1. What is the comparison order that MongoDB uses for type checking of the following BSON types? [Highest to lowest]

(A) Regular Expression

(B) Numbers

(C) ObjectId

(D) Date

A,C,D,B

D,A,B,C

A,D,C,B

C,A,D,B ....correct

2.In "albums" collection, identify the option that retrieves the details of the albums having likes greater than 150 but less than 400 [exclusive of 150 and 400]

1.db.albums.find(

{likes: { $gte: 150), likes: { $ite: 400 } }

)

2.db.albums.find(likes: {$gt: 150, $lt: 400)

3.db.albums.find({likes: ($gt: 150), likes: {$lt: 400}}

)

4.db.albums.find({likes: {$gt: 150, $1t: 400}}...................................co

)

3. What is the MongoDB equivalent of the following SQL query [Choose all the apply]

SELECT * FROM kgroup

WHERE age > 30 AND age < 70

1.db.kgroup.find({age: { $gte: 30, $lte: 70}}

)

2.db.kgroup.find({$and: { age: { $gte: 30}},{age: ($lte: 70}}}

)

3.db.kgroup.find({[$and: { age: { $gte: 30}},{age: { $lte: 70}}]}

4.db.kgroup.find({age: { $gt: 30, $1t: 70}}.............................co

)

4. What does the following query do when performed on the poems collection?

db.poems.update({_id:1), {Title: "Where The Mind Is Without Fear"})

Replaces the complete document with _id as 1, with the document specified in

1.second parameter

2.Syntax error..co

3.Updating a document is possible only with $set parameter

4.Updates the Title field of the poems collection

5. For a given poems collection, chose the option that retrieves the details of the poems with likes greater than 100 but less than 200[exclusive of 100 and 200]

1.db.poems.find({likes: {$gte: 100), likes: {$lte: 200}});

2.db.poems.find(likes:($gt: 100, $lt: 200 });

3.db.poems.find({likes: ($gt: 100), likes: { $it: 200}});

4.db.poems.find({likes: {$gt: 100, $lt: 200}}); ....co

6. What is the MongoDB equivalent of the below SQL query? [Choose all the apply]

SELECT count(*) FROM kgroup;

1.db.kgroup.find(

{count:1}

)

2.db.kgroup.count(

{*}

)

3.db.kgroup.count().....................more co

4.db.kgroup.find().count()................

7.Assume we have orders collection with the following documents:

{"_id": 1, "desc": "One line" }

{"_id": 2, "desc": "Line Line 2")

{"_id": 3, "desc": "Many figures connect one point to another to form a line" }

{"_id":4,"desc": "Multipleline connecting together")

Identify the command that produces the below output:

{"_id": 3, "desc": "Many figures connect one point to another to form a line" }
{"_id":4,"desc": "Multipleline connecting together" }

1.db.orders.find({ desc: { $regex: /m.*line/, $options: 'i' } })......................co

2.db.orders.find({ desc: { $regex: /m.*line/, $options: 'si' } })

3.db.orders.find({ desc: { $regex: /m*line/, $options: 'sxi' }})

4.db.orders.find({ desc: { $regex: /m.*line/, $options: 'xi" } })

1. Given "stock" collection with following documents:

db.stock.find()

{item: "Journal", Instock: [{warehouse:"A",qty: 5}, (warehouse: "C", qty: 15)]},

{item:"notebook",instok:[{warehouse:"C",qty:5}]},

{item: "paper", instock:[{werehouse:"A",qty:60},{werehouse:"B" qty: 15}]},

{item: "planner", instock:[{werehouse:"A",qty:40},{werehouse:"B" qty: 5}]},

(item: "postcard", Instock:[{werehouse:"B",qty: 15},{werehouse:"C" qty: 35}]},

Choose the query that selects all documents where the Instack has its first element as a document having qty less than or equal to 25:

1.db.stock.find(

{"instock.0.qty":{$lte: 25}}.......................co

)

2.db.stock.find({"instock.1.qty": {$lte: 25}}

)

3.db.stock.find({"instock.qty.0":{$lte: 25}}

)

4.db.stock.find({"instock.qty.1":{$lte: 25}}

)

2.Which commands match the documents containing array that satisfy two criteria., i.e, some of the array elements are greater than 91 and some of the array elements of the same document are less than 49?

```
db.marks.insert({id: 1, score: [ 81, 89, 90, 72 ] })

db.marks.insert({id: 2, score: [ 69, 64, 93, 85 ] })

db.marks.insert({ id: 3, score: [ 48, 75, 97, 85 ] })

db.marks.insert( {_id : 4, score: [40, 55, 90, 95 ] })
```

```
1.db.marks.find( { score: { $gt: 91, $1t: 49}})

2.db.marks.find( { $all: { score:[$gt: 91, $lt: 49 ] } })

3.db.marks.find( { score:{ $eleMatch: { $gt: 91, $lt: 49} } })...........................co

4.db.marks.find( { $all: { score:{ $gt: 91, $1t: 49 } } })
```

3. Which of the query below is used to retrieve the top 3 marks after an update is performed?

1.db.emp.update({_id:1}, {$push:{marks: {$each: [80, 89, 90, 72], $slice:3}}})

2.db.emp.update({_id:1}, {$push: { marks: {$each: [80, 89, 90, 72],$slice: -3}}})

3.db.emp.update({_id:1}, { $push: { marks: { $each: [80, 89, 90, 72], $sort:{ marks: 1), $slice: -3}}})................co

4.db.emp.update({_id:1}, { $push: { marks: { $each: [80, 89, 90, 72], $sort: 1, $slice: -3}}})

4. Consider the following documents in the employee collection

{_id:1, skills: ['AI', 'OS', 'Bigdata'] }

{_id:2, skills: ['Datascience', 'java', 'python]}

{_id:3, skills: ['AWS', 'SQL', 'GoogleCloud']}

{_id:4, skills: ['AI', 'SQL', 'GoogleCloud']}

{_id:5, skills: ['Datascience', 'GoogleCloud']}

Which of the following command updates an element without specifying its position???

1.db.employee.update({_id:1, 'skills":"os"}, {$set:{'skills': 'OpenSource' }})

2.db.employee.update({_id:1, 'skills': 'os'}, {$set:{'skills.$':'OpenSource' }}).........................co

3.db.employee.update({_id:1, 'skills': 'OS'}, {$set:{'$skills': 'OpenSource" }})

4.db.employee.update({_id:1, 'skills': 'OS'}, {$set: 'OpenSource' })

1.The output for explain plan for the following query:

db.employee.find({ename: "AAA"),{ename:1,_id:0}).explain("executionStats") looks as follows:

How can we say from the execution stats that it is a covered query?

```
{
"queryPlanner":{
"plannerVersion":1,
"namespace": "sales1.employee",
.........
},
"winningPlan" :(
"stage": "PROJECTION",
............
"executionStats":{
"executionSuccess":true,
"nReturned": 1, "executionTimeMillis":0,
"totalKeysExamined": 1,
"totalDocsExamined":0,
"executionStages":
{ "stage":"PROJECTION",".................
```

1.Because there is a "WinningPlan" with stage "PROJECTION"

2.Because executionTimeMillis is 0

3.Because totalKeysExamined is 1, while executionTimeMillis is 0, the query need not even execute, and has all the results already cached

4.Because totalDocsExamined is 0, while totalKeysExamined is 1, the query never went to the cellection, and is fully satisfied with the index itself....................co

2. An explain plan can tell you which of the following?

(A) How many documents a query examines

(B) what indexes a query uses

(C) what index you should build that you don't already have

(D) How many documents a query returned

(E) How many page faults occurred for the query

(A), (B) and (D)..co

(A) and (D)

(A) and (B)

(A), (B), (D) and (E)

All the given options

1.Consider "albums" collection in mongodb. What is the functionality of the below MongoDB query?

```
db.albums.aggregate([ {$group: {_id: null, count: { $sum: 1}}}]);
```

Counts all the documents from "albums" collection..............................co

Finds the total sum of count field group by _id field

Counts all the documents from "albums" collection having _id value as null

Throws error message