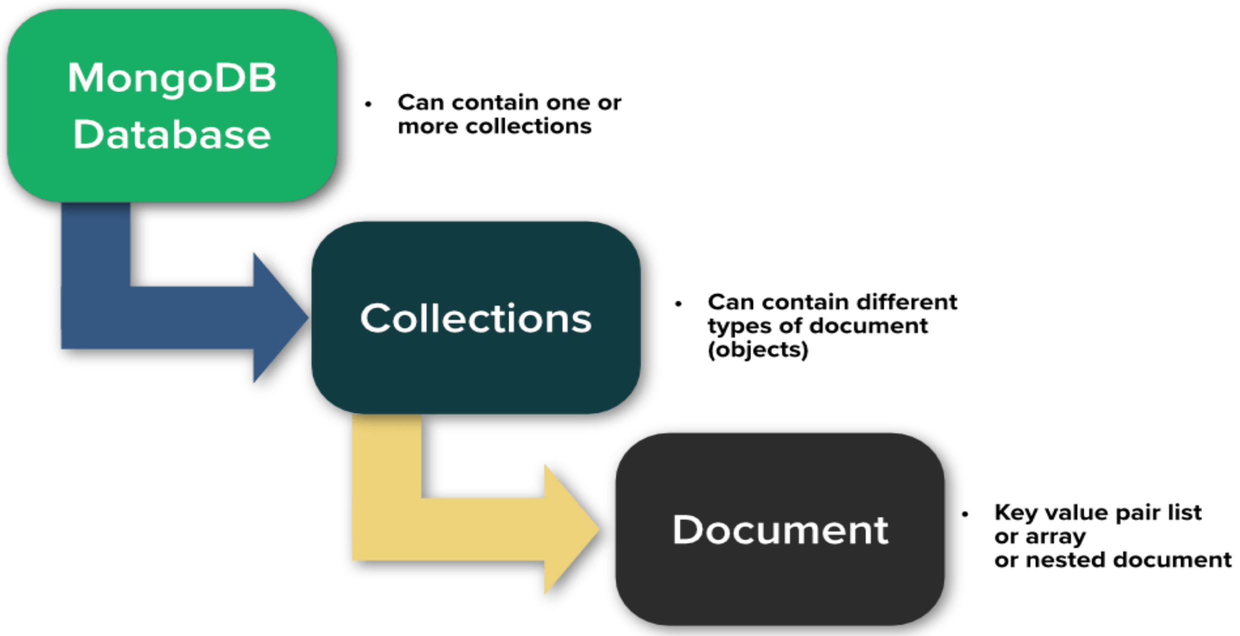
MongoDB

MongoDB – Database, Collection, and Document

Databases, collections, documents are important parts of MongoDB without them you are not able to store data on the MongoDB server.

A Database contains a collection, and a collection contains documents and the documents contain data, they are related to each other.


MongoDB Database
Can contain one or more collections
Collections
Can contain different types of document (objects)
Document
Key value pair list or array or nested document

Database

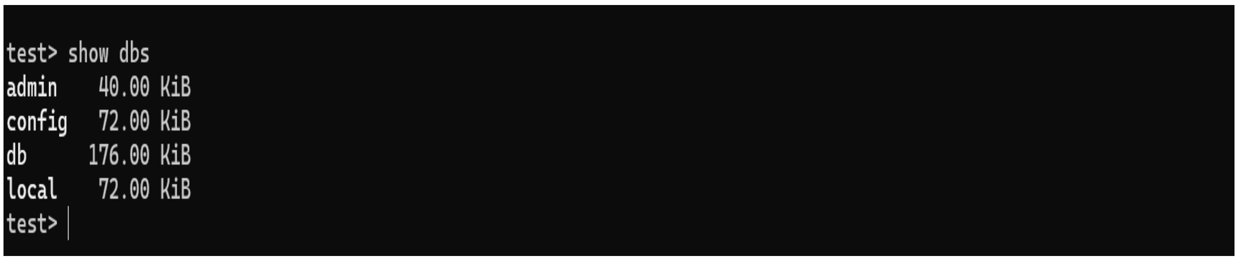
In MongoDB, a database contains the collections of documents.

One can create multiple databases on the MongoDB server.

- Flexible schema
- High availability
- Horizontal scaling

View Database:

To see how many databases are present in your MongoDB server, write the following statement in the mongo shell:



```
test> show dbs
admin    40.00 KiB
config   72.00 KiB
db      176.00 KiB
local    72.00 KiB
test>
```

Naming Restriction for Database:

Before creating a database you should first learn about the naming restrictions for databases:

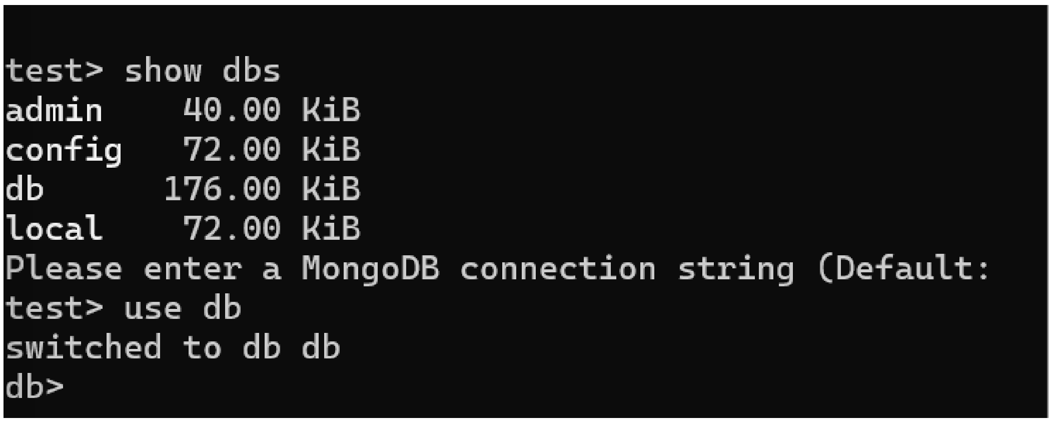
- In MongoDB, the names of the database are case insensitive, but you must always remember that the database names cannot differ only by the case of the characters.
- For windows user, MongoDB database names cannot contain any of these following characters:
- **/\. "$*:|?**
- MongoDB database names cannot contain null characters(in windows, Unix, and Linux systems).
- MongoDB database names cannot be empty and must contain less than 64 characters.

Creating Database:

In the mongo shell, you can create a database with the help of the following command:

```
use database_name
```

This command actually switches you to the new database if the given name does not exist and if the given name exists, then it will switch you to the existing database.



```
test> show dbs
admin    40.00 KiB
config   72.00 KiB
db      176.00 KiB
local    72.00 KiB
Please enter a MongoDB connection string (Default:
test> use db
switched to db db
db>
```

Collection

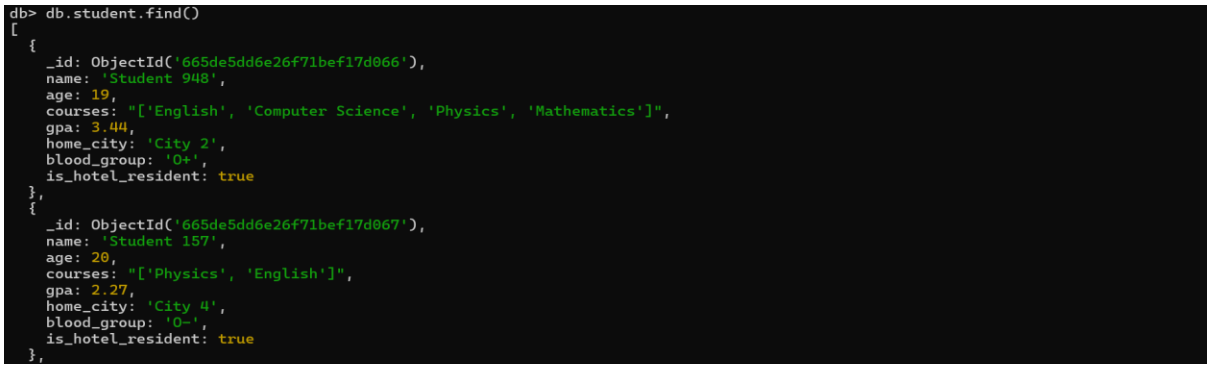
Collections are just like tables in relational databases, they also store data, but in the form of documents.

A single database is allowed to store multiple collections.

- Flexible structure
- Dynamic indexing
- Embedded documents
- Efficient querying

Schemaless:

As we know that MongoDB databases are schemaless. So, it is not necessary in a collection that the schema of one document is similar to another document. Or in other words, a single collection contains different types of documents like as shown in the below example where mystudentData collection contain two different types of documents:



```
db> db.student.find()
[
  {
    _id: ObjectId('665de5dd6e26f71bef17d066'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('665de5dd6e26f71bef17d067'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.27,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
```

Naming Restrictions for Collection:

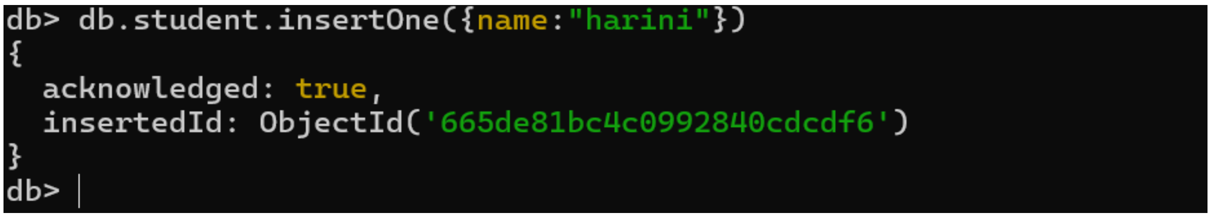
- Collection name must starts with an underscore or a character.
- Collection name does not contain $, empty string, null character and does not begin with system i.e prefix.
- The maximum length of the collection name is 120 bytes (including the database name, dot separator, and the collection name).

Creating collection:

After creating database now we create a collection to store documents. The collection is created using the following syntax:

```
db.collection_name.insertOne({key:value})
```

Here, insertOne() function is used to store single data in the specified collection. And in the curly braces {} we store our data or in other words, it is a document.



```
db> db.student.insertOne({name:"harini"})
{
  acknowledged: true,
  insertedId: ObjectId('665de81bc4c0992840cdcdf6')
}
db> |
```

Document

In MongoDB, the data records are stored as BSON documents.Here, BSON stands for binary representation of JSON documents, although BSON contains more data types as compared to JSON. The document is created using field-value pairs or key-value pairs and the value of the field can be of any BSON type.

- JSON-like structure
- Dynamic schema
- Nested data
- Query efficiency

Syntax:

```
{
field1: value1
field2: value2
....
fieldN: valueN
}
```

Naming restriction of fields:

- The field names are of strings.
- The _id field name is reserved to use as a primary key. And the value of this field must be unique, immutable, and can be of any type other than an array.
- The field name cannot contain null characters.
- The top-level field names should not start with a dollar sign ($).
- Document Size: The maximum size of the BSON document is 16MB. It ensures that the single document does not use too much amount of RAM or bandwidth(during transmission). If a document contains more data than the specified size, then MongoDB provides a GridFS API to store such type of documents.

NOTE:
A single document may contain duplicate fields.

MongoDB always saves the order of the fields in the documents except for the _id field (which always comes in the first place) and the renaming of fields may change the order of the fields in the documents.

_id Field: In MongoDB, every document store in the collection must contain a unique _id field it is just like a primary key in a relational database. The value of the _id field can be set by the user or by the system (if the user does not create an _id field, then the system will automatically generate an ObjectId for _id field).

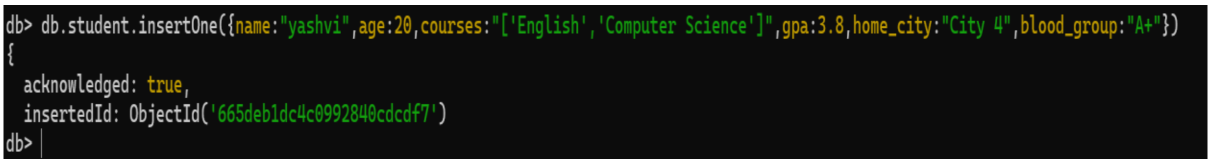
When you create a collection MongoDB automatically creates a unique index on the _id field.

The _id field is the first field of every document.

The value of the _id field can be of any BSON type except arrays.

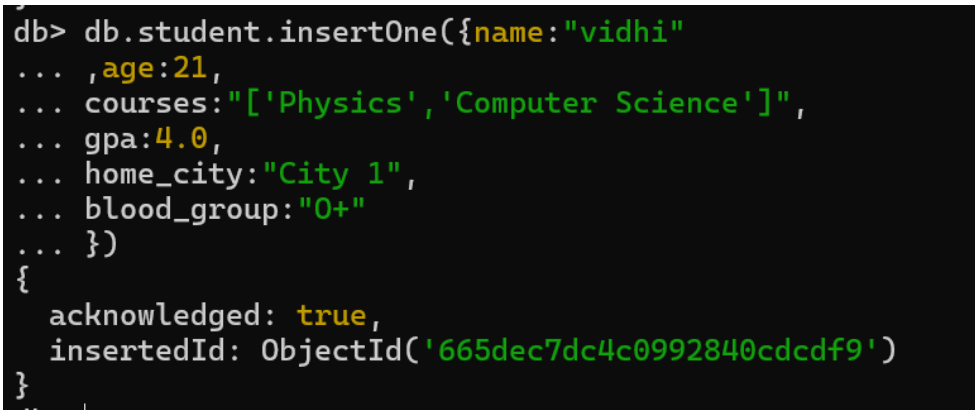
The default value of the _id field is ObjectId.

Example 1:



```
db> db.student.insertOne({name:"yashvi",age:20,courses:"['English','Computer Science']",gpa:3.8,home_city:"City 4",blood_group:"A+"})
{
  acknowledged: true,
  insertedId: ObjectId('665deb1dc4c0992840cdcdf7')
db>
```

Example 2:



```
db> db.student.insertOne({name:"vidhi"
... ,age:21,
... courses:"['Physics','Computer Science']",
... gpa:4.0,
... home_city:"City 1",
... blood_group:"O+"
... })
{
  acknowledged: true,
  insertedId: ObjectId('665dec7dc4c0992840cdcdf9')
}
```