Cheatsheets / **Learn MongoDB**

Indexing in MongoDB

MongoDB Indexes

In MongoDB, an index is a data structure that captures a subset of a collection's data in an easy to traverse form. Indexes store the value of a specific field or fields, and attach a reference to the original document to each index entry. Indexed values are organized in either ascending or descending order.

Single Field Indexes via `.createIndex()`

In MongoDB, a single field index is an index that references one field from a document. A single field index is created using the `.createIndex()` method. `<type>` specifies whether the indexed values should be arranged in ascending or descending order. A value of `1` or `-1` is used to order values in ascending or descending order, respectively.

```
db.<collection>.createIndex({ <field>: <type> })
```

Compound Indexes via `.createIndex()`

In MongoDB, a compound index is an index that contains references to multiple fields within a document. In compound indexes, the order is of particular importance in determining the effectiveness of an index for a given query. Entries for a compound index will be sorted primarily by the first field specified in the creation of the index. Then, within each value of the first field, references will be ordered by the second field, and so on.

`<type>` specifies whether the indexed values should be arranged in ascending or descending order. A value of `1` or `-1` is used to order values in ascending or descending order, respectively.

```
db.<collection>.createIndex({
  <field>: <type>,
  <field2>: <type>,
  ...
})
```

Multikey Indexes

Whenever we index a field that stores an array as its value, MongoDB will automatically create an index key for each element in the array. This type of index is known as a multikey index.

Indexing Advantages

Indexes improve speed and efficiency when querying large collections in MongoDB. Since indexes store only a subset of a collection's data, MongoDB queries can use indexes to limit the number of documents it must scan in order to find documents that match the query.

Indexing Drawbacks

Each time a document is inserted or removed from a collection, MongoDB also updates all of the indexes for that collection, leading to slower write performance for that operation. For this reason, it is important to regularly scrutinize a collection's indexes, and prune any that are unused or redundant.

The `.createIndex()` method

The `.createIndex()` method creates an an index on a collection. It accepts a document as it's first argument, where the field is the document field to be referenced in the index and the value is the type of index. Values can be `1` to order index entries in ascending order or `-1` to order index entries in descending order.
The following example creates a index for a collection named `trees` for the field `genus` in descending order:

```
db.trees.createIndex({ genus: -1
```

```
db.<collection>.createIndex({ <field>:
<type> })
```

The `.dropIndex()` method

The `.dropIndex()` method deletes an index without modifying the original collection. It accepts one argument, a string specifying the name of the index to remove, or a document that specifies the field(s) and corresponding type(s) of index to remove.
The following example drops a index named `last_name` in the `customers` collection:

```
db.customers.dropIndex("last_name
```

```
db.<collection>.dropIndex(<index>)
```

Indexing Performance Insights via `.explain()`

The `.explain()` helper method reveals metadata on the execution of a particular operation. We can provide the string, `"executionStats"`, as an argument to this method to get specific information about a query operation, such as the number of documents examined and the time the query took to execute.
Example query:

```
db.shows.find({ actor: "Tracee El
```

```
db.<collection>.find({ <field>: <parameter> }).explain('executionStats')
```

Compound Indexes

Compound indexes contain references to multiple fields. These types of indexes can support queries on all the specified fields or on any prefix of the indexed fields.

Compound Multikey Indexes

Compound multikey indexes can only support one indexed field whose value is an array. If more than one array field needs to be indexed, two separate indexes for each field would have to be created.

Multikey Index Limitations

In MongoDB, we can create compound multikey indexes as long as only one of the indexed fields stores an array as its value. If more that one field to-be-indexed is an array, multiple indexes will need to be created.
Given the following document:

```
{
  _id: ObjectId(...),
  title: "The Lion King",
  cast: ["James Earl Jones", "Don
  genres: ["live action", "advent
}
```

We would not be able to successfully create a compound index on the fields, `cast` and `genres`, but we would be able to create a compound index on the fields `cast` and `title` or `genres` and `title`.

The `.explain()` Method

In MongoDB, the `.explain()` method can be appended to the `.find()` method to gather useful metadata about the execution of a query operation.
The following example would provide insight on a query for a single field:

```
db.movies.find({ release_year: 19
```

```
db.<collection>.find().explain()
```

Print Share