# MongoDB Lab - Hands-on

**Prerequisites:** You need to have Docker preinstalled. You can install it from here.

## Run MongoDB as a Docker Container

To set up a MongoDB Docker container, we'll use a Docker run command to deploy a MongoDB instance and also give it a container name:

```
docker run -d -p 27017:27017 --name MONGO_CONTAINER mongo:latest
```

## Start the MongoDB shell

The connect to our container named MONGO_CONTAINER using the bash terminal run:

```
docker exec -it MONGO_CONTAINER  bash
```

To launch the MongoDB shell and connect to the MongoDB instance running on localhost with the default port 27017:

```
mongosh
```

## Interact with the MongoDB shell

To list the databases, execute the following command:

```
show dbs
```

Create a new database and switch to it:

```
use lab
```

Create a new collection named Items:

```
db.createCollection('items')
```

Add a document in the colection:

```
db.items.insertOne({name: "item1", qty: 10})
```

Add multiple documents in the colection:

```
db.items.insertMany([{name: "item2", qty: 12}, {name: "item3", qty: 3}])
```

Show all documents in the collection:

```
db.items.find()
```

Find a single document by name:

```
db.items.findOne({ name: "item1" })
```

Find multiple documents:

```
db.items.find({ qty: { $gt: 4 } })
```

Update single document:

```
db.items.updateOne({ name: "item1" }, { $set: { qty: 2 } })
```

Update all matching documents:

```
db.items.updateMany({ qty: { $lt: 4 } }, { $inc: { qty: 5 } })
```

Remove all matching documents:

```
db.items.remove({ qty: { $lte: 10 } })
```

## Load the sample Books dataset

First, exit the mongo shell:

```
exit
```

Then, in the bash shell run the following to update the container libraries and install wget. This is needed in order to later fetch the sample json file from the web.

```
apt-get update && apt-get install wget
```

Fetch the sample json and import it into a collection "books" of database "lab" using **mongoimport**:

```
wget -qO- https://raw.githubusercontent.com/stavmars/MongoDB_Lab/main/books.json | mongoimport -d lab
```

## Explore the Books Dataset

Launch the MongoDB shell again:

```
  mongosh
```

Switch to the new database and view the available collections

```
use lab

show collections
```

Show a single document and examine its structure

```
db.books.find().limit(1)
```

To find the books with a number of pages that is greater or equal than 400 but less than 500, sort them by their publication date descending and print their titles, page counts and date they were published

```
db.books.find({pageCount: {$lt: 500, $gte: 400}}, {
  _id: 0,
  title: 1,
  pageCount: 1,
  publishedDate: 1
}).sort({publishedDate: -1})
```

To find all Python books and print their titles and categories we run the following query.

```
db.books.find({categories: "Python"}, {title: 1, categories: 1})
```

Compare the results of the query above with the following one:

```
db.books.find({categories: ["Python"]}, {title: 1, categories: 1})
```

To find all books that are either about Python or PHP run:

```
db.books.find({categories: {$in: ["Python", "PHP"]}}, {
  title: 1,
  categories: 1,
})
```

Find the top 5 Python books with the most pages and print their titles, categories and page counts.

```
db.books.find({categories: "Python"}, {
    title: 1, categories: 1, pageCount: 1
}).sort({pageCount: -1}).limit(5)
```

Find the books that have as author either "Marc Harter" or "Alex Holmes" and print their titles, authors and categories

```
db.books.find({authors: {$in: ["Marc Harter", "Alex Holmes"]}}, {
    title: 1, categories: 1, authors: 1
})
```

## Indexing Documents

To see the indexes available for a collection run:

```
db.books.getIndexes()
```

To see how indexing helps query performance run the following query and examine its query plans and execution statistics using **explain**:

```
db.books.find({categories: "Python"}).explain("executionStats")
```

Now add an index on the field categories:

```
db.books.createIndex({categories: 1})
```

Now run the same query again and compare the execution stats and query plans followed.

To drop the index we just created we can run:

```
db.books.dropIndex({categories: 1})
```

## Aggregations

This example computes the average number of pages grouped by the "status" field:

```
db.books.aggregate( [
   {
     $group: {
        _id: "$status",
        avgPageCount: { $avg: "$pageCount" }
     }
   }
] )
```

Expand the example above in order to also compute the minimum and maximum number of pages

```
db.books.aggregate([{
    $group: {
        _id: "$status",
        avgPageCount: {$avg: "$pageCount"},
        minPageCount: {$min: "$pageCount"},
        maxPageCount: {$max: "$pageCount"}
    }
}])
```

Now we compute the number of books in the database per year. For this we can use the **$year** operator and add to each document a year field before the **$group** stage:

```
db.books.aggregate([
  {$addFields: {year: {$year: "$publishedDate"}}},
  {
    $group: {
      _id: "$year",
      count: {$sum: 1}
    }
  },
  {$sort: {count: -1}}
])
```

To exclude books with no publication data available, we add a **$match** aggregation stage:

```
db.books.aggregate([
    {$match: {publishedDate: {$ne: null}}},
    {$addFields: {year: {$year: "$publishedDate"}}}, {
        $group: {
            _id: "$year",
            count: {$sum: 1}
        }
    }, {$sort: {count: -1}}])
```

Now expand the query above to find the number of books per year and status.
**Hint:** Use as the _id field in the **$group** stage an object with keys both the year and status: *_id: {year:"$year", status:"$status"}*

```
db.books.aggregate([{$addFields: {year: {$year: "$publishedDate"}}}, {
    $group: {
        _id: {year: "$year", status: "$status"}, count: {$sum: 1}
    }
}, {$sort: {count: -1}}])
```

In the following example, we want to find the average number of pages per book category for all books with status="PUBLISH", and sort the results by the average page count. Remember that categories is an array field with possibly more than one values for every book.

```
db.books.aggregate([
   { $match : {status: "PUBLISH" } },
   { $unwind: "$categories" },
   { $group: { _id: "$categories", avgPageCount: { $avg: "$pageCount" } } },
   { $sort: {avgPageCount: -1} }
])
```

Similarly, find the average number of pages, as well as the number of books per author. Then, find the 5 authors with the most books. Remember that as with the categories field, the authors field is also an array, so use the **$unwind** aggregation stage.

```
db.books.aggregate([{$unwind: "$authors"}, {$match: {authors: {$ne: ""}}}, {
    $group: {
        _id: "$authors", count: {$sum: 1}, avgPageCount: {$avg: "$pageCount"}
    }
}, {$sort: {count: -1}}, {$limit: 5}])
```

Next, we need to find the average number of authors for all books. We use the **$size** operator which counts and returns the total number of items in an array.

```
db.books.aggregate([{$project: {authorsCount: {$size: '$authors'}}}, {
    $group: {
        _id: null,
        avgAuthorsCount: {$avg: '$authorsCount'}
    }
}])
```

In the same fashion, find the average number of categories for every book.

```
db.books.aggregate([{$project: {categoriesCount: {$size: '$categories'}}}, {
    $group: {
        _id: null, avgCategoriesCount: {$avg: '$categoriesCount'}
    }
}])
```

If we wish to find for every author the years that they published a book, we can run the following:

```
db.books.aggregate([
    {$addFields: {year: {$year: "$publishedDate"}}},
    { $unwind: "$authors" },
    { $group: { _id: "$authors", years: { $addToSet: "$year" } } }
])
```

Now, find for every category of book, the years that there were publications belonging to that category:

```
db.books.aggregate([{$addFields: {year: {$year: "$publishedDate"}}}, {$unwind: "$categories"}, {
    $group: {
        _id: "$categories", years: {$addToSet: "$year"}
    }
}])
```

Now modify your previous query to print 'N/A' instead of null in the lists of years.(Hint: Use the ifNull operator when you add the year field). Also, sort the array of years in the final results (Hint: Use the sortArray operator).

```
db.books.aggregate([{$addFields: {year: {$ifNull: [{$year: "$publishedDate"}, "N/A"]}}}, {$unwind: "$c
    $group: {
        _id: "$categories", years: {$addToSet: "$year"}
    }
}, {$project: {years: {$sortArray: {input: "$years", sortBy: 1}}}}])
```

Find the authors who have written books in the most categories. Provide a list of these authors along with the number of categories they've written books in.

```
db.books.aggregate([
  {
    $unwind: "$authors"
  },
  {
    $unwind: "$categories"
  },
  {
    $group: {
      _id: {
        author: "$authors",
        category: "$categories"
      }
    }
  },
  {
    $group: {
      _id: "$_id.author",
      categoriesCount: { $sum: 1 }
    }
  },
  {
    $sort: {
      categoriesCount: -1
    }
  },
  {
    $limit: 5
  },
  {
    $project: {
      author: "$_id",
      categoriesCount: 1,
      _id: 0
    }
  }
])
```