→ update() command

a. Update command in mongodb is used to modify one or more documents in a collection

b. update()

i. Syntax to use `update()`

```
db.collection_name.update(field_to_update, update_query)
```

In above syntax,

ii. field_to_update : we specify the field in document to be updated

iii. update_query : we specify the condition to be used to make modification

iv. Example to use `update()`

```
db.restaurant.update({name:"Wendy'S"}, {$set:{restaurant_id:1}})
```

c. updateMany()

i. If you want to update multiple documents, you'll use updateMany()

ii. It'll help you to update multiple documents in a collection that matches specific condition

iii. Example to use `updateMany()`:

```
db.restaurant.updateMany({cuisine:"Bakery"}, {$set:{cuisine:"Fresh
Bakery"}})
```

d. updateOne()

i. To update a single document in a collection you can use updateOne()

ii. Example to use `updateOne()`:

```
db.restaurant.updateOne({_id:ObjectId('64461f0e78230cd3d4e4588f')},
{$set:{borough:"Bronx"}})
```

e. **`$set:`** operator

i. Example to use `$set:`

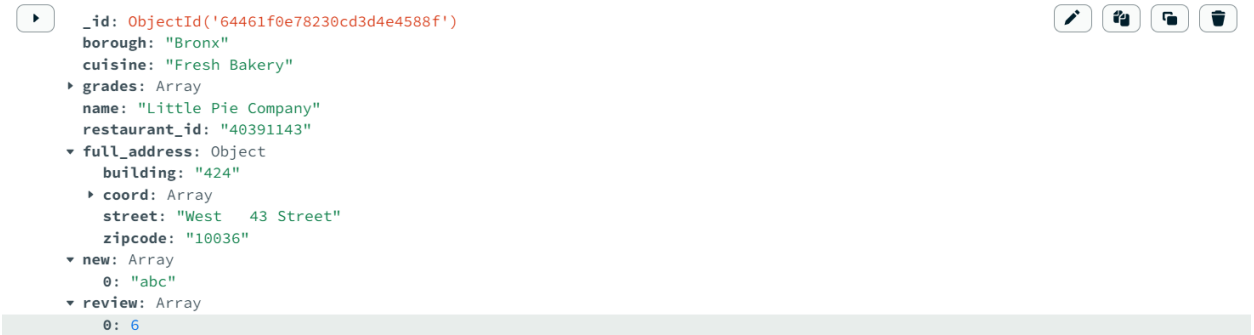
```
db.restaurant.updateMany({cuisine:"Bakery"}, {$set:{cuisine:"Fresh
Bakery"}})
```

f. **`$push:`** operator

i. We can use `$push` operator to push a new key-value pair

ii. Example to use `$push` operator

```
db.restaurant.updateOne({_id:ObjectId('64461f0e78230cd3d4e4588f')},
{$push:{new:"abc", review:6}})
```



g. **`$unset:`** operator
   i. You can use `$unset` operator to remove all the values from a particular field
   ii. Example to use `$unset` operator

```
>db.restaurant.updateOne({_id:ObjectId('64461f0e78230cd3d4e4588f')},
{$unset:{new:"", review:""}})
```

h. **`$rename:`** operator
   i. You can use `$rename:` to rename a field
   ii. Example to use `$rename:` operator

```
>db.restaurant.updateOne({_id:ObjectId('64461f0e78230cd3d4e4588f')},
{$rename:{'address':'full_address'}})
```

i. **`$addtoset:`** operator
   i. This operator adds a value to an array unless the value is already present in the array , in which case `$addToSet` does nothing to that array.
   ii. Example to use `$addtoset` operator

```
>db.restaurant.updateOne({_id:ObjectId('64461f0e78230cd3d4e4588f')},
{$addToSet:{"grades":5}})
```

j. **`$pop:`** operator
   i. If you want to remove the first or last element of an array then use `$pop:` operator
   ii. It takes two values
      1. `'1'` is used to remove the last value
      2. `'-1'` is used to remove the first value
   iii. Example to use `$pop:` operator

```
>db.restaurant.updateOne({_id:ObjectId('64461f0e78230cd3d4e4588f')},
{$pop:{"grades":1}})
```

k. **`$pull:`** operator
   i. `$pull` operator is used to remove all the existing instances or values from existing array
   ii. Syntax to use `$pull` operator

l. **`$pullAll:`** operator
   i. $pullAll operator is used to remove multiple values
   ii. Syntax to use `$pullAll:` operator

m. `save()`
   i. If you want to save the changes that you've made, then use `save()` method
   ii. Syntax to use `save()` method

```
>db.collection.save()
```

   iii. Example to use `save()` method

```
>db.restaurant.save()
```

→ `remove()`

a. `remove()`
   i. If you want to remove all the data from a collection, you'll use `remove()`, which will remove all the data from the collection
   ii. You've to pass `'{}'` in order to remove all data from collection
   iii. Syntax to use `remove({})` :

```
>db.restaurant.remove({})
```

```
>db.restaurant.remove({"salary":2000})
```

iv. You can also use conditional operators with `remove()` method

```
>db.restaurant.remove({"salary":{$lt:2000}})
```

v. You can check for two fields

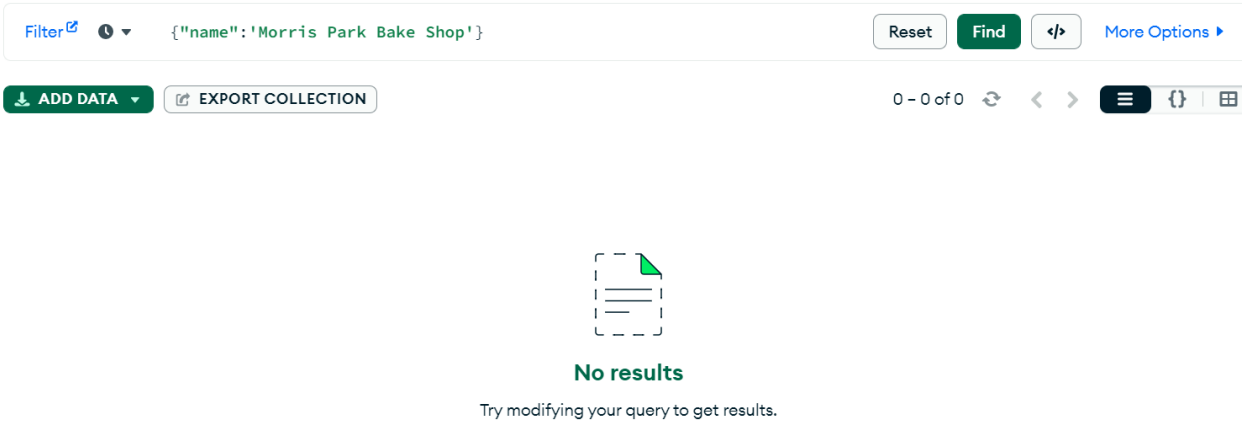
```
>db.restaurant.remove({"name":"Wendy'S", borough:"Manhattan"})
>db.restaurant.remove({borough:{$in:["Manhattan", "Brooklyn"]}})
```

vi. asa

b. `remove({})`

i. Example to use `remove({})` to remove specific field value:

```
>db.restaurant.remove({'name':'Morris Park Bake Shop'})
```



c. drop()

i. Syntax to use drop to remove specific field value:

```
>db.restaurant.drop()
```

d. dropDatabase()

i. If you want to remove entire database, then use `dropDatabase()`

ii. Syntax to use drop to remove specific field value:

```
>db.restaurant.dropDatabase()
```

→ Aggregation pipeline

a. In mongoDB, aggregation pipeline consists of stages and each stage transforms that document as they pass through pipeline

b. MongoDB aggregation pipeline uses following operators:

i. **`$match`** operator

1. It is used to filter documents in a collection based on specific condition
2. It works similar to `WHERE` condition in MySQL

3. The `$match` operator takes a query expression as an argument which may include 1 or more conditions and this will give the result of all the values that matches the condition
4. Example to use `$match` operator

```
>db.restaurant.aggregate([{ $match: { cuisine: 'Hamburgers' } }])
```

```
practice> db.restaurant.aggregate([{ $match: { cuisine: 'Hamburgers' } }])
[
  {
    _id: ObjectId("64461f0e78230cd3d4e45582"),
    address: {
      building: '469',
      coord: [ -73.961704, 40.662942 ],
      street: 'Flatbush Avenue',
      zipcode: '11225'
    },
    borough: 'Brooklyn',
    cuisine: 'Hamburgers',
    grades: [
      {
        date: ISODate("2014-12-30T00:00:00.000Z"),
        grade: 'A',
        score: 8
      },
      {
        date: ISODate("2014-07-01T00:00:00.000Z"),
        grade: 'B',
        score: 23
      },
      {
        date: ISODate("2013-04-30T00:00:00.000Z"),
        grade: 'A',
        score: 12
      },
      {
```

```
>db.emp.aggregate([{$match:{salary:{$lt:87000}}}])
```

```
sample> db.emp.aggregate([{$match:{salary:{$lt:87000}}}])
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d65"),
    name: 'Tom Williams',
    position: 'Sales Representative',
    department: 'Sales',
    salary: 85000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d69"),
    name: 'Samuel Kim',
    position: 'Customer Service Representative',
    department: 'Customer Service',
    salary: 80000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d6a"),
    name: 'Anna Lee',
    position: 'Graphic Designer',
    department: 'Design',
    salary: 85000
  }
]
```

ii. **`$project`** operator

1. This operator is used to select and transform the field to be returned in the query `SELECT`
2. It can be used to modify the structure of the document, rename the field and include/exclude the fields from the result
3. Example

```
db.emp.aggregate([{$project:{name:1, salary:1}}])
```

```
sample> db.emp.aggregate([{$project:{name:1, salary:1}}])
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    name: 'John Doe',
    salary: 90000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d62"),
    name: 'Jane Smith',
    salary: 110000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d63"),
    name: 'Bob Johnson',
    salary: 105000
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d64"),
    name: 'Sara Lee',
    salary: 95000
  },
  {
```

Here, `'1'` means that field should be included in the result

4. To exclude field use `'0'` , Example

```
db.emp.aggregate([{$project:{name:0, salary:0}}])
```

```
sample> db.emp.aggregate([{$project:{name:0, salary:0}}])
[
  {
    _id: ObjectId("6090196dbb1cfc7842916d61"),
    position: 'Software Engineer',
    department: 'Engineering'
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d62"),
    position: 'Project Manager',
    department: 'Management'
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d63"),
    position: 'Marketing Manager',
    department: 'Marketing'
  },
  {
    _id: ObjectId("6090196dbb1cfc7842916d64"),
    position: 'Human Resources',
    department: 'Human Resources'
  },
  {
```

5. **Note:** you just mention only the fields which need to be included as `'1'`, or all the fields which need to be excluded as `'0'`, but don't mix including & excluding. Don't use it like this

❌`db.emp.aggregate([{$project:{name:1, salary:0}}])`❌

```
sample> db.emp.aggregate([{$project:{name:1, salary:0}}])
MongoServerError: Invalid $project :: caused by :: Cannot do exclusion on field salary in inclusion projection
sample>
```

iii. **`$unwind`** operator

1. This operator is used to break an array into individual documents
2. It creates new documents for each element in an array and duplicates values of the other fields in the original documents
3. Example

```
>db.restaurant.aggregate([{$unwind:"$grades"}])
```

```
practice> db.restaurant.aggregate([{$unwind:"$grades"}])
[
  {
    _id: ObjectId("64461f0e78230cd3d4e45582"),
    address: {
      building: '469',
      coord: [ -73.961704, 40.662942 ],
      street: 'Flatbush Avenue',
      zipcode: '11225'
    },
    borough: 'Brooklyn',
    cuisine: 'Hamburgers',
    grades: { date: ISODate("2014-12-30T00:00:00.000Z"), grade: 'A', score: 8 },
    name: "Wendy'S",
    restaurant_id: 1
  },
  {
    _id: ObjectId("64461f0e78230cd3d4e45582"),
    address: {
      building: '469',
      coord: [ -73.961704, 40.662942 ],
      street: 'Flatbush Avenue',
      zipcode: '11225'
    },
    borough: 'Brooklyn',
    cuisine: 'Hamburgers',
```

iv. $group

1. This operator is used to group documents together based on condition and perform aggregate calculations on those groups
2. It creates a new documents for each group and can be used with various aggregate operators like **$sum**, **$avg**, **$min**, **$max**, **$count**
3. Example:

```
>db.emp.aggregate([{$group:{_id:"$salary", count:{$sum:1}}}])
```

```
sample> db.emp.aggregate([{$group:{_id:"$salary", count:{$sum:1}}}])
[
  { _id: 90000, count: 1 },
  { _id: 95000, count: 2 },
  { _id: 120000, count: 1 },
  { _id: 100000, count: 1 },
  { _id: 110000, count: 1 },
  { _id: 80000, count: 1 },
  { _id: 85000, count: 2 },
  { _id: 105000, count: 1 }
]
```

→ Indexing in MongoDB

a. Indexes are very important in databases because indexes can make queries perform in an efficient way

b. If you have a collection with thousands of documents with no indexing. To find a particular document, we need to search all of the documents. But, if you have indexed the documents, then you limit the number of documents to be searched in a collection.

c. An index can either be based on one field or multiple fields in a collection

d.
```
{ _id:’23123’,
emp_id:’12345’,
emp_name:”abc”,
salary:9876,
dept_id:567
```

}

e. **createIndex()**

i. Example to create index using `createIndex()`:

```
>db.emp.createIndex({name:1})
```

```
sample> db.emp.createIndex({name:1})
name_1
```

ii. Example to create index on multiple fields using `createIndex()`:

```
>db.emp.createIndex({name:1, _id:1})
```

```
sample> db.emp.createIndex({name:1, _id:1})
name_1__id_1
```

f. **getIndexes()**

i. Example to show indexes using `getIndexes()`:

```
>db.emp.getIndexes()
```

```
sample> db.emp.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { name: 1 }, name: 'name_1' },
  { v: 2, key: { name: 1, _id: 1 }, name: 'name_1__id_1' }
]
```

→ problem / challenge: WAQ to display all the documents in a collection

```
>db.emp.find()
```

→ problem / challenge: WAQ to display fields: restaurant_id, name, cuisine from all the documents

```
>db.restaurant.aggregate({$project:{name:1, restaurant_id:1, cuisine:1}})
```

```
practice> db.restaurant.aggregate({$project:{name:1, restaurant_id:1, cuisine:1}})
[
  {
    _id: ObjectId("64461f0e78230cd3d4e45582"),
    cuisine: 'Hamburgers',
    name: "Wendy'S",
    restaurant_id: 1
  },
  {
    _id: ObjectId("64461f0e78230cd3d4e45583"),
    cuisine: 'Irish',
    name: 'Dj Reynolds Pub And Restaurant',
    restaurant_id: '30191841'
  },
  {
    _id: ObjectId("64461f0e78230cd3d4e45584"),
    cuisine: 'American ',
    name: 'Riviera Caterer',
    restaurant_id: '40356018'
  },
  {
    _id: ObjectId("64461f0e78230cd3d4e45585"),
    cuisine: 'Jewish/Kosher',
    name: 'Tov Kosher Kitchen',
    restaurant_id: '40356068'
  },
```

→ problem / challenge: WAQ to display all the fields excluding name & _id

```
>db.restaurant.aggregate({$project:{name:0, _id:0}})
```

```
practice> db.restaurant.aggregate({$project:{name:0, _id:0}})
[
  {
    address: {
      building: '469',
      coord: [ -73.961704, 40.662942 ],
      street: 'Flatbush Avenue',
      zipcode: '11225'
    },
    borough: 'Brooklyn',
    cuisine: 'Hamburgers',
    grades: [
      {
        date: ISODate("2014-12-30T00:00:00.000Z"),
        grade: 'A',
        score: 8
      },
      {
        date: ISODate("2014-07-01T00:00:00.000Z"),
        grade: 'B',
        score: 23
      },
      {
        date: ISODate("2013-04-30T00:00:00.000Z"),
        grade: 'A',
        score: 12
      },
      {
```

→ problem / challenge: WAQ to display all the fields: restaurant_id & cuisine, but excluding field name '_id' from all the documents

```
>db.restaurant.find({borough:"Brooklyn"})
```

```
practice> db.restaurant.find({borough:"Brooklyn"})
[
  {
    _id: ObjectId("64461f0e78230cd3d4e45582"),
    address: {
      building: '469',
      coord: [ -73.961704, 40.662942 ],
      street: 'Flatbush Avenue',
      zipcode: '11225'
    },
    borough: 'Brooklyn',
    cuisine: 'Hamburgers',
    grades: [
      {
        date: ISODate("2014-12-30T00:00:00.000Z"),
        grade: 'A',
        score: 8
      },
      {
        date: ISODate("2014-07-01T00:00:00.000Z"),
        grade: 'B',
        score: 23
      },
      {
        date: ISODate("2013-04-30T00:00:00.000Z"),
        grade: 'A',
```

→ problem / challenge: WAQ to display all the restaurants in brooklyn

```
>db.restaurant.find({borough:"Brooklyn"}).limit(5)
```

```
practice> db.restaurant.find({borough:"Brooklyn"}).limit(5)
[
  {
    _id: ObjectId("64461f0e78230cd3d4e45582"),
    address: {
      building: '469',
      coord: [ -73.961704, 40.662942 ],
      street: 'Flatbush Avenue',
      zipcode: '11225'
    },
    borough: 'Brooklyn',
    cuisine: 'Hamburgers',
    grades: [
      {
        date: ISODate("2014-12-30T00:00:00.000Z"),
        grade: 'A',
        score: 8
      },
      {
        date: ISODate("2014-07-01T00:00:00.000Z"),
        grade: 'B',
        score: 23
      },
      {
        date: ISODate("2013-04-30T00:00:00.000Z"),
        grade: 'A',
        score: 12
      },
      {
```

→ WAQ to display first five after skipping first 5 which are in brooklyn

```
>db.restaurant.find({borough:"Brooklyn"}).skip(5).limit(5)
```

```
practice> db.restaurant.find({borough:"Brooklyn"}).skip(5).limit(5)
[
  {
    _id: ObjectId("64461f0e78230cd3d4e4558c"),
    address: {
      building: '7715',
      coord: [ -73.9973325, 40.61174889999999 ],
      street: '18 Avenue',
      zipcode: '11214'
    },
    borough: 'Brooklyn',
    cuisine: 'American ',
    grades: [
      {
        date: ISODate("2014-04-16T00:00:00.000Z"),
        grade: 'A',
        score: 5
      },
      {
        date: ISODate("2013-04-23T00:00:00.000Z"),
        grade: 'A',
        score: 2
      },
      {
        date: ISODate("2012-04-24T00:00:00.000Z"),
        grade: 'A',
        score: 5
      },
      {
        date: ISODate("2011-12-16T00:00:00.000Z"),
```

→ WAQ to display restaurants which have scored more than 90(use `$elemMatch` & `$gt` operator)

```
>db.restaurant.find({grades:{$elemMatch:{score:{$gt:90}}}})
```

```
practice> db.restaurant.find({grades:{$elemMatch:{score:{$gt:90}}}})
[
  {
    _id: ObjectId("64461f0e78230cd3d4e456df"),
    address: {
      building: '65',
      coord: [ -73.9782725, 40.7624022 ],
      street: 'West   54 Street',
      zipcode: '10019'
    },
    borough: 'Manhattan',
    cuisine: 'American ',
    grades: [
      {
        date: ISODate("2014-08-22T00:00:00.000Z"),
        grade: 'A',
        score: 11
      },
      {
        date: ISODate("2014-03-28T00:00:00.000Z"),
        grade: 'C',
        score: 131
      },
      {
        date: ISODate("2013-09-25T00:00:00.000Z"),
        grade: 'A',
        score: 11
      },
```

→ WAQ to display restaurants which have a score of greater than 80 and less than 100

```
>db.restaurant.find({grades:{$elemMatch:{$and:[{score:{$gt:80}},
{score:{$lt:100}}]}}})
```

```
practice> db.restaurant.find({grades:{$elemMatch:{$and:[{score:{$gt:80}}, {score:{$lt:100}}]}}})
[
  {
    _id: ObjectId("64461f0e78230cd3d4e45780"),
    address: {
      building: '345',
      coord: [ -73.9864626, 40.7266739 ],
      street: 'East 6 Street',
      zipcode: '10003'
    },
    borough: 'Manhattan',
    cuisine: 'Indian',
    grades: [
      {
        date: ISODate("2014-09-15T00:00:00.000Z"),
        grade: 'A',
        score: 5
      },
      {
        date: ISODate("2014-01-14T00:00:00.000Z"),
        grade: 'A',
        score: 8
      },
      {
        date: ISODate("2013-05-30T00:00:00.000Z"),
        grade: 'A',
        score: 12
      }
```

```
>db.restaurant.find({grades:{$elemMatch:{score:{$gt:80, $lt:90}}}})
```

```
practice> db.restaurant.find({grades:{$elemMatch:{$and:[{score:{$gt:80}}, {score:{$lt:100}}]}}})
[
  {
    _id: ObjectId("64461f0e78230cd3d4e45780"),
    address: {
      building: '345',
      coord: [ -73.9864626, 40.7266739 ],
      street: 'East 6 Street',
      zipcode: '10003'
    },
    borough: 'Manhattan',
    cuisine: 'Indian',
    grades: [
      {
        date: ISODate("2014-09-15T00:00:00.000Z"),
        grade: 'A',
        score: 5
      },
      {
        date: ISODate("2014-01-14T00:00:00.000Z"),
        grade: 'A',
        score: 8
      },
      {
        date: ISODate("2013-05-30T00:00:00.000Z"),
        grade: 'A',
        score: 12
      },
      {
```

→ WAQ to display restaurants which does not prepare any cuisine of ‘American’ and their grade is more than 70

```
>db.restaurant.find({$and:[ {cuisine:{$ne:"American "}},
{'grades.score':{$gt:70}} ] })
```

```
practice> db.restaurant.find({$and:[ {cuisine:{$ne:"American "}}, {'grades.score':{$gt:70}} ] })
[
  {
    _id: ObjectId("64461f0e78230cd3d4e45780"),
    address: {
      building: '345',
      coord: [ -73.9864626, 40.7266739 ],
      street: 'East 6 Street',
      zipcode: '10003'
    },
    borough: 'Manhattan',
    cuisine: 'Indian',
    grades: [
      {
        date: ISODate("2014-09-15T00:00:00.000Z"),
        grade: 'A',
        score: 5
      },
      {
        date: ISODate("2014-01-14T00:00:00.000Z"),
        grade: 'A',
        score: 8
      },
      {
        date: ISODate("2013-05-30T00:00:00.000Z"),
        grade: 'A',
        score: 12
      },
      {
```

→ WAQ to display restaurant_id, name, cuisine for those restaurants which contains ‘ces’ as the last three characters of its name

```
>db.restaurant.find({name:{$regex:/ces$/}}, {restaurant_id:1, name:1,
cuisine:1})
```

```
practice> db.restaurant.find({name:{$regex:/ces$/}}, {restaurant_id:1, name:1, cuisine:1})
[
  {
    _id: ObjectId("64461f0e78230cd3d4e45a14"),
    cuisine: 'American ',
    name: 'Pieces',
    restaurant_id: '40399910'
  },
  {
    _id: ObjectId("64461f0e78230cd3d4e45ad3"),
    cuisine: 'American ',
    name: 'S.M.R Restaurant Services',
    restaurant_id: '40403857'
  },
  {
    _id: ObjectId("64461f0e78230cd3d4e45ad9"),
    cuisine: 'American ',
    name: 'Good Shepherd Services',
    restaurant_id: '40403989'
  },
  {
    _id: ObjectId("64461f0f78230cd3d4e45f8c"),
    cuisine: 'Ice Cream, Gelato, Yogurt, Ices',
    name: "The Ice Box-Ralph'S Famous Italian Ices",
    restaurant_id: '40690899'
  },
  {
    _id: ObjectId("64461f0f78230cd3d4e4618e"),
```

→ WAQ to find a restaurant which doesn't prepare cuisine “American” and achieved a grade point ‘A’ and does not belong to “Brooklyn” . display the document in descending order of cuisine

```
>db.restaurant.find({cuisine:{$ne:"American "}})
>db.restaurant.find({$and:[{cuisine:{$ne:"American "}},{}, {} ]})
>db.restaurant.find({$and:[{cuisine:{$ne:"American
"}},{'grades.grade':"A"}, {} ]})
>db.restaurant.find({$and:[{cuisine:{$ne:"American
"}},{'grades.grade':"A"}, {borough:{$ne:"Brooklyn"}} ]})
>db.restaurant.find({$and:[{cuisine:{$ne:"American
"}},{'grades.grade':"A"}, {borough:{$ne:"Brooklyn"}}
]}).sort({cuisine:-1})
```

```
practice> db.restaurant.find({$and:[{cuisine:{$ne:"American "}},{'grades.grade':"A"}, {borough:{$ne:"Brooklyn"}} ]}).sor
t({cuisine:-1})
[
  {
    _id: ObjectId("64461f0e78230cd3d4e45c8d"),
    address: {
      building: '89',
      coord: [ -73.9995899, 40.7168015 ],
      street: 'Baxter Street',
      zipcode: '10013'
    },
    borough: 'Manhattan',
    cuisine: 'Vietnamese/Cambodian/Malaysia',
    grades: [
      {
        date: ISODate("2014-08-21T00:00:00.000Z"),
        grade: 'A',
        score: 13
      },
      {
        date: ISODate("2013-08-31T00:00:00.000Z"),
        grade: 'A',
        score: 13
      },
      {
        date: ISODate("2013-04-11T00:00:00.000Z"),
        grade: 'C',
        score: 3
      },
```

→ WAQ to display the cuisine which is most likely to receive ‘C’ grade

```
>db.restaurant.aggregate({$match:{'grades.grade':"C"}})
>
```