José Mariano Portilla Landa 22/03/2022

MongoDB Queries

3.1 Querying the database

It is up to you now. All data is loaded. You will now query the database you just created.

Question 1: List all the restaurants in the collection, sorted in increasing order of names.

Question 2: List all restaurants with "Italian" cuisine and display the name, zip code and geographic coordinates for each of them. Also, make sure the answer is ordered according to the sorting key (increasing postcode, decreasing name).

Question 3: List all Italian restaurants with the postcode "10075" for which the telephone number is provided in the database. Display their name, zip code and phone number.

Question 4: Find all restaurants with at least a score of 50.

Question 5: List all restaurants that are either Italian or have the postcode "10075".

Question 6: List all restaurants with a zip code of "10075" or "10098" with either Italian or American cuisine with a score of at least 50.

Question 7: List all restaurants with at least one score concerning customer service *(C)*, *price (P)* or *quality (Q)*. Simply display the names, cuisine and zip code.

3.2 Updating the database

Question 8: Change the type of cuisine in the restaurant "Juni" to "American (new)". In addition, record the date and time of the system in a "lastModified" field at the time the change is made. If there are several restaurants with the same name, only the first one must be modified.

Question 9: Change the address of the restaurant whose id is "41156888" to "East 31st Street".

Question 10: Change the type of cuisine of all restaurants with a postcode "10016" and the type of cuisine "Other" into "Cuisine to be determined". In addition, record the date and time of the system in a "lastModified" field at the time the change is made.

Question 11: Replace all information about the restaurant with an ID "41154403" with the following information:

```
"name" : "Vella 2",
"address" : {
```



```
  "city" : "1480",
  "street" : "2 Avenue",
  "zipcode" : "10075"
}
```

3.3 Complex querying (aggregation)

Question 12: List the types of cuisine represented in the database. For each one, display the number of associated restaurants. Order the result by decreasing number of restaurants.

Question 13: Show, for each zip code, the number of Italian restaurants with this zip code. Order the result by decreasing number of restaurants.

Question 14: Consider Italian restaurants whose identifier (restaurant_id) is higher or equal to "41205309" and has an "averagePrice" attribute. Calculate the average of these average prices. Then repeat the same operation by calculating the average by zip code.

3.4 References between collections and joins

All this is very interesting, but our collection of restaurants is very lonely now. We will now work with several collections, linked together by references (in the manner of referential integrity constraints, or foreign keys, in relational databases).

Question 15: Create a new collection called "comments" in the same database.

Question 16: Insert three documents into the previously created collection, using the following pattern:

```
{
  "_id" : "----",
  "restaurant_id" : "----",
  "client_id" : "----",
  "comment" : "----",
  "date" : ISODate("----"),
  "type" : "----"
}
```

Some useful details:

- Restaurant identifiers must match existing restaurants in the collection `restaurants`.
- You need to provide feedback from different customers, and for different `restaurants`.
- The type `attribute` can take only the "positive" or "negative" values.

Question 17: List all the comments in your database. Each comment must also contain all the information about the restaurant to which it relates.

Question 18: Insert seven other documents into the comments collection, following the pattern described above, and following these rules:

- Restaurant identifiers must match existing restaurants in the restaurants collection.
- At least one of the restaurants must have several comments.
- At least one of the customers must have commented several times.
- At least one of the customers must have commented several times on the same restaurant.
- The type attribute can take only the "positive" or "negative" values.

Question 19: Find the list of restaurants with reviews and display only the name and comment list for each restaurant. Several strategies are possible.

3.5 Indexing

Let us now deal with the efficiency aspects of our database. We will create indexes to optimize access to our restaurant collection.

Question 20: Create an increasing index on the cuisine attribute of the restaurants collection.

Question 21: Create another index for the restaurants collection, consisting of the cuisine attribute (increasing) and the zipcode attribute (decreasing).

Question 22: List all indexes created on the restaurants collection.

Question 23: Use the explain method to display the execution plan for the query that returns all Italian restaurants. What information is provided by the system?

Question 24: Same question but adding the parameter "executionStats" in the explain method.

Question 25: Drop the two indexes you previously created, and then re-display the statistics on the query execution plan that returns all Italian restaurants. What do you see?

1) db.restaurants.find({}, {"name": 1}).sort({"name": 1})

```
> db.restaurants.find({}, {"name": 1}).sort({"name": 1})
< { _id: ObjectId("61f39e8a0317bfa72f1db6d9"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db6e6"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db6e8"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db6ef"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db6f3"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db6f5"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db6fc"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db6fd"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db704"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db70c"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db70f"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db71f"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db723"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db729"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db72a"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db732"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db734"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db736"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db739"), name: '' }
  { _id: ObjectId("61f39e8a0317bfa72f1db73b"), name: '' }
  Type "it" for more
Atlas atlas-57kx3k-shard-0 [primary] cloudDB>
```

2) db.restaurants.find({"cuisine": "Italian"}, {"name": 1, "address.zipcode": 1, "address.coord": 1, "_id": 0}).sort({"address.zipcode": 1, "name": -1})

```
> db.restaurants.find({"cuisine": "Italian"}, {"name": 1, "address.zipcode": 1, "address.coord": 1, "_id": 0}).sort({"address.zipcode": 1, "name": -1})
< { address: { coord: [ -73.9927809, 40.7451239 ], zipcode: '10001' },
    name: 'Tre Dici' }
  { address: { coord: [ -73.98952609999999, 40.7507917 ], zipcode: '10001' },
    name: 'Stella 34' }
  { address: { coord: [ -73.990708, 40.750403 ], zipcode: '10001' },
    name: 'Spinelli\'S Pizza/Gyro Ii' }
  { address: { coord: [ -73.99556, 40.748852 ], zipcode: '10001' },
    name: 'Salumeria Beillese/ Biricchino Rest' }
  { address: { coord: [ -74.00370749999999, 40.7489719 ], zipcode: '10001' },
    name: 'Pepe Giallo' }
  { address: { coord: [ -73.99426609999999, 40.7508707 ], zipcode: '10001' },
    name: 'Fb8020 Pizza Concession' }
  { address: { coord: [ -73.9930147, 40.7524427 ], zipcode: '10001' },
    name: 'Famous Famiglia' }
  { address: { coord: [ -74.0005178, 40.7471561 ], zipcode: '10001' },
    name: 'Company' }
  { address: { coord: [ -74.003165, 40.748505 ], zipcode: '10001' },
    name: 'Bottino' }
  { address: { coord: [ -73.988411, 40.7196073 ], zipcode: '10002' },
    name: 'Via Tribunali' }
  { address: { coord: [ -73.987714, 40.72184 ], zipcode: '10002' },
    name: 'Tre' }
  { address: { coord: [ -73.988809, 40.7216582 ], zipcode: '10002' },
    name: 'The Meatball Shop' }
```

3) db.restaurants.find({"cuisine": "Italian", "address.zipcode": "10075", "telephoneNumber": {$exists: true}}, {"name": 1, "address.zipcode": 1, "telephoneNumber": 1, "_id": 0})

```
> db.restaurants.find({"cuisine": "Italian", "address.zipcode": "10075", "telephoneNumber": {$exists: true}}, {"name": 1, "address.zipcode": 1, "telephoneNumber'
< { address: { zipcode: '10075' },
    name: 'Due',
    telephoneNumber: '24680356' }
  { address: { zipcode: '10075' },
    name: 'Spigolo',
    telephoneNumber: '67895432' }
```

4) db.restaurants.find({"grades.score": {$gte: 50}})

```
> db.restaurants.find({"grades.score": {$gte: 50}})
< { _id: ObjectId("61f39e7e0317bfa72f1d55d1"),
    address:
     { building: '1269',
       coord: [ -73.871194, 40.6730975 ],
       street: 'Sutter Avenue',
       zipcode: '11208' },
    cuisine: 'Chinese',
    grades:
     [ { date: 2014-09-16T00:00:00.000Z, grade: 'B', score: 21 },
       { date: 2013-08-28T00:00:00.000Z, grade: 'A', score: 7 },
       { date: 2013-04-02T00:00:00.000Z, grade: 'C', score: 56 },
       { date: 2012-08-15T00:00:00.000Z, grade: 'B', score: 27 },
       { date: 2012-03-28T00:00:00.000Z, grade: 'B', score: 27 } ],
    name: 'May May Kitchen',
    restaurant_id: '40358429' }
  { _id: ObjectId("61f39e7e0317bfa72f1d5605"),
    address:
     { building: '261',
       coord: [ -73.94839189999999, 40.7224876 ],
       street: 'Driggs Avenue',
       zipcode: '11222' },
    cuisine: 'Polish',
    grades:
     [ { date: 2014-05-31T00:00:00.000Z, grade: 'A', score: 2 },
       { date: 2013-05-10T00:00:00.000Z, grade: 'A', score: 3 },
       { date: 2012-02-17T00:00:00.000Z, grade: 'A', score: 6 },
       { date: 2011-10-14T00:00:00.000Z, grade: 'C', score: 54 } ],
    name: 'Polish National Home',
    restaurant_id: '40364404' }
  { _id: ObjectId("61f39e7e0317bfa72f1d5626"),
    address:
     { building: '4035',
       coord: [ -73.9395182, 40.8422945 ],
       street: 'Broadway',
       zipcode: '10032' },
    cuisine: 'Pizza',
```

5) db.restaurants.find({$or: [{"cuisine": "Italian"}, {"address.zipcode": "10075"}]})

```
> db.restaurants.find({$or: [{"cuisine": "Italian"}, {"address.zipcode": "10075"}]})
< { _id: ObjectId("61f39e7e0317bfa72f1d55fd"),
    address:
     { building: '10004',
       coord: [ -74.03400479999999, 40.6127077 ],
       street: '4 Avenue',
       zipcode: '11209' },
    cuisine: 'Italian',
    grades:
     [ { date: 2014-02-25T00:00:00.000Z, grade: 'A', score: 12 },
       { date: 2013-06-27T00:00:00.000Z, grade: 'A', score: 7 },
       { date: 2012-12-03T00:00:00.000Z, grade: 'A', score: 10 },
       { date: 2011-11-09T00:00:00.000Z, grade: 'A', score: 12 } ],
    name: 'Philadelhia Grille Express',
    restaurant_id: '40364305' }
  { _id: ObjectId("61f39e7e0317bfa72f1d5603"),
    address:
     { building: '1028',
       coord: [ -73.966032, 40.762832 ],
       street: '3 Avenue',
       zipcode: '10065' },
    cuisine: 'Italian',
    grades:
     [ { date: 2014-09-16T00:00:00.000Z, grade: 'A', score: 13 },
       { date: 2014-02-24T00:00:00.000Z, grade: 'A', score: 10 },
       { date: 2013-05-03T00:00:00.000Z, grade: 'A', score: 10 },
       { date: 2012-08-20T00:00:00.000Z, grade: 'A', score: 7 },
       { date: 2012-02-13T00:00:00.000Z, grade: 'A', score: 9 } ],
    name: 'Isle Of Capri Resturant',
    restaurant_id: '40364373' }
  { _id: ObjectId("61f39e7e0317bfa72f1d5611"),
    address:
     { building: '251',
       coord: [ -73.9775552, 40.7432016 ],
       street: 'East   31 Street',
       zipcode: '10016' },
    cuisine: 'Italian',
```

6) db.restaurants.find({"address.zipcode": {$in: ["10075", "10098"]}, "cuisine": {$in: ["Italian", "American"]}, "grades.score": {$gte: "50"}})

```
> db.restaurants.find({"address.zipcode": {$in: ["10075", "10098"]}, "cuisine": {$in: ["Italian", "American"]}, "grades.score": {$gte: 50}})
< { _id: ObjectId("61f39e800317bfa72f1d6b0b"),
    address:
     { building: '1462',
       coord: [ -73.953769, 40.77037 ],
       street: '1 Avenue',
       zipcode: '10075' },
    cuisine: 'American',
    grades:
     [ { date: 2015-01-16T00:00:00.000Z, grade: 'Z', score: 19 },
       { date: 2014-09-02T00:00:00.000Z, grade: 'C', score: 57 },
       { date: 2014-03-20T00:00:00.000Z, grade: 'A', score: 12 },
       { date: 2013-09-12T00:00:00.000Z, grade: 'B', score: 21 },
       { date: 2013-04-05T00:00:00.000Z, grade: 'B', score: 15 },
       { date: 2012-10-22T00:00:00.000Z, grade: 'A', score: 9 },
       { date: 2012-05-21T00:00:00.000Z, grade: 'B', score: 25 } ],
    name: 'Three Star Diner',
    restaurant_id: '41097286' }
```

7) db.restaurants.find({"grades.category": {$in: ["quality", "customer service", "price"]}}, {"name": 1, "cuisine": 1, "address.zipcode": 1, "_id": 0})

```
> db.restaurants.find({"grades.category": {$in: ["quality", "customer service", "price"]}}, {"name": 1, "cuisine": 1, "address.zipcode": 1, "_id": 0})
< { address: { zipcode: '10001' },
    cuisine: 'Hamburgers',
    name: 'Mcdonald\'S' }
  { address: { zipcode: '11236' },
    cuisine: 'Hamburgers',
    name: 'Burger King' }
  { address: { zipcode: '11222' },
    cuisine: 'Café/Coffee/Tea',
    name: 'Cafe Grumpy' }
  { address: { zipcode: '11435' },
    cuisine: 'Chinese',
    name: 'Loong Hing Chinese Restaurant' }
  { address: { zipcode: '10026' },
    cuisine: 'Hamburgers',
    name: 'Jumbos Hamburger' }
Atlas atlas-57kx3k-shard-0 [primary] cloudDB>
```

8) db.restaurants.updateOne({"name": "Juni"}, {$set: {"cuisine": "American (new)"}, $currentDate: {lastModified: true}})

```
> db.restaurants.updateOne({"name": "Juni"}, {$set: {"cuisine": "American (new)"}, $currentDate: {lastModified: true}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
Atlas atlas-57kx3k-shard-0 [primary] cloudDB>
```

9) db.restaurants.updateOne({"restaurant_id": "41156888"}, {$set: {"address.street": "East 31st Street"}})

```
>   db.restaurants.updateOne({"restaurant_id": "41156888"}, {$set: {"address.street": "East 31st Street"}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 0,
    upsertedCount: 0 }
Atlas atlas-57kx3k-shard-0 [primary] cloudDB>
```

10) db.restaurants.updateMany({"address.zipcode": "10016", "cuisine": "Other"}, {$set: {"cuisine": "Cuisine to be determined"}, $currentDate: {"lastModified": true}})

```
> db.restaurants.updateMany({"address.zipcode": "10016", "cuisine": "Other"}, {$set: {"cuisine": "Cuisine to be determined"}, $currentDate: {"lastModified": true}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 0,
    modifiedCount: 0,
    upsertedCount: 0 }
Atlas atlas-57kx3k-shard-0 [primary] cloudDB>
```

11) db.restaurants.replaceOne({"restaurant_id": "41154403"}, {"name": "Vella 2", "address": {"city": "1480", "street": "2 Avenue", "zipcode": "10075"}})

```
> db.restaurants.replaceOne({"restaurant_id": "41154403"}, {"name": "Vella 2", "address": {"city": "1480", "street": "2 Avenue", "zipcode": "10075"}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 0,
    modifiedCount: 0,
    upsertedCount: 0 }
```

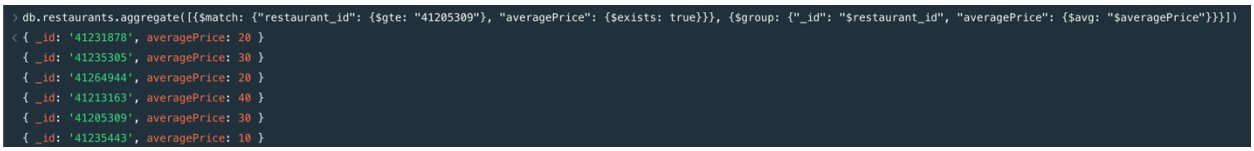
12) db.restaurants.aggregate([{$group: {"_id": "$cuisine", "associatedRestaurants": {$sum: 1}}}, {$sort: {"associatedRestaurants": -1}}])

```
> db.restaurants.aggregate([{$group: {"_id": "$cuisine", "associatedRestaurants": {$sum: 1}}}, {$sort: {"associatedRestaurants": -1}}])
< { _id: 'American', associatedRestaurants: 6179 }
  { _id: 'Chinese', associatedRestaurants: 2417 }
  { _id: 'Café/Coffee/Tea', associatedRestaurants: 1214 }
  { _id: 'Pizza', associatedRestaurants: 1162 }
  { _id: 'Italian', associatedRestaurants: 1070 }
  { _id: 'Other', associatedRestaurants: 990 }
  { _id: 'Latin (Cuban, Dominican, Puerto Rican, South & Central American)',
    associatedRestaurants: 854 }
  { _id: 'Japanese', associatedRestaurants: 759 }
  { _id: 'Mexican', associatedRestaurants: 756 }
  { _id: 'Bakery', associatedRestaurants: 691 }
  { _id: 'Caribbean', associatedRestaurants: 656 }
  { _id: 'Spanish', associatedRestaurants: 637 }
  { _id: 'Donuts', associatedRestaurants: 477 }
  { _id: 'Pizza/Italian', associatedRestaurants: 468 }
  { _id: 'Sandwiches', associatedRestaurants: 459 }
  { _id: 'Hamburgers', associatedRestaurants: 433 }
  { _id: 'Chicken', associatedRestaurants: 410 }
  { _id: 'Ice Cream, Gelato, Yogurt, Ices',
    associatedRestaurants: 348 }
  { _id: 'French', associatedRestaurants: 344 }
  { _id: 'Delicatessen', associatedRestaurants: 321 }
  Type "it" for more
```

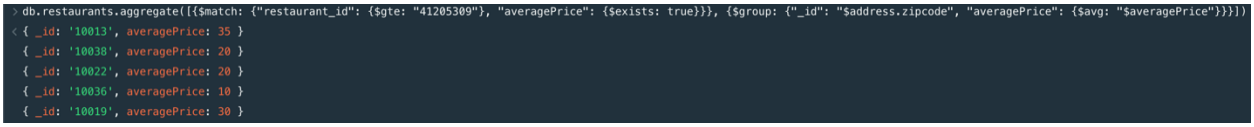
13) db.restaurants.aggregate([{$group: {"_id": "$address.zipcode", "numberOfRestaurants": {$sum: 1}}}, {$sort: {"numberOfRestaurants": -1}}])

```
> db.restaurants.aggregate([{$group: {"_id": "$address.zipcode", "numberOfRestaurants": {$sum: 1}}}, {$sort: {"numberOfRestaurants": -1}}])
< { _id: '10003', numberOfRestaurants: 686 }
  { _id: '10019', numberOfRestaurants: 674 }
  { _id: '10036', numberOfRestaurants: 610 }
  { _id: '10001', numberOfRestaurants: 520 }
  { _id: '10022', numberOfRestaurants: 485 }
  { _id: '10013', numberOfRestaurants: 480 }
  { _id: '10002', numberOfRestaurants: 471 }
  { _id: '10011', numberOfRestaurants: 467 }
  { _id: '10016', numberOfRestaurants: 434 }
  { _id: '10014', numberOfRestaurants: 428 }
  { _id: '10012', numberOfRestaurants: 406 }
  { _id: '11354', numberOfRestaurants: 378 }
  { _id: '10017', numberOfRestaurants: 376 }
  { _id: '11211', numberOfRestaurants: 361 }
  { _id: '11215', numberOfRestaurants: 347 }
  { _id: '10018', numberOfRestaurants: 345 }
  { _id: '11201', numberOfRestaurants: 345 }
  { _id: '11372', numberOfRestaurants: 319 }
  { _id: '10009', numberOfRestaurants: 311 }
  { _id: '11220', numberOfRestaurants: 311 }
  Type "it" for more
```

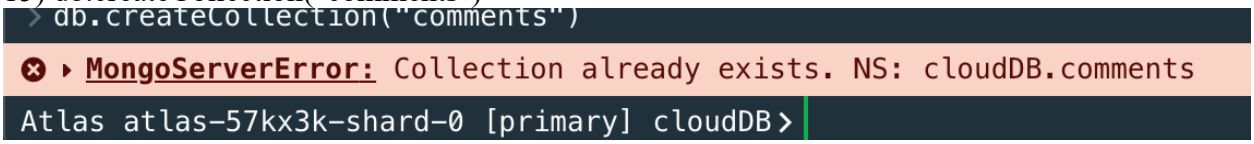
14) db.restaurants.aggregate([{$match: {"restaurant_id": {$gte: "41205309"}, "averagePrice": {$exists: true}}}, {$group: {"_id": "$restaurant_id", "averagePrice": {$avg: "$averagePrice"}}}])



db.restaurants.aggregate([{$match: {"restaurant_id": {$gte: "41205309"}, "averagePrice": {$exists: true}}}, {$group: {"_id": "$address.zipcode", "averagePrice": {$avg: "$averagePrice"}}}])



15) db.createCollection("comments")



I already had created the collection.

16) db.comments.insertMany([

{restaurant_id: "30075445", client_id: "124781426", comment: "Excellent service!", date: new Date("2014-12-03"), type: "positive"},

{restaurant_id: "30112340", client_id: "475732110", comment: "The best food I ever had!", date: new Date("2015-01-21"), type: "positive"},

{restaurant_id: "30191841", client_id: "626141589", comment: "I hated it, never returning.", date: new Date("2020-05-13"), type: "negative"}

])

17) db.comments.aggregate([{$lookup: {"from": "restaurants", "localField": "restaurant_id", "foreignField": "restaurant_id", "as": "restaurant"}}])

```
> db.comments.aggregate([{$lookup: {"from": "restaurants", "localField": "restaurant_id", "foreignField": "restaurant_id", "as": "restaurant"}}])
< { _id: ObjectId("61fd20f8897e0bddfac2d34e"),
    restaurant_id: '30075445',
    client_id: '124781426',
    comment: 'Excelent service!',
    date: 2014-12-03T00:00:00.000Z,
    type: 'positive',
    restaurant:
     [ { _id: ObjectId("61f39e7e0317bfa72f1d55c5"),
         address:
          { building: '1007',
            coord: [ -73.856077, 40.848447 ],
            street: 'Morris Park Ave',
            zipcode: '10462' },
         cuisine: 'Bakery',
         grades:
          [ { date: 2014-03-03T00:00:00.000Z, grade: 'A', score: 2 },
            { date: 2013-09-11T00:00:00.000Z, grade: 'A', score: 6 },
            { date: 2013-01-24T00:00:00.000Z, grade: 'A', score: 10 },
            { date: 2011-11-23T00:00:00.000Z, grade: 'A', score: 9 },
            { date: 2011-03-10T00:00:00.000Z, grade: 'B', score: 14 } ],
         name: 'Morris Park Bake Shop',
         restaurant_id: '30075445' } ] }
  { _id: ObjectId("61fd20f8897e0bddfac2d34f"),
    restaurant_id: '30112340',
    client_id: '475732110',
    comment: 'The best food I ever had!',
    date: 2015-01-21T00:00:00.000Z,
    type: 'positive',
    restaurant:
     [ { _id: ObjectId("61f39e7e0317bfa72f1d55c6"),
         address:
          { building: '469',
            coord: [ -73.961704, 40.662942 ],
            street: 'Flatbush Avenue',
            zipcode: '11225' },
         cuisine: 'Hamburgers',
```

18) db.comments.insertMany([

{restaurant_id: "40356018", client_id: "124781436", comment: "Really good experience!", date: new Date("2011-11-05"), type: "positive"},

{restaurant_id: "40356018", client_id: "412732110", comment: "The best restaurant ever!", date: new Date("2000-01-25"), type: "positive"},

{restaurant_id: "40356018", client_id: "626446789", comment: "Probably returning.", date: new Date("2019-03-14"), type: "positive"},

{restaurant_id: "40356068", client_id: "572941028", comment: "Not my kind of restaurant.", date: new Date("2022-01-13"), type: "negative"},

{restaurant_id: "40356068", client_id: "572941028", comment: "I hated it!", date: new Date("2013-08-21"), type: "negative"},

{restaurant_id: "40356068", client_id: "572941028", comment: "Not that bad.", date: new Date("2018-04-06"), type: "positive"},

{restaurant_id: "40356151", client_id: "412732110", comment: "There are better restaurants for the same price.", date: new Date("2017-04-19"), type: "negative"},

])

```
> db.comments.insertMany([
    {restaurant_id: "40356018", client_id: "124781436", comment: "Really good experience!", date: new Date("2011-11-05"), type: "positive"},
    {restaurant_id: "40356018", client_id: "412732110", comment: "The best restaurant ever!", date: new Date("2000-01-25"), type: "positive"},
    {restaurant_id: "40356018", client_id: "626446789", comment: "Probably returning.", date: new Date("2019-03-14"), type: "positive"},
    {restaurant_id: "40356068", client_id: "572941028", comment: "Not my kind of restaurant.", date: new Date("2022-01-13"), type: "negative"},
    {restaurant_id: "40356068", client_id: "572941028", comment: "I hated it!", date: new Date("2013-08-21"), type: "negative"},
    {restaurant_id: "40356068", client_id: "572941028", comment: "Not that bad.", date: new Date("2018-04-06"), type: "positive"},
    {restaurant_id: "40356151", client_id: "412732110", comment: "There are better restaurants for the same price.", date: new Date("2017-04-19"), type: "negative"},
])
< { acknowledged: true,
  insertedIds:
   { '0': ObjectId("6212e1fc583038de6daa74b0"),
     '1': ObjectId("6212e1fc583038de6daa74b1"),
     '2': ObjectId("6212e1fc583038de6daa74b2"),
     '3': ObjectId("6212e1fc583038de6daa74b3"),
     '4': ObjectId("6212e1fc583038de6daa74b4"),
     '5': ObjectId("6212e1fc583038de6daa74b5"),
     '6': ObjectId("6212e1fc583038de6daa74b6") } }
```

19) db.restaurants.aggregate([{$lookup: {"from": "comments", "localField": "restaurant_id", "foreignField": "restaurant_id", "as": "comments"}}, {$match: {"comments": {$ne: []}}}, {$project: {"_id": 0, "name": 1, "comments": 1}}])

```
> db.restaurants.aggregate([{$lookup: {"from": "comments", "localField": "restaurant_id", "foreignField": "restaurant_id", "as": "comments"}}, {$match: {"comments": {$ne: []}}}, {$project: {"_id": 0, "name": 1, "comments": 1}}])
< { name: 'Morris Park Bake Shop',
  comments:
   [ { _id: ObjectId("61fd20f8897e0bddfac2d34e"),
       restaurant_id: '30075445',
       client_id: '124781426',
       comment: 'Excelent service!',
       date: 2014-12-03T00:00:00.000Z,
       type: 'positive' } ] }
{ name: 'Wendy\'S',
  comments:
   [ { _id: ObjectId("61fd20f8897e0bddfac2d34f"),
       restaurant_id: '30112340',
       client_id: '475732110',
       comment: 'The best food I ever had!',
       date: 2015-01-21T00:00:00.000Z,
       type: 'positive' } ] }
{ name: 'Dj Reynolds Pub And Restaurant',
  comments:
   [ { _id: ObjectId("61fd20f8897e0bddfac2d350"),
       restaurant_id: '30191841',
       client_id: '626141589',
       comment: 'I hated it, never returning.',
       date: 2020-05-13T00:00:00.000Z,
       type: 'negative' } ] }
{ name: 'Riviera Caterer',
  comments:
   [ { _id: ObjectId("6200873c6d3204c3160bc38f"),
       restaurant_id: '40356018',
       client_id: '124781436',
       comment: 'Really good experience!',
       date: 2011-11-05T00:00:00.000Z,
       type: 'positive' },
     { _id: ObjectId("6200873c6d3204c3160bc390"),
       restaurant_id: '40356018',
       client_id: '412732110',
       comment: 'The best restaurant ever!',
```

20) db.restaurants.createIndex({"cuisine": 1})

```
>   db.restaurants.createIndex({"cuisine": 1})
< 'cuisine_1'
Atlas atlas-57kx3k-shard-0 [primary] cloudDB>
```

21) db.restaurants.createIndex({"cuisine": 1, "address.zipcode": -1})

```
> db.restaurants.createIndex({"cuisine": 1, "address.zipcode": -1})
< 'cuisine_1_address.zipcode_-1'
```

22) db.restaurants.getIndexes()

```
> db.restaurants.getIndexes()
< [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    { v: 2, key: { cuisine: 1 }, name: 'cuisine_1' },
    {
      v: 2,
      key: { cuisine: 1, 'address.zipcode': -1 },
      name: 'cuisine_1_address.zipcode_-1'
    }
  ]
```

23) db.restaurants.find({"cuisine": "Italian"}).explain()

The system provides information about the query plan:

- plannerVersion: Version of the planner
- namespace: The collection and database where the query is run
- indexFilterSet: Indicate if MongoDB applied an index filter for the query shape
- parsedQuery: The parsed query
- winningPlan: Document that details plan selected by query optimizer
- stage: Name of the stage
- inputStage: Document that describes the child stage, which provides documents or index keys to its parent
- rejectedPlans: Array of candidate plans considered rejected by the query optimizer
- indexName: Name of the index used
- keyPattern: Pattern of the index used
- serverInfo: Information about the server such as host, port and version

```
< { queryPlanner:
    { plannerVersion: 1,
      namespace: 'cloudDB.restaurants',
      indexFilterSet: false,
      parsedQuery: { cuisine: { '$eq': 'Italian' } },
      winningPlan:
       { stage: 'FETCH',
         inputStage:
          { stage: 'IXSCAN',
            keyPattern: { cuisine: 1 },
            indexName: 'cuisine_1',
            isMultiKey: false,
            multiKeyPaths: { cuisine: [] },
            isUnique: false,
            isSparse: false,
            isPartial: false,
            indexVersion: 2,
            direction: 'forward',
            indexBounds: { cuisine: [ '["Italian", "Italian"]' ] } } },
      rejectedPlans:
       [ { stage: 'FETCH',
           inputStage:
            { stage: 'IXSCAN',
              keyPattern: { cuisine: 1, 'address.zipcode': -1 },
              indexName: 'cuisine_1_address.zipcode_-1',
              isMultiKey: false,
              multiKeyPaths: { cuisine: [], 'address.zipcode': [] },
              isUnique: false,
              isSparse: false,
              isPartial: false,
              indexVersion: 2,
              direction: 'forward',
              indexBounds:
               { cuisine: [ '["Italian", "Italian"]' ],
                 'address.zipcode': [ '[MaxKey, MinKey]' ] } } } ] },
```

```
serverInfo:
 { host: 'cloudcomputing-shard-00-02.hhqdg.mongodb.net',
   port: 27017,
   version: '4.4.12',
   gitVersion: '51475a8c4d9856eb1461137e7539a0a763cc85dc' },
ok: 1,
'$clusterTime':
 { clusterTime: Timestamp({ t: 1644210500, i: 1 }),
   signature:
    { hash: Binary(Buffer.from("4152a94832dabbe0057abd4ee636a13f583d58b3", "hex"), 0),
      keyId: 7028277190319931000 } },
operationTime: Timestamp({ t: 1644210500, i: 1 }) }
```

24) db.restaurants.find({"cuisine": "Italian"}).explain("executionStats")

The system provides the same information as query 23 in addition to statistics that describe the completed query execution for the winning plan:

- executionSuccess: Whether the execution was a success or not
- nReturned: Number of documents that match the query condition
- executionTimeMillis: Total time in milliseconds required for query plan selection and query execution
- totalKeysExamined: Number of index entries scanned
- totalDocsExamined: Number of documents examined during query execution
- executionStages: Details the completed execution of the winning plan
- executionTimeMillisEstimate: Estimated amount of time in milliseconds for query execution
- works: Number of "work units" performed by the query execution stage
- advanced: Number of intermediate results returned to its parent stage
- needTime: Number of work cycles that did not return an intermediate result to its parent stage
- needYield: Number of times that the storage layer requested that the query stage suspend processing and yield its locks
- saveState: Number of times that the query stage suspended processing and saved its current execution state
- restoreState: Number of times that the query stage restored a saved execution state

- isEOF: Specifies if execution stage has reached end of stream
- keysExamined: Number of in-bounds and out-of-bounds keys that are examined in the process of the index scan
- docsExamined: Number of documents scanned during the query execution stage
- seeks: Number of times that we had to seek the index cursor to a new position in order to complete the index scan

```
> db.restaurants.find({"cuisine": "Italian"}).explain("executionStats")
< { queryPlanner:
    { plannerVersion: 1,
      namespace: 'cloudDB.restaurants',
      indexFilterSet: false,
      parsedQuery: { cuisine: { '$eq': 'Italian' } },
      winningPlan:
       { stage: 'FETCH',
         inputStage:
          { stage: 'IXSCAN',
            keyPattern: { cuisine: 1 },
            indexName: 'cuisine_1',
            isMultiKey: false,
            multiKeyPaths: { cuisine: [] },
            isUnique: false,
            isSparse: false,
            isPartial: false,
            indexVersion: 2,
            direction: 'forward',
            indexBounds: { cuisine: [ '["Italian", "Italian"]' ] } } },
      rejectedPlans:
       [ { stage: 'FETCH',
           inputStage:
            { stage: 'IXSCAN',
              keyPattern: { cuisine: 1, 'address.zipcode': -1 },
              indexName: 'cuisine_1_address.zipcode_-1',
              isMultiKey: false,
              multiKeyPaths: { cuisine: [], 'address.zipcode': [] },
              isUnique: false,
              isSparse: false,
              isPartial: false,
              indexVersion: 2,
              direction: 'forward',
              indexBounds:
               { cuisine: [ '["Italian", "Italian"]' ],
                 'address.zipcode': [ '[MaxKey, MinKey]' ] } } } ] },
```

```
executionStats:
 { executionSuccess: true,
   nReturned: 1070,
   executionTimeMillis: 3,
   totalKeysExamined: 1070,
   totalDocsExamined: 1070,
   executionStages:
    { stage: 'FETCH',
      nReturned: 1070,
      executionTimeMillisEstimate: 2,
      works: 1071,
      advanced: 1070,
      needTime: 0,
      needYield: 0,
      saveState: 1,
      restoreState: 1,
      isEOF: 1,
      docsExamined: 1070,
      alreadyHasObj: 0,
      inputStage:
```

```
        { stage: 'IXSCAN',
          nReturned: 1070,
          executionTimeMillisEstimate: 1,
          works: 1071,
          advanced: 1070,
          needTime: 0,
          needYield: 0,
          saveState: 1,
          restoreState: 1,
          isEOF: 1,
          keyPattern: { cuisine: 1 },
          indexName: 'cuisine_1',
          isMultiKey: false,
          multiKeyPaths: { cuisine: [] },
          isUnique: false,
          isSparse: false,
          isPartial: false,
          indexVersion: 2,
          direction: 'forward',
          indexBounds: { cuisine: [ '["Italian", "Italian"]' ] },
          keysExamined: 1070,
          seeks: 1,
          dupsTested: 0,
          dupsDropped: 0 } } },
  serverInfo:
   { host: 'cloudcomputing-shard-00-02.hhqdg.mongodb.net',
     port: 27017,
     version: '4.4.12',
     gitVersion: '51475a8c4d9856eb1461137e7539a0a763cc85dc' },
  ok: 1,
  '$clusterTime':
   { clusterTime: Timestamp({ t: 1644208765, i: 4 }),
     signature:
      { hash: Binary(Buffer.from("878c3908a342774eb7cb051d474338e2249943a2", "hex"), 0),
        keyId: 7028277190319931000 } },
  operationTime: Timestamp({ t: 1644208765, i: 4 }) }
```

https://docs.mongodb.com/manual/reference/explain-results/#mongodb-data-explain.queryPlanner

25) db.restaurants.dropIndexes()

db.restaurants.find({"cuisine": "Italian"}).explain("executionStats")

```
> db.restaurants.find({"cuisine": "Italian"}).explain("executionStats")
< { queryPlanner:
    { plannerVersion: 1,
      namespace: 'cloudDB.restaurants',
      indexFilterSet: false,
      parsedQuery: { cuisine: { '$eq': 'Italian' } },
      winningPlan:
       { stage: 'COLLSCAN',
         filter: { cuisine: { '$eq': 'Italian' } },
         direction: 'forward' },
      rejectedPlans: [] },
   executionStats:
    { executionSuccess: true,
      nReturned: 1070,
      executionTimeMillis: 20,
      totalKeysExamined: 0,
      totalDocsExamined: 25356,
      executionStages:
       { stage: 'COLLSCAN',
         filter: { cuisine: { '$eq': 'Italian' } },
         nReturned: 1070,
         executionTimeMillisEstimate: 10,
         works: 25358,
         advanced: 1070,
         needTime: 24287,
         needYield: 0,
         saveState: 25,
         restoreState: 25,
         isEOF: 1,
         direction: 'forward',
         docsExamined: 25356 } },
```

```
serverInfo:
 { host: 'cloudcomputing-shard-00-02.hhqdg.mongodb.net',
   port: 27017,
   version: '4.4.12',
   gitVersion: '51475a8c4d9856eb1461137e7539a0a763cc85dc' },
ok: 1,
'$clusterTime':
 { clusterTime: Timestamp({ t: 1644210646, i: 7 }),
   signature:
    { hash: Binary(Buffer.from("9912c5894d29f128d48529eced6e996a7c8c3a6d", "hex"), 0),
      keyId: 7028277190319931000 } },
operationTime: Timestamp({ t: 1644210646, i: 7 }) }
```

Impact of indexes

If we do not provide the system an index, then we can see that the winning plan has a different stage called “COLLSCAN” instead of “FETCH” and “IXSCAN”, this stage is very short, having a filter that is the same as the parsedQuery and a forward direction. There are also no rejected plans.

On the execution stats we can see that the executionTimeMillis was of 20 rather than 3, which means that it was slower than having indexes. It didn’t examined keys (previous version examined 1070) but it examined 25356 documents instead of 1070 of the previous version. We can also see that the stage is called “COLLSCAN” instead of “FETCH” and “IXSCAN”, and that the executionTimeMillisEstimate was of 10 instead of 2, they were 25358 works compared to 1071 and 24287 number of work cycles that did not return an intermediate result to its parent stage. The number of times that the query stage suspended processing and saved its current execution state, and the number of times that the query stage restored a saved execution state was of 25 vs 1 when having indexes.

In general terms, the query without indexes took more time to execute than the one with indexes because it had to do more work to find what we were asking for (all Italian restaurants). Given that the collection did not have a considerable amount of data (around 25,000 documents) the execution times were pretty small and almost imperceptible when querying, however as our database expands the execution time will also grow and can become a problem if indexes are not implemented. It is also worth noting that if we have a large database in which we are constantly writing data, then implementing indexes may cause the opposite effect by slowing down the system due to the reindexing process executing each time a new record is written, if we read more than write then indexes can be a powerful solution.

Indexes and execution plans

Without indexes we proceed to a stage called COLLSCAN automatically, we did not even have rejected plans. This stage means that we are doing a collection scan, affecting performance, however, when we implemented indexes, the system did not considered the COLLSCAN stage, instead, it executed a FETCH stage and then an IXSCAN stage. FETCH stage retrieves documents and IXSCAN scans index keys, which makes the process faster.