mongoDB

By Naveen

The database for modern applications

As a programmer, you think in objects. Now your database is also in object model.

MongoDB is a document database, which means it stores data in JSON-like documents. We believe this is the most natural way to think about data, and is much more expressive and powerful than the traditional row/column model.

Rich JSON Documents

- The most natural and productive way to work with data.
- Supports arrays and nested objects as values.
- Allows for flexible and dynamic schemas.

Powerful query language

- Rich and expressive query language that allows you to filter and sort by any field, no matter how nested it may be within a document.
- Support for aggregations and other modern use-cases such as geo-based search, graph search, and text search.
- Queries are themselves JSON, and thus easily composable. No more concatenating strings to dynamically generate SQL queries.

What is MongoDB ?

MongoDB, developed by MongoDB Inc., is a free and open source cross-platform database program.

It is categorized as a NoSQL database program which stores data in JSON-like documents.

A **document** is a set of key-value pairs defined by a **schema** which defines how the data is structured and organized in a database.

A group of documents makes up a **collection,** and documents within the same collection can have different schema. This is referred to as **dynamic schema**.

Some of the users of MongoDB are Google, Facebook, Lyft, Adobe, etc.

Used by millions of developers to power the world's most innovative products and services
invision
ebay
Adobe
Google
SQUARESPACE
coinbase
SEGA
intuit
eharmony
EA
verizon
shutterfly
GOV.UK
SAP

Advantages

- High scalability
- High availability
- Big data capability
- Easy replication
- Fast performance
- High flexibility

What is Schema ?

A **schema** is the structure behind data organization.

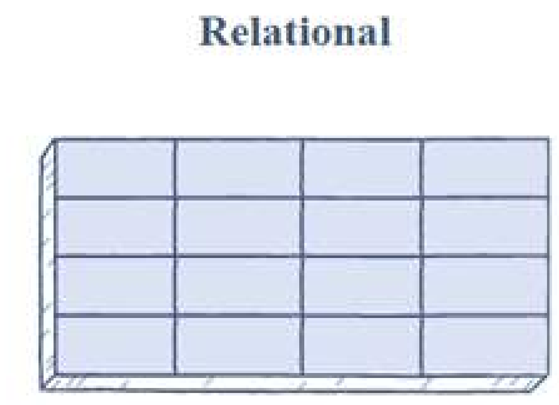
What is SQL?

Structured Query language (SQL) **pronounced as "S-Q-L" or sometimes as "See-Quel"**is actually the standard language for dealing with Relational Databases.

SQL programming can be effectively used to insert, search, update, delete database records.

Relational databases like MySQL Database, Oracle, Ms SQL server, Sybase, etc uses SQL.

- **Relational database**: is a collective set of multiple data sets organized by tables, records and columns.



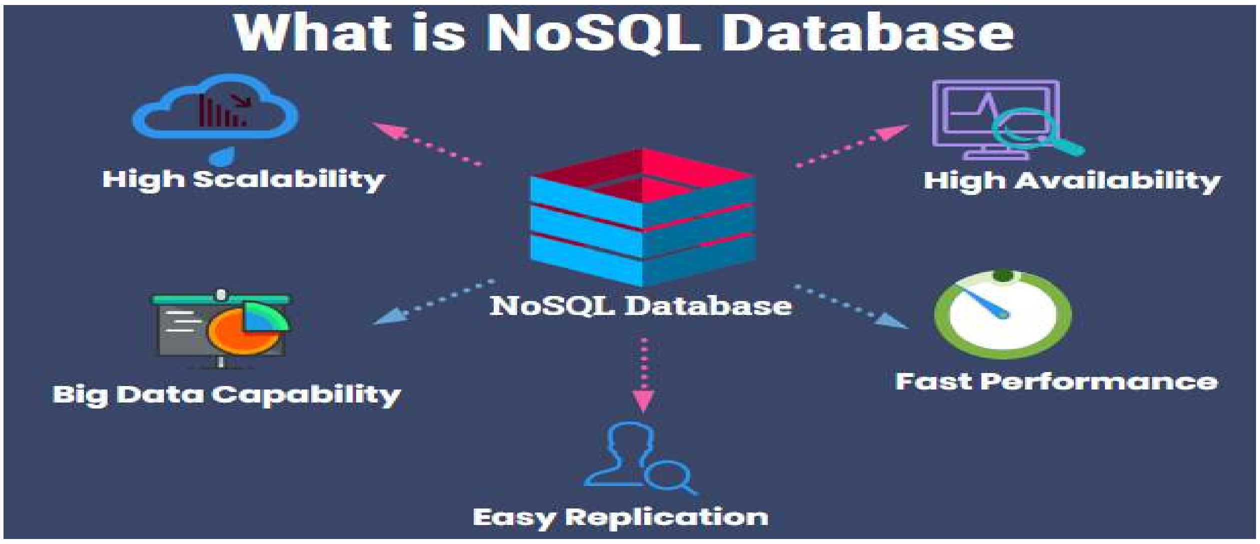
NoSQL Database

NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications.

NOSQL means **"Not only SQL".**

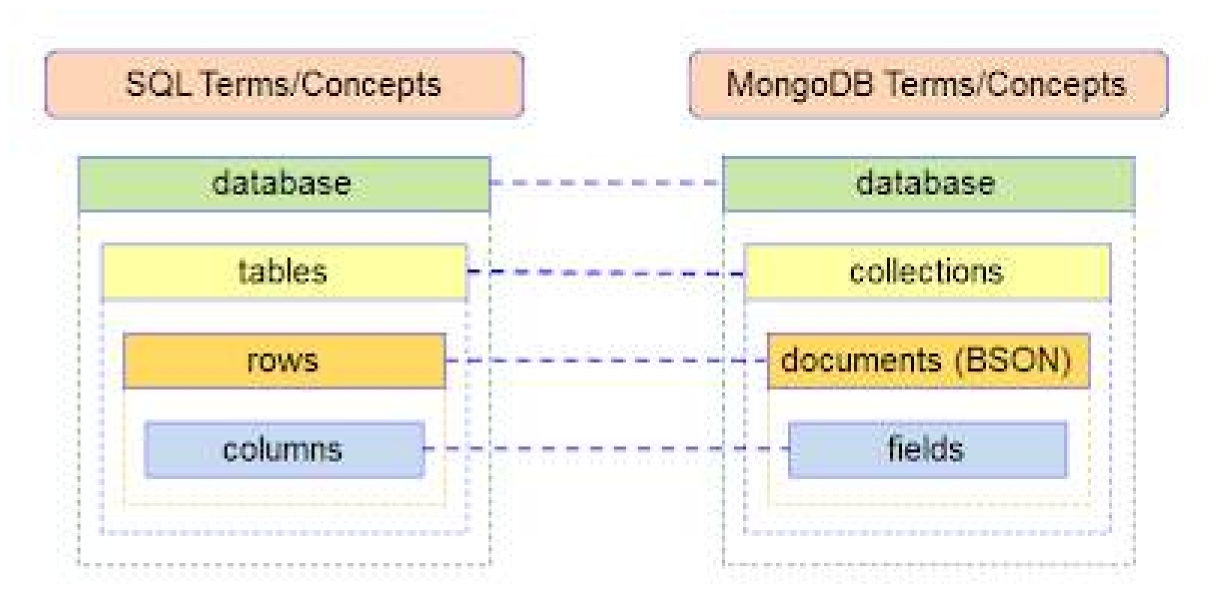
NoSQL databases are categorized as Non-relational or distributed database system.

MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase, Neo4j, CouchDB etc. are the example of nosql database.



Differences between SQL and NoSQL database:

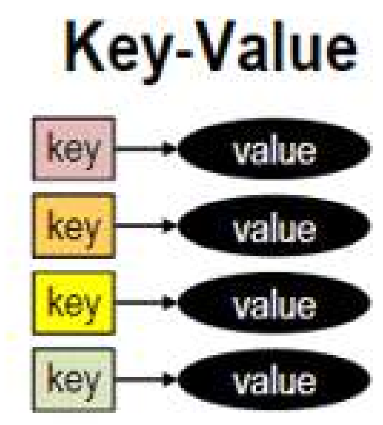
No	SQL	NoSQL
1)	Databases are categorized as Relational Database Management System (RDBMS).	NoSQL databases are categorized as Non-relational or distributed database system.
2)	SQL databases have fixed or static or predefined schema.	NoSQL databases have dynamic schema.
3)	SQL databases display data in form of tables so it is known as table-based database.	NoSQL databases display data as collection of key-value pair, documents, graph databases or wide-column stores.
4)	SQL databases are vertically scalable.	NoSQL databases are horizontally scalable.
5)	SQL databases use a powerful language "Structured Query Language" to define and manipulate the data.	In NoSQL databases, collection of documents are used to query the data. It is also called unstructured query language. It varies from database to database.
6)	SQL databases are best suited for complex queries.	NoSQL databases are not so good for complex queries because these are not as powerful as SQL queries.
7)	SQL databases are not best suited for hierarchical data storage.	NoSQL databases are best suited for hierarchical data storage.
8)	MySQL, Oracle, Sqlite, PostgreSQL and MS-SQL etc. are the example of SQL database.	MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase, Neo4j, CouchDB etc. are the example of nosql database



Types of NoSQL Database

There are four primary types of NoSQL database i.e. **Key-value stores, Document databases, wide- column stores and graph stores**.

1. **Key-Value Stores:** These databases work on a simple data model that has a pair of unique keys and a value associated with it. These databases perform efficiently and show high scalability for caching in web applications and session management.



Examples of Key-Value Stores

Phone Directory

Key	Value
Bob	(123) 456-7890
Jane	(234) 567-8901
Tara	(345) 678-9012
Tiara	(456) 789-0123

Stock Trading

This example uses a list as the value.

The list contains the stock ticker, whether its a “buy” or “sell” order, the number of shares, and the price.

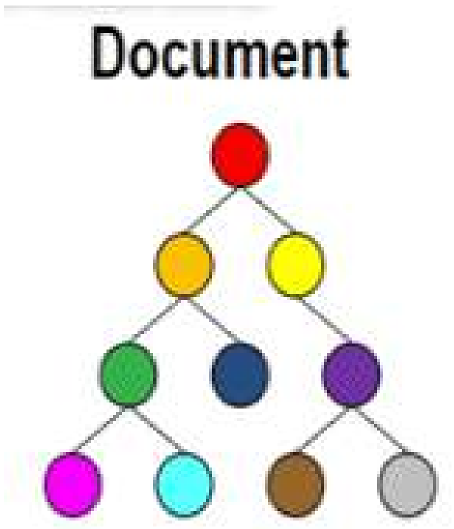
Key	Value
123456789	APPL, Buy, 100, 84.47
234567890	CERN, Sell, 50, 52.78
345678901	JAZZ, Buy, 235, 145.06
456789012	AVGO, Buy, 300, 124.50

Examples of Key-Value Database Management Systems

Here are some of the DBMSs that use the key-value approach.

- Redis
- Oracle NoSQL Database
- Voldemorte
- Aerospike
- Oracle Berkeley DB

2. **Document databases:** These databases store semi-structured data and their descriptions in document format. They do not refer to master schema for creating and updating programs. Their usage has increased with the use of JavaScript and JSON (JavaScript Object Notation). These are used for mobile application data handling and content management.



Example

```
[
0 : {    "train_num": 12235,
          "name":  "Dibrugarh - New Delhi Rajdhani Express",
        "train_from": "DBRG",
        "train_to": "NDLS",
        "data":  {   ...  }
   }
1 : {  "train_num": 12236,
        "name": "New Delhi - Dibrugarh Rajdhani Express",
        "train_from": "NDLS",
        "train_to":  "DBRG",
        "data":{ ... }
}
2 : {  "train_num": 12301 ,
        "name":  "Howrah - New Delhi Rajdhani Express (via
Gaya)",
        "train_from":  "HWH",
        "train_to": "NDLS",
        "data":{  ...  }
} ]
```

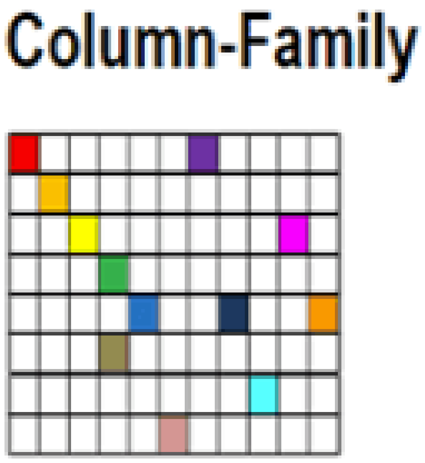
Examples of Document Store DBMSs

There are many document oriented database management systems available. Some are open source, others are proprietary.

Here are examples of some of the leading document store DBMSs.

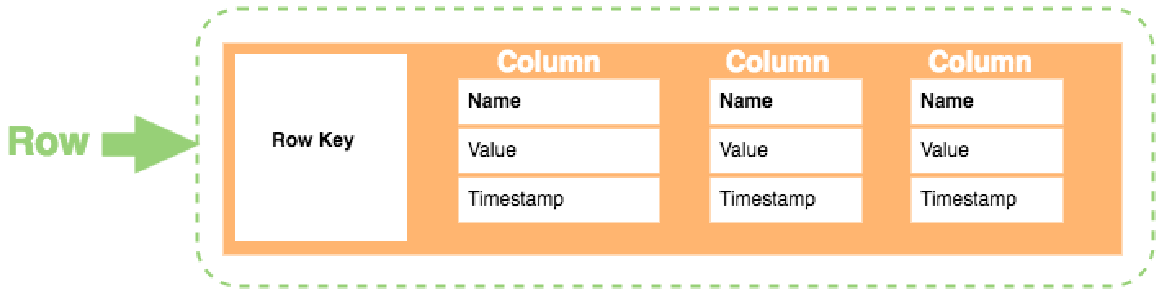
- MongoDB
- DocumentDB
- CouchDB
- MarkLogic
- OrientDB

3. **Wide- column stores:** These databases organize data in columns instead of rows. They can query large data sets faster than other conventional databases. These are used for catalogs, fraud detection, and recommendation engines.



- A **column family** consists of multiple rows.
- Each **row** can contain a different number of columns to the other rows. And the columns don't have to match the columns in the other rows (i.e. they can have different column names, data types, etc).
- Each **column** is contained to its row. It doesn't span all rows like in a relational database. Each column contains a name/value pair, along with a timestamp.

  Here's how each row is constructed:



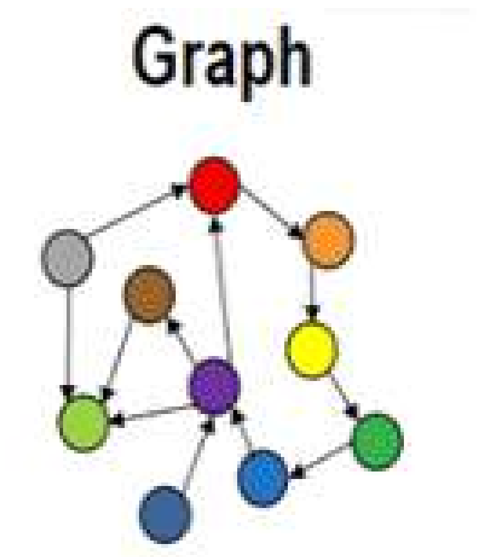
Here's a breakdown of each element in the row:

- **Row Key**. Each row has a unique key, which is a unique identifier for that row.
- **Column**. Each column contains a name, a value, and timestamp.
- **Name**. This is the name of the name/value pair.
- **Value**. This is the value of the name/value pair.
- **Timestamp**. This provides the date and time that the data was inserted. This can be used to determine the most recent version of data.

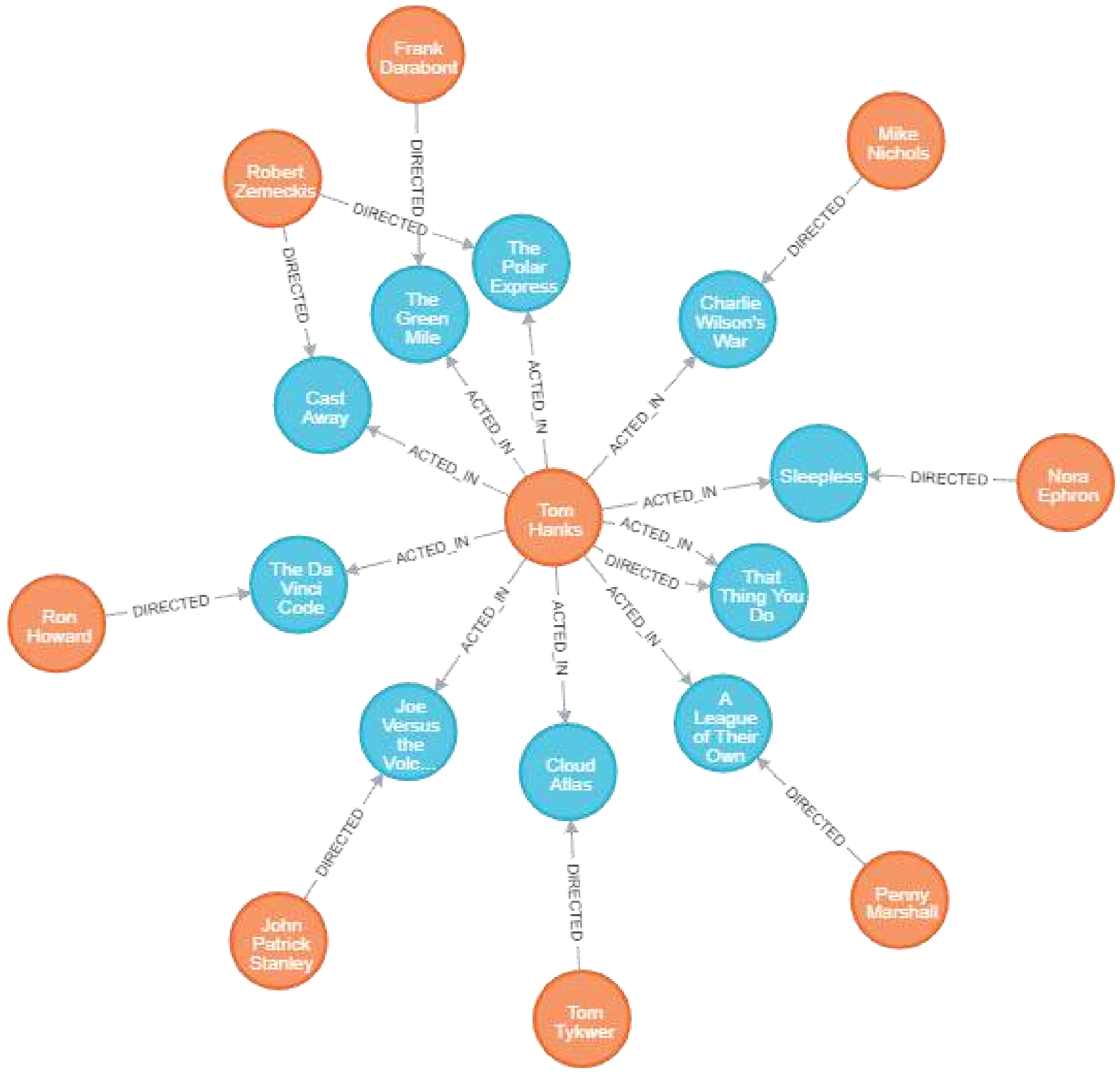
Examples of Column Store DBMSs

- Bigtable
- Cassandra
- HBase
- Vertica
- Druid
- Accumulo
- Hypertable

4. **Graph Stores:** These databases organize data as nodes and edges that show connections between nodes. These are used where map relationships are needed like customer relationship or reservation systems management.



Example



Examples of Graph Databases

Examples of graph databases include

Neo4j

Blazegraph

OrientDB.

Using Python to Perform Operations on MongoDB

Step 1 : Import the *MongoClient* from *PyMongo.*

```
from pymongo import MongoClient
```

Step 2: Create a client connection to the MongoDb instance running on the local machine.

```
client = MongoClient('localhost:27017')
```

Step 3: Define a database object to insert data into the MongoDb database.

```
db = client.EmployeeData
```

Step 4: Use the mongo db API to **insert** document to the database .

```
db.Employees.insert_one(
    {
    "id": employeeId,
      "name":employeeName,
    "age":employeeAge,
    "country":employeeCountry
    })
```

Step 5: For **Reading** a document from MongoDb we have to call a ***find*** method on the collection and it would return all the documents in the collection.

```
empCol = db.Employees.find()
for emp in empCol:        print emp
```

Step 6: Update a document from the collection, we'll set a criteria according to which we'll update the collection. In this case, we'll consider the employee id to update the document in the collection.

```
db.Employees.update_one(
    {"id": criteria},
    {
    "$set": {
        "name":name,
        "age":age,
        "country":country
    }
    }
)
```

Step 7: To **remove** a document from the collection we use **delete** method need. We remove the document based on the employee id.

```
db.Employees.delete_many({"id":101})
print ('Deletion successful' )
```

Installing PyMongo

The following command is used to install PyMongo.

```
pip install pymongo
```

We install PyMongo with pip.

PyMongo create collection

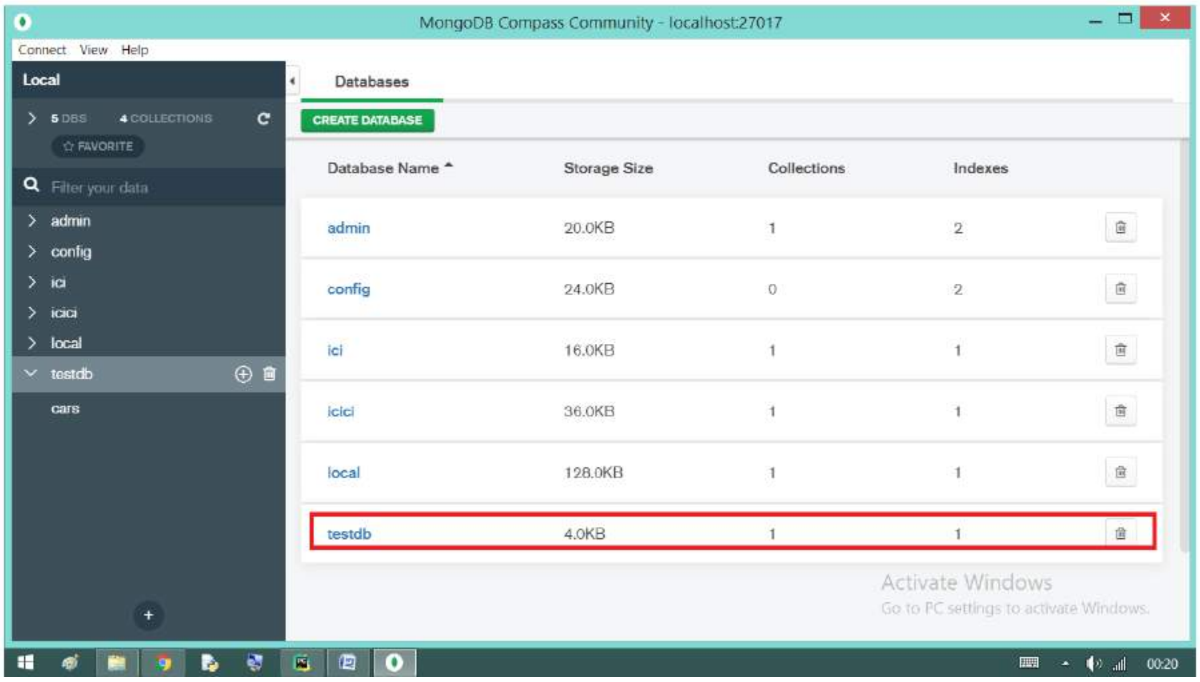
In the first example, we create a new collection. MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

create_collection.py

```
from pymongo import MongoClient

cars = [ {'name': 'Audi', 'price': 52642},

    {'name': 'Mercedes', 'price': 57127},

    {'name': 'Skoda', 'price': 9000},

    {'name': 'Volvo', 'price': 29000},

    {'name': 'Bentley', 'price': 350000},

    {'name': 'Citroen', 'price': 21000},

    {'name': 'Hummer', 'price': 41400},

    {'name': 'Volkswagen', 'price': 21600} ]

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    db.cars.insert_many(cars)
```

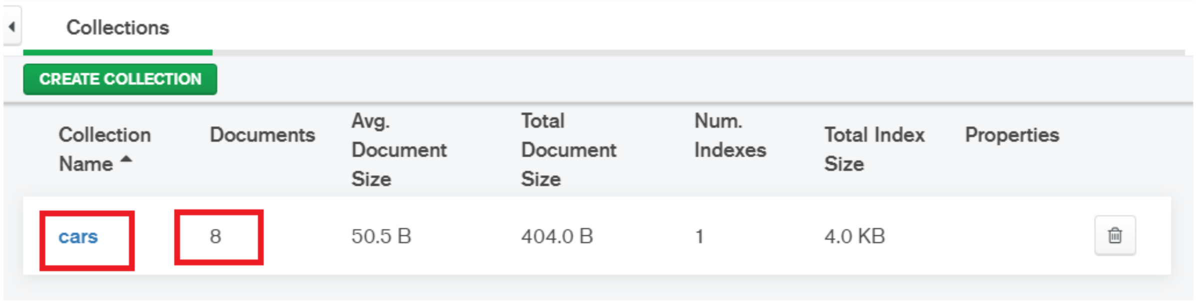
Note: With insert_many() method, we insert eight documents into the cars collection, which is automatically created as well

A New database is created.



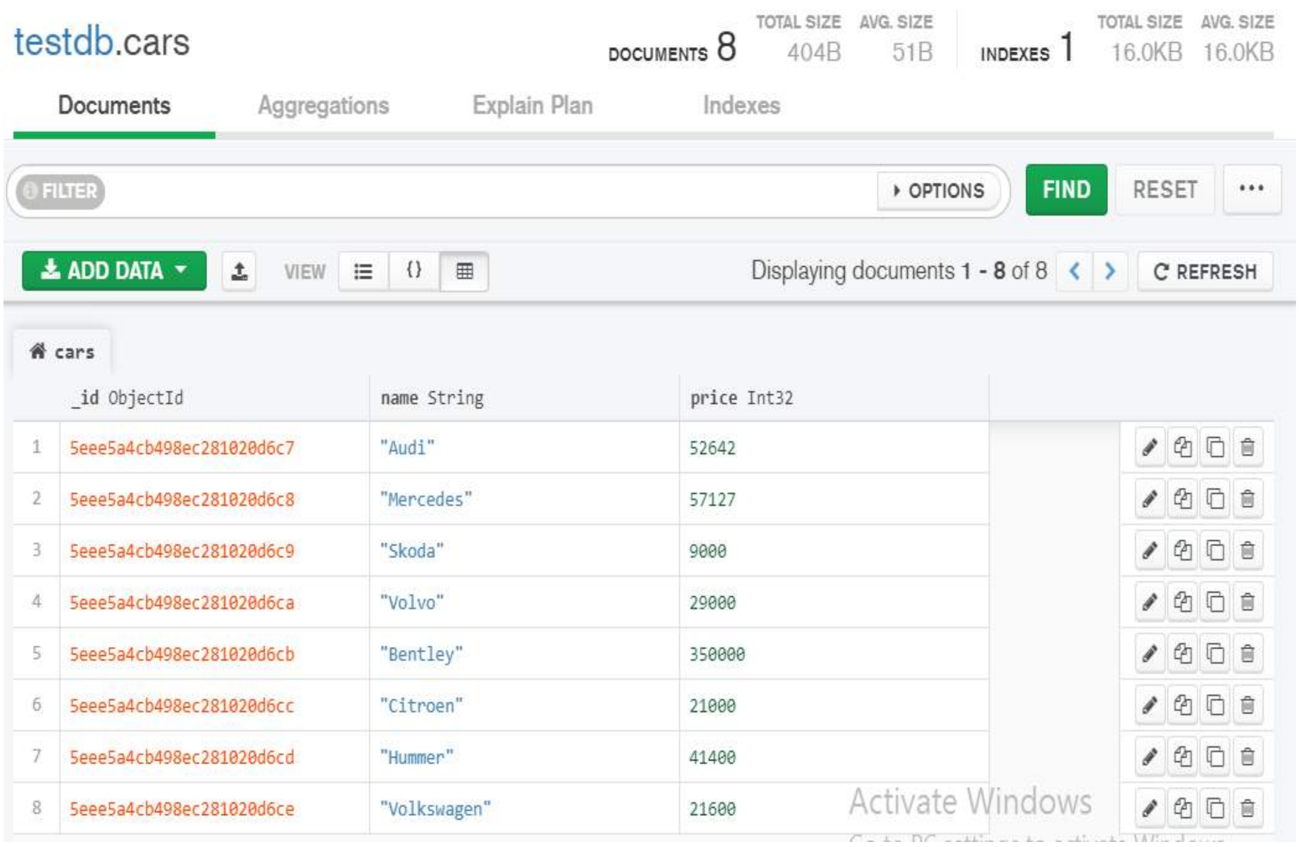
Database Name	Storage Size	Collections	Indexes
admin	20.0KB	1	2
config	24.0KB	0	2
ici	16.0KB	1	1
icici	36.0KB	1	1
local	128.0KB	1	1
testdb	4.0KB	1	1

A New Collection is CREATED



Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
cars	8	50.5 B	404.0 B	1	4.0 KB	

Documents in Cars collection



	_id ObjectId	name String	price Int32
1	5eee5a4cb498ec281020d6c7	"Audi"	52642
2	5eee5a4cb498ec281020d6c8	"Mercedes"	57127
3	5eee5a4cb498ec281020d6c9	"Skoda"	9000
4	5eee5a4cb498ec281020d6ca	"Volvo"	29000
5	5eee5a4cb498ec281020d6cb	"Bentley"	350000
6	5eee5a4cb498ec281020d6cc	"Citroen"	21000
7	5eee5a4cb498ec281020d6cd	"Hummer"	41400
8	5eee5a4cb498ec281020d6ce	"Volkswagen"	21600

PyMongo list collections

With **collection_names(),** we get list available collections in the database.

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    print(db.collection_names())
```

The example prints collections in the testdb database.

Output:

```
['cars']
```

PyMongo drop collection

The **drop()** method removes a collection from the database.

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    db.cars.drop()
```

PyMongo running commands

We can issue commnads to MongoDB with **command().** The server Status command returns the status of the MongoDB server.

```
from pymongo import MongoClient

from pprint import pprint

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    status = db.command("serverStatus")

    pprint(status)
```

Example 2

The dbstats command returns statistics that reflect the use state of a single database.

```
from pymongo import MongoClient

from pprint import pprint

client = MongoClient('mongodb://localhost:27017/')

with client:

   db = client.testdb

   print(db.collection_names())

   status = db.command("dbstats")

   pprint(status)
```

PyMongo cursor

The find methods return a PyMongo cursor, which is a reference to the result set of a query.

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

   db = client.testdb

   cars = db.cars.find()

   print(cars.next())

   print(cars.next())
   print(cars.next())
   cars.rewind()
   print(cars.next())
   print(cars.next())
   print(cars.next())
   print(list(cars))
```

cars = db.cars.find()

The find() method returns a PyMongo cursor.

print(cars.next())

With the next() method, we get the next document from the result set.

cars.rewind()

The rewind() method rewinds the cursor to its unevaluated state.

print(list(cars))

With the list() method, we can transform the cursor to a Python list. It loads all data into the memory.

PyMongo read all data

In the following example, we read all records from the collection. We use Python for loop to traverse the returned cursor.

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find()

    for car in cars:

        print('{0} {1}'.format(car['name'],  car['price']))
```

cars = db.cars.find()

The **find()** method selects documents in a collection or view and returns a cursor to the selected documents. A cursor is a reference to the result set of a query.

Output

```
Audi 52642

Mercedes 57127

Skoda 9000

Volvo 29000 ......
```

PyMongo count documents

The number of documents is retrieved with the **count()** method.

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')
```

```
with client:
    db = client.testdb
    n_cars = db.cars.find().count()
    print("There are {} cars".format(n_cars))
```

Output:

```
There are 8 cars
```

PyMongo filters

The first parameter of **find()** and **find_one()** is a filter. The filter is a condition that all documents must match.

```
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')
with client:
    db = client.testdb
    expensive_cars = db.cars.find({'price': {'$gt': 50000}})
    for ecar in expensive_cars:
        print(ecar['name'])
```

The example prints the names of cars whose price is greater than 50000.

```
expensive_cars = db.cars.find({'price': {'$gt': 50000}})
```

The first parameter of the find() method is the filter that all returned records must match. The filter uses the $gt operator to return only expensive cars.

Output

```
Audi

Mercedes

Bentley
```

PyMongo sorting documents

We can sort documents with sort().

```
from pymongo import MongoClient, DESCENDING

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find().sort("price", DESCENDING)

    for car in cars:

        print('{0} {1}'.format(car['name'], car['price']))
```

Output

```
Bentley 350000

Mercedes 57127

Audi 52642

Hummer 41400

Volvo 29000

Volkswagen 21600 ...
```

Example 2 ASCENDING

```
from pymongo import MongoClient
from pymongo import ASCENDING
client = MongoClient('mongodb://localhost:27017/')
with client:
    db = client.testdb
    cars = db.cars.find().sort("price", ASCENDING)
    for car in cars:
        print('{0} {1}'.format(car['name'], car['price']))
```

Output

```
Skoda 9000

Citroen 21000

Volkswagen 21600

Volvo 29000 ......
```

PyMongo limit data output

The limit query option specifies the number of documents to be returned and the **skip()** option some documents.

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')

with client:

    db = client.testdb

    cars = db.cars.find().skip(2).limit(3)

    for car in cars:
```

```
    print('{0}: {1}'.format(car['name'], car['price']))
```

The example reads from the cars collection, skips the first two documents, and limits the output to three documents.

```
cars = db.cars.find().skip(2).limit(3)
```

The **skip()** method skips the first two documents and the **limit()** method limits the output to three documents.

Output

```
Skoda: 9000

Volvo: 29000

Bentley: 350000
```

Where Clouse Example

```
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')
with client:
    db = client.testdb
    cars = db.cars.find().where('this.price<15000')
    print(list(cars))
```

Find one record Example

```
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')
with client:
    db = client.testdb
    car = db.cars.find_one({'name': 'Volvo'})
    print(car)
```