Our application will be used to teach the learners how to use MongoDB to build an application with Python programming. In that case, it is the most important for learners to learn MongoDB sentences during the process of realizing this application. Therefore, we have designed several homework to help learners to understand MongoDB operations.

**First** I want to introduce a basic homework concerning the operation in MongoDB, which is just like a certain kind of SQL statement. We have already known this kind of SQL sentence:

**SELECT user\_id, status**

**FROM users**

**WHERE age > 25**

**AND age <= 50**

This sentence represent that I want to select all the users’ user\_id and status and these users’ ages are greater than 25 but smaller than or equal to 50. In MongoDB, we can make it in a different way.

**db.users.find(**

**{ age: { $gt: 25, $lte: 50 } }**

**{ user\_id: 1, status: 1, \_id: 0 }**

**)**

In this statement, users is the name of this collection. The find way can help us select what we want. The first brace is used to clarify the condition that the age should be greater than 25 but smaller than or equal to 50. The sign ’$gt’ means greater than and the sign ‘$lte’ means less than or equal to. In the second brace, the attributions that we want would be set to 1, otherwise 0.

For example, we need to find records of the cities in a certain region that we choose, which means the cities must meet the condition that we set. The latitude, longitude and date should be in a range. Therefore the answer will be:

**temperature.find({"LATITUDE" : {'$gt': Range[1], '$lt': Range[0]},**

**"LONGITUDE" : {'$gt':Range[3], '$lt': Range[2]},**

**"DATE" : {'$gt': earliest, '$lt': latest}}):**

The condition is the latitude is greater than Range[1] and less than Range[0], the longitude is greater than Range[3] and less than Range[2], and the date is greater than earliest and less than latest.

This is the **first** homework that I design.

I have designed **another** homework as well, which is about insert operation in MongoDB. Here is an insert operation in SQL:

**insert into instructor**

**values(10211, ’Smith’, ‘Biology’);**

In this statement, 10211 is the ID number. Smith represents the name and Biology is the department that this instructor belongs to. After this operation, a new record would be added into the table instructor.

This statement can also be realized in MongoDB. This is the way:

**db.users.insert( {**

**ID: "10211",**

**name: "Smith",**

**dept\_name: "Biology"**

**} )**

In our application, we need to get the average temperature of a whole year in every city. Therefore, the homework is to ask learners to write the sentence. We have already informed the learners that the empty collection city\_ave\_year\_temp is used to store the records that are inserted. The attributions that should be included are cities’ names, years and the average temperature.

**city\_ave\_year\_temp.insert({"CityName":item,"Year":i,"AveTemp":ave\_year\_t})**

**Thirdly**, the homework has something to do with delete operation. This time I also start with an SQL sentence:

**DELETE FROM users**

**WHERE status = "D"**

This sentence represents that we want to remove all the users whose status is D. This can also be realized by using MongoDB statement:

**db.users.remove( { status: "D" } )**

The brace means the standard that the users are supposed to meet.

Just imagine that for some reason, the weather data of a city in a certain date range are wrong so we have to delete the improper data. The third homework is to tell the learners to finish this function.

**temperature.remove({"STATION\_NAME":city,"DATE" : {'$gte': min, '$lte': max}})**

The **4th** homework that I design is about update in database. This time we also start with an SQL sentence:

**UPDATE users**

**SET status = "C"**

**WHERE age > 25**

This sentence means I want to set the users’ status to A and the users’ ages should be greater than 25. Similarly, we can realize this function in a MongoDB way:

**db.users.update(**

**{ age: { $gt: 25 } },**

**{ $set: { status: "C" } },**

**{ multi: true }**

**)**

I have to say something about the set operator. The $set operator replaces value of a field with specified value. The $set operator expression has the following form: { $set: { <field1>: <value1>, ... } }

Since we have already deleted the wrong data we have to calculate the average temperature of a whole year in every city again. And then the new outcome must be updated. The final homework is update the data. The new data is stored in variable changed. Consequently the answer should be:

**city\_ave\_year\_temp.update({"CityName":city},{'$set':{'AveTemp':changed}})**

Now we will continue with aggregation. Aggregation is just like ‘group by’ in the SQL database. The aggregation pipeline allows MongoDB to provide native aggregation capabilities that corresponds to many common data aggregation operations in SQL. This is an instance about adopting group by in SQL:

**SELECT cust\_id,**

**count(\*)**

**FROM orders**

**GROUP BY cust\_id**

**HAVING count(\*) > 1**

This SQL statement can show all the customers’ IDs and the count of ordering records. This sentence can also be realized in MongoDB:

**result = orders.aggregate( [**

**{**

**$group: {**

**\_id: "$cust\_id",**

**count: { $sum: 1 }**

**}**

**},**

**{ $match: { count: { $gt: 1 } } }**

**] )**

We can have a look on it. For cust\_id with multiple records, he cust\_id and the corresponding record count would be returned. I want to make some explanations. The output contain an \_id field which contains the distinct group by key. The attributions in the brace which is right after the sign ’$group’ are grouped. The operator ‘$match’ can match the specified conditions, which is like ‘having’ or ‘where’ in SQL.

Right now, we wanna make a conclusion about the highest temperature. The learners are asked to calculate the average highest temperatures in a certain date range (start from 2000.01.01 to 2010.01.01) for every city. The result should be stored in the variable ‘pipeline’ and the note is ‘$and’ means logical and.

A possible answer is:

**pipeline = [{'$match':{'$and':[{"DATE":{'$lt':20000101}},{"DATE":{'$gt':20100101}}]}},**

**{'$group':{'\_id' :'$STATION\_NAME',"ave\_high\_temp":{'$avg':'$TMIN'}}}]**

We can see the condition is that the date is supposed to be greater than 20000101 and smaller than 20100101. The operator ‘$avg’ is used to calculate the average.

Finally we have something about index, which is an easy and quick way to get the result. Indexes support the efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a *collection scan*, i.e. scan every document in a collection, to select those documents that match the query statement. If an appropriate index exists for a query, MongoDB can use the index to limit the number of documents it must inspect. This is an example:

**temperature.create\_index([("STATION\_NAME", pymongo.DESCENDING)])**

This statement means ‘station\_name’ is used to created an index as the primary key and the result would be sorted descendingly.

At the present the learners should create an index by using the date as the primary key. Here is the answer:

**date = temperature.create\_index([("DATE", pymongo.DESCENDING)])**

Additionally, if we want to delete the index that we have already created we can just use the ‘drop\_index’. For example, we wanna delete the date index:

**temperature.drop\_index(date)**

What’s more, if we wanna delete all the indexes, what we need to do is make the brace empty:

**temperature.drop\_indexes()**