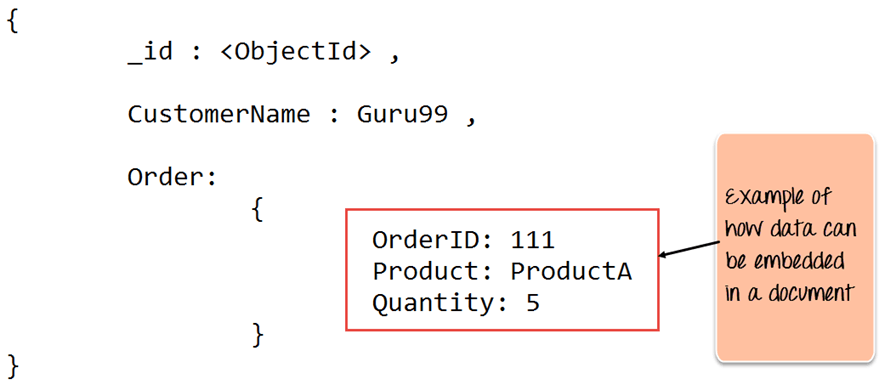
# Introduction to MongoDB

## What is MongoDB

1. MongoDB is a document database. Each database contains collections which in turn contains documents. Each document can be different with varying number of fields. The size and content of each document can be different from each other.
2. The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clear structure with key-value pairs.
3. As seen in the introduction with NoSQL databases, the rows (or documents as called in MongoDB) doesn't need to have a schema defined beforehand. Instead, the fields can be created on the fly.
4. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
5. Scalability – The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database

The below example shows how a document can be modeled in MongoDB.

1. The \_id field is added by MongoDB to uniquely identify the document in the collection.
2. What you can note is that the Order Data ( OrderID , Product and Quantity ) which in RDBMS will normally be stored in a separate table, while in MongoDB it is actually stored as an embedded document in the collection itself. This is one of the key differences of how data is modelled in MongoDB.

[](https://www.guru99.com/images/MongoDB/112015_1051_Introductio1.png)

## What Is Meant By NoSQL

NoSQL is not a relational database. It provides more flexibility since all records are not restricted by the same column names and types defined across the entire table. The below example will give a better idea of what is NoSQL.

Following 2 tables are simple example of a Customer table and an Order table wherein the Customer's table is linked to the Order's table via a relationship.

* **Customer Table**

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **CustomerName** | **OrderID** |
| 11 | Guru99 | 111 |
| 22 | Trevor Smith | 222 |
| 33 | Nicole | 333 |

* **Order Table**

|  |  |  |
| --- | --- | --- |
| **OrderID** | **Product** | **Quantity** |
| 111 | ProductA | 5 |
| 222 | ProductB | 8 |
| 333 | ProductC | 10 |

In NoSQL, the tables can probably look like the ones as shown below

* **Customer Table**

|  |  |  |  |
| --- | --- | --- | --- |
| CustomerID 11 | CustomerName Guru99 | OrderID 111 | City US |
| CustomerID 22 | CustomerName Trevor Smith | OrderID 222 | Status Privilege |
| CustomerID 33 | CustomerName Nicole | OrderID 333 |  |

* **Order Table**

|  |  |  |  |
| --- | --- | --- | --- |
| OrderID 111 | Product ProductA | Quantity 5 | Shipment Date 22-Mar-15 |
| OrderID 222 | Product ProductB | Quantity 8 |  |
| OrderID 333 | Product ProductC | Quantity 10 |  |

1. The first thing you will notice straightaway is that you don't have columns with special column names defined, but instead each field has a key-value pair.
2. You will notice that in the customer's table that the first 3 keys are the same for all 3 rows, but the fourth key (City and Status) is different for the first 2 rows and not applicable for the third row.
3. Likewise, in the Orders tables, the 2nd and 3rd row have no values defined for the 4 column (shipment date).

This is what makes NoSQL so special and unique and also very flexible. In our dynamic and ever changing technology world, business owners now demand for a faster turnaround time to software solutions.

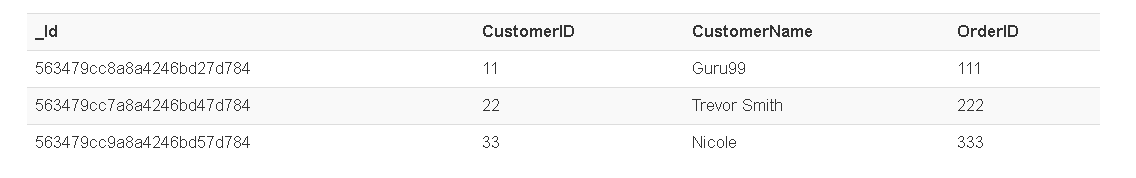
By using flexible databases such as NoSQL databases, we can inculcate a faster turnaround time, because we have more flexibility and less constraints in the way data can be defined.

Just imagine the amount of time spent in adding or editing columns to existing tables in a relational database compared to the amount of effort required in adding the same in a NoSQL database.

## Common Terms in MongoDB

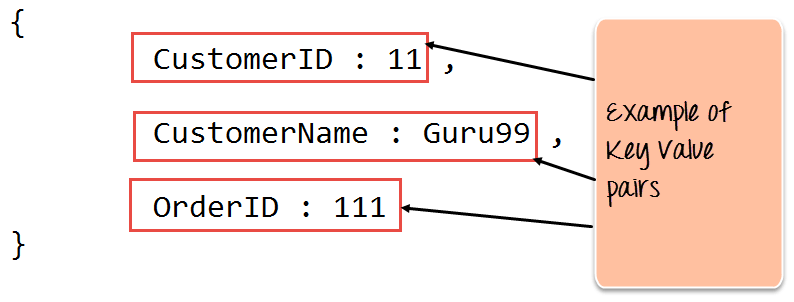
Below are the a few of the common terms used in MongoDB

1. **\_id** – This is a field required in every MongoDB document. The \_id field represents a unique value in the MongoDB document. The \_id field is like the document's primary key. If you create a new document without an \_id field, MongoDB will automatically create the field. So for example, if we see the example of the above customer table, Mongo DB will add a 24 digit unique identifier to each document in the collection.



1. **Collection** – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL. A collection exists within a single database. As seen from the introduction collections don't enforce any sort of structure.
2. **Cursor** – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.
3. **Database** – This is a container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.
4. **Document** - A record in a MongoDB collection is basically called a document. The document in turn will consist of field name and values.
5. **Field** - A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.

The following diagram shows an example of Fields with Key value pairs. So in the example below CustomerID and 11 is one of the key value pair's defined in the document.

[](https://www.guru99.com/images/MongoDB/112015_1051_Introductio2.png)

1. **JSON** – This is known as[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)Object Notation. This is a human-readable, plain text format for expressing structured data. JSON is currently supported in many programming languages.

Just a quick note on the key difference between the \_id field and a normal collection field. The \_id field is used to uniquely identify the documents in a collection and is automatically added by MongoDB when the collection is created.

## Why to Use MongoDB

Below are the few of the reasons as to why one should start using MongoDB

1. Document-oriented – Since MongoDB is a NoSQL type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situation and requirements.
2. Ad hoc queries - MongoDB supports searching by field, range queries, and regular expression searches. Queries can be made to return specific fields within documents.
3. Indexing - Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.
4. Replication - MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.
5. Load balancing - MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

## Data Modelling

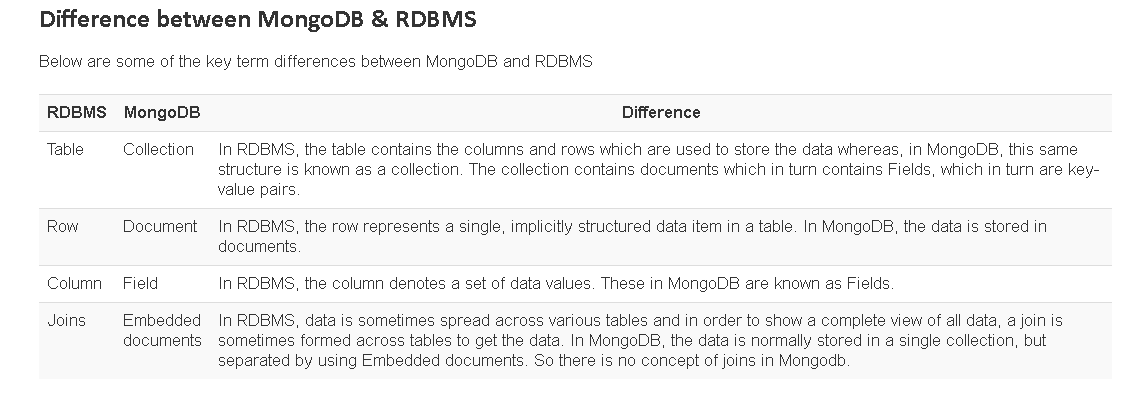
As we have seen from the Introduction section, the data in MongoDB has a flexible schema. Unlike in[SQL](https://www.guru99.com/sql.html)databases, where you must have a table's schema declared before inserting data, MongoDB's collections do not enforce document structure. This sort of flexibility is what makes MongoDB so powerful.

When modeling data in Mongo, keep the following things in mind

1. What are the needs of the application – Look at the business needs of the application and see what data and the type of data needed for the application. Based on this, ensure that the structure of the document is decided accordingly.
2. What are data retrieval patterns – If you foresee a heavy query usage then consider the use of indexes in your data model to improve the efficiency of queries.
3. Are frequent insert's, updates and removals happening in the database – Reconsider the use of indexes or incorporate sharding if required in your data modeling design to improve the efficiency of your overall MongoDB environment.

## Difference between MongoDB & RDBMS

Below are some of the key term differences between MongoDB and RDBMS



Apart from the terms differences, a few other differences are shown below

1. Relational databases are known for enforcing data integrity. This is not an explicit requirement in MongoDB.
2. RDBMS requires that data be [normalized](https://www.guru99.com/database-normalization.html) first so that it can prevent orphan records and duplicates Normalizing data then has the requirement of more tables, which will then result in more table joins, thus requiring more keys and indexes.

As databases start to grow, performance can start becoming an issue. Again this is not an explicit requirement in MongoDB. MongoDB is flexible and does not need the data to be normalized first.

<https://www.guru99.com/working-mongodb-indexes.html>

# MongoDB Indexing Tutorial - createIndex()

Indexes are very important in any database, and with MongoDB it's no different. With the use of Indexes, performing queries in MongoDB becomes more efficient.

If you had a collection with thousands of documents with no indexes, and then you query to find certain documents, then in such case MongoDB would need to scan the entire collection to find the documents. But if you had indexes, MongoDB would use these indexes to limit the number of documents that had to be searched in the collection.

Indexes are special data sets which store a partial part of the collection's data. Since the data is partial, it becomes easier to read this data. This partial set stores the value of a specific field or a set of fields ordered by the value of the field.

In this tutorial, you will learn –

* [Understanding Impact of Indexes](https://www.guru99.com/working-mongodb-indexes.html#1)
* [Create Indexes](https://www.guru99.com/working-mongodb-indexes.html#2)
* [Finding Indexes](https://www.guru99.com/working-mongodb-indexes.html#3)
* [Dropping Indexes](https://www.guru99.com/working-mongodb-indexes.html#4)

## Understanding Impact of Indexes

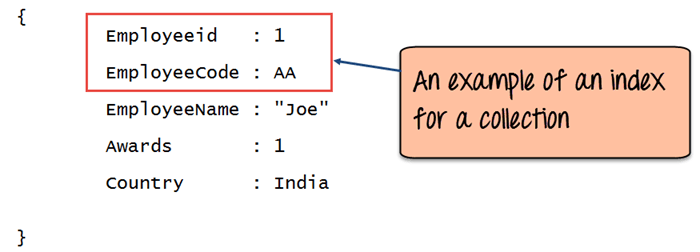
Now even though from the introduction we have seen that indexes are good for queries, but having too many indexes can slow down other operations such as the Insert, Delete and Update operation.

If there are frequent insert, delete and update operations carried out on documents, then the indexes would need to change that often, which would just be an overhead for the collection.

The below example shows an example of what field values could constitute an index in a collection. An index can either be based on just one field in the collection, or it can be based on multiple fields in the collection.

In the example below, the Employeeid "1" and EmployeeCode "AA" are used to index the documents in the collection. So when a query search is made, these indexes will be used to quickly and efficiently find the required documents in the collection.

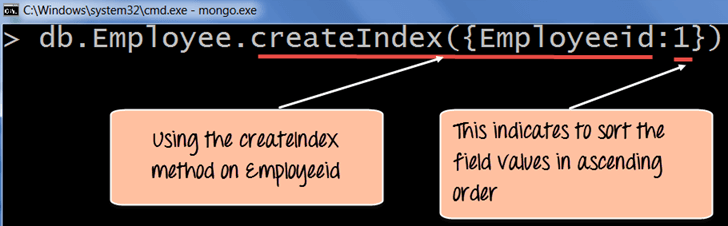
So even if the search query is based on the EmployeeCode "AA", that document would be returned.

[](https://www.guru99.com/images/MongoDB/112115_0455_Workingwith1.png)

## Mongodb createIndex() – Create Indexes

Creating an Index in MongoDB is done by using the "**createIndex**" method.

The following example shows how this can be done. Let's assume that we have our same Employee collection which has the Field names of "Employeeid" and "EmployeeName".

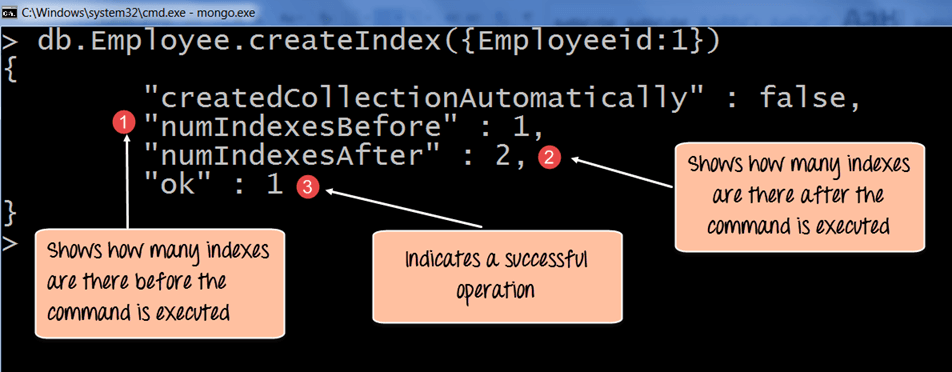
[](https://www.guru99.com/images/MongoDB/112115_0455_Workingwith2.png)

**Code Explanation:**

1. The **createIndex** method is used to create an index based on the "Employeeid" of the document.
2. The '1' parameter indicates that when the index is created with the "Employeeid" Field values, they should be sorted in ascending order. Please note that this is different from the \_id field (The id field is used to uniquely identify each document in the collection) which is created automatically in the collection by MongoDB. The documents will now be sorted as per the Employeeid and not the \_id field.

If the command is executed successfully, the following Output will be shown:

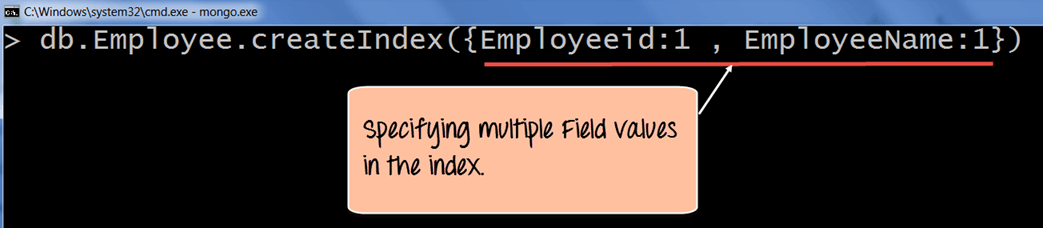
**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0455_Workingwith3.png)

1. The numIndexesBefore: 1 indicates the number of Field values (The actual fields in the collection) which were there in the indexes before the command was run. Remember that each collection has the \_id field which also counts as a Field value to the index. Since the \_id index field is part of the collection when it is initially created, the value of numIndexesBefore is 1.
2. The numIndexesAfter: 2 indicates the number of Field values which were there in the indexes after the command was run.
3. Here the "ok: 1" output specifies that the operation was successful, and the new index is added to the collection.

The above code shows how to create an index based on one field value, but one can also create an index based on multiple field values.

The following example shows how this can be done;

[](https://www.guru99.com/images/MongoDB/112115_0455_Workingwith4.png)

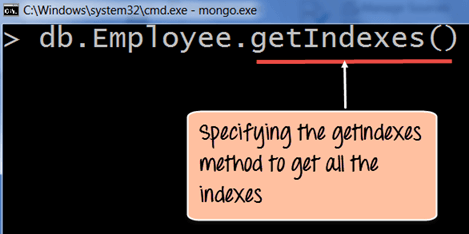
**Code Explanation:**

1. The createIndex method now takes into account multiple Field values which will now cause the index to be created based on the "Employeeid" and "EmployeeName". The Employeeid:1 and EmployeeName:1 indicates that the index should be created on these 2 field values with the :1 indicating that it should be in ascending order.

## Mongodb getindexes() – Finding Indexes

Finding an Index in MongoDB is done by using the **"getIndexes"**method.

The following example shows how this can be done;

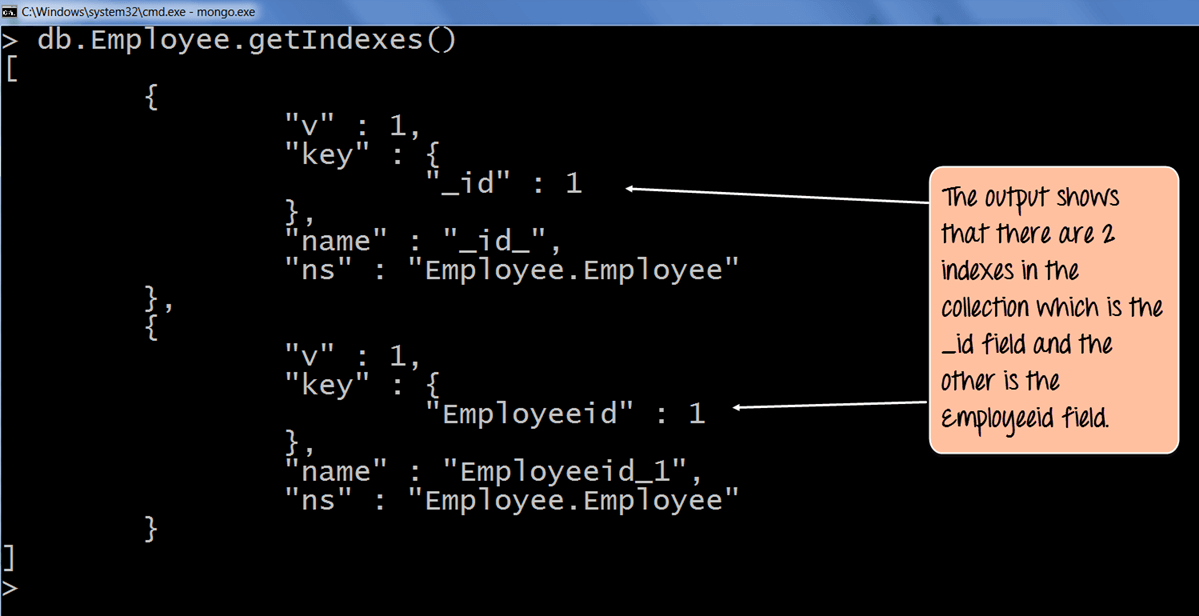
[](https://www.guru99.com/images/MongoDB/112115_0455_Workingwith5.png)

**Code Explanation:**

1. The getIndexes method is used to find all of the indexes in a collection.

If the command is executed successfully, the following Output will be shown:

**Output:**

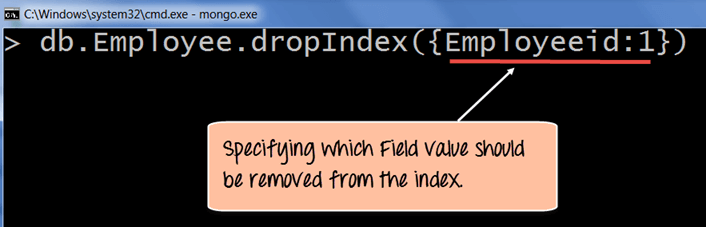
[](https://www.guru99.com/images/MongoDB/112115_0455_Workingwith6.png)

1. The output returns a document which just shows that there are 2 indexes in the collection which is the \_id field, and the other is the Employee id field. The :1 indicates that the field values in the index are created in ascending order.

## Mongodb dropindex() – Dropping Indexes

Removing an Index in MongoDB is done by using the dropIndex method.

The following example shows how this can be done;

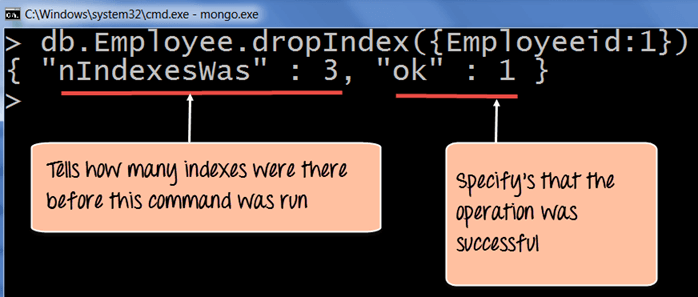
[](https://www.guru99.com/images/MongoDB/112115_0455_Workingwith7.png)

**Code Explanation:**

1. The dropIndex method takes the required Field values which needs to be removed from the Index.

If the command is executed successfully, the following Output will be shown:

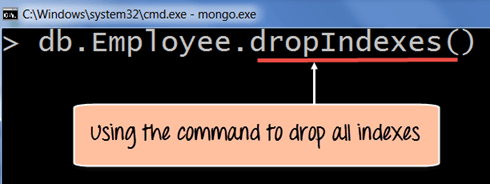
**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0455_Workingwith8.png)

1. The nIndexesWas: 3 indicates the number of Field values which were there in the indexes before the command was run. Remember that each collection has the \_id field which also counts as a Field value to the index.
2. The ok: 1 output specifies that the operation was successful, and the "Employeeid" field is removed from the index.

To remove all of the indexes at once in the collection, one can use the dropIndexes command.

The following example shows how this can be done.

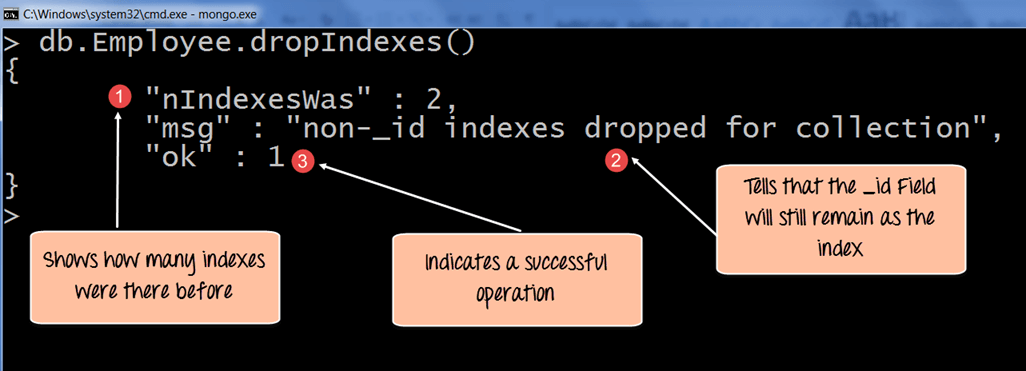
[](https://www.guru99.com/images/MongoDB/112115_0455_Workingwith9.png)

**Code Explanation:**

1. The dropIndexes method will drop all of the indexes except for the \_id index.

If the command is executed successfully, the following Output will be shown:

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0455_Workingwith10.png)

1. The nIndexesWas: 2 indicates the number of Field values which were there in the indexes before the command was run.
2. Remember again that each collection has the \_id field which also counts as a Field value to the index, and that will not be removed by MongoDB and that is what this message indicates.
3. The ok: 1 output specifies that the operation was successful.

**Summary**

* Defining indexes are important for faster and efficient searching of documents in a collection.
* Indexes can be created by using the createIndex method. Indexes can be created on just one field or multiple field values.
* Indexes can be found by using the getIndexes method.
* Indexes can be removed by using the dropIndex for single indexes or dropIndexes for dropping all indexes.