**•Find all the books with a number of pages greater than 250**

db.book.find({n: {$gt: 250 }})

**•Find all the books authored by Mario Rossi**

db.book.find({"author.name": "Mario", "author.surname": "Rossi" })

**•Find all the books with a price less than 20 € for the country Italy (IT)**

db.book.find({"price": {$elemMatch: {"v": {$lt: 20}, "country": "IT" }}} )

**•Increase the review score of 0.1 for all the books with the tag database**

db.book.updateMany({tag: "database" }, { $inc: {review\_score: 0.2} })

**•Insert the tag “NoSQL” for all the books with tag “mongodb”**

db.book.updateMany( {"author.name": "Mario", "author.surname": "Rossi"}, {$set: {publisher: {name:"Polito", city:"Turin"}}} )

**•Insert the publisher for all the documents authored by Mario Rossi with the default value {‘name’: ‘Polito’, city:’Turin’}** db.book.updateMany({tag: "mongodb" }, { $addToSet: {tag: "NoSQL"} })

**•Find the maximum, the minum and the average price of all the books with tag “database”**

db.book.aggregate([ {$match: {tag: "database" }},

{$unwind: "$price"},

{$group: {\_id: null, avg: {$avg: "$price.v"} , min: {$min: "$price.v"} , max: {$max: "$price.v"} } } ])

**•Compute the number of books authored by Mario Rossi**

db.book.count({ "author.name": "Mario", "author.surname": "Rossi" })

db.book.find({ "author.name": "Mario", "author.surname": "Rossi" }).count()

"\_id": "10006546",

"url": "https://www.airbnb.com/rooms/10006546",

"name": "Ribeira Charming Duplex",

"description": "Fantastic duplex apartment with . . .",

"property\_type": "House",

"minimum\_nights": 2,

"maximum\_nights": 30,

"beds": 5,

"number\_of\_reviews": 51,

"amenities": [ "TV", "Wifi", "Smoking allowed", "Pets allowed", "Dryer", "Heating" ],

"price": 80,

"host": { "host\_id": "51399391", "name": "Ana&Gonçalo", "city": "Porto", "country": "Portugal" }

**Select all properties that have wifi and dryer as amenities** **sorted by increasing price. Display only the name, url and price.**

db.listingsAndReviews.find(

{ amenities: {$all: ["Wifi", "Dryer"] } }, {\_id:0, name: 1, price:1, url: 1} ).sort( {price:1} )

**For all House type properties located in the city of Turin, set the minimum number of nights to 3. If the minimum number of nights is greater than 3, it should not be updated.**

db.listingsAndReviews.updateMany(

{"property\_type": "House", "host.city": "Turin", "minimum\_nights": {lt: 3 } },

{$set: {"minimum\_nights": 3}} )

**Calculate the median price of all House type properties.**

db.listingsAndReviews.aggregate( [

{ $match: {property\_type: "House"}},

{ $sort: {price: 1}},

{ $group: { '\_id': null,

'value': {'$push': '$price'}}},

{ $project: { \_id: 1,

"median": {$arrayElemAt:["$value", {$floor: {$multiply: [0.50,{$size: "$value"} ]}} ]}}} ] )

EXAMS

{ "\_id": "56d5f7eb604eb380b0d8d8d2",

"student\_id": 0,

"scores": [ { "type": "exam", "score": 41.25131199553351 },

{ "type": "quiz", "score": 91.7351500084582 },

{ "type": "homework", "score": 24.198828271948415 },

{ "type": "homework", "score": 79.77471812670814 } ],

"class\_id": 391}

**A) For each type of test, calculate the minimum, maximum, and average grades taken by students. Display this information only for the types with more than 50 tests taken.**

db.grades.aggregate( [

{$unwind: {path: '$scores'}},

{$group: { \_id: '$scores.type',

avg: {$avg: '$scores.score'},

min: {$min: '$scores.score'},

max: {$max: '$scores.score'},

count: {$sum: 1}}},

{$match: { count: {$gt:50}}}] )

**B) Considering only the sufficient scores (grade equal to or higher than 60) of the exam type, calculate for each class the number of exams taken and the average of the grades obtained**

db.grades.aggregate( [

{$unwind: { path: "$scores", }},

{$match: { "scores.type":"exam",

"scores.score": {$gte: 60} }},

{$group: { \_id: "$class\_id",

c: {$sum:1},

avg: {$avg:"$scores.score"} }}]

**C) Enter the certficate grade of 85 for the student with identifier of 1000 enrolled in grade 150. Also set a flag named "completed"**

db.grades.updateOne(

{ "student\_id": 1000, class\_id:150},

{$push: {scores: {"type": "certificate", "grade": 85 }},

$set: {completed: true} })

{"\_id":ObjectId("573a13f3f29313caabddeb3c"),  
"title":"Si accettano miracoli",  
"countries":["Italy"],  
"genres":["Comedy"],  
"runtime":110,  
"cast":["Alessandro Siani","Fabio De Luigi","Ana Caterina Morariu","Serena Autieri"],  
"languages":["Italian"],  
"released": Date("2015-01-01"),  
"directors":["Alessandro Siani"],  
"writers":["Gianluca Ansanelli","Tito Buffolini","Alessandro Siani"],  
"awards":{  
    "wins":1,  
    "nominations":0},  
"year":2015,  
"imdb":{  
    "rating":4.9,  
    "votes":269,  
},  
"type":"movie"  
}

**Select the top 10 genres with the highest average imdb rating.**

db.movies.aggregate([{$unwind: { path: "$genres",}},

{$group: { \_id: "$genres",

avg\_rating: {$avg: "$imdb.rating"}

}}, {$sort: { avg\_rating: -1}}, {$limit: 10}])

**Select all comedy movies in Italian language with an imdb rating greater than 4.5. Display only the title, the release date and the imdb rating.**

db.movies.find ( {languages:"Italian", "imdb.rating":{$gte: 4.5}, "genres": "Comedy"}, {\_id:0, title: 1, released:1, "imdb.rating":1})

  "\_id": "10006546",  
  "url": "https://www.airbnb.com/rooms/10006546",  
  "name": "Ribeira Charming Duplex",  
  "description": "Fantastic duplex apartment with . . .",  
  "property\_type": "House",  
  "minimum\_nights": 2,  
  "maximum\_nights": 30,  
  "beds": 5,  
  "number\_of\_reviews": 51,  
  "amenities": [  
    "TV",  
    "Wifi",  
    "Smoking allowed",  
    "Pets allowed",  
    "Dryer",  
    "Heating"  
  ],  
  "price": 80,  
  "host": {  
    "host\_id": "51399391",  
    "name": "Ana&Gonçalo",  
    "city": "Porto",  
    "country": "Portugal"

**Calculate the median price of all properties with the minimum number of nights greater than 3.**

db.hotel.aggregate([

{$match : {'minimum\_nights': {$gt : 3}}}

{$sort : {'price' : 1}},

{$group : {\_id:null,

                x : {$push : '$price'}}}

{$project : {\_id : 1 , mead : {$arrayElemAt : [x , {$floor : {$multiply : [0.5 , {$size : '$x'}]}}]}

])

**Select all properties that have TV and Wifi as amenities sorted by decreasing price. Display only the name, url, price and host.**

db.hotels.find({amenities : {$all : ['tv' , 'wifi']}},

{\_id:0 , name : 1 , url : 1 , price : 1 , host : 1}).sort({'price' : -1})

{"\_id": ObjectId(“xyz”),

"title": "Python 3.9",

"teacher":{ "name": "John", “surname”: “Doe”, "webiste": "https://www.doe.com/", “nation”: “USA” },

"published": Date("2019-02-13T00:00:00.000Z"),

"category": "Computer Science",

“tags : [“Python”, “Coding”],

"price": 99,

“avg\_score”: 4.8,

“number\_reviews”: 47,

“enrolled\_students”: 1234,

“details: { “hour\_length”: 12, “number\_of\_lessons: 38, “final\_test”: false }}

**Write a MongoDB query to display only the title, the category, and the price of courses containing the tag “Databases”, published in 2019, and whose length is less than 10 hours.**

db.courses.find(

{ tag: “Databases”,

published: { $gte: new Date('2019-01-01'), $lt: new Date('2020-01-01') },

‘details.hour\_length’: { $lt: 10 } },

{“title”:1, "category":1, “price”:1, “\_id”:0} )

**Considering only courses in the category Computer Science published in the year 2020, for each tag, select the average price and the maximum number of enrolled students.**

db.courses.aggregate([

{$match: {“published”: {$gte: new Date(‘2020-01-01’), $lt: new Date('2021-01-01')} },

“category”: “Computer Science” },

{ $unwind: ‘$tags’},

{$group:

{ '\_id': ‘$tags’, 'avg\_price': {'$avg': '$price'} 'max\_students': {'$max': '$enrolled\_students’} } } ])

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

{ "address":

{"building":"768",

"coord":[-73.9685872,40.7679509],

"street":"Madison Avenue",

"zipcode":"10065",

"borough":"Manhattan",

"city": "New York"},

"sold\_items": ["Smartphones", "PC", "TV"],

"reviews":[

{"date": {date:{"2019-11-05"}}, "score":10, "description": "Lorem ipsum"},

{"date": {date("2020-02-21"}}, "score":8, "description": "Lorem ipsum"}     ],

"name":"Elettronic-store" }

**Select all the shops located in Rome that sell smartphones or TV and received at least one review with a score greater than 8. Show only the name, the street and the building**

db.shops.find(

{sold\_items:{$in: ["Electronics", "Home"]},

 "address.city":"Rome",

"reviews.score":{$gt:8} },

{\_id:0, name: 1, "address.street":1,"address.building":1 })

**For each city, compute the average and the maximum review score.  Show only the first 10 cities with the highest number of reviews**

db.collection.aggregate([

{$unwind: "$reviews"},

{$group: { \_id: "$address.city",

'countReviews': {$sum: 1},

'maxReviewScore': {$max: '$reviews.score'}, 'avgReviewScore': {$avg: '$reviews.score'} }},

{$sort: {countReviews: -1} },

{$limit: 10} ])

Given the following document structure, representing the measurements received by **sensors**, where each document collects the measures received in one day: {"\_id":ObjectId("5553a998e4b02cf7151190b8"),

"start": Date("2021-02-01T00:00:00.000Z"),

"end": Date("2021-02-01T23:00:00.000Z"),

“sensor”:

{ “\_id”: 1000,

"position":{"type":"Point","coordinates":[-47.9,47.6]},

"elevation":200, “city”: “Turin”, “country”: “Italy” },

"temperature":[

{ts: Date("2021-02-01T00:00:00.000Z"), value: 12},

{ts: Date("2021-02-01T01:00:00.000Z"), value: 11}, …

{ts: Date("2021-02-01T23:00:00.000Z"), value: 9} ],

nTemp: 24, // total number of elements in the temperature list sumTemp: 372 //

**sum of the values of all elements in the temperature list }**

**Update the document of the sensor with "\_id" equal to 1000 by adding a new "temperature" measurement with "value" 16 received at the timestamp "ts" 2021-02- 02T01:10:00.000Z.**

**Also concurrently update the corresponding statistics (i.e., "nTemp" and "sumTemp"). Suppose that the document with "start" attribute equal to "2021-02-02" exists. N.B. Use the syntax new Date (string) to manage date attributes, e.g., "start": new Date("2021-02-02")**

db.measures.updateOne(

{ 'sensor.\_id': 1000,

'start': new Date("2021-02-02"), },

{ $inc:{ nTemp: 1, sumTemp: 16},

$push: { temperature: { ts: new Date(“2021-02-20 10:00”), value: 16}} })

**Considering the sensor located in Italy and the measures received in the month of January 2021, show the sensor id, sensor city and the date in which the average measure of the sensor was greater than or equal to 15.**

db.measures.aggregate([

{$match: { “sensor.country”: "Italy",

“start”: {$gte: new Date (“2021-01-01”)},

“start”: {$lte: new Date (“2021-01-31”)},

{$addFields: { avg: { $divide: [“$tot”, “$n” ]} }},

{ $match : {avg: {$gte: 15}}},

{$project: { “sensor.\_id”: 1, “sensor.city”: 1, start: 1}} ])

The following document structure represents cameras sold by an e-commerce. Each document collects the aggregated metrics of one day.

{"\_id": "nikon\_d3500",

"model": "D3500",

"brand":{ "name": "Nikon",

"url": "https://www.nikon.it/" }

"releaseDate": Date("2018-08-28T00:00:00.000Z"),

"category": "DSRL",

"price": 435,

"specs":{ "resolution": 24,

"technology": "APS-C CMOS",

"min\_ISO": 100,

"max\_ISO": 25600,

"weight": 365,

"viewfinder": "optical",

"video\_resolution": "1920 x 1080" },

"scores": { "overall": 57,

"image\_quality": 48,

"versatility": 62,

"comfort": 85,

"speed": 41 } }

**Write a MongoDB query to display only the model, the price, and the brand name of cameras released in 2021, belonging to the “laser” category, and whose overall score is in the 70-90 range.**

db.cameras.find( { category: ‘laser',

releaseDate: { $gte: new Date('2021-01-01'),

$lt: new Date('2022-01-01') }, ‘scores.overall’: { $gte: 70, $lte: 90 } },

{model:1, "brand.name":1, price:1, \_id:0} )

**Considering only cameras released since 2015, for each release year and for each category, select the median overall score. N.B. Use the operator $year to extract the year from the date, e.g., $year: “$releaseDate”**

db.measures.aggregate([

{$match:

{releaseDate: {$gte: new Date(‘2015-01-01’)} } },

{$sort: {‘$scores.overall’: 1} },

{$group: { '\_id': { 'cat': '$category',

'y': { $year: "$releaseDate" } },

'value': {'$push': '$scores.overall'} } },

{$project: { \_id: 1, "median": { $arrayElemAt: ["$value",

{ $floor: { $multiply: [0.50, {$size: "$value"}] } }] } ])

document of a set of events measured by a sensor.

{"\_id": ObjectId(“xyz”),

"sensor":{ “id”: 1,

“location”:{ “building”: “A”

“floor”: 1,

“type”: “bedroom” } },

"start”: Date("2022-01-15T00:00:00.000Z"),

“measures : [

{“ts”: Date("2022-01-15T01:59:59.000Z"), “temperature”: 21},

{“ts”: Date("2022-01-15T23:59:59.000Z"), “temperature”: 21.5} ],

“n”: 2,

“sum\_temp”: 42.5 }

**Write a MongoDB query to insert a new measure for the sensor 5 acquired on 2021-12- 01 at 08:00 with a temperature equal to 19. The document to be updated is related to the sensor 5 and to the set of measures of the day 2021-12-01 (start attribute). Increase also the statistics (i.e., attributes “n” and “sum\_temp”) stored in the document, which describe the number of measurements and the sum of the temperatures.**

db.timeserie.updateOne (

{'sensor.id': 5,

'start': new Date(2021-12-01T00:00:00.000Z) },

{'$push': {'measures':{ts: new Date("2021-12-01T08:00:00.000Z"),

“temperature”: 19 } },

'$inc':{'n':1, ‘sum\_temp’: 19}, })

**Considering only measurements acquired in June 2021 by sensors located at floor 2, for each building, select the average and the maximum temperature.**

db.collection\_name.aggregate([

{$match: { “sensor.location.floor”: 2,

“start”: { $gte: new Date(‘2021-06-01’), $lte: new Date('2021-06-30')} } },

{ $unwind: ‘$measures’},

{$group: { '\_id': ‘$sensor.location.building’,

'avg\_temp': {'$avg': '$measures.temperature'},

'max\_temp': {'$max': '$measures.temperature'} } } ])