**MongoDB Querying Questions**

1. **Write a MongoDB query to display all the documents in the collection restaurants.**

**Query:**

> db.Restaurant\_Data.find({}).pretty()

1. **Write a MongoDB query to display the fields restaurant\_id, name, borough and cuisine for all the documents in the collection restaurant.**

**Query:**

>db.Restaurant\_Data.find({},{"restaurant\_id":1,"name":1,"borough":1,"cuisine":1}).pretty()

1. **Write a MongoDB query to display the fields restaurant\_id, name, borough and cuisine, but exclude the field \_id for all the documents in the collection restaurant.**

**Query:**

>db.Restaurant\_Data.find({},{"restaurant\_id":1,"name":1,"borough":1,"cuisine":1,"\_id":0}).pretty()

1. **Write a MongoDB query to display the fields restaurant\_id, name, borough and zip code, but exclude the field \_id for all the documents in the collection restaurant.**

**Query:**

>db.Restaurant\_Data.find({},{"restaurant\_id":1,"name":1,"borough":1,"zipcode":1,"\_id":0}).pretty()

1. **Write a MongoDB query to display all the restaurant which is in the borough Bronx.**

**Query:**

> db.Restaurant\_Data.find({"borough":"Bronx"}).pretty()

1. **Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx.**

**Query:**

> db.Restaurant\_Data.find({"borough":"Bronx"}).limit(5).pretty()

1. **Write a MongoDB query to display the next 5 restaurants after skipping first 5 which are in the borough Bronx.**

**Query:**

>db.Restaurant\_Data.find({"borough":"Bronx"}).skip(5).limit(5).pretty()

1. **Write a MongoDB query to find the restaurants who achieved a score of more than 90.**

**Query:**

> db.Restaurant\_Data.find({"grades.score":{"$gt":90}}).pretty()

1. **Write a MongoDB query to find the restaurants that achieved a score of more than 80 but less than 100.**

**Query:**

>db.Restaurant\_Data.find({"grades.score":{"$gt":80,"$lt":100}}).pretty()

1. **Write a MongoDB query to find the restaurants which are located in latitude value less than -95.754168.**

**Query:**

>db.Restaurant\_Data.find({"address.coord":{"$lt":-95.754168}}).pretty()

1. **Write a MongoDB query to find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and latitude less than -65.754168.**

**Query:**

>db.Restaurant\_Data.find({$and:[{"address.coord":{"$lt":-65.754168}},{"cuisine":{"$ne":"American "}},{"grades.score":{"

$gt":70}}]}).pretty()

1. **Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American' and achieved a score of more than 70 and located in the longitude less than -65.754168. Note : Do this query without using $and operator.**

**Query:**

>db.Restaurant\_Data.find({"address.coord":{"$lt":-65.754168},"cuisine":{"$ne":"American "},"grades.score":{"$gt":70}}).

pretty()

1. **Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American ' and achieved a grade point 'A' not belong to the borough Brooklyn. The document must be displayed according to the cuisine in descending order.**

**Query:**

>db.Restaurant\_Data.find({$and:[{"cuisine":"American"},{"grades.grade":"A"},{"borough":{"$nin":["Brooklyn"]}}]}).sort({"cuisine":-1}).pretty()

1. **Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'Wil' as first three letters for its name.**

**Query:**

>db.Restaurant\_Data.find({"name":/wil/i},{"restaurant\_id":1,"name":1,"borough":1,"cuisine":1,"\_id":0}).pretty()

1. **Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which contain 'ces' as the last three letters for its name.**

**Query:**

>db.Restaurant\_Data.find({"name":/ces$/},{"restaurant\_id":1,"name":1,"borough":1,"cuisine":1,"\_id":0}).pretty()

**MongoDB Indexing**

1. **Default \_id:**
2. **Single Field Indexing:**

db.items.createIndex( { “item" : 1 } )

1. **Compound Indexes:**

> db.items.createIndex({math:1, science:-1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 2,

"numIndexesAfter" : 3,

"ok" : 1

}

> db.items.getIndexes()

[

{

"v" : 2,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "test.items"

},

{

"v" : 2,

"key" : {

"item" : 1

},

"name" : "item\_1",

"ns" : "test.items"

},

{

"v" : 2,

"key" : {

"math" : 1,

"science" : -1

},

"name" : "math\_1\_science\_-1",

"ns" : "test.items"

}

]

1. **Multikey Index**

> db.items.createIndex({ \_id: 1, product\_id: [ 1, 2 ], retail\_id: [ 100, 200 ], category: "both fields are arrays" })

1. **Hashed Index**

db.items.createIndex( { item: "hashed" } )

1. **Geospatial Index**

db.collection.createIndex( { : "2dsphere" } )

1. **To Drop Indexes**

db.items.dropIndex( { “item" : 1 } )

db.collection.dropIndexes()

**MongoDB Aggregation**

1. **Single Purpose Aggregation Operations**

db.orders.aggregate([

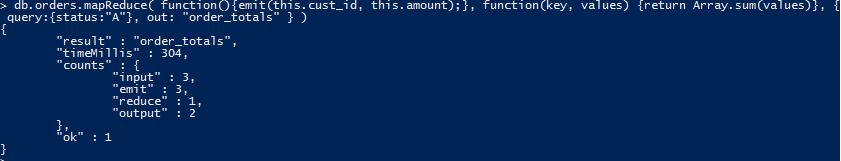
{ $match: { status: "A" } },

{ $group: { \_id: "$cust\_id", total: { $sum: "$amount" } } }

])

1. **Map Reduce**

db.orders.mapReduce( function(){emit(this.cust\_id, this.amount);}, function(key, values) {return Array.sum(values)}, {query:{status:"A"}, out: "order\_totals" } )

****

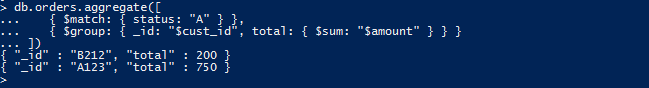
1. **Pipeline**

db.orders.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$cust\_id", total: { $sum: "$amount" } } }

])

****

**MongoDB DATABASE CONNECTIVITY**

1. **Python Connection:**

**Code:**

from pymongo import MongoClient

client = MongoClient(port=27017)

db=client.student\_02

1. **Generating sample data code example:**

**Code:**

from pymongo import MongoClient

from random import randint

client = MongoClient(port=27017)

db=client.business\_02

names = ['Kitchen','Animal','State', 'Tastey', 'Big','City','Fish', 'Pizza','Goat', 'Salty','Sandwich','Lazy', 'Fun']

company\_type = ['LLC','Inc','Company','Corporation']

company\_cuisine = ['Pizza', 'Bar Food', 'Fast Food', 'Italian', 'Mexican', 'American', 'Sushi Bar', 'Vegetarian']

for x in range(1, 501):

business = {

'name' : names[randint(0, (len(names)-1))] + ' ' + names[randint(0, (len(names)-1))] + ' ' + company\_type[randint(0, (len(company\_type)-1))],

'rating' : randint(1, 5),

'cuisine' : company\_cuisine[randint(0, (len(company\_cuisine)-1))]

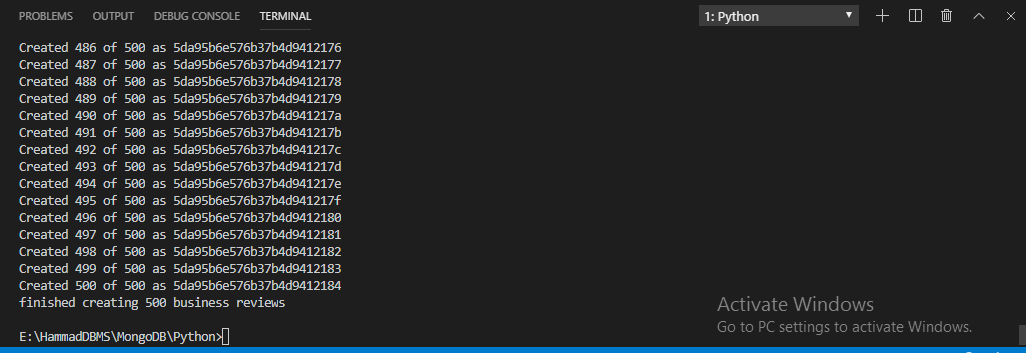
}

result=db.reviews.insert\_one(business)

print('Created {0} of 500 as {1}'.format(x,result.inserted\_id))

print('finished creating 500 business reviews')

**Output:**

****

1. **Exploring business review data with aggregation, pipeline, update and delete.**

**Code:**

from pymongo import MongoClient

from random import randint

from pprint import pprint

client = MongoClient(port=27017)

db=client.business\_02

fivestar = db.reviews.find\_one({'rating': 5})

print(fivestar)

print('The number of 5 star reviews:')

fivestarcount = db.reviews.count\_documents({'rating': 5})

print(fivestarcount)

print('\nThe sum of each rating occurance across all data grouped by rating ')

stargroup=db.reviews.aggregate(

[

{ '$group':

{ '\_id': "$rating",

"count" :

{ '$sum' :1 }

}

},

{"$sort": { "\_id":1}

}

] )

for group in stargroup:

print(group)

ASingleReview = db.reviews.find\_one({})

print('A sample document:')

pprint(ASingleReview)

result = db.reviews.update\_one({'\_id' : ASingleReview.get('\_id') }, {'$inc': {'likes': 1}})

print('Number of documents modified : ' + str(result.modified\_count))

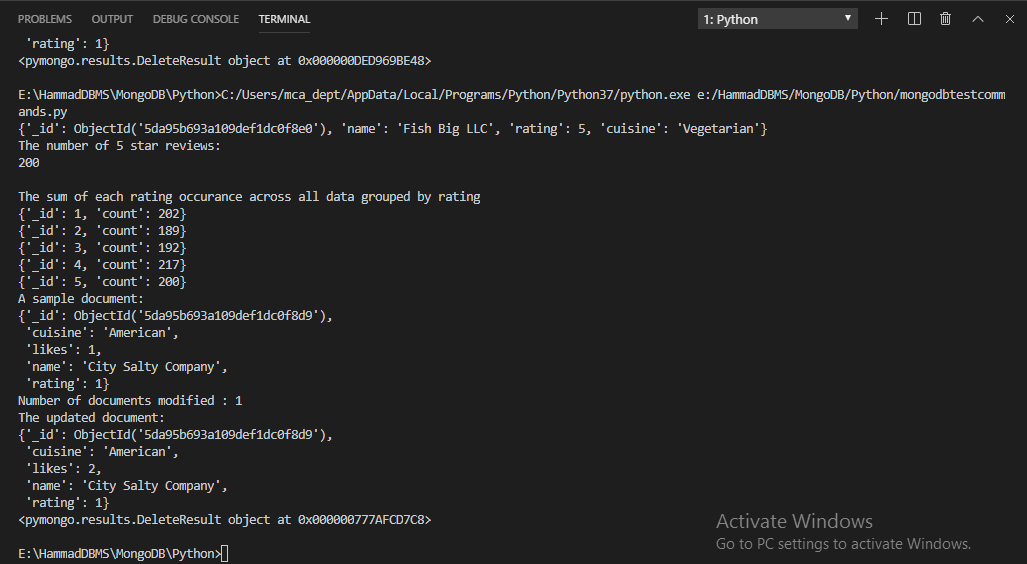
UpdatedDocument = db.reviews.find\_one({'\_id':ASingleReview.get('\_id')})

print('The updated document:')

pprint(UpdatedDocument)

result = db.reviews.delete\_many({'cuisine': 'Bar Food'})

**Output:**

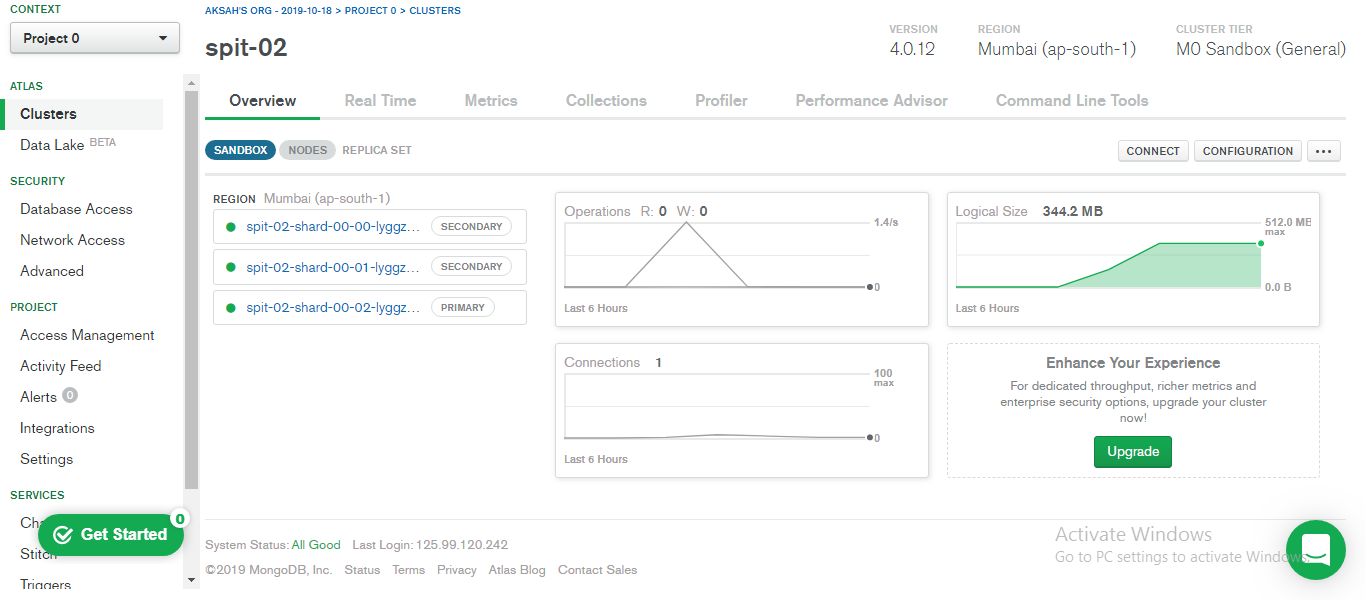
****

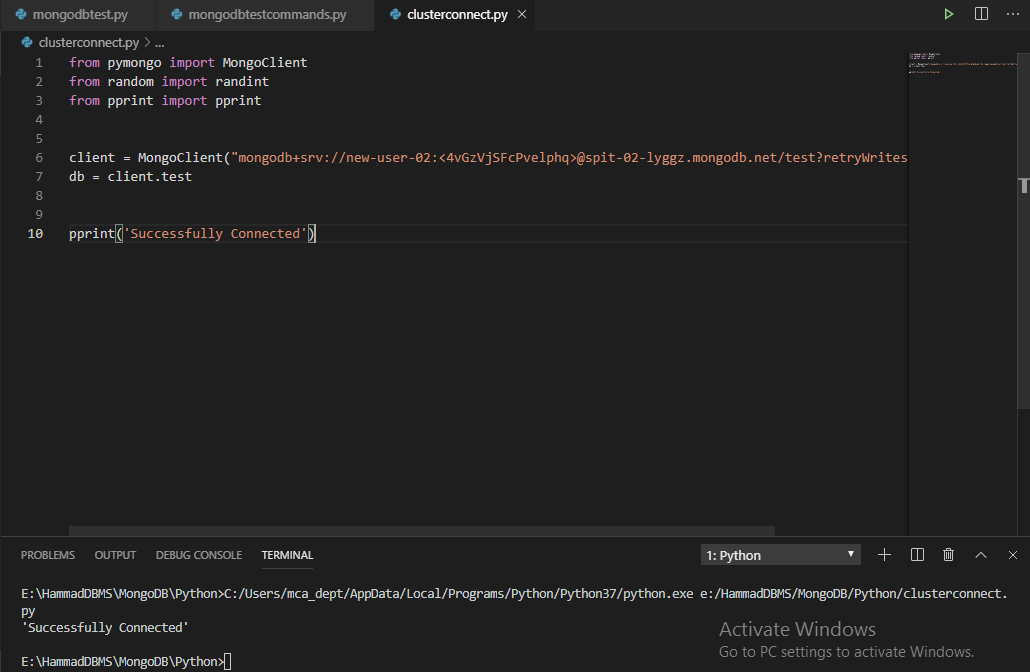
**MongoDB Atlas DATABASE CONNECTIVITY**

1. **Creating a cluster:**

**Cluster Name : spit-02**

**Cluster User : new-user-02**

****

****