

MongoDB hierarchy : Server → Database → Collections → Documents

CRUD (Create, Read, Update, Delete) operations allow you to work with the data stored in MongoDB. The CRUD operation documentation is categorized in two sections: **Read Operations** **find** and **return** documents stored within your MongoDB database. **Write Operations** **insert**, **modify**, or **delete** documents in your MongoDB database.

API stands for **Application Programming Interface**. It is a set of definitions and protocols that enable two software components to communicate with each other.

Basic Commands

1. **Use a Database:** Switch to a specific database or create a new one if it doesn't exist.
 - `use mydatabase`
2. **Create a Collection:**
 - `db.createCollection("mycollection_name")`
 - `db.createCollection("mycollection_name", {
 capped: false, // Optional: Set to true for a capped collection
 size: 100000, // Optional: Specifies the maximum size in bytes for a capped collection
 max: 50 // Optional: Specifies the maximum number of documents for a capped collection
})`
3. **Insert Document:** Insert One or Many document into a collection.
 - `db.mycollection.insertOne({ key1: "value1", key2: "value2",...})`
 - `db.mycollection.insertMany([{ key1: "value1", key2: "value2",...}, {key1: "value3", key2: "value4",...}, ... additional documents])`
 - `db.mycollection.insert({ key: "value", another_key: "another_value" })`
4. **Query Document:** Retrieve documents from a collection.
 - `db.mycollection.find()`
 - `db.mycollection.find({Document based filter},{Field or feater based filter})`
 - `var result = db.mycollection.findOne({ key1: "value1" })`
 - `var cursor = db.mycollection.find({ key1: "value1", key2: "value2"})`
5. **Update a Document:** Update a document in a collection.
 - `db.mycollection.update({ key: "value" }, { $set: { another_key: "new_value" } })`
6. **Delete Documents:** Remove documents from a collection based on conditions.
 - `db.mycollection.remove({ key: "value" })`

Advanced Queries

1. **Projection (Selecting Fields):** Specify fields to include/exclude in the query result.
 - `db.mycollection.find({ key: "value" }, { key: 1, _id: 0 })`
2. **Sorting:** Sort documents based on a field.

- `db.mycollection.find().sort({ key: 1 })`
3. **Limit and Skip:** Limit the number of documents returned, and skip a specified number of documents.
 - `db.mycollection.find().limit(5).skip(2)`
 4. **Aggregation:** Use aggregation pipelines for complex data transformations.
 - `db.mycollection.aggregate([{ $group: { _id: "$key", count: { $sum: 1 } } }])`

Miscellaneous Commands

1. **show dbs (Show Databases)** - Display a list of available databases.
2. **show collections** - Display a list of collections in the current database.
3. **db.dropDatabase()** - Delete the current database.
4. **db.mycollection.drop()** - Delete a collection.

Query Operators

Comparison Operators

1. **\$eq:** Values are equal
2. **\$ne:** Values are not equal
3. **\$gt:** Value is greater than another value
4. **\$gte:** Value is greater than or equal to another value
5. **\$lt:** Value is less than another value
6. **\$lte:** Value is less than or equal to another value
7. **\$in:** Value is matched within an array
8. **\$nin:** Value is not matched within an array

Logical Operators

1. **\$and:** Returns documents where both queries match
2. **\$or:** Returns documents where either query matches
3. **\$nor:** Returns documents where both queries fail to match
4. **\$not:** Returns documents where the query does not match

Update Operators

Fields - The following operators can be used to update fields:

1. **\$set:** Sets the value of a field
2. **\$unset:** Removes the field from the document
3. **\$rename:** Renames the field
4. **\$inc:** Increments the field value
5. **\$currentDate:** Sets the field value to the current date

Array - The following operators assist with updating arrays.

1. **\$addToSet:** Adds distinct elements to an array
2. **\$pop:** Removes the first or last element of an array
3. **\$pull:** Removes all elements from an array that match the query
4. **\$push:** Adds an element to an array

Aggregations

1. **\$group:** Groups documents by a specified expression and can perform aggregation on those grouped documents.
2. **\$limit:** Limits the number of documents passed to the next stage in the pipeline.
3. **\$project:** Reshapes documents by including, excluding, or renaming fields, as well as creating computed fields.
4. **\$sort:** Sorts documents based on specified fields.
5. **\$match:** Filters documents to pass only those that match a specified condition to the next stage.
6. **\$addField:** Adds new fields to documents. Similar to \$project, but it adds fields to all documents in the pipeline.
7. **\$count:** Returns the number of documents that passed through the pipeline.
8. **\$lookup:** Performs a left outer join to another collection in the same database to filter documents from the input collection.
9. **\$out:** Writes the resulting documents of the aggregation pipeline to a specified collection.

Integrating python with SQL

```
import pymongo #pip install pymongo
```

```
# Connect to MongoDB
```

```
client (object) = pymongo.MongoClient ("your_mongodb_connection_string")
```

Inseting collections to MongoDB through Python

```
# Select database and collection
```

```
data_base (object) = client["data_base_name_in_this_client"]
```

```
collection (object) = data_base ["collection_name_in_this_data_base "]
```

```
# Insert single document
```

```
x={"item":"pizza","price":200}
```

```
collection.insert_one(x)
```

```
# Insert multiple documents
```

```
y=[{"item":"Ice cream","price":250}, {"item":"Burger","price":150}, {"item":"Coke","price":100}, [{"item":"Ice cream","price":250}]
```

```
collection.insert_many(y)
```

```
# Update documents
```

```
collection.update_one({"item":"Coke"}, {"$set":{"Location":"Chennai","price":250}})
```

```
collection.update_many({"item":"Ice cream"}, {"$set":{"Location":"Hyderabad","price":199}})
```

Inseting json file to MongoDB

```
boopathi = client["boopathi"]
```

```
collect = boopathi["sample"]
```

```
import json
```

```
# Open JSON file
```

```
file (object) = open("fail_path_from_where_you_need_to_open")
```

```
data (object) = json.load(file)
```

```
# Insert JSON data into MongoDB
```

```
collect.insert_many(data)
```

Data conversion using request

```
collection = boopathi.satelite
```

```
import requests
```

```
import time
```

```

# Get live data
iss=requests.get("API_Key_from_where_you_are_requesting") #Receiving live data using API Key
x=iss.text #storing the received data
y=json.loads(x) #converting the data (string) into json formate
collection.insert_one(y)

# Fetch data for 12 times with a delay of 5 seconds
for i in range(12):
    iss=requests.get("http://api.open-notify.org/iss-now.json")
    x=iss.text
    y=json.loads(x)
    collection.insert_one(y)
    time.sleep(5)

```

Data conversion using pandas

```

collect = boopathi["house"]
import pandas as pd

# Read CSV file
data_frame = pd.read_csv("/content/sample_data/california_housing_test.csv")

# Convert DataFrame to dictionary
cad = data_frame.to_dict("records")

# Insert dictionary into MongoDB collection
collect.insert_many(cad)

collect=boopathi["house_2"]
# Insert each record individually
for i in cad:
    collect.insert_one(i)

```

Note:

```

#insert_one -{ }
#insert_many -[]

```