

COMP5347/COMP4347: Web Application Development

Week 7 Tutorial: MongoDB Basic Queries

Learning Objectives

Practice basic MongoDB features with given JSON document:

- import data into mongodb from JSON file
- basic read query
- basic aggregation query
- indexing
- get familiar with MongoDB query reference document

Part 1 Option 1: Start MongoDB server

Option 1: Lab machine

MongoDB is installed on all Lab PCs under the directory `C:/Program Files/MongoDB/`. It requires a data directory to store all data. It is recommended that you create this data directory on U: drive. This way you can access your data from any lab machine. `mongod` is the primary demon process of the MongoDB database engine. It can be started by a regular user with the command `mongod.exe`.

Task: Below are the basic steps for starting MongoDB database engine the first time:

- Create a directory `comp5347/mongodb` on your U drive.
- Open a command window or power shell window and change to directory `C:/Program Files/MongoDB/Server/3.4/bin`
- run the following command

```
mongod.exe --dbpath U:/comp5347/mongodb --smallfiles
```

If the database engine starts correctly, you will see a few lines of initialization message and the last line should be "waiting for connections on port 27017". Leave this window open unless you are ready to shut down the server.

Option 2: Your own machine

Please go to Canvas – Modules – Week 7 and refer to the file named `Week 7 Tutorial - MongoDB Installation.pdf` for guide on how to install MongoDB on your own machine.

Part 2: Start MongoDB Compass as client shell GUI

Although you can run MongoDB's interactive JavaScript shell in the command line interface, there are also various client software compatible with MongoDB that could make things much easier. Robo 3T (previously known as Robomongo) is installed on all lab computers. Alternatively, if you're using your personal devices, we recommend MongoDB's official client tool MongoDB Compass. When you installed and opened it for the first time, you will be prompted to start a new connection.

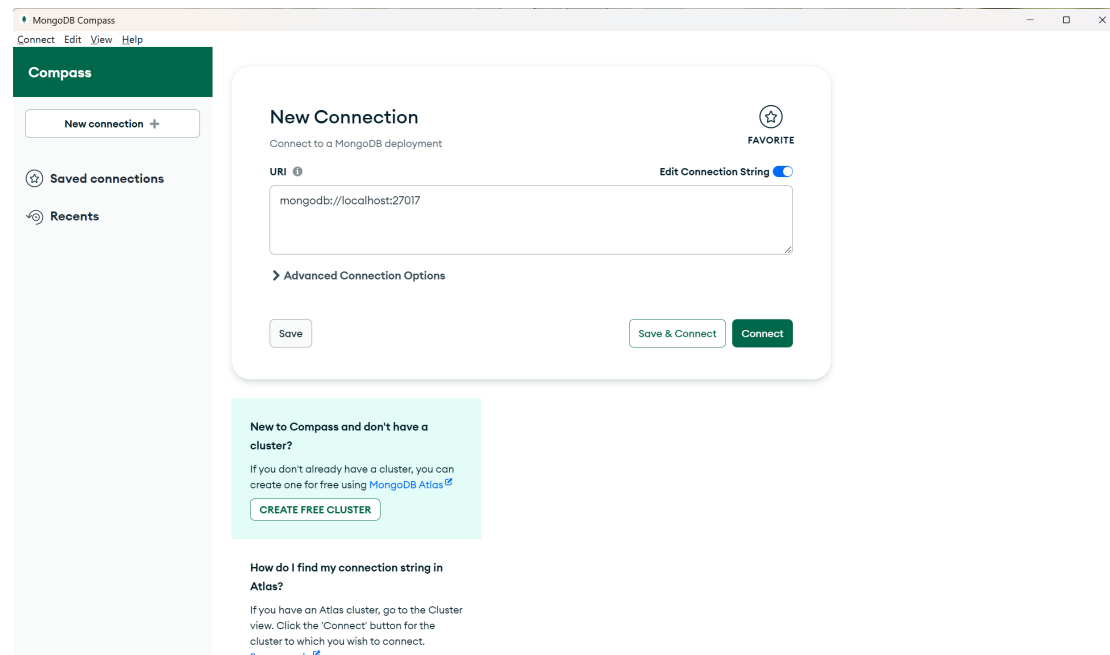


Figure 1: MongoDB Compass New Connection screen

Task: To create a new local connection, keep the default URI and click “Connect” or “Save and Connect”. Later in this course if you wish to connect to different servers, enter the URI with the format `mongodb://[Username]:[Password]@[Server hostname]:[Port number]`. Alternatively, you can click on “Advanced Connection Options” and enter them manually.

Once you're connected, the “main screen” of Compass will show up. On the left you will see a list of default databases that are contained in any MongoDB server. They store configurations and other settings of the database server application, which you are **NOT** recommended to alter in anyway.

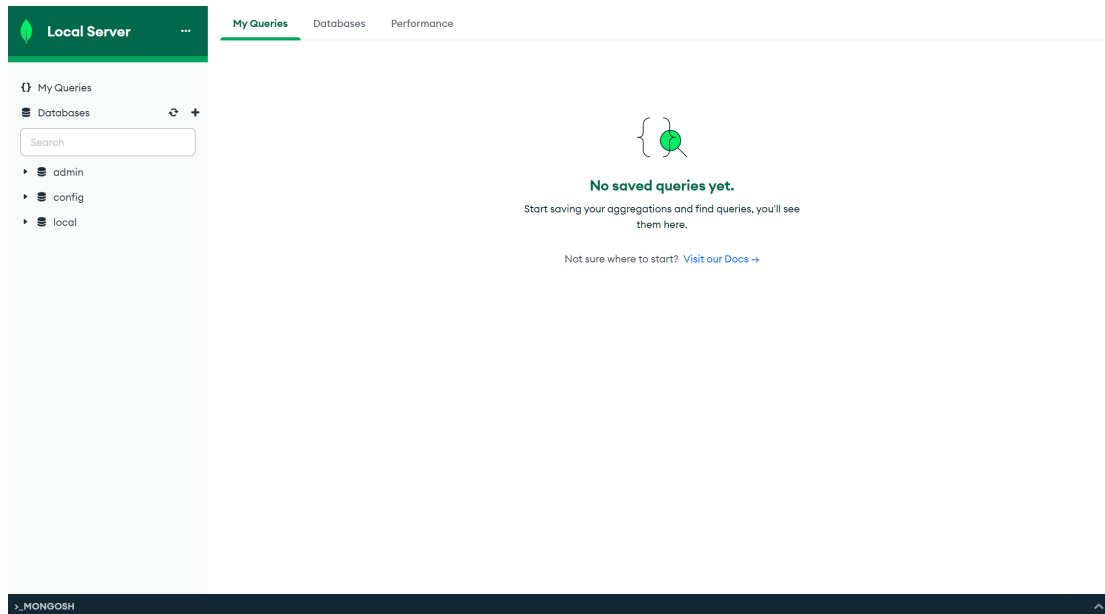


Figure 2: Create new connection in MongoDB Compass

From there, you can create a new database. Click “+” next to “Databases”, and a “Create Database” window will pop up. In this tutorial we will call the new database `wikipedia`, and the collection `revisions` (This will be the first collection in your new database). We will use this collection to store the wiki revision and user data.

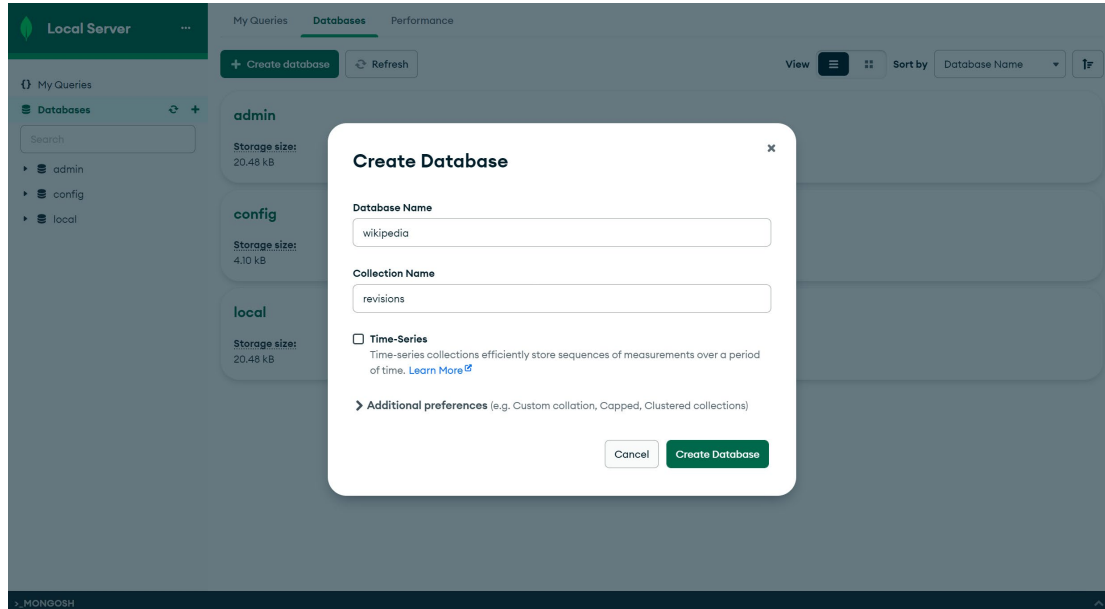


Figure 3: Create new database in MongoDB Compass

After creating the database, you will be directed to the collection's screen. You can also access it later by navigating on the left panel, select the collection under your database.

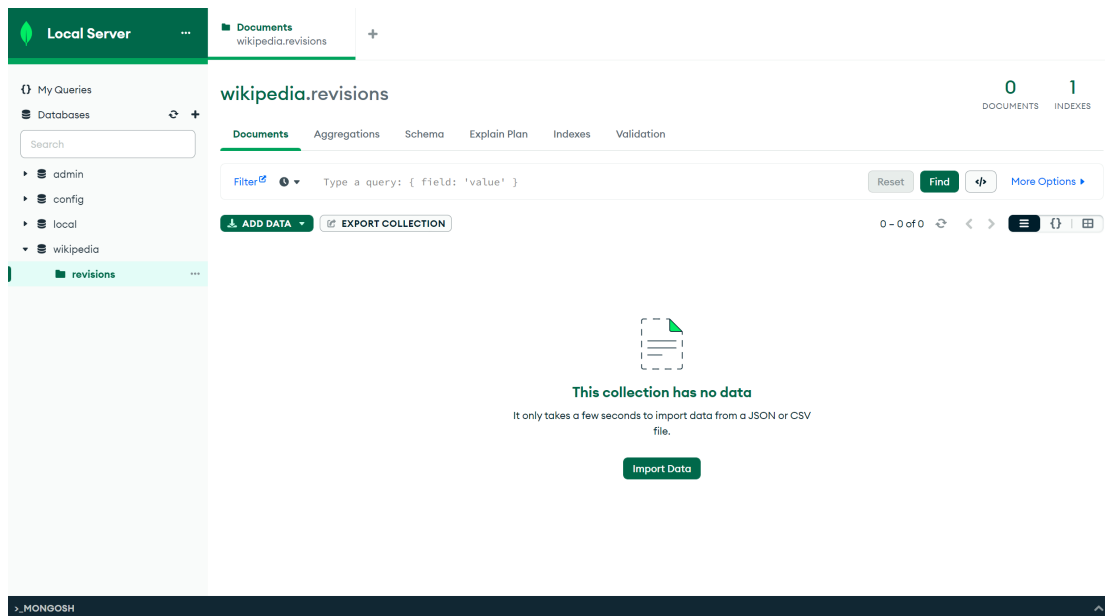


Figure 4: Collection screen in MongoDB Compass

Part 3: Import JSON file into MongoDB collection

Option 1: MongoDB Compass (GUI)

On the collection screen, click on “ADD DATA” button and select “Import JSON or CSV file”, and choose the two JSON files you downloaded from Canvas, one at a time.

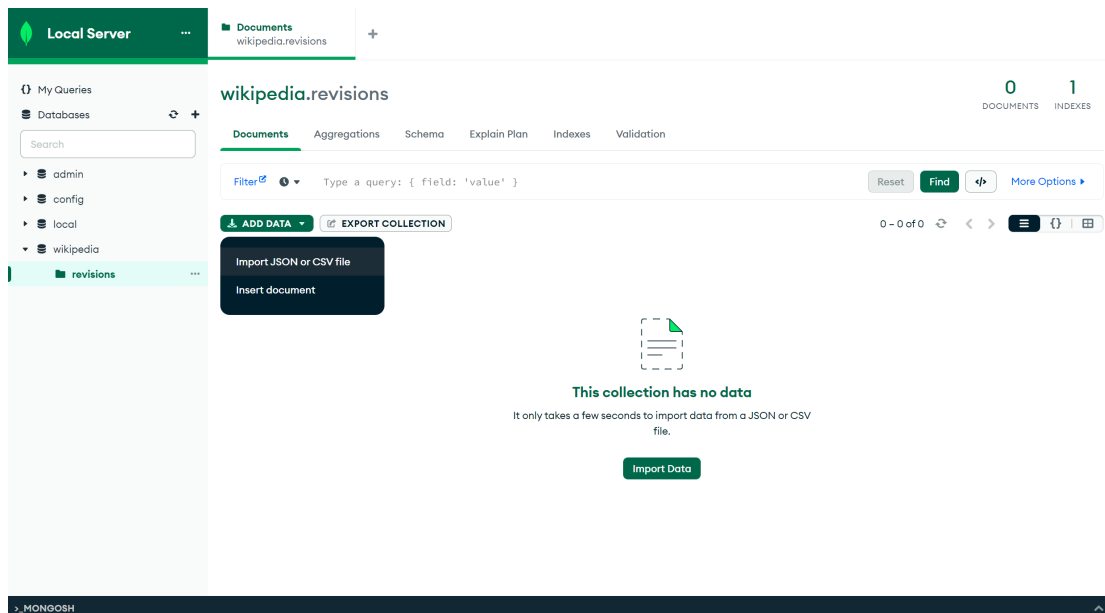


Figure 5: Import JSON/CSV file in MongoDB Compass

When it finishes import, Compass will show you the status “Import completed” with a total number of documents you imported. In MongoDB, each object in a collection is

called a “document”.

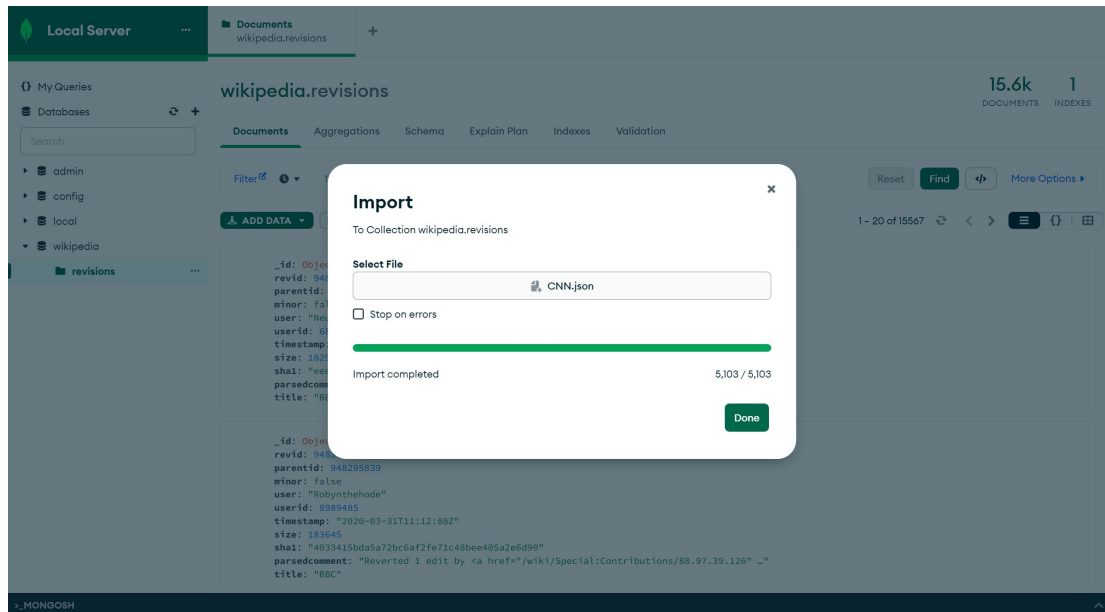


Figure 6: Import finishes in MongoDB Compass

Option 2: mongoimport (CLI)

From Canvas Week 7 Module, download a zip file named “week7-data.zip” containing two JSON files: `BBC.json` and `CNN.json`. You can inspect the content of the files using any text editor (e.g. Notepad or Sublime Text). They contain revision data of the two wikipedia articles: BBC and CNN respectively. Data about each revision is stored as an object with a number of properties: `title`, `timestamp`, `user` and so on. The collection of revisions is stored as a large array.

MongoDB provides a simple tool `mongoimport` to import JSON, CSV or TSV files to mongo collection. We will use it to import the revision data in the newly created collection revisions.

Task: For Windows users: Open a new command window or power shell window and change to directory `C:\Program Files\MongoDB\Server\6.0\bin`. Your installation path may not be the same, so if you cannot run the command, inspect where your MongoDB server is installed and change the path accordingly. For Mac users: you may try to run `mongoimport` directly, but if you're met with a Command Not Found error, install this utility by running `brew install mongodb/brew/mongodb-database-tools` and try again.

Run the following command to import a file to revisions collection.

```
➤ mongoimport --jsonArray --db wikipedia --collection revisions --file [Path-to-unzipped-lab7-files]
```

If the command executes successfully, you will see information such as how many documents are imported. Each object in the JSON file will be imported as a document

in the specified collection.

Part 4: Simple MongoDB read queries

Close the import window, and make sure you are on the right screen by navigating the left panel: Databases – wikipedia – revisions. It will show you a list of all (at least a part) of all documents in this collection. Compass was able to do this by running a `find` query for you in the background, which is similar to `select` in RDBMS. Notice a `_id` field is automatically generated for each document, with the type `ObjectId`. This serves as an identifier to each document, even across different collections. By importing a JSON/CSV file, MongoDB is smart enough to infer simple data types such as integer and string for each field. On the top-right corner of the screen, there's also a number of total documents in this collection.

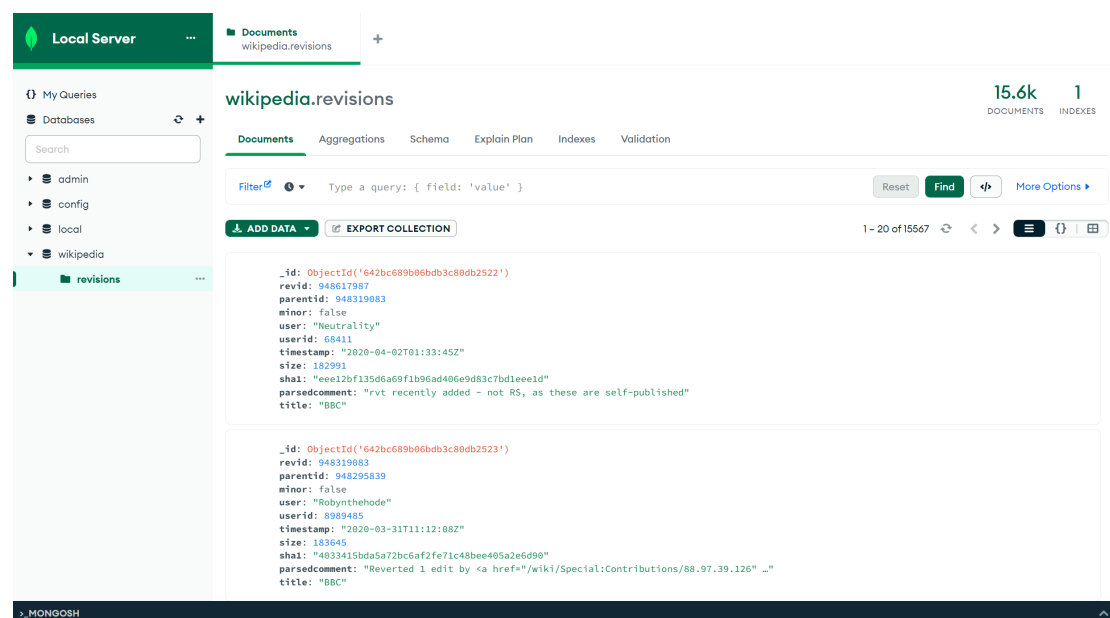


Figure 7: List of documents in a collection in MongoDB Compass

Double-clicking on any of the document's field reveals the type of each field. You will notice that all fields in the imported documents are either assigned a `String`, `Boolean` or `Int32` type. This includes the `timestamp` field.



Figure 8: A sample of fields and their types in a MongoDB document

Although Compass handles most of the queries for you, you may still do it yourself using commands. To do this, click on the `>_MONGOSH` banner on the bottom of this screen, and you will see a shell interface for you to run MongoDB commands. Mongosh uses the test database by default, type `use wikipedia` to switch to the right database we are working on.

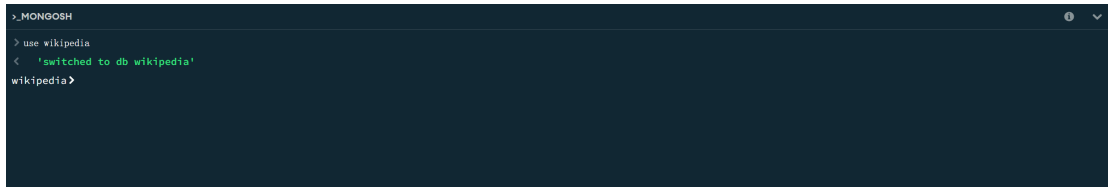


Figure 9: Mongosh interface in MongoDB Compass

Task: A few read queries you can try:

1. Find the number of documents in the revisions collection:

```
db.getCollection('revisions').find({}).count()
```

2. Find distinct users in the revisions collection:

```
db.getCollection('revisions').distinct('user')
```

3. Find distinct user from article 'BBC':

```
db.getCollection('revisions').distinct('user', {title:'CNN'})
```

4. Find the first revision of article CNN

```
db.getCollection('revisions').find({title:'CNN'}).sort({timestamp:1}).limit(1)
```

The last query starts by retrieving all documents with title 'CNN' in the revisions collection (`db.getCollection('revisions').find({title:'CNN'})`), then sort them by the field 'timestamp' in ascending order (`.sort({timestamp:1})`) and output only the first document in the results (`.limit(1)`). The 'timestamp' field currently is of type String with format 'YYYY-MM-DDTHH:MM:SS', sorting by string results would be similar to sorting by Date result.

Part 5: Aggregation

Aggregation framework allows us to run more complex queries on the collection by grouping documents based on certain criteria and summarizing the results in different ways.

Task: Run the following aggregation to find out which five editors made the most revisions on CNN page:

```
db.getCollection("revisions").aggregate([
  {$match:{title:"CNN"}},
  {$group:{_id:"$user", numOfEdits: {$sum:1}}},
  {$sort:{numOfEdits: -1}},
  {$limit:5}
])
```

The aggregation has four stages: the first stage (`$match`) finds all revision documents belong to article 'CNN'; the second stage groups (`$group`) the documents based on the "user" field value, and create a new field called "numOfEdits", the value of which is set to the number of document with that particular "user" value; the third stage (`$sort`) sorts the resulting documents by the field "numOfEdits" in descending order; the last stage (`$limit`) limits the output to the first five documents.

Part 6: JavaScript shell scripting

The mongodb shell is written in JavaScript and simple JavaScript statements/functions can be used to query or manipulate data. In particular, a read query `db.collection.find()` always returns a `cursor` object to the results. The interactive JavaScript shell by default iterates through the cursor for up to 50 items by calling the `next()` method and prints the matching documents out. In addition to the `next()` method, a lot of other methods can be used to traverse or manipulate the result set. The reference document for all cursor methods can be found from

<https://docs.mongodb.com/manual/reference/method/#cursor>

In the following example, we use simple script to update the data type of existing documents in `revisions`.

```
db.revisions.find({}).forEach(function(doc) {
  db.revisions.updateOne(
    { "_id": doc._id },
    { "$set": { "timestamp": new ISODate(doc.timestamp) } }
  );
});
```

We start by a read query `db.revisions.find({})` to find all documents in the revisions

collection. This returns a cursor object. We use the `forEach()` method of the cursor to apply a JavaScript function to each document returned (`.forEach(function(doc){...})`).

The function is an anonymous function defined right at the spot with a `db.revisions.updateOne` statement. The first parameter to this statement `{ "_id": doc._id }` identifies the document to be updated is the one we are currently iterating through. The update object `{ "$set": { "timestamp": new ISODate(doc.timestamp) } }` converts the `timestamp` field to type `ISODate`. Because we used `updateOne()` function and the ID match filter mentioned above, the new one will override the old one.

<https://docs.mongodb.com/manual/reference/method/db.collection.save/>

JavaScript statements also allows you to run a query with subsequent ones built on results from previous ones. The following example uses two queries to find out set of registered users who have made revisions on both articles. The first query finds out distinct regular users in article “CNN” and save the results as in variable `users`. The second query uses the variable to finds out the users also appear in article “BBC”.

```
users = db.revisions.distinct("user",{title:"CNN", anon:{$exists:false}});
```

```
common_users = db.revisions.distinct("user",{title:"BBC", user:{$in:users}})
```

It is possible to save script as a file and use the shell command `mongo` to run it

<https://docs.mongodb.com/manual/tutorial/write-scripts-for-the-mongo-shell/>

Part 7: Setting up indexes

Run the following command to set up an index on revisions collection.

```
db.revisions.createIndex({timestamp:1})
```

The `explain` method on cursor object can be used to see how those indexes are used in query execution. Run the following command to see how index is used.

```
db.getCollection("revisions").find(  
  {title:"CNN", timestamp:{$gte: new Date("2019-01-01")}}  
)  
.explain("executionStats");
```

The query tries to find the revisions of article “CNN” made in 2019. The query condition contains two fields: `title` and `timestamp`. There is an index on the `timestamp` field. The `executionStats` shows that the `nReturned` is 272, and this is the result of examining 680 documents. Figure 5 shows a sample output. The query runs in two stages: an input stage and a filter stage. The `timestamp` index is used in the input stage and returns 680 documents satisfy the timestamp condition. The filter stages examine the 680

documents by comparing their titles with “CNN”, which returns 272 Documents as the query results.

```
executionStages: {
  stage: 'FETCH',
  filter: {
    title: {
      '$eq': 'CNN'
    }
  },
  nReturned: 272,
  executionTimeMillisEstimate: 0,
  works: 681,
  advanced: 272,
  needTime: 408,
  needYield: 0,
  saveState: 0,
  restoreState: 0,
  isEOF: 1,
  docsExamined: 680,
  alreadyHasObj: 0,
```

Figure 10: MONGOSH explain query execution results

Part 8: Write your own query/aggregation

Task: Write your own query or aggregation to find out:

- All revisions after 2020 in article 'BBC'.
- The number of minor revisions in the `revisions` collection. A minor revision has a `minor` field with value: `true`.
- The number of minor revisions belonging to each article: 'BBC' and 'CNN'