collection. These text indexes do not store language-specific stop words (e.g. “the”, “a”, “or”) and stem the words in a collection to only store root words.

MongoDB Indexing and Text Searching

Index Types

- **Hashed Index**
  - To support hash based sharding, MongoDB provides a hashed index type, which indexes the hash of the value of a field. These indexes have a more random distribution of values along their range, but only support equality matches and cannot support range-based queries.
- **Unique Indexes**
  - The unique property for an index causes MongoDB to reject duplicate values for the indexed field.
  - Other than the unique constraint, unique indexes are functionally interchangeable with other MongoDB indexes.

MongoDB Indexing and Text Searching

Index Properties

- **Partial Indexes**
  - Partial indexes only index the documents in a collection that meet a specified filter expression.
  - By indexing a subset of the documents in a collection, partial indexes have lower storage requirements and reduced performance costs for index creation and maintenance.
  - Partial indexes offer a superset of the functionality of sparse indexes and should be preferred over sparse indexes.

MongoDB Indexing and Text Searching

Index Properties

- **Sparse Indexes**
  - The sparse property of an index ensures that the index only contain entries for documents that have the indexed field. The index skips documents that do not have the indexed field.
  - You can combine the sparse index option with the unique index option to reject documents that have duplicate values for a field but ignore documents that do not have the indexed key.

MongoDB Indexing and Text Searching

Index Properties

- **TTL Indexes**
  - TTL indexes are special indexes that MongoDB can use to automatically remove documents from a collection after a certain amount of time.
  - This is ideal for certain types of information like machine generated event data, logs, and session information that only need to persist in a database for a finite amount of time.

MongoDB Indexing and Text Searching

Indexing Text

- MongoDB provides text indexes to support text search queries on string content. text indexes can include any field whose value is a string or an array of string elements.
- A collection can have at most one text index.
- Text index works by tokenizing the text and adding the tokens to the text index. Token delimiters include: quotes, terminal punctuation, and white space.
- Tokenization is language specific. For example, given a string "Il a dit qu'il «était le meilleur joueur du monde»", the text index treats «, », and spaces as delimiters.

MongoDB Indexing and Text Searching

Indexing Text

- MongoDB supports text search for various languages. text indexes drop language-specific stop words (e.g. in English, the, an, a, and, etc.) and use simple language-specific suffix stemming.
- You can specify a language for the text index; if you specify “none”, simple tokenization with no stop words and no stemming is performed.

MongoDB Indexing and Text Searching

Indexing Text

- **Creating a Text Index**
  - To create a text index, use the *db.collection.createIndex()* method from the MongoDB shell.
  - To index a field that contains a string or an array of string elements, include the field and specify the string literal "text" in the index document.

    db.reviews.createIndex({ comments: "text" })
  - To specify a default language for text , include a “default_language” attribute:

    ```
    db.reviews.createIndex(
        { comments: "text" },
        { default_language: “spanish”}
    )
    ```

MongoDB Indexing and Text Searching

Indexing Text

- **Creating a Text Index**
  - Each document or sub-document can have its own language that overrides the default language when indexing the text in the document:

```
{
    _id: 1,
    language: "portuguese",
    original: "A sorte protege os audazes.",
    translation: [
        {
            language: "english",
            quote: "Fortune favors the bold."
        },
        {
            language: "spanish",
            quote: "La suerte protege a los audaces."
        }
    ]
}
```

MongoDB Indexing and Text Searching

Indexing Text

- **Creating a Text Index in Java**

```
MongoDatabase database = mongo.getDatabase("myDb");
MongoCollection<Document> revuews= database.getCollection(”reviews");
BasicDBObject index = new BasicDBObject(”comments", “text”);
reviews.createIndex(index);
```

MongoDB Indexing and Text Searching

Indexing Text

- **Creating a Multi-field Text Index**
  - You can index multiple fields for the text index. This example creates a text index on the fields subject and comments:

```
db.reviews.createIndex(
        {
                subject: "text",
                comments: "text"
        }
)
```

  - A compound index can include text index keys in combination with ascending/descending index keys.
  - To drop a text index, use the index name

MongoDB Indexing and Text Searching

Indexing Text

- **Specifying Weights**
  - To create a text index with different field weights for the content field and the keywords field, include the weights option to the *createIndex()* method.

```
db.reviews.createIndex(
        {
                subject: "text",
                comments: "text"
        },
        {
                weights: {
                        subject: 10,
                        comments: 5
                },
                name: “TextIndex”
        }
)
```

MongoDB Indexing and Text Searching

Indexing Text

- **Naming an Index**
  - The **name** field in the index just shows gives the index a specific name.
  - The default name for an index is based on its field names and the field types. The default index name would be “subject_text_comments_text”.
  - The name “TextIndex” is more convenient when used to top the index:

    db.reviews.dropIndex(“TextIndex”);

MongoDB Indexing and Text Searching

Indexing Text

- **Wildcard Text Indexes**
  - When creating a text index on multiple fields, you can also use the wildcard specifier ($**).
  - With a wildcard text index, MongoDB indexes every field that contains string data for each document in the collection. Here is how to create a text index using the wildcard specifier:

    ```
    db.collection.createIndex( { "$**": "text" } )
    ```

  - This index allows for text search on all fields with string content. Such an index can be useful with highly unstructured data if it is unclear which fields to include in the text index or for ad-hoc querying.
  - Wildcard text indexes are text indexes on multiple fields. As such, you can assign weights to specific fields during index creation to control the ranking of the results

MongoDB Indexing and Text Searching

Indexing Text

- **Wildcard Text Indexes**
  - Wildcard text indexes, as with all text indexes, can be part of a compound indexes. For example, the following creates a compound index on the field a as well as the wildcard specifier:

    db.collection.createIndex({ **a**: 1, "$**": "text" })

  - As with all compound text indexes, since the **a** precedes the text index key, in order to perform a **$text** search with this index, the query predicate must include an equality match condition for **a**.

MongoDB Indexing and Text Searching

Indexing Text

- **Finding Indexed Text**
  - To find indexed text in a container, use the **$text** and **$search** query operators:

    db.reviews.find({ rating: { $ge 10}, $text: {$search: “falafel” })
  - This finds all reviews that received a rating of at least 10 and mentioned the word “falafel” in one of the indexed text fields.
  - The $text operator tokenizes the search string using whitespace, and most punctuations as delimiters and performs a logical OR of all search tokens in the search string.
  - For example, you could use the following query to find all stores containing any terms from the list “coffee”, “shop”, and “java”.

    db.stores.find({ $text: { $search: "java coffee shop" } })

MongoDB Indexing and Text Searching

Indexing Text

- **Finding Indexed Text**
  - You can search for exact phrases by wrapping the in double-quotes. For example, the following will find all documents containing “java” or “coffee shop”

    ```
    db.stores.find( { $text: { $search: "java \"coffee shop\"" } } )
    ```

  - To exclude a word, you can prepend a “-” character. For example, to find all stores containing “java” or “shop” but not “coffee”, use the following:

    ```
    db.stores.find( { $text: { $search: "java shop -coffee" } } )
    ```

MongoDB Indexing and Text Searching

Indexing Text

- **Finding Indexed Text**
  - MongoDB returns its results in unsorted order by default. However, text search queries will compute a relevance score for each document that specifies how well a document matches the query.
  - To sort the results in order of relevance score based on weighting, you must explicitly project the $meta” textScore field and sort on it:

```
db.stores.find(
   { $text: { $search: "java coffee shop" } },
   { score: { $meta: "textScore" } }
).sort( { score: { $meta: "textScore" } } )
```