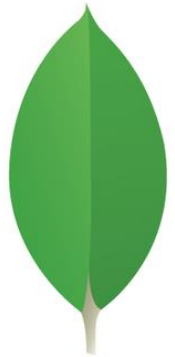
Cognixia

A Collabera LEARNING SOLUTIONS COMPANY



mongoDB.

MongoDB – Best NoSQL DB

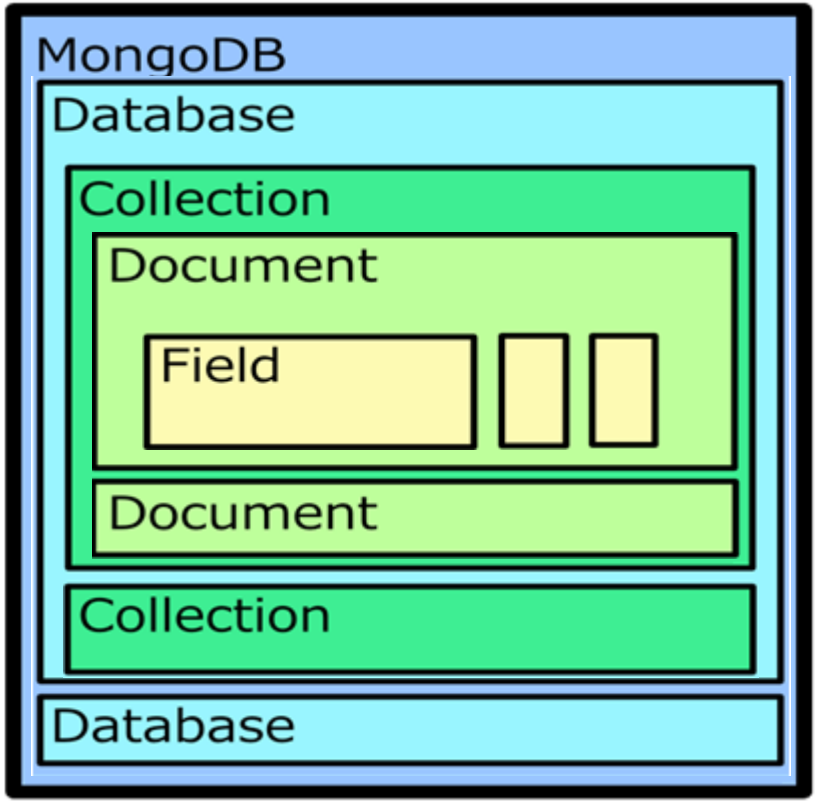
Introduction

- MongoDB is an open-source document database and leading NoSQL database.
- MongoDB is written in C++.
- It is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability.
- Works on concept of collection and document.

Introduction

- Database is a physical container for collections.
- Each database gets its own set of files on the file system.
- A single MongoDB server typically has multiple databases.

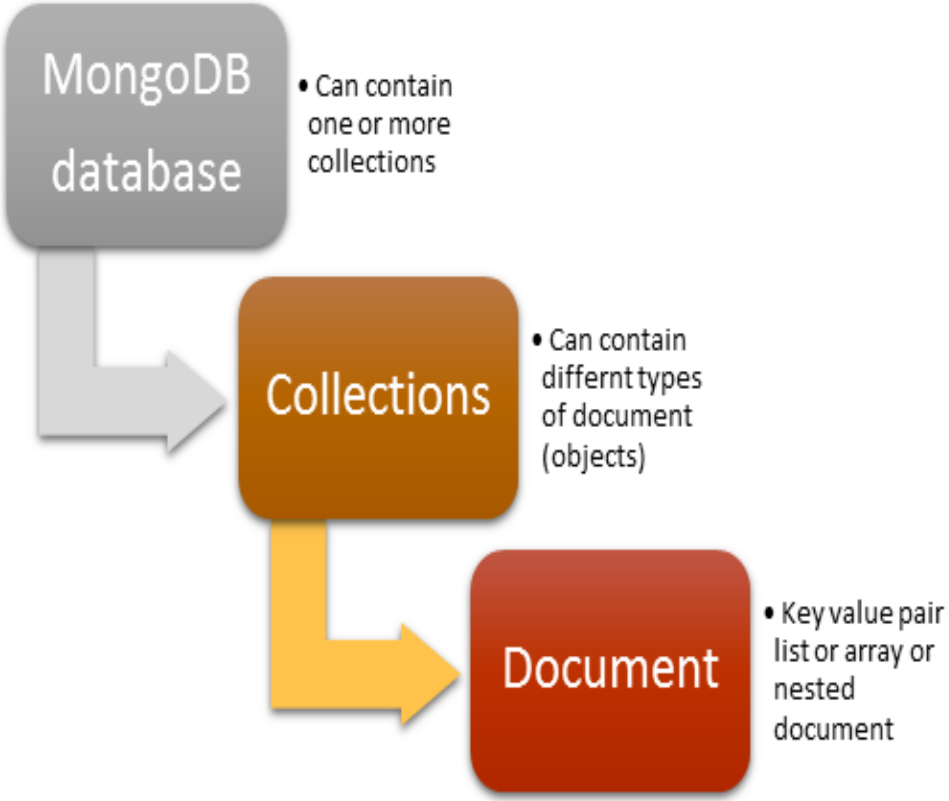
Introduction



Introduction

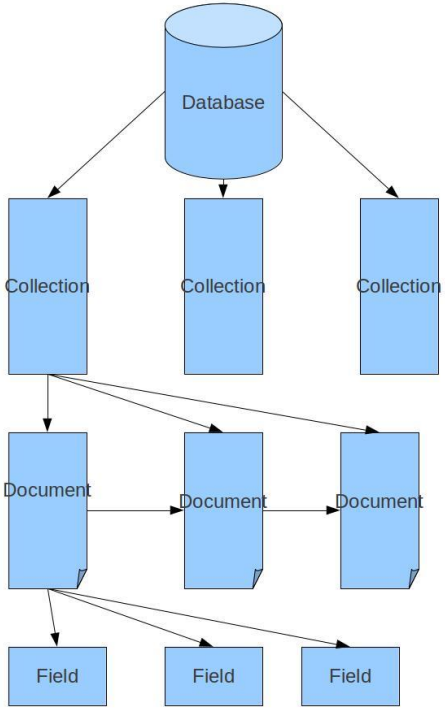
- Collection is a group of MongoDB documents.
- All documents in a collection are of similar or related purpose.

Introduction



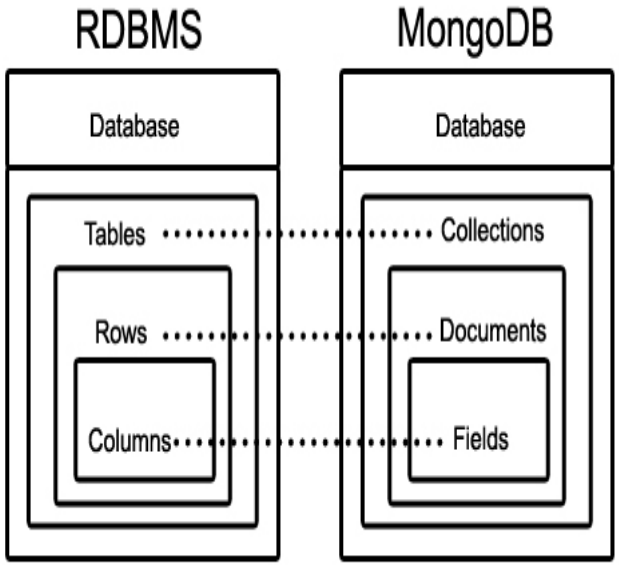
Introduction

- A document is a set of key-value pairs.
- Documents have dynamic schema.
- Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.



MongoDB Vs RDBMS

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo



Advantages of MongoDB over RDBMS

- **Schema less** − MongoDB is a document database in which one collection holds different documents.
- Number of fields, content and size of the document can differ from one document to another.
- Structure of a single object is clear.
- No complex joins.
- Deep query-ability.
- Tuning.
- Ease of scale-out
- Conversion/mapping of application objects to database objects not needed.
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Why Use MongoDB?

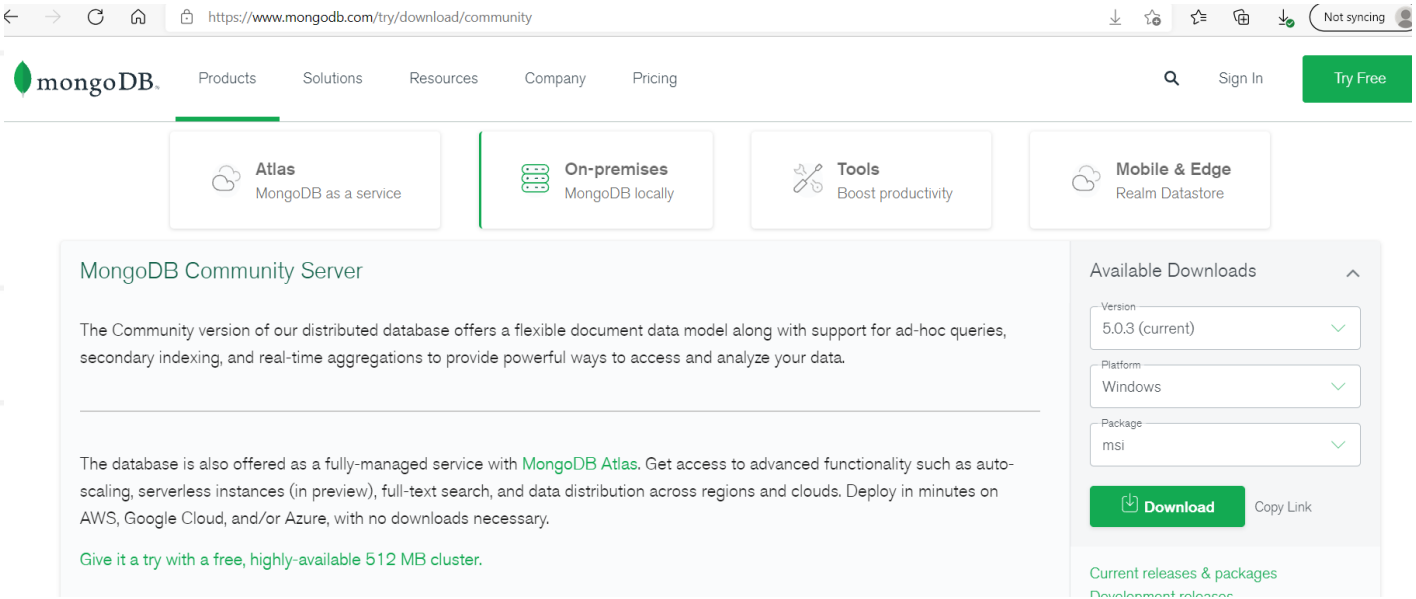
- **Document Oriented Storage** – Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

Where to Use MongoDB?

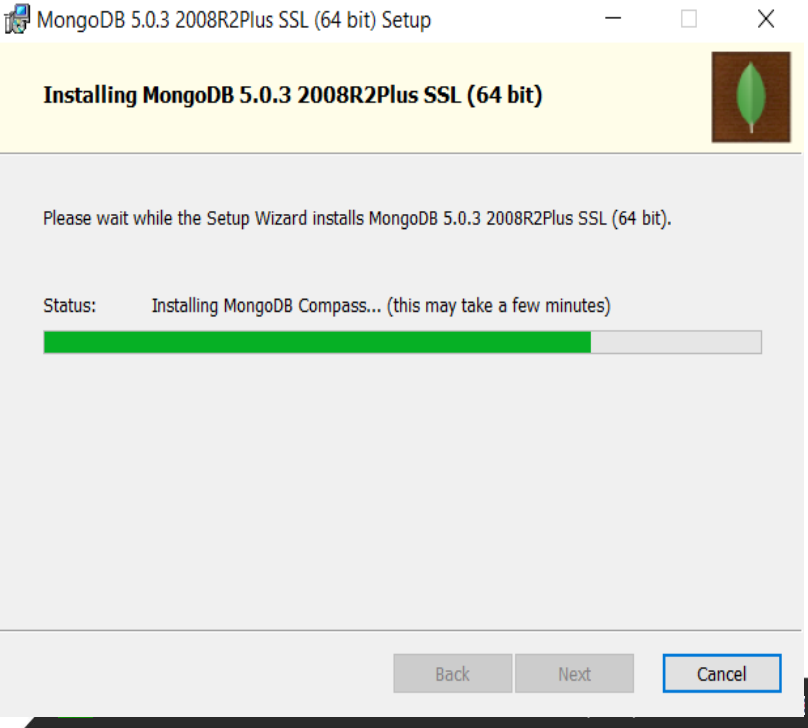
- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

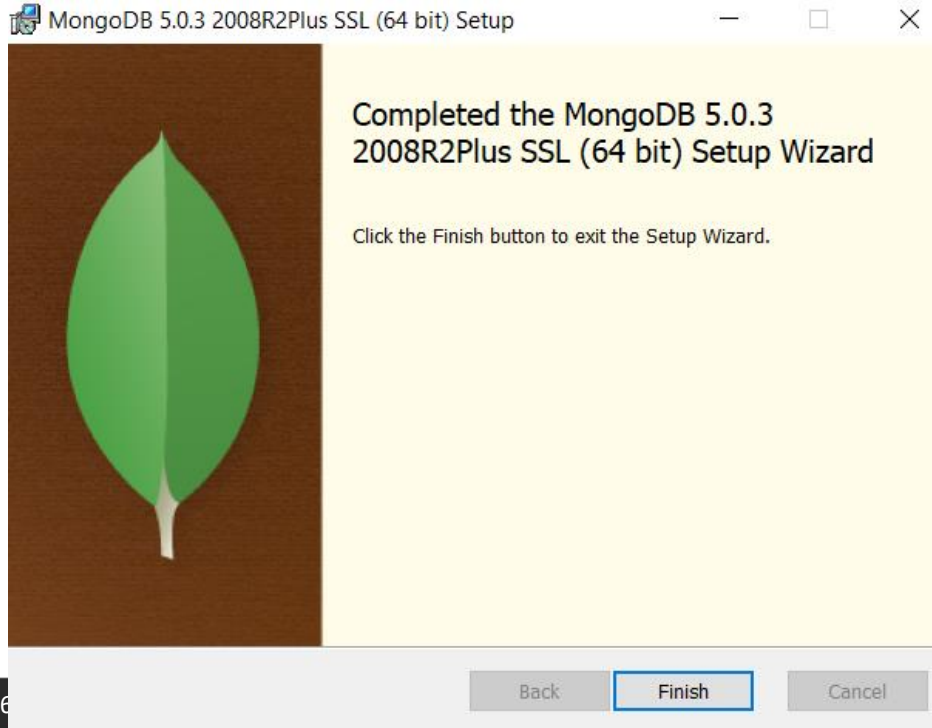
Download and Install MongoDB

- https://www.mongodb.com/try/download/community
- For Mac using Homebrew: Install MongoDB Community Edition on macOS — MongoDB Manual



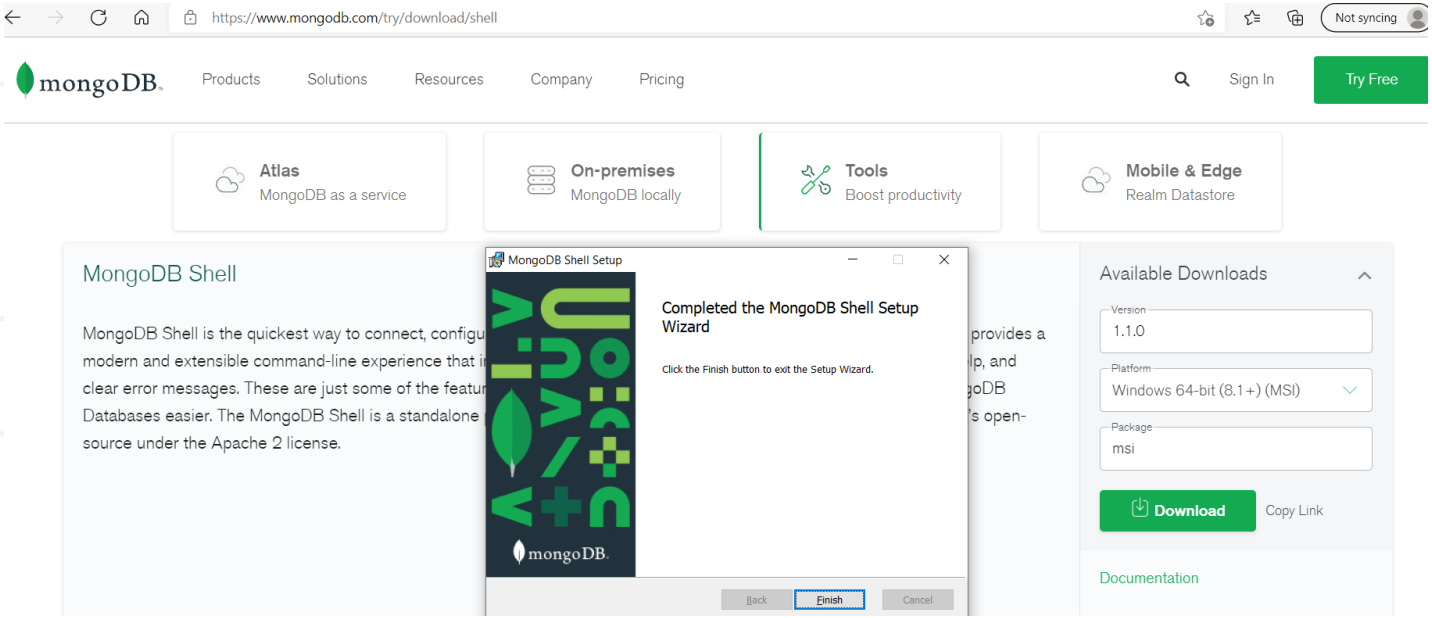
Double-click the .msi file. Don’t change any defaults. Click Install and finish.





Download and Install MongoDB Shell

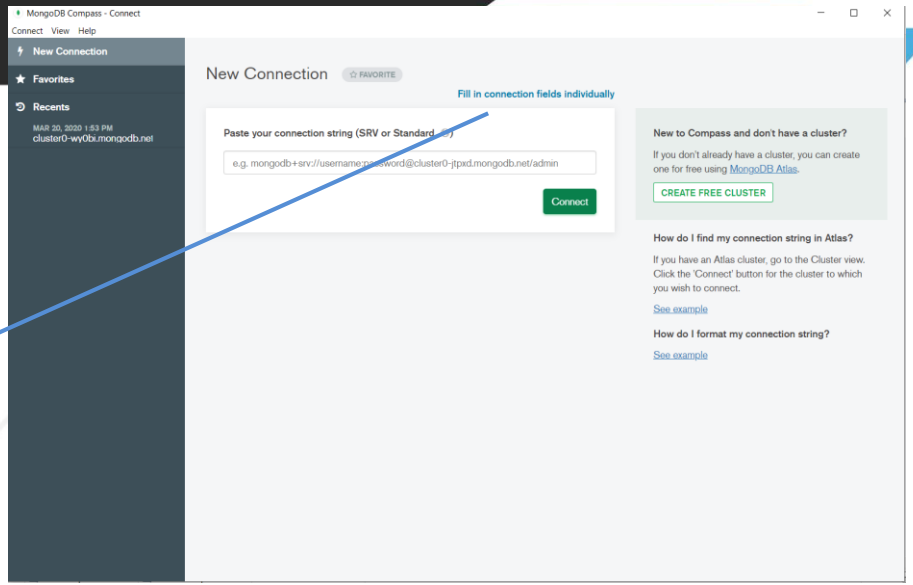
- [MongoDB Shell Download | MongoDB]



Start the Shell – Ensure you are logged in in MongoDb Compass

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000

C:\Program Files\MongoDB\Server\5.0\bin>mongosh
Current Mongosh Log ID: 6171d25e1f485a65b3d84070
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:          5.0.3
Using Mongosh:          1.1.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting:
   2021-10-21T16:11:16.197-04:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

test> show dbs;
MongoshInvalidInputError: [COMMON-10001] 'dbs;' is not a valid argument for "show".
test> show dbs;
admin         41 kB
config       111 kB
local         41 kB
my_first_db   41 kB
test>
```

Log into MongoDB Compass



MongoDB Compass - localhost:27017

Connect View Help

Local

3 DBS 1 COLLECTIONS

FAVORITE

HOST
localhost:27017

CLUSTER
Standalone

EDITION
MongoDB 5.0.3 Community

Filter your data

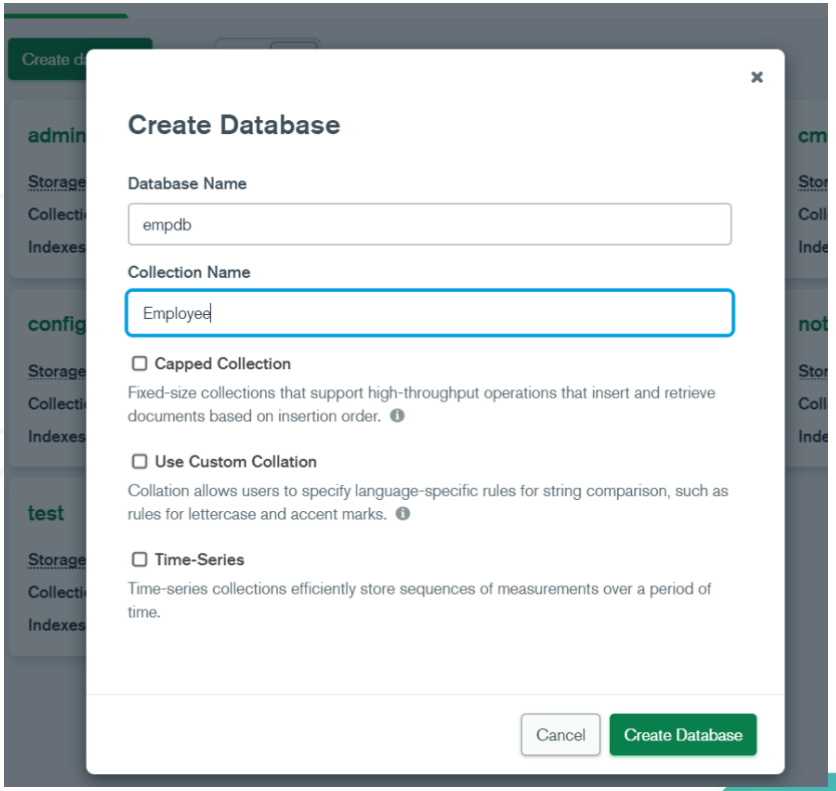
- admin
- config
- local

Databases Performance

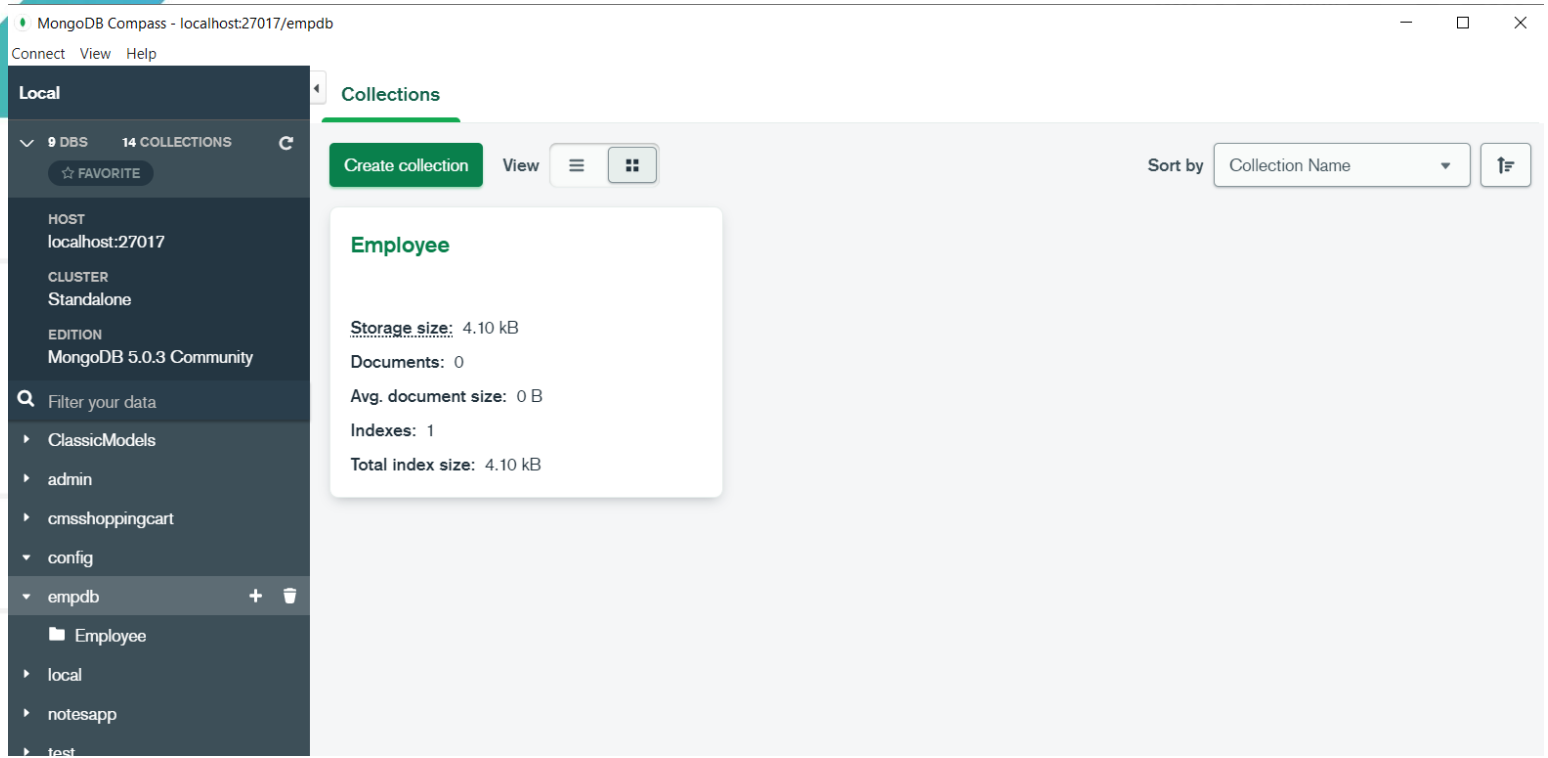
CREATE DATABASE

Database Name	Storage Size	Collections	Indexes
admin	20.0KB	0	1
config	4.0KB	0	2
local	20.0KB	1	1

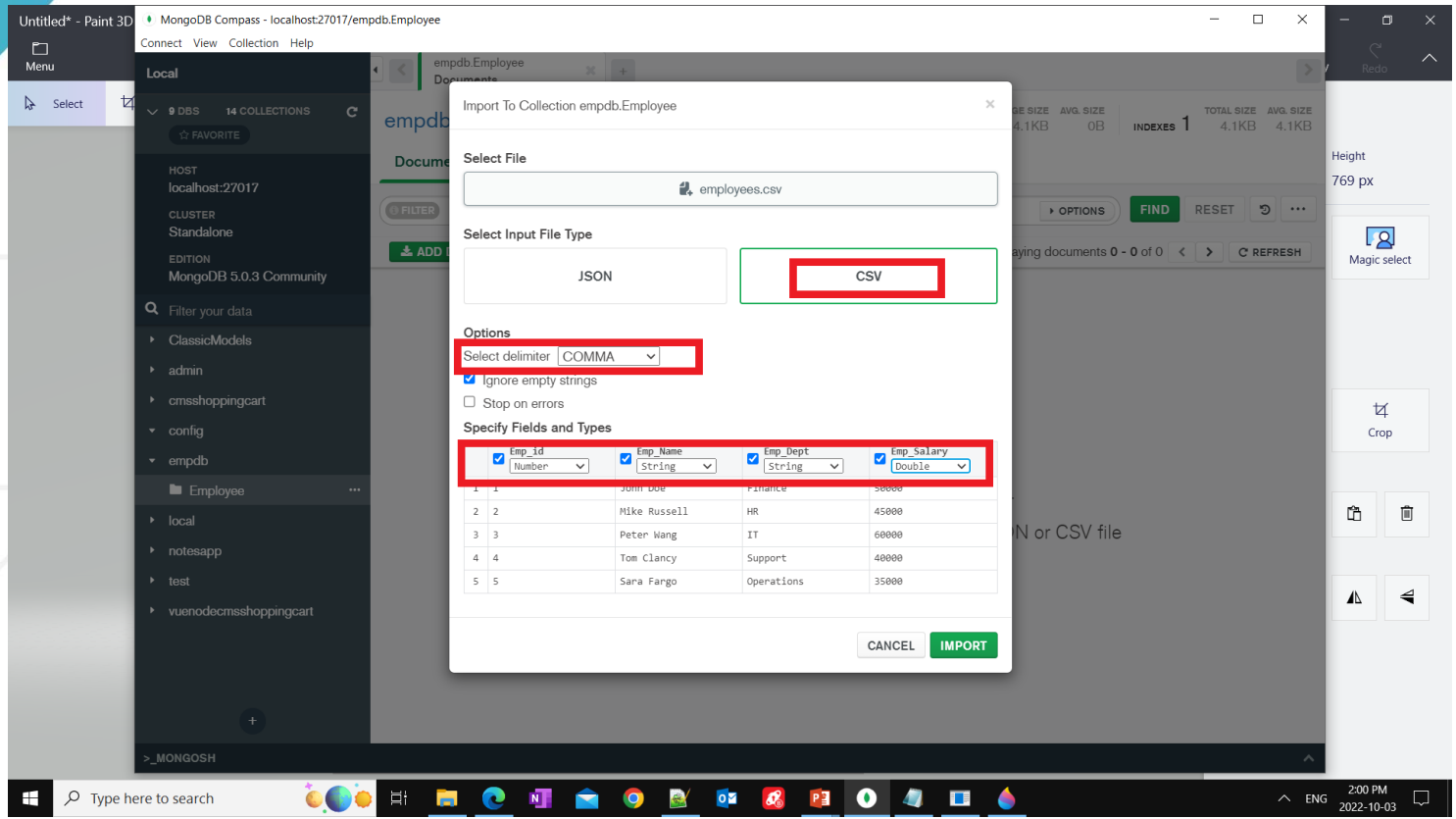
Create a new Database



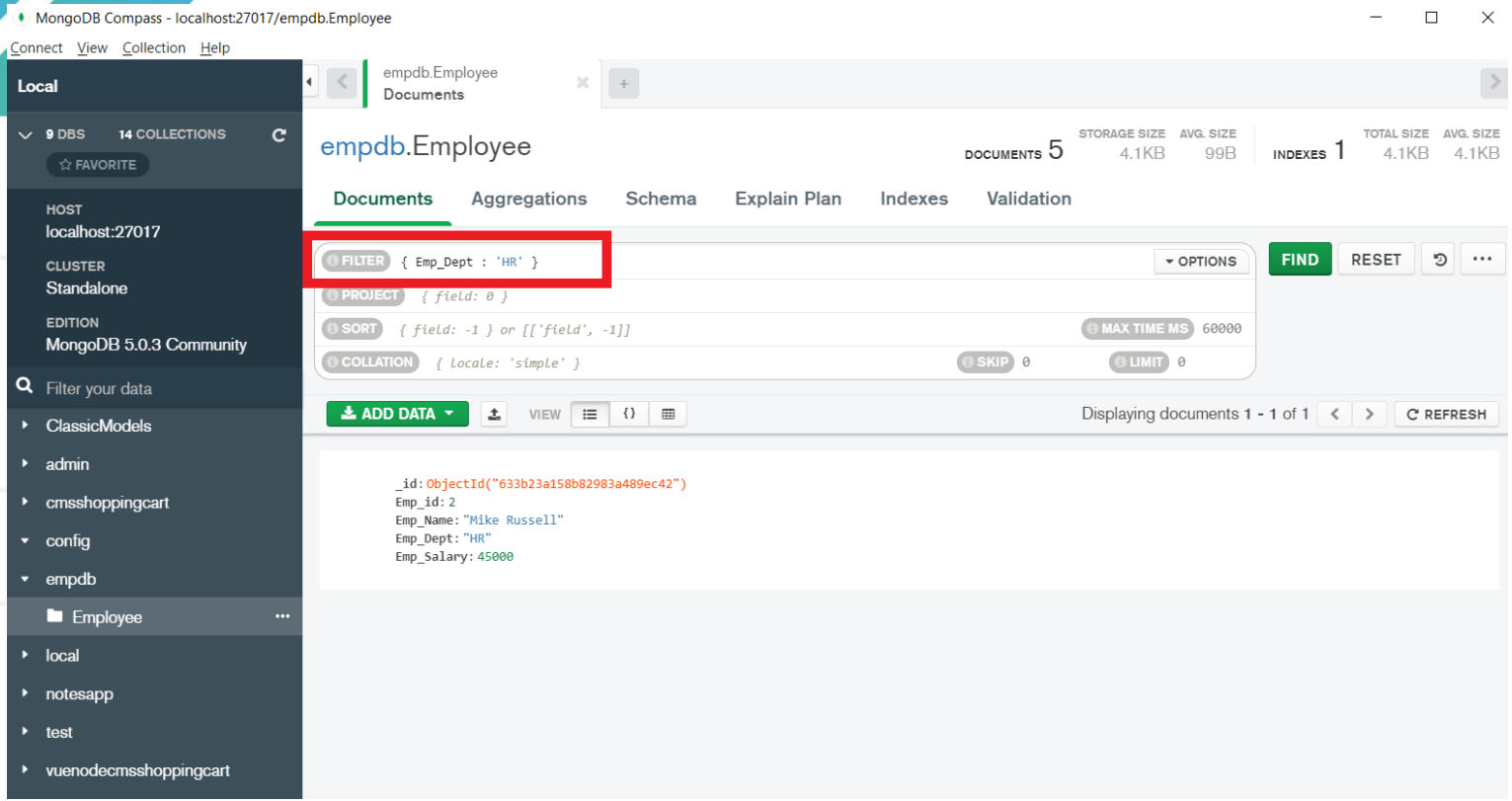
Create a new Collection



Import Data

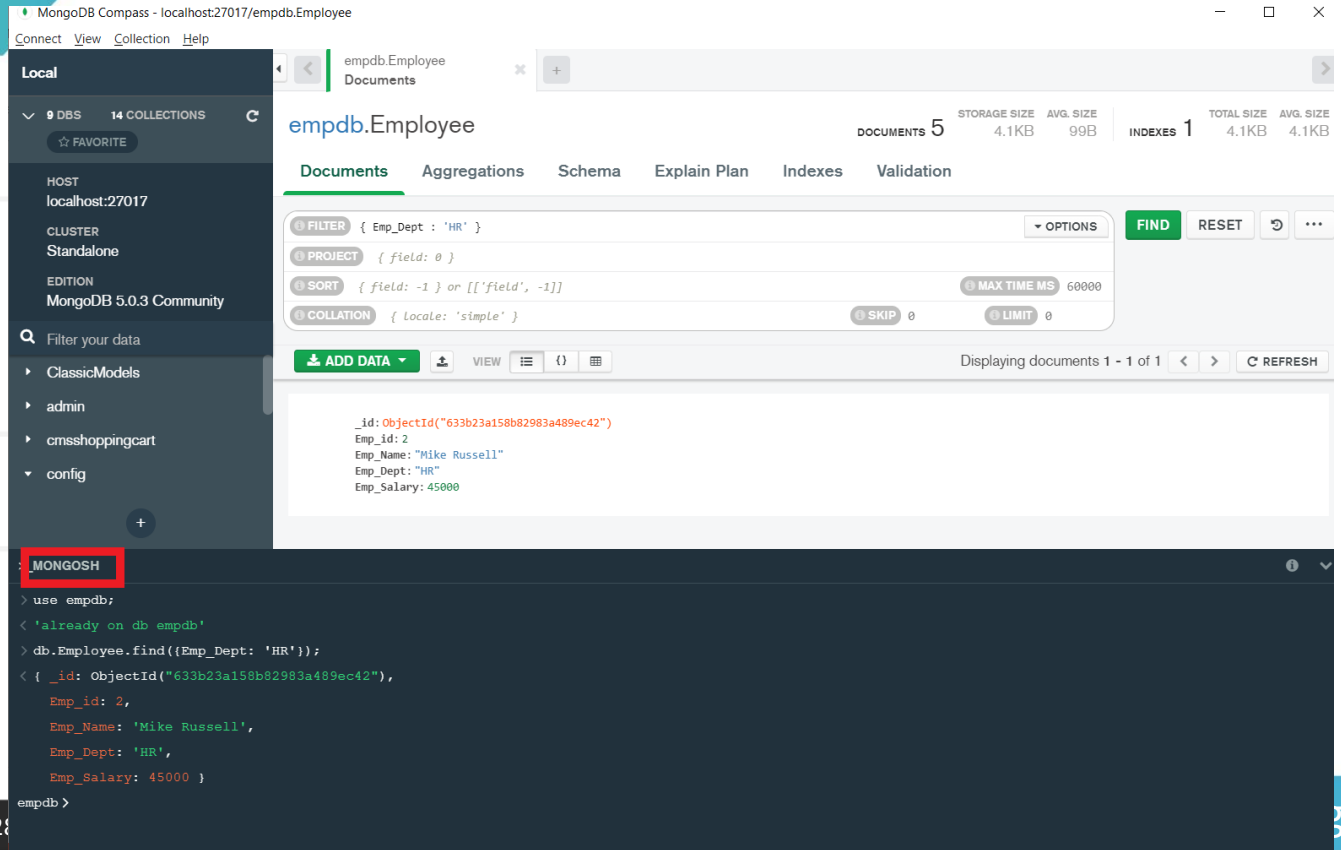

Untitled* - Paint 3D
MongoDB Compass - localhost:27017/empdb.Employee
Connect View Collection Help
Local
empdb.Employee
Import To Collection empdb.Employee
Select File
employees.csv
Select Input File Type
JSON
CSV
Options
Select delimiter COMMA
Ignore empty strings
Stop on errors
Specify Fields and Types
Emp_id Number
Emp_Name String
Emp_Dept String
Emp_Salary Double
1 1 John Doe Finance 50000
2 2 Mike Russell HR 45000
3 3 Peter Wang IT 60000
4 4 Tom Clancy Support 40000
5 5 Sara Fargo Operations 35000
CANCEL
IMPORT
FIND
RESET
REFRESH
Magic select
Crop
Type here to search
ENG
2:00 PM
2022-10-03

Commands on MongoDB Compass



Basic commands on mongo shell

[What is MongoDB? — MongoDB Manual]



Basic commands on mongo shell

- Show all available databases:

```
> show dbs;
admin   0.000GB
config  0.000GB
local   0.000GB
```

- Select a particular database to access, e.g. mydb. This will create mydb if it does not already exist:

```
> use mydb;
switched to db mydb
```

Basic commands on mongo shell

- Show all functions that can be used with the database:

```
>  db.mydb.help()
```

- To check your currently selected database, use the command db

```
> Db
mydb
```

- command is used to drop a existing database.

```
> db.dropDatabase()
{ "dropped" : "mydb", "ok" : 1 }
```

Inserting records to ‘test’ collection

- We add three records to our collection test as:

```
> db.test.insert({"key":"value1","key2":"Val2","key3":"val3"})
WriteResult({ "nInserted" : 1 })
> db.test.insert({"key":"value2","key2":"Val21","key3":"val31"})
WriteResult({ "nInserted" : 1 })
```

- To find our test collections that we have added:

```
> db.test.find()
{ "_id" : ObjectId("5c2f07431cee9c4433924d3b"), "key" : "value1", "key2" : "Val2", "key3" : "val3" }
{ "_id" : ObjectId("5c2f07891cee9c4433924d3c"), "key" : "value2", "key2" : "Val21", "key3" : "val31" }
```

Find ‘test’ collections

- To find our test collection that we have added and make them pretty readable, use the **pretty**() function.

```
> db.test.find().pretty()
{
        "_id" : ObjectId("5c2f07431cee9c4433924d3b"),
        "key" : "value1",
        "key2" : "Val2",
        "key3" : "val3"
}
{
        "_id" : ObjectId("5c2f07891cee9c4433924d3c"),
        "key" : "value2",
        "key2" : "Val21",
        "key3" : "val31"
}
```

CRUD Operations : CREATE

- The difference with save is that if the passed document contains an _id field, if a document already exists with that _id it will be updated instead of being added as new.

```
> db.people.insert({name: 'Tom', age: 28});
Or
> db.people.save({name: 'Tom', age: 28});
```

- Use insertOne to insert only one record:

```
> db.people.insertOne({name: 'Tom', age: 28});
```

- Use insertMany to insert multiple records:

```
>  db.people.insertMany([{name: 'Tom', age: 28},{name: 'John', age: 25}]);
```

CRUD Operations : READ

- Query for all the docs in the people collection that have a name field with a value of 'Tom'

```
> db.people.find({name: 'Tom'})
```

- Finding the first record in a collection:

```
> db.people.findOne({name: 'Tom'})
```

- You can also specify which fields to return by passing a field selection parameter. The following will exclude the _id field and only include the age field:

```
> db.people.find({name: 'Tom'}, {_id: 0, age: 1})
```

CRUD Operations : READ

- If you want to find sub record like address object contains country, city, etc.

```
> db.people.find({'address.country': 'US'})
```

- Also specify the field too if required

```
> db.people.find({'address.country': 'US'}, {'name': true, 'address.city': true})
```

- Remember that the result has a `.pretty()` method that pretty-prints resulting JSON:

```
> db.people.find().pretty()
```

CRUD Operations : UPDATE

- Update the **entire** object:

```
> db.people.update({name: 'Tom'}, {age: 29, name: 'Tom'});

// New in MongoDB 3.2
> db.people.updateOne({name: 'Tom'},{$set: {age: 29, name: 'Tom'}});
//Will replace only first matching document.

> db.people.updateMany({name: 'Tom'},{$set: {age: 29, name: 'Tom'}});
//Will replace all matching documents.
```

- just update a single field of a document. In this case age:

```
> db.people.update({name: 'Tom'}, {$set: {age: 29}});
```

CRUD Operations : UPDATE

- You can also update multiple documents simultaneously by adding a third parameter. This query will update all documents where the name equals Tom:

```
> db.people.update({name: 'Tom'}, {$set: {age: 29}}, {multi: true})

// New in MongoDB 3.2
> db.people.updateOne({name: 'Tom'}, {$set: {age: 29}});
//Will updateonly first matching document.

> db.people.updateMany({name: 'Tom'}, {$set: {age: 31}});
//Will update all matching documents.

>  db.people.updateMany({name: 'Tom'},{$set:{age: 30, salary:50000}});
// Document will have `salary` field as well.
```

CRUD Operations : DELETE

- Deletes all documents matching the query parameter:

```
// New in MongoDB 3.2
> db.people.deleteMany({name: 'Tom'})

// All versions
> db.people.remove({name: 'Tom'})
```

- Delete one document:

```
// New in MongoDB 3.2
> db.people.deleteOne({name: 'Tom'})

// All versions
> db.people.remove({name: 'Tom'}, true)
```

CRUD Operations : DELETE

- MongoDB's remove() method. If you execute this command without any argument or without empty argument it will remove all documents from the collection.

```
> db.people.remove({});
```

Update of embedded documents

- For the following schema:

  {name: 'Tom', age: 28, marks: [50, 60, 70]}

- Update Tom's marks to 55 where marks are 50 (Use the positional operator $):

```
> db.people.update({name: "Tom", marks: 50}, {"$set": {"marks.$": 55}})
```

- For the following schema:

  {name: 'Tom', age: 28, marks: [{subject: "English", marks: 90},
  {subject: "Maths”,marks: 100},{subject: "Computes", marks: 20}]}

- By using {name: "Tom", "marks.subject": "English"} you will get the position of the object in the marks array, where subject is English. In "marks.$.marks", $ is used to update in that position of the marks array

```
> db.people.update({name: "Tom", "marks.subject": "English"},{"$set":{"marks.$.marks": 85}})
```

Update of embedded documents

- Update values in an Array

1. The positional $ operator identifies an element in an array to update without explicitly specifying the position of the element in the array.
2. Consider a collection students with the following documents:

   { "_id" : 1, "grades" : [80, 85, 90] }

   { "_id" : 2, "grades" : [88, 90, 92] }

   { "_id" : 3, "grades" : [85, 100, 90] }
3. To update 80 to 82 in the grades array in the first document

```
> db.students.update({ _id: 1, grades: 80 },{ $set: { "grades.$" : 82 } });
```

Getting database information

- List all collections in database

```
>  show collections
          or
> show tables
          or
>  db.getCollectionNames()
```

- List all databases

```
> show dbs
          or
> db.adminCommand('listDatabases')
          or
>db.getMongo().getDBNames()
```

Querying for Data

- Retrive all documents in a collection

```
> db.people.find({});
```

- Retrieve documents in a collection using a condition (similar to WHERE in MYSQL)

```
db.collection.find({key: value});
example :
    > db.people.find({"name":"Tom"});
```

- Retrieve documents in a collection using Boolean conditions (Query Operators)

```
//AND operator
db.collection.find({ $and:[ { key:value },{ key: value } ] });
example :
     > db.people.find({$and:[{"name":"Tom"},{"age": 28}]});
```

Querying for Data

```
//OR operator
db.collection.find({ $or:[ { key:value },{ key: value } ] });
 example :
       > db.people.find({$or:[{"name":"Tom"},{"age": 28}]});

//NOT operator
db.inventory.find( { key: { $ne: value } } );
 example:
        > db.people.find({ "name": { $ne : "billl" } });
```

//similarly we have following operators

- $eq: Matches values that are equal to a specified value.
- $gt: Matches values that are greater than a specified value.
- $gte: Matches values that are greater than or equal to a specified value.
- $lt: Matches values that are less than a specified value.$lte:
- $ne: Matches values that are less than or equal to a specified value.
- $in: Matches all values that are not equal to a specified value.
- $nin: Matches none of the values specified in an array.

For more Information or set up an appointment kindly contact us today.