INT305 Lab 10 Querying Data

DB Query:

- **db.collection.find(query, projection):** Selects documents in a collection or view and returns a cursor to the selected documents.

query	Optional. Specifies selection filter using query operators. To return all documents in a collection, omit this parameter or pass an empty document ({}).
projection	Optional. Specifies the fields to return in the documents that match the query filter. To return all fields in the matching documents, omit this parameter. For details, see Projection.
Returns:	A cursor to the documents that match the query criteria. When the find() method "returns documents," the method is actually returning a cursor to the documents.

Comparison Operator:

Syntax: { <field> : {<operator> : <value>} }

$eq = EQual to , **$nq** != Not Equal to, **$gt** > Greater Than,

$gte >= Greater Than or Equal to, **$lt** < Less Than,

$lte <= Less Than or Equal to

Logical Operator:

Syntax: { <operator> : [{clause1}, {clause2},…] }

Syntax: { $not : {clause1} }

$and	Match all of the specified query clauses
$or	At least one of the query clauses is matched
$nor	Fail to match both of given clauses
$not	Negates the query requirement

Cursor Methods: These methods modify the way that the underlying query is executed.

Some Cursor Methods:

cursor.count()	Modifies the cursor to return the number of documents in the result set rather than the documents themselves.
cursor.limit()	Constrains the size of a cursor's result set. db.collection.find(<query>).limit(<number>) You must specify a numeric value for limit(). A limit() value of 0 (i.e. .limit(0)) is equivalent to setting no limit.
cursor.pretty()	Configures the cursor to display results in an easy-to-read format.
cursor.sort()	Returns results ordered according to a sort specification. sort({field : value}, {...}) Specify in the sort parameter the field or fields to sort by and a value of `1` or `-1` to specify an ascending or descending sort Respectively. Example: sort({ age : -1, posts: 1 })

More cursor methods: https://docs.mongodb.com/v4.2/reference/method/js-cursor/

Example:

```
db.trips.find({ "tripduration": { "$lte" : 70 }).limit(3);

db.trips.find({"tripduration":{"$lte":70}).sort({"tripduration":-1});
```

Resource:

https://docs.mongodb.com/v4.2/reference/method/db.collection.find/

- **db.collection.aggregate(pipeline):** Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

  - pipeline array (A sequence of data aggregation operations or stages)

    pipeline = [{ $match : { … },
    { $group : { … },
    { $sort : { … },
    …]

Aggregation Operators

$match	Used to match documents (like SQL where clause)
$project	Used to populate specific field's value(s)
$group	Used to group documents by specific field
$sum	Used to count or sum the values inside a group.
$avg	Calculates average of specific field's value per group.

$min	Finds the minimum value of a field in each group.
$max	Finds the maximum value of a field in each group.
$first, $last	Getting specific field's value from first and last document of each group: Works well when document result is sorted.
$limit	Passes the first *n* documents unmodified to the pipeline where *n* is the specified limit. For each input document, outputs either one document (for the first *n* documents) or zero documents (after the first *n* documents).
$skip	Skips the first *n* documents where *n* is the specified skip number and passes the remaining documents unmodified to the pipeline. For each input document, outputs either zero documents (for the first *n* documents) or one document (if after the first *n* documents).
$sort	Reorders the document stream by a specified sort key. Only the order changes; the documents remain unmodified. For each input document, outputs one document.
$lookup	Used to perform an equality match between a field from the input documents with a field from the documents of the "joined" collection

Example:

```
db.employees.aggregate( [
       {"$match": {"totalExp": {$gt: 3}}},
       {"$group": {"_id": "$dept",     "average_age": {"$avg":
"$age"}}},
{"$match": {"average_age": {"$lte": 35}}} ] )
```

```
Output:
{ _id: 'Facilities', average_age: 33.5 }
{ _id: 'Admin', average_age: 32.5 }
```

```
db.employees.aggregate([{$group:{"_id":"$dept",
"noOfEmployee":{$sum:1},
"minExp":{$min:"$totalExp"}}}]);
```

```
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "minExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "minExp" : 4 }
{ "_id" : "Admin", "noOfEmployee" : 2, "minExp" : 10 }
```

```
db.employees.aggregate([{$group:{"_id":"$dept",
"noOfEmployee":{$sum:1},
"maxExp":{$max:"$totalExp"}}}]);
```

```
Output:
{ "_id" : "HR", "noOfEmployee" : 2, "maxExp" : 3 }
{ "_id" : "Facilities", "noOfEmployee" : 2, "maxExp" : 14 }
{ "_id" : "Admin", "noOfEmployee" : 2, "maxExp" : 11 }
```

Resource: https://docs.mongodb.com/manual/reference/method/db.collection.aggregate/#mongodb-method-db.collection.aggregate

Task 1: Querying data using an aggregate()/distinct() collection method.

- Use the “zips” collection of the “sample_training” database to answer the questions 1.1-1.6.

1.1. Write a MongoDB query to retrieve the number of cities grouped by state. Arrange the result by the number of cities in descending order. List only the first 5 documents.

```
> db.zips.aggregate([
    {$group : {_id : {state : "$state"}, numOfCity : {$sum : 1}}},
    {$sort : {"numOfCity" : -1}},
    {$limit : 5}
    ]);
< { _id: { state: 'TX' }, numOfCity: 1676 }
  { _id: { state: 'NY' }, numOfCity: 1596 }
  { _id: { state: 'CA' }, numOfCity: 1523 }
  { _id: { state: 'PA' }, numOfCity: 1458 }
  { _id: { state: 'IL' }, numOfCity: 1240 }
```

1.2. Write a MongoDB query to retrieve the total number of population, grouped by city; retrieve the results only the state “AL” and the first 10 documents.

```
> db.zips.aggregate([
    {$match : {state : "AL"}},
    {$group : {_id : {city : "$city"}, numOfPop : {$sum : "$pop"}}},
    {$limit : 10}
    ]);
< { _id: { city: 'PANSEY' }, numOfPop: 595 }
  { _id: { city: 'LEXINGTON' }, numOfPop: 2241 }
  { _id: { city: 'GRADY' }, numOfPop: 1811 }
  { _id: { city: 'MOULTON' }, numOfPop: 17288 }
  { _id: { city: 'NEW MARKET' }, numOfPop: 5825 }
  { _id: { city: 'FOSTERS' }, numOfPop: 2100 }
  { _id: { city: 'BIGBEE' }, numOfPop: 264 }
  { _id: { city: 'ROBERTSDALE' }, numOfPop: 3519 }
  { _id: { city: 'FALKVILLE' }, numOfPop: 5392 }
  { _id: { city: 'GADSDEN' }, numOfPop: 7002 }
```

1.3. Write a MongoDB query to retrieve the maximum and minimum population, grouped by state. Display the states that the minimum population is greater than 20 and arrange the results by the maximum population in ascending order.

```
> db.zips.aggregate([
    {$group : {_id : {state : "$state"}, minPop : {$min : "$pop"}, maxPop : {$max : "$pop"}}},
    {$match : {"minPop" : {$gt : 20}}},
    {$sort : {"maxPop" : 1}}
    ]);
< { _id: { state: 'NH' }, minPop: 27, maxPop: 41438 }
  { _id: { state: 'DE' }, minPop: 108, maxPop: 50573 }
  { _id: { state: 'RI' }, minPop: 45, maxPop: 53733 }
  { _id: { state: 'IN' }, minPop: 23, maxPop: 56543 }
  { _id: { state: 'CT' }, minPop: 25, maxPop: 60670 }
  { _id: { state: 'OH' }, minPop: 38, maxPop: 66674 }
```

1.4. Write a MongoDB query to retrieve the maximum population grouped by state of 'AK', 'AL', 'AR', 'AZ'. Display the states that the maximum population is greater than 30000 and arrange the results by the state in descending order.

```
> db.zips.aggregate([
    {$match : {"$or" : [{state : "AK"}, {state : "AL"}, {state : "AR"}, {state : "AZ"}]}},
    {$group : {_id : {state : "$state"}, maxPop : {$max : "$pop"}}},
    {$match : {"maxPop" : {$gt : 30000}}},
    {$sort : {"_id.state" : -1}}
    ]);
< { _id: { state: 'AZ' }, maxPop: 57131 }
  { _id: { state: 'AR' }, maxPop: 53532 }
  { _id: { state: 'AL' }, maxPop: 44165 }
  { _id: { state: 'AK' }, maxPop: 32383 }
```

1.5. Write a MongoDB query to display the state, the average population and the number of cities grouped by states. Arrange the results by the average population in descending order and display only the first 2 documents.

```
> db.zips.aggregate([
    {$group : {_id : {state : "$state"}, avgPop : {$avg : "$pop"}, nomOfCity : {$sum: 1}}},
    {$sort : {"avgPop" : -1}},
    {$limit : 2}
    ]);
< { _id: { state: 'DC' }, avgPop: 25287.5, nomOfCity: 24 }
  { _id: { state: 'CA' },
    avgPop: 19540.39461588969,
    nomOfCity: 1523 }
```

1.6. Write a MongoDB query to display all distinct values of the states.

```
> db.zips.distinct("state")
< [
    'AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT',
    'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID',
    'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD',
    'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC',
    'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY',
    'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD',
    'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI',
    'WV', 'WY'
  ]
```

- Use the “sample_analytics” database to answer the questions 1.7-1.9.

1.7. Evaluate all collections of the “sample_analytics” database, please identify fields that represent the relationship among the collections.

มี 3 collection คือ accounts , customers และ transactions

- **accounts** ไม่ได้มีตัวเชื่อม
- **customers เชื่อมกับ accounts** ด้วย field accounts ของ customers กับ field account_id ของ accounts
- **transactions เชื่อมกับ accounts** ด้วย field account_id ของ transactions กับ field account_id ของ accounts

1.8. Write a MongoDB query to display all documents of the “accounts” collection that maps to the documents of the “customers” collection. List only the first 3 documents.

```
> db.accounts.aggregate([
    {$lookup : {
        from : "customers",
        localField : "account_id",
        foreignField : "accounts",
        as : "customer_docs"
        }},
    {$limit : 3}
    ])
```

```
{ _id: ObjectId("5ca4bbc7a2dd94ee5816238c"),
  account_id: 371138,
  limit: 9000,
  products: [ 'Derivatives', 'InvestmentStock' ],
  customer_docs:
   [ { _id: ObjectId("5ca4bbcea2dd94ee58162a68"),
       username: 'fmiller',
       name: 'Elizabeth Ray',
       address: '9286 Bethany Glens\nVasqueztown, CO 22939',
       birthdate: 1977-03-02T02:20:31.000Z,
       email: 'arroyocolton@gmail.com',
       active: true,
       accounts: [ 371138, 324287, 276528, 332179, 422649, 387979 ],
       tier_and_details:
        { '0df078f33aa74a2e9696e0520c1a828a':
           { tier: 'Bronze',
             id: '0df078f33aa74a2e9696e0520c1a828a',
             active: true,
             benefits: [ 'sports tickets' ] },
          '699456451cc24f028d2aa99d7534c219':
           { tier: 'Bronze',
             benefits: [ '24 hour dedicated line', 'concierge services' ],
             active: true,
             id: '699456451cc24f028d2aa99d7534c219' } } } ] }
```

1.9. Write a MongoDB query to display all documents of the “transactions” collection that maps to the documents of the “customers” collection. List only one document with three fields: account_id, transaction_count and the output mapping field.

```
db.transactions.aggregate([
  {$lookup : {
      from : "customers",
      localField : "account_id",
      foreignField : "accounts",
      as : "customer_transaction"
      }},
  {$project : {account_id : 1, transaction_count : 1, customer_transaction : 1, _id : 0}},
  {$limit : 1}
  ])
```

```
{ account_id: 443178,
  transaction_count: 66,
  customer_transaction:
   [ { _id: ObjectId("5ca4bbcea2dd94ee58162b2b"),
       username: 'lejoshua',
       name: 'Michael Johnson',
       address: '15989 Edward Inlet\nLake Maryton, NC 39545',
       birthdate: 1971-09-23T02:01:15.000Z,
       email: 'courtneypaul@gmail.com',
       accounts: [ 470650, 443178 ],
       tier_and_details:
        { b5f19cb532fa436a9be2cf1d7d1cac8a:
           { tier: 'Silver',
             benefits: [ 'dedicated account representative' ],
             active: true,
             id: 'b5f19cb532fa436a9be2cf1d7d1cac8a' } } } ] }
```

1.10. Select one database and create your own questions and answer those by writing MongoDB queries. **Do at least 3 questions** with different types of answers.

- นับจำนวนตาม type movie และ series ที่ปล่อยออกมาตั้งแต่ปี 2000 เป็นต้นไป

```
> db.movies.aggregate([
    {$match : {year : {$gte : 2000}}},
    {$group : {_id : {type : "$type"}, count : {$sum : 1}}}
    ]);
{ _id: { type: 'series' }, count: 138 }
 { _id: { type: 'movie' }, count: 13581 }
```

- แสดงชื่อหนังที่แต่ละคนได้ comment ไว้

```
> db.comments.aggregate([
    {$lookup : {
      from : "movies",
      localField : "movie_id",
      foreignField : "_id",
      as : "movie_comment"
    }},
    {$project : {name : 1, _id : 0, "movie_comment.title" : 1}},
    {$limit : 6}
    ]);
{ name: 'Mercedes Tyler',
   movie_comment: [ { title: 'The Land Beyond the Sunset' } ] }
 { name: 'John Bishop',
   movie_comment: [ { title: 'A Corner in Wheat' } ] }
 { name: 'Jaqen H\'ghar',
   movie_comment: [ { title: 'In the Land of the Head Hunters' } ] }
 { name: 'Taylor Scott',
   movie_comment: [ { title: 'Traffic in Souls' } ] }
 { name: 'Yara Greyjoy',
   movie_comment: [ { title: 'Regeneration' } ] }
 { name: 'Gregor Clegane',
   movie_comment: [ { title: 'Hell\'s Hinges' } ] }
```

- Average rating ของ imdb และ tomato ของหนังในแต่ละ rated และให้เรียงลำดับตามตัวอักษรของ rated แบบ ascending

```
> db.movies.aggregate([
    {$group : {_id : {rated : "$rated"}, avgImdb : {$avg : "$imdb.rating"},
              avgTomato : {$avg : "$tomatoes.viewer.rating"}}},
    {$sort : {"_id.rated" : 1}},
    {$limit : 10}
    ]);
< { _id: { rated: null },
    avgImdb: 6.825887137773259,
    avgTomato: 3.2930053063193436 }
  { _id: { rated: 'AO' },
    avgImdb: 5.3999999999999995,
    avgTomato: 3.1333333333333333 }
  { _id: { rated: 'APPROVED' },
    avgImdb: 7.168829337094499,
    avgTomato: 3.5644067796610166 }
  { _id: { rated: 'Approved' },
    avgImdb: 7.040000000000001,
    avgTomato: 3.8200000000000003 }
  { _id: { rated: 'G' },
    avgImdb: 6.6280922431865825,
    avgTomato: 3.4214285714285717 }
```

Task 2: **Querying data using a find() collection method.**

- Use the “restaurants” collection of the “sample_restaurants” database to answer the questions 2.1-2.10.

2.1. 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.

```
> db.restaurants.find({}, {
    "restaurant_id" : 1, borough : 1,
    name : 1, "address.zipcode" : 1, _id : 0
  })
< { address: { zipcode: '11224' },
    borough: 'Brooklyn',
    name: 'Riviera Caterer',
    restaurant_id: '40356018' }
  { address: { zipcode: '11234' },
    borough: 'Brooklyn',
    name: 'Wilken\'S Fine Food',
    restaurant_id: '40356483' }
  { address: { zipcode: '10314' },
    borough: 'Staten Island',
    name: 'Kosher Island',
    restaurant_id: '40356442' }
  { address: { zipcode: '11225' },
    borough: 'Brooklyn',
    name: 'Wendy\'S',
    restaurant_id: '30112340' }
  { address: { zipcode: '10462' },
    borough: 'Bronx',
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445' }
```

2.2. Write a MongoDB query to display the first 3 restaurants which are in the borough Bronx.

```
> db.restaurants.find({borough : "Bronx"}).limit(3)
< { _id: ObjectId("5eb3d668b31de5d588f4292e"),
    address:
     { building: '1007',
       coord: [ -73.856077, 40.848447 ],
       street: 'Morris Park Ave',
       zipcode: '10462' },
    borough: 'Bronx',
    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("5eb3d668b31de5d588f42933"),
  address:
   { building: '2300',
     coord: [ -73.8786113, 40.8502883 ],
     street: 'Southern Boulevard',
     zipcode: '10460' },
  borough: 'Bronx',
  cuisine: 'American',
  grades:
   [ { date: 2014-05-28T00:00:00.000Z, grade: 'A', score: 11 },
     { date: 2013-06-19T00:00:00.000Z, grade: 'A', score: 4 },
     { date: 2012-06-15T00:00:00.000Z, grade: 'A', score: 3 } ],
  name: 'Wild Asia',
  restaurant_id: '40357217' }
```

```
{ _id: ObjectId("5eb3d668b31de5d588f4294a"),
  address:
   { building: '1006',
     coord: [ -73.84856870000002, 40.8903781 ],
     street: 'East 233 Street',
     zipcode: '10466' },
  borough: 'Bronx',
  cuisine: 'Ice Cream, Gelato, Yogurt, Ices',
  grades:
   [ { date: 2014-04-24T00:00:00.000Z, grade: 'A', score: 10 },
     { date: 2013-09-05T00:00:00.000Z, grade: 'A', score: 10 },
     { date: 2013-02-21T00:00:00.000Z, grade: 'A', score: 9 },
     { date: 2012-07-03T00:00:00.000Z, grade: 'A', score: 11 },
     { date: 2011-07-11T00:00:00.000Z, grade: 'A', score: 5 } ],
  name: 'Carvel Ice Cream',
  restaurant_id: '40363093' }
```

2.3. Write a MongoDB query to display the next 5 restaurants after skipping the first 10 which are in the borough Bronx.

```
> db.restaurants.find({borough : "Bronx"})
  .skip(10)
  .limit(5)
< { _id: ObjectId("5eb3d668b31de5d588f429cd"),
    address:
     { building: '21',
       coord: [ -73.9168424, 40.8401362 ],
       street: 'East  170 Street',
       zipcode: '10452' },
    borough: 'Bronx',
    cuisine: 'American',
    grades:
     [ { date: 2014-12-16T00:00:00.000Z, grade: 'B', score: 22 },
       { date: 2014-08-12T00:00:00.000Z, grade: 'A', score: 7 },
       { date: 2014-03-03T00:00:00.000Z, grade: 'B', score: 22 },
       { date: 2013-08-29T00:00:00.000Z, grade: 'A', score: 12 },
       { date: 2012-08-29T00:00:00.000Z, grade: 'A', score: 10 },
       { date: 2012-02-13T00:00:00.000Z, grade: 'B', score: 16 } ],
    name: 'Munchtime',
    restaurant_id: '40366748' }
  { _id: ObjectId("5eb3d668b31de5d588f429ce"),
    address:
     { building: '4340',
       coord: [ -73.8194559, 40.8899176 ],
       street: 'Boston Road',
```

2.4. Write a MongoDB query to find the restaurants which do not prepare any cuisine of 'American' and achieved a score more than 70 and located in the longitude less than -65.754168.

```
> db.restaurants.find({$and : [
   {cuisine : {$ne : "American"}},
   {"grades.score" : {$gt : 70}},
   {"address.coord.0" : {$lt : -65.754168}},
   ]})
< { _id: ObjectId("5eb3d668b31de5d588f42b26"),
   address:
    { building: '345',
      coord: [ -73.9864626, 40.7266739 ],
      street: 'East 6 Street',
      zipcode: '10003' },
   borough: 'Manhattan',
   cuisine: 'Indian',
   grades:
    [ { date: 2014-09-15T00:00:00.000Z, grade: 'A', score: 5 },
      { date: 2014-01-14T00:00:00.000Z, grade: 'A', score: 8 },
      { date: 2013-05-30T00:00:00.000Z, grade: 'A', score: 12 },
      { date: 2013-04-24T00:00:00.000Z, grade: 'P', score: 2 },
      { date: 2012-10-01T00:00:00.000Z, grade: 'A', score: 9 },
      { date: 2012-04-06T00:00:00.000Z, grade: 'C', score: 92 },
      { date: 2011-11-03T00:00:00.000Z, grade: 'C', score: 41 } ],
   name: 'Gandhi',
   restaurant_id: '40381295' }
  { _id: ObjectId("5eb3d668b31de5d588f42c88"),
   address:
    { building: '130',
```

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

```
> db.restaurants.find({$and : [
    {cuisine : {$ne : "American"}},
    {borough : {$ne : "Brooklyn"}},
    {"grades.grade" : "A"}
    ]})
    .sort({cuisine : -1})
< { _id: ObjectId("5eb3d668b31de5d588f43060"),
    address:
     { building: '89',
       coord: [ -73.9995899, 40.7168015 ],
       street: 'Baxter Street',
       zipcode: '10013' },
    borough: 'Manhattan',
    cuisine: 'Vietnamese/Cambodian/Malaysia',
    grades:
     [ { date: 2014-08-21T00:00:00.000Z, grade: 'A', score: 13 },
       { date: 2013-08-31T00:00:00.000Z, grade: 'A', score: 13 },
       { date: 2013-04-11T00:00:00.000Z, grade: 'C', score: 3 },
       { date: 2012-10-17T00:00:00.000Z, grade: 'A', score: 4 },
       { date: 2012-05-15T00:00:00.000Z, grade: 'A', score: 10 } ],
    name: 'Thai Son',
    restaurant_id: '40559606' }
  { _id: ObjectId("5eb3d668b31de5d588f430eb"),
    address:
     { building: '8278',
       coord: [ -73.88143509999999, 40.7412552 ],
```

2.6. Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepare either American or Chinese dishes.

```
> db.restaurants.find({$and : [{borough : "Bronx"}, {
    $or : [{cuisine : "American"}, {cuisine : "Chinese"}]
  }]})
< { _id: ObjectId("5eb3d668b31de5d588f42933"),
    address:
     { building: '2300',
       coord: [ -73.8786113, 40.8502883 ],
       street: 'Southern Boulevard',
       zipcode: '10460' },
    borough: 'Bronx',
    cuisine: 'American',
    grades:
     [ { date: 2014-05-28T00:00:00.000Z, grade: 'A', score: 11 },
       { date: 2013-06-19T00:00:00.000Z, grade: 'A', score: 4 },
       { date: 2012-06-15T00:00:00.000Z, grade: 'A', score: 3 } ],
    name: 'Wild Asia',
    restaurant_id: '40357217' }
  { _id: ObjectId("5eb3d668b31de5d588f4294b"),
    address:
     { building: '1236',
       coord: [ -73.8893654, 40.81376179999999 ],
       street: '238 Spofford Ave',
       zipcode: '10474' },
    borough: 'Bronx',
    cuisine: 'Chinese',
    grades:
```

2.7. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which are not belonging to the borough Staten Island or Queens or Bronx or Brooklyn.

```
> db.restaurants.find({borough : {$nin : ["Staten Island", "Queens", "Bronx", "Brooklyn"]}}, {
    "restaurant_id" : 1, borough : 1, name : 1, cuisine : 1, _id : 0
  })
< { borough: 'Manhattan',
    cuisine: 'American',
    name: '1 East 66Th Street Kitchen',
    restaurant_id: '40359480' }
  { borough: 'Manhattan',
    cuisine: 'Irish',
    name: 'Dj Reynolds Pub And Restaurant',
    restaurant_id: '30191841' }
  { borough: 'Manhattan',
    cuisine: 'Delicatessen',
    name: 'Bully\'S Deli',
    restaurant_id: '40361708' }
  { borough: 'Manhattan',
    cuisine: 'American',
    name: 'Glorious Food',
    restaurant_id: '40361521' }
  { borough: 'Manhattan',
    cuisine: 'American',
    name: 'P & S Deli Grocery',
    restaurant_id: '40362264' }
  { borough: 'Manhattan',
    cuisine: 'Chicken',
    name: 'Harriet\'S Kitchen',
    restaurant_id: '40362098' }
```

2.8. Write a MongoDB query to find the restaurant Id, name, borough and cuisine for those restaurants which achieved a score which is not more than 10. Arrange the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.

```
> db.restaurants.find({"grades.score" : {$lte : 10}}, {
    "restaurant_id" : 1, borough : 1, name : 1, cuisine : 1, _id : 0
  })
  .sort({cuisine : 1, borough : -1})
< { borough: 'Queens',
    cuisine: 'Afghan',
    name: 'Afghan Kebob House',
    restaurant_id: '41535706' }
  { borough: 'Queens',
    cuisine: 'Afghan',
    name: 'Choopan Kabab Restaurant',
    restaurant_id: '41569155' }
  { borough: 'Queens',
    cuisine: 'Afghan',
    name: 'Bakhtar Kabab',
    restaurant_id: '41661199' }
  { borough: 'Queens',
    cuisine: 'Afghan',
    name: 'Bakhter Afghan Halal Kabab',
    restaurant_id: '50010813' }
```

```
  { borough: 'Manhattan',
    cuisine: 'Afghan',
    name: 'Afghan Kebab House',
    restaurant_id: '40552806' }
  { borough: 'Manhattan',
    cuisine: 'Afghan',
    name: 'Khyber Pass',
    restaurant_id: '40589545' }
  { borough: 'Manhattan',
    cuisine: 'Afghan',
    name: 'Afghan Kebab House #1',
    restaurant_id: '40616799' }
  { borough: 'Manhattan',
    cuisine: 'Afghan',
    name: 'Ariana Kebab House',
    restaurant_id: '40868400' }
  { borough: 'Brooklyn',
    cuisine: 'Afghan',
    name: 'Bahar Masala',
    restaurant_id: '41670224' }
```

2.9. Write a MongoDB query to find the restaurant Id, name, and grades for those restaurants which achieved a grade of "A" and scored 11 on an ISODate "2014-08-11T00:00:00Z" among many of survey dates. Arrange the date in descending order.

```
> db.restaurants.find({"grades.grade" : "A", "grades.score" : 11,
  "grades.date" : ISODate("2014-08-11T00:00:00Z")}, {
    "restaurant_id" : 1, name : 1, grades : 1, _id : 0
  })
  .sort({"grades.date" : -1})
< { grades:
     [ { date: 2015-01-17T00:00:00.000Z, grade: 'A', score: 6 },
       { date: 2014-08-11T00:00:00.000Z, grade: 'A', score: 11 },
       { date: 2013-12-26T00:00:00.000Z, grade: 'A', score: 7 },
       { date: 2013-06-12T00:00:00.000Z, grade: 'A', score: 9 },
       { date: 2012-05-08T00:00:00.000Z, grade: 'A', score: 5 } ],
    name: 'Sweet Afton',
    restaurant_id: '41432714' }
  { grades:
     [ { date: 2015-01-15T00:00:00.000Z, grade: 'A', score: 11 },
       { date: 2014-08-11T00:00:00.000Z, grade: 'A', score: 9 },
       { date: 2013-06-18T00:00:00.000Z, grade: 'A', score: 7 },
       { date: 2013-01-24T00:00:00.000Z, grade: 'A', score: 2 },
       { date: 2012-06-11T00:00:00.000Z, grade: 'A', score: 11 } ],
    name: 'Shoolbred\'S',
    restaurant_id: '41302014' }
  { grades:
     [ { date: 2015-01-12T00:00:00.000Z, grade: 'A', score: 10 },
       { date: 2014-08-11T00:00:00.000Z, grade: 'A', score: 9 },
       { date: 2014-01-14T00:00:00.000Z, grade: 'A', score: 13 },
```

2.10. Write a MongoDB query to find the restaurant Id, name, address and geographical location for those restaurants where 2nd element of the coord array contains a value which is more than 42 and up to 52.

```
> db.restaurants.find({$and : [{"address.coord.1" : {$gt : 42}}, {"address.coord.1" : {$lte : 52}}]}, {
    "restaurant_id" : 1, name : 1, "address.coord" : 1, _id : 0
  })
< { address: { coord: [ -78.877224, 42.89546199999999 ] },
    name: 'T.G.I. Friday\'S',
    restaurant_id: '40387990' }
  { address: { coord: [ -0.7119979, 51.6514664 ] },
    name: 'T.G.I. Fridays',
    restaurant_id: '40388936' }
  { address: { coord: [ -87.86567699999999, 42.61150920000001 ] },
    name: 'Di Luvio\'S Deli',
    restaurant_id: '40402284' }
  { address: { coord: [ -78.589606, 42.8912372 ] },
    name: 'La Caridad 78',
    restaurant_id: '40568285' }
  { address: { coord: [ -84.9751215, 45.4713351 ] },
    name: 'Bijan\'S',
    restaurant_id: '40876618' }
  { address: { coord: [ -88.0778799, 42.4154769 ] },
    name: 'Hyatt, Ny Central/Room Service',
    restaurant_id: '40879243' }
  { address: { coord: [ -111.9975205, 42.0970258 ] },
    name: 'Sports Center At Chelsea Piers (Sushi Bar)',
    restaurant_id: '40882356' }
  { address: { coord: [ -72.4751457, 43.2956803 ] },
    name: 'Fino Ristorante',
```