Sample MongoDB Associate Developer Exam

Điểm: 35/53

✓ **Đúng** 1/1 Điểm

1. Given the following documents in a collection:

   {_id: 1, txt: "just some text"}
   {_id: 2, txt: "just some text"}

   Which two documents can successfully be added in the same collection?

   (Choose 2)

   - [✓] { _id: 0, txt: "just some text" }
   - [ ] { _id: 1, txt: "just some text" }
   - [ ] { _id: [4], txt: "just some text"}
   - [✓] { _id: 3, txt: "just some text" }

✗ **Không chính xác** 0/1 Điểm

2. Given the following documents in a collection:

{ _id: 1, n: [1,2,5], p: 0.75, c: 'Green' }
{ _id: 2, n: 'Orange', p: 'Blue', c: 42, q: 14 }
{ _id: 3, n: [1,3,7], p: 0.75, c: 'Orange' }

Which two documents can successfully be added in the same collection?

(Choose 2)

- [✓] { _id: 1, n: [1,2,5], p: 0.75, c: 'Green' }
- [✓] { _id: 5, n: [1,2,5], p: 0.75, c: 'Green' }
- [ ] { _id: 2, n: [1,2,5], p: 0.75, c: 'Green' }
- [ ] { _id: 6, n: [1,3,7], p: 0.85, c: 'Orange }

✕ **Không chính xác** 0/1 Điểm

3. Given the following documents in a collection:

{ _id: 1, n: [1,2,5], p: 0.75, c: 'Green' }
{ _id: 2, n: 'Orange', p: 'Blue', c: 42, q: 14 }
{ _id: 3, n: [1,3,7], p: 0.55, c: 'Orange' }

Suppose we have created a unique index at field p through:

db.collection.createIndex({ p: 1 }, { unique: true })

Which two documents can successfully be added in the same collection?

(Choose 2)

- [✓] { _id: 3, n: [1,2,5], p: 'Orange', c: 'Green' }
- [ ] { _id: 4, n: [1,2,5], p: 0.75, c: 'Green' }
- [✓] { _id: 5, n: [1,2,5], p: 0.85, c: 'Green' }

☐ { _id: 6, n: [1,3,7], p: 'Green', c: 'Orange }

✓ **Đúng** 1/1 Điểm

4. Suppose you have a developers collection with the following document structure:

```
{
    _id: 1,
    fname: 'Bob',
    lname: 'Smith',
    tech_stack: [ 'git', 'c++', 'sqlite', 'linux' ]
}
```

Select the query that can be executed without errors.

(Choose 1)

○ db.developers.updateOne({ _id: 1 }, { $unset: { _id: '' } })

○ db.developers.updateOne({ _id: 1 }, { $set: { _id: 3 } })

○ db.developers.updateOne({ _id: 1 }, { $set: { _id: 3, fname: 'Bob' } })

[✓ db.developers.updateOne({ _id: 1 }, { $set: { fname: 'John' } })]

✗ **Không chính xác** 0/1 Điểm

5. Given the following sample documents in products collection:

{ "name" : "XPhone", "price" : 799, "color" : ["white", "black"], "storage" : [64, 128, 256] }
{ "name" : "XPad", "price" : 899, "color" : ["white", "black", "purple"], "storage" : [128, 256, 512] }
{ "name" : "GTablet", "price" : 899, "color" : ["blue"], "storage" : [16, 64, 128] }
{ "name" : "GPad", "price" : 699, "color" : ["white", "orange", "gold", "gray"], "storage" : [128, 256, 1024] }
{ "name" : "GPhone", "price" : 599, "color" : ["white", "orange", "gold", "gray"], "storage" : [128, 256, 512] }

Given the following query:

```
db.products.find( { $and : [ { "price" : { $lte : 800 } },
                             { $or : [ { "color" : "purple" }, { "storage" : 1024 } ] } ] } )
```

What is the correct output of the query?

(Choose 1)

[✓ { "name" : "XPhone", "price" : 799, "color" : ["white", "black"], "storage" : [64, 128, 256] }]

{ "name" : "XPad", "price" : 899, "color" : ["white", "black", "purple"], "storage" : [128, 256, 512] }

{ "name" : "GTablet", "price" : 899, "color" : ["blue"], "storage" : [16, 64, 128] }

{ "name" : "GPad", "price" : 699, "color" : ["white", "orange", "gold", "gray"], "storage" : [128, 256, 1024] }

✓ **Đúng** 1/1 Điểm

6. A collection has documents like the following:

```
{ _id: 1, name: 'Oatmeal Fruit Cake with Gummy Bears ', price: 11 },
{ _id: 2, name: 'Cheesecake Trifle with Chocolate Sprinkles ', price: 14 },
{ _id: 3, name: 'Pistachio Brownie with Walnuts ', price: 5},
{ _id: 4, name: 'Strawberry Ice Cream Cake with Butterscotch Syrup ', price: 3 }
```

Which tokenizer can be used to create an autocomplete index to look for matches at the beginning of a word on the name field?

(Choose 1)

regexCaptureGroup

[✓ edgeGram]

nGram

○ matchNGram

✗ **Không chính xác** 0/1 Điểm

7. There is a gamers collection in your database with the following document structure:

{ _id: 1, level: 15, is_active: true }
{ _id: 2, level: 14, is_active: true }
{ _id: 3, level: 7, is_active: false }

How do you update the value of the level field to 20 for a player with an id = 2?
Expected result:

{ _id: 1, level: 15, is_active: true }
{ _id: 2, level: 20, is_active: true }
{ _id: 3, level: 7, is_active: false }

(Choose 1)

○ db.gamers.updateOne({ _id: 2 }, { $unset: { level: 20 } })

○ db.gamers.updateOne({ _id: 2 }, { $inc: { level: 20 } })

○ db.gamers.updateOne({ _id: 2 }, { level: 20 })

○ db.gamers.updateOne({ _id: 2 }, { $set: { level: 20 } })

✗ **Không chính xác** 0/1 Điểm

8. Which two statements are used to insert a field of type date?

(Choose 2)

✓ db.collection.insertOne({ date: new Date("2022-01-01T00:00:00Z") })

✓ db.collection.insertOne({ date: "2022-01-01T00:00:00Z" })

☐ db.collection.insertOne({ date: ISODate("2022-01-01T00:00:00Z") })

☐ db.collection.insertOne({ date: 2022-01-01T00:00:00Z })

✗ **Không chính xác** 0/1 Điểm

9. Given the following documents in a collection:

{ _id: 1, a: 1, b: 2, c: 3, d: 4}
{ _id: 2, a: 4, b: 5, c: 4}
{ _id: 3, a: 6, b: 7, c: 5}

What is the output after executed the statement:

db.collection.replaceOne({ a: { $gte: 1 } }, { a: 4, b: 5, c: 6 });

(Choose 1)

○ { _id: 1, a: 4, b: 5, c: 6 }, { _id: 2, a: 4, b: 5, c: 4 }, { _id: 3, a: 6, b: 7, c: 5 }

○ { _id: 1, a: 4, b: 5, c: 6 }, { _id: 2, a: 4, b: 5, c: 6 }, { _id: 3, a: 6, b: 7, c: 5 }

[✓ { _id: 1, a: 4, b: 5, c: 6, d: 4 }, { _id: 2, a: 4, b: 5, c: 6 }, { _id: 3, a: 6, b: 7, c: 5 }]

○ { _id: 1, a: 1, b: 2, c: 3, d: 4}, { _id: 2, a: 4, b: 5, c: 4}, { _id: 3, a: 6, b: 7, c: 5}

✓ **Đúng** 1/1 Điểm

10. When there are many transactions running on the same collection, what method should we use to safely update a document?

(Choose 1)

○ updateOne

○ updateOneSafely

[✓ findOneAndUpdate]

○ updateOneSecurely

✓ **Đúng** 1/1 Điểm

11. What is the result of the function findOneAndDelete?

(Choose 1)

○ There is no such method

[✓ Deleted document]

○ The number of documents that were deleted

○ Wrong syntax

✓ **Đúng** 1/1 Điểm

12. Given the following documents in a collection:

```
{ _id: 6305, name : "A. MacDyver", "assignment" : 5, "points" : 24 }
{ _id: 6308, name : "B. Batlock", "assignment" : 3, "points" : 22 }
{ _id: 6312, name : "M. Tagnum", "assignment" : 5, "points" : 30 }
{ _id: 6319, name : "R. Stiles", "assignment" : 2, "points" : 12 }
{ _id: 6322, name : "A. MacDyver", "assignment" : 2, "points" : 14 }
{ _id: 6234, name : "R. Stiles", "assignment" : 1, "points" : 10 }
```

What is the result after executed db.scores.findOneAndDelete({ "name" : "A. MacDyver" }, { sort : { "points" : 1 } })

(Choose 1)

○ { _id: 6305, name : "A. MacDyver", "assignment" : 5, "points" : 24 }

[✓ { _id: 6322, name: "A. MacDyver", "assignment" : 2, "points" : 14 }]

○ { _id: 6234, name : "R. Stiles", "assignment" : 1, "points" : 10 }

○ { _id: 6319, name : "R. Stiles", "assignment" : 2, "points" : 12 }

✗ **Không chính xác** 0/1 Điểm

13. Given the following documents in a collection:

{ _id: 1, name: "John", age: 25 }
{ _id: 2, name: "Alice", age: 30 }
{ _id: 3, name: "Bob", age: 35 }

What is the result after executed

db.collection.updateOne({ _id: 4 }, { $set: { name: "Eve", age: 40 } }, { upsert: true });

(Choose 1)

○ { _id: 1, name: "Eve", age: 40 }, { _id: 2, name: "Alice", age: 30 }, { _id: 3, name: "Bob", age: 35 }

○ { _id: 1, name: "John", age: 25 }, { _id: 2, name: "Alice", age: 30 }, { _id: 3, name: "Bob", age: 35 }

○ { _id: 1, name: "John", age: 25 }, { _id: 2, name: "Alice", age: 30 }, { _id: 3, name: "Bob", age: 35 }, { _id: 4, name: "Eve", age: 40 }

[✓ { _id: 4, name: "Eve", age: 40 }, { _id: 2, name: "Alice", age: 30 }, { _id: 3, name: "Bob", age: 35 }]

✓ **Đúng** 1/1 Điểm

14. What is the correct syntax to insert multiple documents into a MongoDB collection named sales using the bulk write operation?

(Choose 1)

[✓ db.collection.bulkWrite([{ insertOne: { document: { _id: 3, type: "beef", size: "medium", price: 6 } } }, { insertOne: { document: { _id: 4, type: "sausage", size: "large", price: 10 } } }])]

- ○ db.collection.bulkWrite({ insertMany: [{ document: { _id: 3, type: "beef", size: "medium", price: 6 } }, { document: { _id: 4, type: "sausage", size: "large", price: 10 } }] })
- ○ db.collection.insertMany([{ document: { _id: 3, type: "beef", size: "medium", price: 6 } }, { document: { _id: 4, type: "sausage", size: "large", price: 10 } }], { bulkWrite: true})
- ○ db.collection.insertMultiple([{ document: { _id: 3, type: "beef", size: "medium", price: 6 } }, { document: { _id: 4, type: "sausage", size: "large", price: 10 } }])

✕ **Không chính xác** 0/1 Điểm

15. Given the following documents in collection 'first_collection':

{ _id: 1, name: "John", age: 25 }
{ _id: 2, name: "Alice", age: 30 }
{ _id: 3, name: "Bob", age: 35 }

Given the following documents in collection 'result':

{ _id: 4, name: "Eve", age: 40 }
{ _id: 5, name: "Charlie", age: 45 }

How do the collection 'result' look like after executed

db.first_collection.aggregate([{ $match: { age: { $gte: 30 } } },{ $out: "result" }])

(Choose 1)

- ○ { _id: 1, name: "John", age: 25 }, { _id: 2, name: "Alice", age: 30 }, { _id: 3, name: "Bob", age: 35 }
- [✓ { _id: 4, name: "Eve", age: 40 }, { _id: 5, name: "Charlie", age: 45 }]
- ○ { _id: 4, name: "Eve", age: 40 }, { _id: 5, name: "Charlie", age: 45 }, { _id: 2, name: "Alice", age: 30 }, { _id: 3, name: "Bob", age: 35 }
- ○ { _id: 2, name: "Alice", age: 30 }, { _id: 3, name: "Bob", age: 35 }

✗ **Không chính xác** 0/1 Điểm

16. Given the following documents in a collection:

{ _id: 1, name: "John", age: 25 }
{ _id: 2, name: "Alice", age: 30 }
{ _id: 3, name: "Bob", age: 40 }
{ _id: 5, name: "Charlie", age: 45 }

What is the output of

db.collection.find({ age: { $in: [25, 40] } });

(Choose 2)

- [✓ { _id: 1, name: "John", age: 25 }]
- [✓ { _id: 2, name: "Alice", age: 30 }]
- { _id: 3, name: "Bob", age: 40 }
- { _id: 5, name: "Charlie", age: 45 }

✓ **Đúng** 1/1 Điểm

17. Which of the following commands will delete a collection named restaurants?

(Choose 1)

- db.restaurants.dropCollection()
- db.restaurants.delete()
- [✓ db.restaurants.drop()]
- db.restaurants.remove()

✓ **Đúng** 1/1 Điểm

18. You have a developers collection with the following document structure:

```
{
   _id: 1,
   fname: 'John',
   lname: 'Smith',
   tech_stack: ['sql', 'git', 'python', 'linux', 'django', 'aws']
}

{
   _id: 2,
   fname: 'Michael',
   lname: 'Doe',
   tech_stack: ['git', 'python', 'sqlite', 'linux', 'flask']
}
```

Which of the following queries will return only the first three elements of the array in the tech_stack field ?

(Choose 1)

- ○ db.developers.find({ tech_stack: { $slice: [0, 3] } })
- ○ db.developers.find({}, { tech_stack: [0, 1, 2] })
- [✓ db.developers.find({}, { tech_stack: { $slice: [0, 3] } })]
- ○ db.developers.find({}, { tech_stack: [0, 3] })

✓ **Đúng** 1/1 Điểm

19. We have a movies collection with the following document structure:

```
{
   _id: ObjectId("573a1390f29313caabcd60e4"),
   title: 'The Immigrant',
   released: ISODate("1917-06-17T00:00:00.000Z"),
    rated: 'UNRATED',
    year: 1917,
    imdb: { rating: 7.8, votes: 4680, id: 8133 }
}
```

We need to filter those movies where the imdb rating is greater then 7. Which query should we use?

(Choose 1)

- [● db.movies.find({ "imdb.rating": { "$gt": 7 } })]
- ○ db.movies.find({ "imdb.rating": { "$lt": 7 } })
- ○ db.movies.find({ "imdb.rating": { "$gte": 7 } })
- ○ db.movies.find({ imdb.rating: { "$gt": 7 } })

✓ **Đúng** 1/1 Điểm

20. There is a collection named products in MongoDB database. Your coworker wants to know how many products are in this collection (number of documents in the collection). Which query should you use? Select all correct answers.

(Choose 2)

- ☐ db.products.aggregate()
- [✓ db.products.countDocuments()]
- ☐ db.products.total()
- [✓ db.products.count()]

✓ **Đúng** 1/1 Điểm

21. We have a movies collection with the following document structure:

```
{
   _id: ObjectId("573a1390f29313caabcd6223"),
   genres: [ 'Comedy', 'Drama', 'Family' ],
   title: 'The Poor Little Rich Girl',
   released: ISODate("1917-03-05T00:00:00.000Z"),
   year: 1917,
   imdb: { rating: 6.9, votes: 884, id: 8443 }
}
```

We need to use Aggregation Framework to fetch all movies from this collection where 'Drama' is in genres list and the minimum 'imdb.votes' is at least 100.
Additionally, in the projection stage, we want to leave only the following fields:
-> title
-> genres
-> imdb.votes

Example output:

```
{
   title: 'Miss Lulu Bett',
   genres: [ 'Comedy', 'Drama' ],
   imdb: { votes: 204 }
}
```

Which pipeline should you use?

(Choose 1)

- ○ [{ $match: { genres: { $nin: ['Drama'] }, 'imdb.votes': { $gte: 100} } }]
- ○ [{ $match: { genres: { $in: ['Drama'] } } }, { $project: { _id: 0, title: 1, genres: 1, 'imdb.votes': 1} }]
- ○ [{ $project: { _id: 0, title: 1, genres: 1, 'imdb.votes': 1 } }]
- [✓ [{ $match: { genres: { $in: ['Drama'] }, 'imdb.votes': { $gte: 100 } } }, { $project: { _id: 0, title: 1, genres: 1, 'imdb.votes': 1 } }]]

✓ **Đúng** 1/1 Điểm

22. What is the correct syntax to perform a $lookup operation in MongoDB to combine documents from two collections, employees and departments, based on the "department_id" field in the employees collection and the "_id" field in the departments collection?

(Choose 1)

- ○ db.employees.aggregate({ $lookup: { from: "departments", localField: "department_id", foreignField: "_id", as: "department_info" } })
- [✓ db.employees.aggregate([{ $lookup: { from: "departments", localField: "department_id", foreignField: "_id", as: "department_info" } }])]
- ○ db.aggregate([{ $lookup: { from: "employees", to: "departments", on: "department_id", as: "department_info"} }])
- ○ db.aggregate({ $lookup: { from: "employees", to: "departments", on: "department_id", as: "department_info" } })

✕ **Không chính xác** 0/1 Điểm

23. Consider a MongoDB database containing a collection of documents representing product information for an e-commerce website. The documents have the following structure:

```
{
   "_id": ObjectId("5f95a1d11a12b400001b75c0"),
   "product_name": "Smartphone",
   "brand": "Apple",
   "price": 800,
   "categories": [
      "Electronics",
      "Smartphones"
   ],
   "reviews": [
      {
         "username": "user1",
         "rating": 4,
         "comment": "Great product!"
      },
      {
            "username": "user2",
            "rating": 5,
            "comment": "Excellent!"
        },
        {
            "username": "user3",
            "rating": 3,
            "comment": "Good but overpriced."
        }
    ]
}
```

Select the MongoDB aggregation pipeline that returns the average rating of all products grouped by brand. The result should include only brands with an average rating greater than or equal to 4. The output should have the following format:

```
{
    "brand": "Apple",
    "avg_rating": 4.33
}
```

(Choose 1)

- ○ [{ "$unwind": "$reviews" }, { "$group": { "_id": "$brand", "avg_rating": { "$avg": "$reviews.rating" } } }, { "$match": { "avg_rating": { "$gte": 4 } } }, { "$project": { "brand": "$_id", "avg_rating": "$avg_rating", "_id": 0 } }]
- ○ [{ "$unwind": "$reviews" }, { "$group": { "_id": "$brand", "avg_rating": { "$avg": "$reviews.rating" } } }, { "$match": { "avg_rating": { "$gte": 4 } } }]
- [✓ [{ "$group": { "_id": "$brand", "avg_rating": { "$avg": "$reviews.rating" } } }, { "$match": { "avg_rating": { "$gte": 4 } } }, { "$project": { "brand": "$_id", "avg_rating": "$avg_rating", "_id": 0 } }]]
- ○ [{ "$unwind": "$reviews" }, { "$group": { "_id": "$brand", "avg_rating": { "$avg": "$reviews.rating" } } }, { "$project": { "brand": "$_id", "avg_rating": "$avg_rating", "_id": 0 } }]

✕ **Không chính xác** 0/1 Điểm

24. Consider a MongoDB collection named employees with the following document structure:

```
{
   "_id" : ObjectId("5f7f39d8dbdfgbcabcabcabc"),
   "name" : "John Doe",
   "age" : 32,
   "position" : "Manager",
   "department" : "Sales",
   "salary" : 60000
}
```

You are required to find all employees who earn a salary greater than or equal to $60,000 and who work in the "Sales" department.Additionally, you are required to retrieve the average salary for all employees in the "Sales" department.Which of the following queries would perform the most efficient collection scan ?

- [✓ db.employees.find({ department: "Sales", salary: { $gte: 60000 } }).forEach(function (doc) { // process the documents });]
- db.employees.aggregate([{ $match: { department: "Sales", salary: { $gte: 60000 } } }, { $group: { _id: "$department", avg_salary: { $avg: "$salary" } } }]);
- db.employees.aggregate([{ $match: { department: "Sales" } }, { $group: { _id: "$department", avg_salary: { $avg: "$salary" } } }, { $project: { _id: 0, avg_salary: { $gte: ["$avg_salary", 60000] } } }]);
- db.employees.aggregate([{ $match: { department: "Sales" } }, { $group: { _id: "$department", avg_salary: { $avg: "$salary" } } }, { $match: { avg_salary: { $gte: 60000 } } }]);

✓ **Đúng** 1/1 Điểm

25. Suppose we have a document:

```
{
   "_id": ObjectId("5f7f39d8dbdfgbcabcabcabc"),
   "a": 1
}
```

What is the result after the operation:

```
db.users.updateOne(
   { "_id": ObjectId("5f7f39d8dbdfgbcabcabcabc") },
```

{ $set: { "b.c": "value" } }
)

(Choose 1)

- ○ { "_id": ObjectId("5f7f39d8dbdfgbcabcabcabc"), "a": 1, "b.c": "value" }
- [✓ { "_id": ObjectId("5f7f39d8dbdfgbcabcabcabc"), "a": 1, "b": { "c": "value } }]
- ○ { "_id": ObjectId("5f7f39d8dbdfgbcabcabcabc"), "a": 1 }
- ○ Statement failed to execute.

✓ **Đúng** 1/1 Điểm

26. Given the following documents in a collection:

{ "_id": 1, "name": "pen", "color": "red" }
{ "_id": 2, "name": "eraser", "color": ["red", "blue"] }
{ "_id": 3, "name": "ruler", "color": "blue" }

What is the output of

db.collection.find({ $or: [{ name: "pen" }, { color: "red" }] }, { _id: 0, name: 1 })

(Choose 2)

- [✓ { name: "pen" }]
- [✓ { name: "eraser" }]
- ☐ { name: "ruler" }
- ☐ Nothing returned

✗ **Không chính xác** 0/1 Điểm

27. Given the following documents in a collection

```
{
    "_id": 1,
    "name": "The Sun Rises",
    "genre": "horror",
    "reviews": {
        "reviewer": "Amelia",
        "score": 7.3
    }
}

{
    "_id": 2,
    "name": "Nothing in this world",
    "genre": "comedy",
    "reviews": [
        {
            "reviewer": "James",
            "score": 5
        }
    ]
}

{
    "_id": 3,
    "name": "Number the Star",
    "genre": "horror",
    "reviews": [
        {
            "reviewer": "Smith",
            "score": 5
        },
        {
            "reviewer": "Jack",
            "score": 8
        }
    ]
}

{
    "_id": 4,
    "name": "The movement",
    "genre": "comedy",
  "reviews": [
    {
      "reviewer": "James",
      "score": 7
    }
  ]
}
```

What is the output of db.collection.find({ genre: { $in: ["comedy", "horror"] }, reviews: { $elemMatch: { score: { $gte: 7 } } } })

(Choose 2)

- [✓] { "_id": 1, "name": "The Sun Rises", "genre": "horror", "reviews": { "reviewer": "Amelia", "score": 7.3 }}
- [ ] { "_id": 2, "name": "Nothing in this world", "genre": "comedy", "reviews": [{ "reviewer": "James", "score": 5 }]}
- [ ] { "_id": 3, "name": "Number the Star", "genre": "horror", "reviews": [{ "reviewer": "Smith", "score": 5 }, { "reviewer": "Jack", "score": 8 }]}
- [✓] { "_id": 4, "name": "The movement", "genre": "comedy", "reviews": [{ "reviewer": "James", "score": 7 }]}

✓ **Đúng** 1/1 Điểm

28. Which of the following rules (when ordering) should be followed when building query indexes?

(Choose 1)

- ○ Range, Sort, Equality
- ○ Equality, Range, Sort
- [✓ Equality, Sort, Range]
- ○ There is no specific rule.

✓ **Đúng** 1/1 Điểm

29. Which of the following methods can be used to create an index on MongoDB?

(Choose 1)

[✓ createIndex()]

addIndex()

index()

ensureIndex()

✓ **Đúng** 1/1 Điểm

30. Which method can be used to retrieve all the indexes of a collection in MongoDB?

(Choose 1)

[✓ getIndexes()]

findIndexes()

getAllIndexes()

retrieveIndexes()

✓ **Đúng** 1/1 Điểm

31. You have a collection named "books" that contains documents with the following structure:

```
{
  _id: ObjectId("5f9e7a1a2c9d440000f2e5a1"),
  title: "The Great Gatsby",
```

author: "F. Scott Fitzgerald",
year: 1925
}

You need to create a compound index on the "books" collection to optimize queries for sorting by author in ascending order and year in descending order. Which index definition should you use?

(Choose 1)

[✓ db.books.createIndex({ author: 1, year: -1 })]

db.books.createIndex({ author: -1, year: 1 })

db.books.createIndex({ author: -1 }, { year: -1 })

db.books.createIndex({ year: -1 }, { author: 1 })

✓ **Đúng** 1/1 Điểm

32. Which method is used to drop an index?

(Choose 1)

db.collection.clearIndex()

[✓ db.collection.dropIndex()]

db.collection.removeIndex()

db.collection.deleteIndex()

✓ **Đúng** 1/1 Điểm

33. We have the following index in a movies collection:

{ author: 1, genres: 1 }

We want to insert the following document:

```
{
   "title": "The Immigrant",
   "year": 1917,
   "author": [
      "James",
      "Smith"
   ],
   "genres": [
      "Short",
      "Comedy",
      "Drama"
   ]
}
```

How many index entries will be created?

(Choose 1)

- ○ 3
- ○ 4
- ○ 5
- [✓ 6]

× **Không chính xác** 0/1 Điểm

34. We have the following index in a movies collection:

```
{ title: 1, genres: 1 }
```

We want to insert the following document:

```
{
   "title": "The Immigrant",
   "year": 1917,
   "genres": [
      "Short",
        "Comedy",
        "Drama"
    ]
}
```

Select the appropriate index entries.

(Choose 1)

- ○ "The Immigrant" - "Comedy", "The Immigrant" - "Drama", "The Immigrant" - "Short"
- [✓ "The Immigrant" - "Short", "The Immigrant" - "Comedy", "The Immigrant" - "Drama"]
- ○ "Short" - "The Immigrant", "Comedy" - "The Immigrant", "Drama" - "The Immigrant"
- ○ "Short", "Comedy", "Drama"

✓ **Đúng** 1/1 Điểm

35. Which of the following queries can use an index on the "title" field? Select all correct answers.

(Choose 2)

- [✓ db.movies.find({ "title": "Titanic" })]
- ☐ db.movies.find({ "genres": "Drama", "type": "series" })
- ☐ db.movies.find()
- [✓ db.movies.find({ "title": "Death Note" })]

✗ **Không chính xác** 0/1 Điểm

36. We have a query:

```
db.cars.find(
{
    manufacturer: 'Ford',
    cost: { $gt: 15000 }
}).sort( { model: 1 } )
```

Which index should be created to optimize the query above?

(Choose 1)

- ○ { cost: 1, model: 1, manufacturer: 1 }
- ● { manufacturer: 1, cost: 1, model: 1 }
- ○ { manufacturer: 1, model: 1, cost: 1 }
- ○ { model: 1, manufacturer: 1, cost: 1 }

✓ **Đúng** 1/1 Điểm

37. We have the following schema for a movies collection:

```
{
    _id: ObjectId,
    title: String,
    genres: Array,
    languages: Array,
    year: 32-bit integer
}
```

And the following index on the movies collection:

{ title: 1 }

Which of the following queries will use the given index to perform the sorting stage? Select all correct answers.

(Choose 3)

- [✓] db.movies.find({}).sort({ title: 1 })
- [ ] db.movies.find({ genres: "Drama" }).sort({ year: 1 })

- [✓] db.movies.find({ genres: "Drama" }).sort({ title: 1 })
- [✓] db.movies.find({}).sort({ title: -1 })

✓ **Đúng** 1/1 Điểm

38. Create an compound index as below:

    db.collection.createIndex({ score: -1, username: 1 })

    Which of the following queries uses the created index effectively?

    (Choose 2)

- [ ] db.collection.find({ score: 1, username: 1 })
- [✓] db.collection.find({ score: -1, username: 1 })
- [ ] db.collection.find({ score: -1, username: -1 })
- [✓] db.collection.find({ score: 1, username: -1 })

✓ **Đúng** 1/1 Điểm

39. Given the following sample documents:

    {_id:1, name: "Quesedillas Inc.", active: true },
    {_id:2, name: "Pasta Inc.", active: true },
    {_id:3, name: "Tacos Inc.", active: false },
    {_id:4, name: "Cubanos Inc.", active: false },
    {_id:5, name: "Chicken Parm Inc.", active: false }

    A company wants to create a mobile app for users to find restaurants by name. The developer wants to show the user restaurants that match their search. An Atlas Search index has already been created to support this query.

    What query satisfies these requirements?

(Choose 1)

○ db.restaurants.aggregate([{ "$search": { "text": { "path": "name", "synonym": "cuban"} } }])

[✓ db.restaurants.aggregate([{ "$search": { "text": { "path": "name", "query": "cuban"} } }])]

○ db.restaurants.aggregate([{ "$search": { "text": { "field": "name", "query": "cuban"} } }])

○ db.restaurants.aggregate([{ "$search": { "text": { "field": "name", "synonym": "cuban"} } }])

✓ **Đúng** 1/1 Điểm

40. You are designing a MongoDB database for an e-commerce website. The website should track the customers, their orders, and the products in each order. Which of the following data models would best fit this requirement?

(Choose 1)

○ One collection for customers, orders, and products, with each document representing an order and containing embedded documents for the customer and an array of embedded documents for the products in the order.

[✓ One collection for customers, one collection for orders, and one collection for products. Each order document would contain a reference to the customer who made the order and an array of references to the products in the order.]

○ One collection for customers and orders, with each document representing an order and containing references to both the customer and the products in the order.

○ One collection for customers, one collection for orders, and one collection for products. Each customer document would contain an array of references to their orders and an array of references to the products in each order.

✓ **Đúng** 1/1 Điểm

41. Which of the following is considered an anti-pattern in MongoDB data modeling?

(Choose 1)

○ Using embedded documents to model a one-to-one relationship between entities.

○ Denormalizing data to reduce the number of read operations required to retrieve related data.

○ Using references to model a many-to-many relationship between entities.

[✓ Using a single collection for all entities, regardless of their relationship to each other.]

✓ **Đúng** 1/1 Điểm

42. A typical products collection is in an e-commerce database.

What schema is the most effective?

(Choose 1)

○ Orders for products should be embedded as an array in each product document.

○ Reviews for products should be embedded as an array in each product document.

○ Current and historical prices for product should be embedded in each product document.

[✓ Current inventory/availability for product should be embedded in each product document.]

✓ **Đúng** 1/1 Điểm

43. In the context of MongoDB, consider the following statements related to connection pooling as managed by the MongoDB driver. Select the correct statement.

(Choose 1)

○ Connection pooling allows multiple client applications to use a single database connection simultaneously, thus improving overall system performance.

[✓ Connection pooling involves the process of creating and managing a pool of sockets, which are then used by client applications to connect and interact with the MongoDB database.]

○ Connection pooling in MongoDB works by creating new connections for every client request, discarding it after the request is processed.

○ Connection pooling in MongoDB does not support idle connections, i.e., if a connection is not in use, it is immediately terminated.

✓ **Đúng** 1/1 Điểm

44. You're developing a service in Node.js using the MongoDB Node.js driver. You have to handle a scenario where multiple clients need to access and update certain fields of a document simultaneously. You decided to use the findAndModify method. Given the following options, which of them is the correct approach for handling the document modification?

(Choose 1)

○ A. Using the findAndModify method with { upsert: true } option.

○ B. Using the findAndModify method without any concurrency control options.

○ C. Using the findAndModify method with { returnNewDocument: true } option.

[✓ D. Using the findAndModify method with { new: true } option.]

✓ **Đúng** 1/1 Điểm

45. A developer is tasked with inserting a new document into a MongoDB collection using the Node.js MongoDB driver. The developer needs to add a new customer to the customers collection. The customer's information is stored in a JavaScript object called "newCustomer". Which of the following represents the correct syntax to accomplish this task?

(Choose 1)

- [✓ db.collection('customers').insertOne(newCustomer)]
- db.collection('customers').add(newCustomer)
- db.customers.save(newCustomer)
- db.getCollection('customers').insert(newCustomer)

✓ **Đúng** 1/1 Điểm

46. You are developing an application that uses MongoDB as its primary database. You have been tasked with designing a system that minimizes downtime and maximizes data availability. You need to connect to the database using MongoClient from the Node.js MongoDB driver. Which of the following MongoDB URI connection strings correctly connects MongoClient to a replica set named 'myReplicaSet' spread across three servers, and configures read preference to 'nearest'?

(Choose 1)

- mongodb+srv://localhost:27017,localhost:27018,localhost:27019/mydatabase?replicaSet=myReplicaSet
- mongodb://localhost:27017,localhost:27018,localhost:27019/mydatabase?readPreference=nearest
- mongodb+srv://username:password@localhost:27017,localhost:27018,localhost:27019/mydatabase?replicaSet=myReplicaSet&readPreference=secondary

[✓ mongodb://localhost:27017,localhost:27018,localhost:27019/mydatabase?replicaSet=myReplicaSet&readPreference=nearest]

✓ **Đúng** 1/1 Điểm

47. As a MongoDB Developer, you are tasked with creating an aggregation pipeline using the MongoDB Node.js driver that will transform the documents in the "orders" collection. The pipeline should match all documents where the "status" field is "In Process", then group them by "customer_id", and finally calculate the total and average "amount" for each group. Identify the correct syntax for achieving this.

(Choose 1)

- const pipeline = [{ $find: { status: "In progress" } }, { $group: { _id: "$customer_id", total: { $sum: "$amount" }, average: { $avg: "$amount" } } }]; db.orders.aggregate(pipeline);
- const pipeline = [{ $match: { status: "In progress" } }, { $group: { _id: "$customer_id", total: { $add: "$amount" }, average: { $avg: "$amount" } } }]; db.orders.aggregate(pipeline);
- const pipeline = [{ $where: { status: "In progress" } }, { $group: { _id: "$customer_id", total: { $sum: "$amount" }, average: { $avg: "$amount" } } }]; db.orders.aggregate(pipeline);
- [✓ const pipeline = [{ $match: { status: "In progress" } }, { $group: { _id: "$customer_id", total: { $sum: "$amount" }, average: { $avg: "$amount" } } }]; db.orders.aggregate(pipeline);]

✓ **Đúng** 1/1 Điểm

48. Given the following documents:

{_id:1, restaurant: "Quesadillas Inc.", rating: 4.5 },
{_id:2, restaurant: "Pasta Inc.", rating: 3.9},
{_id:3, restaurant: "Tacos Inc.", rating: 2.5}

A developer wants to find the highest rated restaurant in a list. An index has been created on the appropriate field.

What query satisfies the requirements?

(Choose 1)

○ const pipeline = [{ $sort: { rating : -1, limit: 1 } }]; const aggCursor = coll.runAggregation(pipeline);

○ const pipeline = [{ $sort: { rating : -1 } }, { $limit: 1 }]; const aggCursor = coll.runAggregation(pipeline);

○ const pipeline = [{ $sort: { rating : -1 , limit: 1} }]; const aggCursor = coll.aggregate(pipeline);

[✓ const pipeline = [{ $sort: { rating : -1 } }, { $limit: 1 }]; const aggCursor = coll.aggregate(pipeline);]

✗ **Không chính xác** 0/1 Điểm

49. Given the following document from the cakeFlavors collection. All documents in this collection have the same schema.

```
{
  "_id" : 1,
  "flavor" : "chocolate",
  "number" : 15
}
```

What operation on the cakeFlavors collection will update the value of the number field to 100 for a document with a "strawberry" flavor value and insert a new document if it does not exist?

(Choose 1)

○ db.cakeFlavors.updateOne({ flavor: "strawberry"} , { $set: { number: 100 } }, { $upsert: true })

○ db.cakeFlavors.insertOne({ flavor: "strawberry"} , { $set: { number: 100 } }, { $upsert: true })

[✓ db.cakeFlavors.insertOne({ flavor: "strawberry"} , { $set: { number: 100 } }, { upsert: true })]

◯ db.cakeFlavors.updateOne({ flavor: "strawberry"} , { $set: { number: 100 } }, { upsert: true })

✕ **Không chính xác** 0/1 Điểm

50. What are two valid method names for MongoClient class?

(Choose 2)

☐ db()

☑ open()

☐ destroy()

☑ close()

✓ **Đúng** 1/1 Điểm

51. What are two advantages of using Connection Pooling within the Javascript (NodeJS) Driver?

(Choose 2)

☑ Reduce application latency

☐ Remove the need for the application to authenticate

☐ Remove the need to open and close connections in the application

☑ Limit the number of connections to the server

✕ **Không chính xác** 0/1 Điểm

52. Given the following collection named movies, containing the following six documents:

{ _id: 1, "title": ''A.", "genre": "sci-fi'', "year": "1983" }
{ _id: 2, "title": ''B.", "genre": "comedy'', "year": "1990" }
{ _id: 3, "title": ''C.", "genre": "documentary', "year": "1995" }
{ _id: 4, "title": ''D.", "genre": "sci-fi'', "year": "1985" }
{ _id: 5, "title": ''E.", "genre": "drama'', "year": "2000" }
{ _id: 6, "title": ''F.", "genre": "comedy'', "year": "1977" }

What syntax will update multiple documents in the collection?

(Choose 1)

○ const filter = { genre: "sci-fi" }; const updateDoc = { $set: { genre: "science fiction" } }; const result = await movies.updateMany(filter, updateDoc);

○ const filter = { genre = "sci-fi" }; const updateDoc = { $set = { genre = "science fiction", review = 'While very fictional, I found the science lacking' } }; const result = await movies.updateMany(filter, updateDoc);

[✓ ● const filter = { genre = "sci-fi" }; const updateDoc = { $set: { genre = "science fiction", review = 'I found it neither scientific nor fictional' } }; const result = await movies.update(filter, updateDoc);]

○ const filter = { genre: "sci-fi" }; const updateDoc = { $set: { genre: "science fiction", review: 'I found it neither scientific nor fictional' } }; const result = await movies.update(filter, updateDoc);

✕ **Không chính xác** 0/1 Điểm

53. A Cooking dataset is in Atlas. There is a Recipes database with a Desserts collection.

How can one document be found that provides a recipe for cookies without chocolate using Atlas Data Explorer?

(Choose 1)

○ 1. Select the collection on the left-hand side. 2. Select the "Aggregation" view. 3. Specify the first stage as `$match` query: {dessert_type: "Cookie"} 4. Specify the second stage as `$match` query: {ingredients: {$all: ["chocolate"]}} 5. Set `$limit` to 1.

○ 1. Select the collection on the left-hand side. 2. Select the "Aggregation" view. 3. Specify the first stage as `$match` query: {dessert_type: "Cookie", ingredients: {$nin: ["chocolate"]}} 4. Set `$limit` to 1.

[✓ 1. Select the collection on the left-hand side. 2. Specify the "Find" view. 3. Specify the filter query: {dessert_type: "Cookie", ingredients: {$nin: ["chocolate"]} 4. Specify the project query: {dessert_type: 1}]

○ 1. Select the collection on the left-hand side. 2. Specify the "Find" view. 3. Specify the filter query with limit: {dessert_type: "Cookie", ingredients: {$nin: ["chocolate"], $limit: 1}