1. Which of the following are valid types in BSON? Check all that apply.

2. Why does MongoDB use BSON rather than JSON? Check all that apply.

------------------------------------------

3. Which writes are atomic in MongoDB 4.0? Check all that apply.

* An update to a single document in a replica set
* An update to multiple documents in a replica set using transactions
* An update to a single document in a sharded cluster

------------------------------------------

4. On which of the following CRUD operations can you specify a write concern? Check all that apply.

* db.collection.insertOne()
* db.collection.find()
* db.collection.updateMany()

----------------------------------------

5. db.people.updateMany( { "city" : "Seattle", "state" : "WA" }, { "$addToSet" : { "likes" : "forest" } } )

* { "\_id" : ObjectId("57fd48257268886f789b33ff"), "firstName" : "Arthur", "lastName" : "Aaronson", "state" : "WA", "city" : "Seattle", "likes" : [ "dogs", "cats", "hiking" ] }
* { "\_id" : ObjectId("57fd48257268886f789b3400"), "firstName" : "Beth", "lastName" : "Barnes", "state" : "WA", "city" : "Richland", "likes" : [ "forest", "desert" ] }
* { "\_id" : ObjectId("57fd48257268886f789b3401"), "firstName" : "Charlie", "lastName" : "Carlson", "state" : "CA", "city" : "San Diego", "likes" : [ "desert", "beach" ] }
* { "\_id" : ObjectId("57fd48257268886f789b3402"), "firstName" : "Dawn", "lastName" : "Davis", "state" : "WA", "city" : "Seattle", "likes" : [ "forest", "mountains", "hiking" ] }

6. db.sayings.find( { $text : { $search : "fact find" } } )

* { \_id : 1, quote : “That’s a fact, Jack.” }
* { \_id : 2, quote : “Find out if that fact is correct.” }
* { \_id : 3, quote : “Nobody will ever catch me.” }

7. db.team.find()

* { "\_id" : 12, scores : [ 3, 5, 7, 2, 1, -4, 3, 12 ] }
* { "\_id" : 12, "scores" : [ 3, 5, 7, 2, 1 ] }

---------------------------------------

8. db.sample.find( { "$or" : [ { "a" : { "$in" : [ 3, 10] } }, { "b" : { "$lt" : 2 } } ] } )

* { "\_id" : 1, "a" : 0, "c" : 0, "b" : 2 }
* { "\_id" : 2, "a" : 2, "c" : 0, "b" : 1 }
* { "\_id" : 3, "a" : 4, "c" : 0, "b" : 14 }
* { "\_id" : 4, "a" : 5, "c" : 0, "b" : 17 }
* { "\_id" : 5, "a" : 3, "c" : 0, "b" : 12 }
* { "\_id" : 6, "a" : 1, "c" : 1, "b" : 5 }
* { "\_id" : 7, "a" : 8, "c" : 1, "b" : 7 }
* { "\_id" : 8, "a" : 11, "c" : 1, "b" : 0 }
* { "\_id" : 9, "a" : 17, "c" : 1, "b" : 1 }
* { "\_id" : 10, "a" : 3, "c" : 1, "b" : 1 }

------------------------------------------------

9. Consider the following example document from the sample collection. All documents in this collection have the same schema.

{ "\_id" : 3, "a" : 7, "b" : 4 }

Which of the following queries will replace this with the document,

{"\_id" : 7, "c" : 4 }

* db.sample.replaceOne( { "\_id" : 3 } , { "\_id" : 7 , "c" : 4 } )
* db.sample.updateOne( { "\_id" : 3 } , { "$set" : { "\_id" : 7 , "c" : 4 } } )
* db.sample.updateOne( { "\_id" : 3 } , { "\_id" : 7 , "c" : 4 , { "$unset" : [ "a" , "b" ] } } )
* db.sample.updateOne( { "\_id" : 3 } , { "\_id" : 7 , "c" : 4 })
* This operation cannot be done with a single query.

---------------------------------------------------------------------

10. db.foo.insertOne( { } );

* An empty document
* A document with an \_id assigned to be an ObjectId
* A document that matches the collection's existing schema, but with null fields
* No document will be inserted; an error will be raised
* A document will be inserted with the same \_id as the last document inserted

--------------------------------------------------------------

11. Which of the following are good reasons to denormalize and create copies of your data in different collections? Check all that apply.

* When you want to optimize for your most common read use case(s).
* There is never a good reason to denormalize.
* To avoid having to join data in the application layer.

--------------------------------------------------------------

11. What setting can be specified to disable journaling for replica set members using the WiredTiger storage engine?

* storage.journal.enabled: false
* --nojournal
* Journaling is an opt in feature so doesn't need to be explicity disabled
* Journaling cannot be disabled for replica set members

------------------------------------------------------------

12. In MongoDB, the WiredTiger storage engine provides concurrency at what level?

* Server-level concurrency
* Database level concurrency
* Collection level concurrency
* Document level concurrency
* Field level concurrency

-------------------------------------------------------

13. After adding a new consumer service, you begin to receive reports of poor query performance. You suspect disk latency may be a cause. Which tool can you use to confirm or rule out your suspicion?

* mongostat
* mongoreplay
* mongotop
* mongofiles

---------------------------------------------------

14. Given the following example document:

{

"\_id": ObjectId("5360c0a0a655a60674680bbe"),

"user": {

"login": "ir0n",

"description": "Made of metal"

"date": ISODate("2014-04-30T09:16:45.836Z"),

}

}

and the following index:

db.users.createIndex( { "user.login": 1, "user.date": -1 }, "myIndex" )

When performing the following query:

db.users.find( { "user.login": /^ir.\*/ }, { "user":1, "\_id":0 } ).sort( { "user.date":1 } )

* As a covered query using "myIndex" because we are filtering out "\_id" and only returning "user.login"
* As an index scan that uses "myIndex" because field "user.login" is indexed
* As an optimized sort query (no explicit sort stage) using "myIndex" because we are sorting on an indexed field
* MongoDB will need to do a table/collection scan to find matching documents
* None of the above

-------------------------------------------------------------------

15. db.sample.createIndex( { "foo" : -1 } )

For which of the following queries can MongoDB efficiently look at only a subset of the index entries, rather than all of the index entries? Check all that apply.

* db.sample.find( { "foo" : /a.\*b/ } )
* db.sample.find( { "foo" : /^c.\*d/ } )
* db.sample.find( { "foo" : /^e.\*f/ } )

------------------------------------------------

16. Adding an index on { a : 1 } can potentially decrease the speed of which of the following operations? Check all that apply.

* db.collection.find( { a : 232 } )
* db.collection.updateOne( { b : 456 }, { $inc : { a : 1 } } )
* db.collection.insertOne( { a : 341 } )

---------------------------------------------

17. You have the following indexes on the things collection:

[{

"v" : 1,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "test.things"

},

{

"v" : 1,

"key" : {

"a" : 1

},

"name" : "a\_1",

"ns" : "test.things"

},

{

"v" : 1,

"key" : {

"c" : 1,

"b" : 1,

"a" : 1

},

"name" : "c\_1\_b\_1\_a\_1",

"ns" : "test.things"

}]

* db.things.find( { b : 1 } ).sort( { c : 1, a : 1 } )
* db.things.find( { c : 1 } ).sort( { a : 1, b : 1 } )
* db.things.find( { a : 1 } ).sort( { b : 1, c : 1 } )

------------------------------------------------------------------------------

18. Which of the following statements are true of unique indexes? Check all that apply.

* The only possible unique index is the "\_id" field.
* The "unique" constraint on an index ensures that no two (or more) documents can share a value for that field in a collection
* Hashed indexes cannot be unique.

-------------------------------------------------------------------------

19. You have the following index on the toys collection:

{ "manufacturer" : 1, "name" : 1, "date" : -1 }

* db.toys.find( { manufacturer : "Matteo", name : "Barbara", date : "2018-07-02" } )
* db.toys.find( { name : "Big Rig Truck", date : "2018-02-01", manufacturer : "Tanko" } )
* db.toys.find( { date : "2018-03-01", manufacturer : "Loggo", name : "Brick Set" } )

20. You have the following indexes on your collection:

[{

"v" : 1,

"key" : {

"\_id" : 1

},

"name" : "\_id\_",

"ns" : "test.sample"

},

{

"v" : 1,

"key" : {

"name" : 1,

"date" : 1,

"phone" : 1

},

"name" : "name\_1\_date\_1\_phone\_1",

"ns" : "test.sample"

}]

Which of the following queries will use an index? Check all that apply.

* db.sample.find( { \_id : 22, date: ISODate("2012-07-04" } )
* db.sample.find( { date: ISODate("2011-07-04"), name : "Alice" } )
* db.sample.find( { title : "DBA" } )
* db.sample.find( { phone : "123-456-7890"), info : "201-555-5792" } )

21. Which of the following is true of covered queries? Check all that apply.

* MongoDB can satisfy covered queries using only index keys.
* MongoDB can satisfy covered queries efficiently without the use of an index.
* Covered queries ordinarily have slower response times than queries that are not covered.

22. Which of the following must be true for a query to be a covered query? Check all that apply.

Check all that apply:

* All fields used in the selection filter of the query must be in the index that the query uses
* All fields returned in the results must be in the index that the query uses
* All fields returned in the results must be fields in the selection filter of the query

--------------------------------------------------------------

23. You have created the following index on the foo collection:

> db.foo.createIndex( { a : 1, b : -1, c : -1, d: 1 } )

Which of the following queries will be able to fulfill the query without an in-memory sort (i.e., it's able to use the index to sort)? Check all that apply.

* db.foo.find( { a : { $gt : 100 } } ).sort( { c : -1 } )
* db.foo.find( { a : 100 } ).sort( { b : 1, c : 1 } )
* db.foo.find( { a : 200, b : { $lt : 100 } } ).sort( { b : 1 } )

----------------------------------------------------------------

24. The users collection contains the following documents:

> db.users.find()

* { "\_id" : 1, "name" : "dave123", "q1" : true, "q2" : true }
* { "\_id" : 2, "name" : "dave2", "q1" : false, "q2" : false }
* { "\_id" : 3, "name" : "ahn", "q1" : true, "q2" : true }
* { "\_id" : 4, "name" : "li", "q1" : true, "q2" : false }
* { "\_id" : 5, "name" : "annT", "q1" : false, "q2" : true }
* { "\_id" : 6, "name" : "li", "q1" : true, "q2" : true }
* { "\_id" : 7, "name" : "ty", "q1" : false, "q2" : true }

- You perform the following query:

db.users.aggregate( [ { $sample : { size : 3 } } ] )

Which of the following are possible outputs from the database? Check all that apply.

Check all that apply:

* { "\_id" : 7, "name" : "ty", "q1" : false, "q2" : true }
* { "\_id" : 2, "name" : "dave2", "q1" : false, "q2" : false }
* { "\_id" : 4, "name" : "li", "q1" : true, "q2" : false }
* { "\_id" : 1, "name" : "dave123", "q1" : true, "q2" : true }
* { "\_id" : 2, "name" : "dave2", "q1" : false, "q2" : false }
* { "\_id" : 3, "name" : "ahn", "q1" : true, "q2" : true }
* { "\_id" : 5, "name" : "annT", "q1" : false, "q2" : true }
* { "\_id" : 6, "name" : "li", "q1" : true, "q2" : true }
* { "\_id" : 7, "name" : "ty", "q1" : false, "q2" : true }

25 Aggregation: Question 2

Which of the following statements are true about the $match pipeline stage? Check all that apply.

Check all that apply:

* You should use it as early as possible in the pipeline.
* It can be used as many times as needed.
* It has a syntax similar to find() commands.

26. Suppose you have the following collection in MongoDB. The collection is named alpha.

* {"\_id" : 1, "region" : "NW1", "leads" : 1, "email" : "mlangley@co1.com"}
* {"\_id" : 2, "region" : "NW1", "leads" : 1, "email" : "jpicoult@co4.com"}
* {"\_id" : 3, "region" : "NW1", "leads" : 2, "email" : "zzz@company2.com"}
* {"\_id" : 4, "region" : "SE1", "leads" : 8, "email" : "mary@hssu.edu" }
* {"\_id" : 5, "region" : "SE2", "leads" : 4, "email" : "janet@col.edu"}
* {"\_id" : 6, "region" : "SE2", "leads" : 2, "email" : "bill@uni.edu"}
* {"\_id" : 7, "region" : "SE2", "leads" : 4, "email" : "iii@company1.com"}
* {"\_id" : 8, "region" : "SW1", "leads" : 1, "email" : "phil@co3.com"}
* {"\_id" : 9, "region" : "SW1", "leads" : 2, "email" : "thomas@company.com"}
* {"\_id" : 10, "region" : "SW2", "leads" : 2, "email" : "sjohnson@uchi.edu"}
* {"\_id" : 11, "region" : "SW2", "leads" : 5, "email" : "tsamuel@someco.com"}

How many documents will be returned in the result set in response to the following aggregation query?

db.alpha.aggregate( [

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

"count" : { "$sum" : 1 } } } ,

{ "$match" : { "count" : { "$gte" : 3 } } }

] )

Choose the best answer:

* 0
* 1
* 2
* 3
* 4

27. Suppose you have the following collection with only 2 documents:

> db.people.find()

* { "\_id" : "apples", "traits" : [ "sweet" , "crispy" ] }
* { "\_id" : "oranges", "traits" : [ "sweet" , "orange" , "juicy" ] }

You run an aggregation query and begin with { $unwind : "$traits" } .

How many documents will you have in the pipeline after the $unwind stage?

Choose the best answer:

* 1
* 2
* 3
* 4
* 5

28. Which of the following is a correct definition for idempotence?

Choose the best answer:

Affecting the same fields in successive queries. e.g., db.collection.updateOne( { \_id : 3 } , { $inc : { a : 7 , b : 6 } } ) and db.collection.updateOne( { \_id : 3 } , { $inc : { a : 1 , b : 2 } } ) are idempotent with respect to one another

* If an action may be performed multiple times and have the same effect as if it had been performed once, it is idempotent. For example, $set: {a: 3} is idempotent, but $inc: {a: 1} is not.
* Having the property that order does not matter. I.e., if A and B are update operations, then they are idempotent if A(B(document)) = B(A(document))
* Requiring the same amount of data to be transferred makes two queries idempotent with respect to one another.
* Setting an entire document's state with an update, rather than just modifying a subset in the argument of the update, is an idempotent operation.

29. Which of the following describe the primary reasons MongoDB supports replication? Check all that apply.

* To provide high availability
* To enable horizontal scaling
* To prevent downtime in case of a disaster at a data center

30. Replication: Question 3

What is the principal advantage of having a delayed replica set member?

Choose the best answer:

* It allows the load on the secondary servers to be more evenly spread.
* It allows you to perform queries against historical versions of the data.
* It increases write speed to the primary.
* It makes it easier to upgrade the system without downtime.
* It provides a window of time to recover from an operator error.

----------------------------------------------------------------------------------

31. Replication: Question 4

What settings can be controlled by rs.reconfig()? Check all that apply.

Check all that apply:

* Priority for each replica set member
* Which replica set members are hidden
* Which version of MongoDB each replica set member is running

32. The people collection contains the following documents:

{ "\_id" : ObjectId("57fd59a2d630a0fd9685a148"), "firstName" : "Arthur", "lastName" : "Aaronson", "state" : "WA", "city" : "Seattle", "likes" : [ "dogs", "cats" ] }

{ "\_id" : ObjectId("57fd59a2d630a0fd9685a149"), "firstName" : "Beth", "lastName" : "Barnes", "state" : "WA", "city" : "Richland", "likes" : [ "forest", "desert" ] }

{ "\_id" : ObjectId("57fd59a2d630a0fd9685a14a"), "firstName" : "Charlie", "lastName" : "Carlson", "state" : "CA", "city" : "San Diego", "likes" : [ "desert", "beach" ] }

{ "\_id" : ObjectId("57fd59a2d630a0fd9685a14b"), "firstName" : "Dawn", "lastName" : "Davis", "state" : "WA", "city" : "Seattle", "likes" : [ "forest", "mountains" ] }

You perform the following query:

db.people.deleteMany( { state : "NB" } )

How many oplog entries are created as a result of this query?

Choose the best answer:

* 0
* 1
* 2
* 3
* 4

33. Your top three queries for a collection consist of the following query patterns:

90% db.employees.find( { lastName : 1, firstName : 1, currentEmployee : 1, company : 1 }, { supervisor : 1, teamName : 1, position: 1, duties : 1, salary : 1 } )

5% db.employees.updateOne( { employeeId: 1 }, { $set : { position : 1, teamName : 1, salary : 1, duties : 1 } } )

1% db.employees.updateOne( { employeeId: 1 }, { $set : { currentEmployee : 1, hireDate: 1 } } )

Your system is outgrowing the servers you currently have it running on, and you would like to shard the collection. Which of the following shard keys will be most performant?

Choose the best answer:

* { company : 1, lastName: 1 }
* { supervisor : 1, teamName : 1, position : 1, duties : 1, salary : 1 }
* { employeeId : 1 }
* { currentEmployee : 1, hireDate : 1 }
* { currentEmployee : 1, company : 1 }

34. Which of the following are traits of a hashed \_id as a shard key, relative to an unhashed \_id? Check all that apply.

Check all that apply:

* Inserts on auto-generated ObjectId’s will be distributed across shards
* Range queries (to find documents created on a particular week, for example) may be less efficient
* Increased security due to hashing

-------------------------------------------------------------------

35. Sharding: Question 3

In a sharded collection, which of the following is true of the primary shard?

Choose the best answer:

* It processes queries from the application layer and determines the location of this data in the sharded cluster.
* It is a shard on which the primary for a replica set is found.
* It holds the unsharded collections for the database.
* It is simply the first shard initialized in a sharded cluster.
* It holds configuration information for the sharded cluster.
* Your answer was saved at 10:06:32 AM UTC

36. Suppose you have a sharded cluster with a status as follows:

mongos> sh.status()

--- Sharding Status ---

sharding version: {

"\_id" : 1,

"version" : 3,

"minCompatibleVersion" : 3,

"currentVersion" : 4,

"clusterId" : ObjectId("51de8630162b88d59cd7b006")

}

shards:

* { "\_id" : "shard0000", "host" : "localhost:30000" }
* { "\_id" : "shard0001", "host" : "localhost:30001" }
* { "\_id" : "shard0002", "host" : "localhost:30002" }
* { "\_id" : "shard0003", "host" : "localhost:30003" }
* { "\_id" : "shard0004", "host" : "localhost:30004" }
* { "\_id" : "shard0005", "host" : "localhost:30005" }
* { "\_id" : "shard0006", "host" : "localhost:30006" }
* { "\_id" : "shard0007", "host" : "localhost:30007" }

databases:

{ "\_id" : "admin", "partitioned" : false, "primary" : "config" }

{ "\_id" : "test", "partitioned" : true, "primary" : "shard0001" }

test.products

shard key: { "productId" : 1 }

chunks:

* + - shard0000 2
    - shard0002 2
    - shard0003 1
    - shard0004 1
    - shard0005 1
    - shard0006 1
    - shard0007 1
    - shard0001 2

{ "productId" : { "$minKey" : 1 } } -->> { "productId" : 9294 } on : shard0000 { "t" : 2, "i" : 0 }

{ "productId" : 9294 } -->> { "productId" : 18684 } on : shard0002 { "t" : 3, "i" : 0 }

{ "productId" : 18684 } -->> { "productId" : 27851 } on : shard0003 { "t" : 4, "i" : 0 }

{ "productId" : 27851 } -->> { "productId" : 36852 } on : shard0004 { "t" : 5, "i" : 0 }

{ "productId" : 36852 } -->> { "productId" : 46047 } on : shard0005 { "t" : 6, "i" : 0 }

{ "productId" : 46047 } -->> { "productId" : 55450 } on : shard0006 { "t" : 7, "i" : 0 }

{ "productId" : 55450 } -->> { "productId" : 64644 } on : shard0007 { "t" : 8, "i" : 0 }

{ "productId" : 64644 } -->> { "productId" : 73769 } on : shard0000 { "t" : 9, "i" : 0 }

{ "productId" : 73769 } -->> { "productId" : 82950 } on : shard0002 { "t" : 10, "i" : 0 }

{ "productId" : 82950 } -->> { "productId" : 91983 } on : shard0001 { "t" : 10, "i" : 1 }

{ "productId" : 91983 } -->> { "productId" : { "$maxKey" : 1 } } on : shard0001 { "t" : 1, "i" : 10 }

Which shards would be involved in answering the following query:

use test;

db.products.find({"productId" : {"$gte" : 20000, "$lte" : 40000}});

Check all that apply:

* shard0000
* shard0001
* shard0002
* shard0003
* shard0004
* shard0005
* shard0006
* shard0007

-------------------------------------------------------------------------

37. When a chunk is in flight from one shard to another during a migration process, where are reads to that chunk directed?

Choose the best answer:

* To the shard from which it is being migrated
* To both the shard from which it is being migrated and the shard to which it is being migrated
* To the shard to which it is being migrated
* Documents in the chunk are locked during migration. Reads are queued.
* To the config server

38. In which of the following situations can we assume sharding will be an effective strategy?

Choose the best answer:

* A single MongoDB instance cannot keep up with your application's write load and you have exhausted other options.
* Your data set is too big to fit in a single MongoDB instance.
* You would like to improve read performance for your application.
* The data set is taking too much time to backup and restore.
* All of the above

39. Consider a collection of users with the following fields and possible values:

phone\_number -- a 10-digit telephone number (string)

eye\_color -- "brown", "hazel", "blue", "green", or "other" (string)

weight -- an integer in pounds; known for about half the users

started\_driving\_at -- the age at which the user got their driver's license in years. For most users this is 15, 16, 17, or 18. (integer)

\_id -- automatically created on insert (ObjectId)

Assuming the data-access patterns also support your choice, which of these fields would make the best shard key?

Choose the best answer:

* phone\_number
* eye\_color
* weight
* started\_driving\_at
* \_id

40. In a sharded cluster, which of the following indexes should contain only unique values? Check all that apply. Check all that apply:

* The \_id index
* The shard key index
* A single-field index

----------------------------------------------------------------------------------------

41. An insufficiently granular (“low cardinality”) shard key can result in which type of problems? Check all that apply.Check all that apply:

* Large chunks that cannot be split.
* More shards getting hit on certain queries.
* Limits to document size.

---INDEXES-------------------------------------------------------------------------------------

42. Which of the following statements is/are true?

* Compound indexes can service queries that filter on a prefix of the index keys.
* By default, the explain() command will execute your query.
* Compound indexes can service queries that filter on any subset of the index keys.
* If no indexes can be used then a collection scan will be necessary.
* Query plans are removed from the plan cache on index creation, destruction, or server restart.

--------------------------------------------------------------------------------------------------

43. Given the following indexes:

1. { categories: 1, price: 1 }

2. { in\_stock: 1, price: 1, name: 1 }

**The following documents:**

1. { price: 2.99, name: "Soap", in\_stock: true, categories: ['Beauty', 'Personal Care'] }

2. { price: 7.99, name: "Knife", in\_stock: false, categories: ['Outdoors'] }

**And the following queries:**

1. db.products.find({ in\_stock: true, price: { $gt: 1, $lt: 5 } }).sort({ name: 1 })

2. db.products.find({ in\_stock: true })

3. db.products.find({ categories: 'Beauty' }).sort({ price: 1 })

Which of the following is/are true?

* Index #2 can be used by both query #1 and #2.
* Index #1 would provide a sort to query #3.
* Index #2 properly uses the equality, sort, range rule for query #1.
* There would be a total of 4 index keys created across all of these documents and indexes.

No, there would be 5 total index keys:

{ categories: 'Beauty', price: 2.99 }

{ categories: 'Personal Care', price: 2.99 }

{ categories: 'Outdoors', price: 7.99 }

{ in\_stock: true, price: 2.99, name: 'Soap' }

{ in\_stock: false, price: 7.99, name: 'Knife'}

The additional index keys are due to the multikey index on categories.

----------------------------------------------------------------------------------------------------

44. Which of the following statements is/are true?

* -Indexes can only be traversed forward.
* -The ideal ratio between nReturned and totalKeysExamined is 1.
* -An index doesn't become multikey until a document is inserted that has an array value.
* -Running performance tests from the mongo shell is an acceptable way to benchmark your database.
* -You can use the --wiredTigerDirectoryForIndexes option to place your indexes on a different disk than your data.

-----------------------------------------------------------------------------------------------------

45. Which of the following statements is/are true?

* Compound indexes can service queries that filter on a prefix of the index keys.
* By default, the explain() command will execute your query.
* Compound indexes can service queries that filter on any subset of the index keys.
* If no indexes can be used then a collection scan will be necessary.
* Query plans are removed from the plan cache on index creation, destruction, or server restart.

------------------------------------------------------------------------------------------------------

46. Which of the following statements is/are true?

* Indexes are fast to search because they're ordered such that you can find target values with few comparisons.
* Under heavy write load you should scale your read throughput by reading from secondaries.
* On a sharded cluster, aggregation queries using $lookup will require a merge stage on a random shard.
* When you index on a field that is an array it creates a partial index.
* Indexes can solve the problem of slow queries.

------------------------------------------------------------------------------------------------------

47. Which of the following statements is/are true?

* Background index builds block all reads and writes to the database that holds the collection being indexed.
* By default, all MongoDB user-created collections have an \_id index.
* MongoDB indexes are markov trees.
* Collations can be used to create case insensitive indexes.
* It's common practice to co-locate your mongos on the same machine as your application to reduce latency.

-------------------------------------------------------------------------------------------------------

48. Which of the following statements is/are true?

* Indexes can decrease insert throughput.
* Indexes can be walked backwards by inverting their keys in a sort predicate.
* Partial indexes can be used to reduce the size requirements of the indexes.
* It's important to ensure that secondaries with indexes that differ from the primary not be eligible to become primary.
* It's important to ensure that your shard key has high cardinality

--------------------------------------------------------------------------------------------------------

49. Which of these statements is/are true?

* Write concern has no impact on write latency.
* You can index multiple array fields in a single document with a single compound index.
* Creating an ascending index on a monotonically increasing value creates index keys on the right-hand side of the index tree.
* Covered queries can sometimes still require some of your documents to be examined.
* A collection scan has a logarithmic search time.

--------------------------------------------------------------------------------------------------------

50. From a performance standpoint, when working with a distributed database it's important to consider...

* Latency
* Routed Queries
* Reading from secondaries only

----------------------------------------------------------------------------------------------------------

51. Which of the following is/are true?

* Vertical scaling is generally cheaper than horizontal scaling.
* Picking a good shard key is one of the most important parts of sharding.
* Ordered bulk operations are faster than unordered.

52. Should you ever read from secondaries on a sharded cluster?

- As of MongoDB 3.6, because of changes to both logic in chunk migration and read guarantees, it is now safe to read from secondaries as long as the appropriate read concern is specified.

----------------------------------------------------------------------------------------------------------

53. Which of the following conditions apply when creating indexes on secondaries?

* A secondary should never be allowed to become primary [TRUE]
* These indexes can only be set on secondary nodes
* We can create specific indexes on secondaries, even if they are not running in standalone mode

-----------------------------------------------------------------------------------------------------------

54. What operators will cause a merge stage on the primary shard for a database?

* $lookup - Yes

- $group - No, $group can potentially cause a merge stage, but a random shard will be selected for the merging.

- $out - Yes

----------------------------------------------------------------------------------------------------------

56. When building indexes to service your queries, which of the following is the general rule of thumb you should keep when ordering your index keys?

Note, use the following definitions to for this question:

* equality: indexed fields on which our queries will do equality matching
* range: indexed fields on which our queries will have a range condition
* sort: indexed fields on which our queries will sort on-

1. equality, range, sort
2. sort, range, equality
3. sort, equality, range
4. equality, sort, range [True]
5. range, sort, equality
6. range, equality, sort

------------------------------------------------------------------------------------------------------------

57. Given the following indexes:

* { \_id: 1 }
* { name: 1, dob: 1 }
* { hair: 1, name: 1 }

Which of the following queries could be covered by one of the given indexes?

1. db.example.find( { \_id : 1117008 }, { \_id : 0, name : 1, dob : 1 } )
2. db.example.find( { name : { $in : [ "Alfred", "Bruce" ] } }, { name : 1, hair : 1 } )
3. db.example.find( { name : { $in : [ "Bart", "Homer" ] } }, {\_id : 0, hair : 1, name : 1} )
4. db.example.find( { name : { $in : [ "Bart", "Homer" ] } }, {\_id : 0, dob : 1, name : 1} )

--------------------------------------------------------------------------------------------------------------

58. Given the following index:

> db.products.createIndex({ productName: 1 })

And the following query:

> db.products.find({ productName: /^Craftsman/ })

1. The query will need to do a collection scan.
2. The query will do an index scan.
3. The query will likely need to look at all index keys.
4. The query would match a productName of "Screwdriver - Craftsman Brand"

---------------------------------------------------------------------------------------------------------------

59. Which of the following decreases the write performance of your MongoDB cluster?

1. Adding indexes
2. Increasing the number of members we acknowledge writes from
3. Upgrading to MongoDB 3.4

------------------------------------------------------------------------------------------------------

60. Why is it important to maintain the same data type for fields across different documents?

1. It's just a best practice; all drivers will deal with data type issues by default
2. It helps to simplify the client application logic [True]
3. Because it aligns well with cosmetic shapes of documents
4. - To avoid application data consistency problems [True]

-------------------------------------------------------------------------------------------------------

61. With regards to aggregation performance, which of the following are true?

1. You can increase index usage by moving $match stages to the end of your pipeline
2. Passing allowDiskUsage to your aggregation queries will seriously increase their performance
3. When $limit and $sort are close together a very performant top-k sort can be performed
4. Transforming data in a pipeline stage prevents us from using indexes in the stages that follow

---------------------------------------------------------------------------------------------------------

62. Which of the following are true of index build operations?

1. Foreground index builds block all reads and writes to the collection being indexed.
2. Foreground index builds block all reads and writes to the database that holds the collection being indexed.
3. Background index builds do not impact the query performance of the MongoDB deployment while running.
4. Background index builds take longer to complete than foreground index builds.
5. Background index builds block reads and writes to the collection being indexed.

--------------------------------------------------------------------------------------------------------

63. Which of the following is/are true concerning query plans?

1. MongoDB's query optimizer is statistically based, where collection heuristics are used to determine which plan wins.
2. Query plans are cached so that plans do not need to be generated and compared against each other every time a query is executed. [TRUE]
3. When query plans are generated, for a given query, every index generates at least one query plan.
4. If an index can't be used, then there is no query plan for that query.

------------------------------------------------------------------------------------------------------------

64. With the output of an explain command, what can you deduce?

1. The index used by the chosen plan
2. If a sort was performed by walking the index or done in memory
3. All the available indexes for this collection
4. All the different stages the query needs to go through with details about the time it takes, the number of documents processed and returned to the next stage in the pipeline
5. The estimation of the cardinalities of the distribution of the values

--------------------------------------------------------------------------------------------------------------

65. What is the method that forces MongoDB to use a particular index?

1. force()
2. suggest()
3. hint()
4. index()

-----------------------------------------------------------------------------

66. Which of the following statements apply to index resource allocation?

1. For the fastest processing, we should ensure that our indexes fit entirely in RAM
2. Index information does not need to completely allocated in RAM since MongoDB only uses the right-end-side to the index b-tree, regardless of the queries that use index.
3. Indexes are not required to be entirely placed in RAM, however performance will be affected by constant disk access to retrieve index information.

67. What type of strategy and tools should we be using to performance benchmark a MongoDB installation?

1. Publicly available tools, including correct database variations
2. Mongo shell to test read performance
3. Test transfer ratio using mongodump

----------------------------------------------------------------------------

68. The following query was ran:

> var exp = db.restaurants.explain("executionStats")

> exp.find({ "address.state": "NY", stars: { $gt: 3, $lt: 4 } }).sort({ name: 1 }).hint(REDACTED)

Which resulted in the following output:

{

"queryPlanner": {

"plannerVersion": 1,

"namespace": "m201.restaurants",

"indexFilterSet": false,

"parsedQuery": "REDACTED",

"winningPlan": {

"stage": "SORT",

"sortPattern": {

"name": 1

},

"inputStage": {

"stage": "SORT\_KEY\_GENERATOR",

"inputStage": {

"stage": "FETCH",

"inputStage": {

"stage": "IXSCAN",

"keyPattern": "REDACTED",

"indexName": "REDACTED",

"isMultiKey": false,

"isUnique": false,

"isSparse": false,

"isPartial": false,

"indexVersion": 1,

"direction": "forward",

"indexBounds": "REDACTED"

}

}

}

},

"rejectedPlans": [ ]

},

"executionStats": {

"executionSuccess": true,

"nReturned": 3335,

"executionTimeMillis": 20,

"totalKeysExamined": 3335,

"totalDocsExamined": 3335,

"executionStages": "REDACTED"

},

"serverInfo": "REDACTED",

"ok": 1

}

which index was used to satisfy a query given its explain output

1. { "address.state": 1 }
2. { "address.state": 1, "name": 1, "stars": 1 }
3. { "address.state": 1, "name": 1 }
4. { "address.state": 1, "stars": 1, "name": 1 } [Yes, this query wouldn't need to examine any extra index keys, so since nReturned and totalKeysExamined are both 3,335 we know this index was used.]

---------------------------------------------------------------------------------

69. Which of the following statements regarding indexes are true?

1. Indexes are used to increase the speed of our queries. [TRUE]
2. The \_id field is automatically indexed on all collections. [TRUE]
3. Indexes reduce the number of documents MongoDB needs to examine to satisfy a query. [TRUE]
4. Indexes can decrease write, update, and delete performance. [TRUE]

--------------------------------------------------------------------------

70. Which of the following queries can use an index on the zip field?

1. db.addresses.find( { zip : 55555 } )
2. db.addresses.find( { city : "Newark", state : "NJ" } )
3. db.addresses.find()

-------------------------------------------------------------------

71. Given the following schema for the products collection:

{

"\_id": ObjectId,

"product\_name": String,

"product\_id": String

}

And the following index on the products collection: { product\_id: 1 }

1. db.products.find({ product\_name: 'Wax' }).sort({ product\_name: 1 })
2. db.products.find({ product\_name: 'Soap' }).sort({ product\_id: 1 })
3. db.products.find({}).sort({ product\_id: -1 })
4. db.products.find({}).sort({ product\_id: 1 })
5. db.products.find({ product\_id: '57d7a1' }).sort({ product\_id: -1 })

----------------------------------------------------------------------

72. Which of the following statements are true?

1. Index prefixes can be used in query predicates to increase index utilization.
2. Index prefixes can be used in sort predicates to prevent in-memory sorts.
3. We can invert the keys of an index in our sort predicate to utilize an index by walking it backwards.
4. It's impossible to have a sorted query use an index for both sorting and filtering.

-----------------------------------------------------------------------

73. Given the following index:

{ name: 1, emails: 1 }

When the following document is inserted, how many index entries will be created?

{ "name": "Beatrice McBride",

"age": 26,

"emails": [

"puovvid@wamaw.kp",

"todujufo@zoehed.mh",

"fakmir@cebfirvot.pm"

]

}

* 1
* 2
* 3
* 4

----------------------------------------------------------------------

74. Which of the following is true regarding partial indexes?

1. Partial indexes represent a superset of the functionality of sparse indexes.
2. Partial indexes can be used to reduce the number of keys in an index.
3. Partial indexes support compound indexes.

The following is not true:

- Partial indexes don't support a uniqueness constraint.

No, you can still specify a uniqueness constraint with a partial index. However, uniqueness will be limited to the keys covered by the partial filter expression.

----------------------------------------------------------------------

75. Which other type of index is mostly closely related to text indexes?

* Single-key indexes
* Compound indexes
* Multi-key indexes [TRUE]
* Partial indexes

----------------------------------------------------------------------------------------------

76. Which of the following statements are true regarding collations on indexes?

* MongoDB only allows collations to be defined at collection level
* Collations allow the creation of case insensitive indexes. [TRUE]
* Creating an index with a different collation from the base collection implies overriding the base collection collation.
* We can define specific collations in an index. [TRUE]

-------------------------------------------------------------------------------------------------

77. Given the following index:

{ "first\_name": 1, "address.state": -1, "address.city": -1, "ssn": 1 }

Which of the following queries are able to use it for both filtering and sorting?

1. db.people.find({ "address.city": "West Cindy" }).sort({ "address.city": -1 })
2. - db.people.find({ "first\_name": "Jessica" }).sort({ "address.state": 1, "address.city": 1 }) [TRUE]
3. db.people.find({ "first\_name": { $gt: "J" } }).sort({ "address.city": -1 })
4. db.people.find({ "first\_name": "Jessica", "address.state": { $lt: "S"} }).sort({ "address.state": 1 }) [TRUE]
5. db.people.find({ "address.state": "South Dakota", "first\_name": "Jessica" }).sort({ "address.city": -1 })

---------------------------------------------------------------------------------------------------------------------

78. In this lab you're going to examine several examples queries and determine which compound index will best service them

> db.people.find({

"address.state": "Nebraska",

"last\_name": /^G/,

"job": "Police officer"

})

> db.people.find({

"job": /^P/,

"first\_name": /^C/,

"address.state": "Indiana"

}).sort({ "last\_name": 1 })

> db.people.find({

"address.state": "Connecticut",

"birthday": {

"$gte": ISODate("2010-01-01T00:00:00.000Z"),

"$lt": ISODate("2011-01-01T00:00:00.000Z")

}

})

79. If you had to build one index on the people collection, which of the following indexes would best sevice all 3 queries?

1. { "address.state": 1, "job": 1, "first\_name": 1 }
2. { "address.state": 1, "job": 1 }
3. { "job": 1, "address.state": 1 }
4. { "address.state": 1, "last\_name": 1, "job": 1 } [TRUE]
5. { "job": 1, "address.state": 1, "last\_name": 1 }
6. { "job": 1, "address.state": 1, "first\_name": 1 }

----------------------------------------------------------------------------------------

80. Regarding the performance implications, which of the following statements are correct?

1. MongoDB does not benefit from adding more RAM to your servers.
2. Disk RAID architecture can impact the performance of your MongoDB deployment.
3. Switching from HDDs to SSDs does not bring a benefit to the performance of MongoDB.
4. CPU availability impacts the performance of MongoDB, especially if using the WiredTiger storage engine.

---------------------------------------------------------------------------------------

81. Which of the following statements is true?

1. Some expressions can only be used in certain stages.
2. Only one expression per stage can be used.
3. An aggregation pipeline is an array of stages.

---------------------------------------------------------------------------------------

82. Which of the following is true about pipelines and the Aggregation Framework?

1. Stages cannot be configured to produce our desired output.
2. Documents flow through the pipeline, passing from one stage to the next
3. The Aggregation Framework provides us many stages to filter and transform our data
4. Pipelines must consist of at least two stages.

---------------------------------------------------------------------------------------

83. Which of the following statements is true?

1. Some expressions can only be used in certain stages.
2. Only one expression per stage can be used.
3. An aggregation pipeline is an array of stages.

----------------------------------------------------------------------------------------

84. Help MongoDB pick a movie our next movie night! Based on employee polling, we have decided that potential movies must meet the following criteria.

1. imdb.rating is at least 7
2. genres does not contain "Crime" or "Horror"
3. rated is either "PG" or "G"
4. languages contains "English" and "Japanese"
5. Assign the aggregation to a variable named pipeline, like

Assign the aggregation to a variable named pipeline, like:

- var pipeline = [ { $match: { ... } } ]

As a hint, your aggregation should return 23 documents. You can verify this by typing db.movies.aggregate(pipeline).itcount()

- Load validateLab1.js into mongo shell

- load('validateLab1.js')

And run the validateLab1 validation method

validateLab1(pipeline)

----------------------------------------------------------------------------------------

85. Which of the following statements are true of the $project stage?

1. Once we specify a field to retain or perform some computation in a $project stage, we must specify all fields we wish to retain. The only exception to this is the \_id field.
2. Beyond simply removing and retaining fields, $project lets us add new fields.
3. $project can only be used once within an Aggregation pipeline.
4. $project cannot be used to assign new values to existing fields.

The correct answers are the following:

Once we specify a field to retain or perform some computation in a $project stage, we must specify all fields we wish to retain. The only exception to this is the \_id field.

$project implicitly removes all other fields once we have retained, reshaped, or computed a new field. The exception to this is the \_id field, which we must explicitly remove.

Beyond simply removing and retaining fields, $project lets us add new fields.

We can add new fields and reassign the values of existing ones, shaping the documents into different datastructures and computing values using expressions.

The remaining options are incorrect.

---------------------------------------------------------------------------------------

86. Which of the following statements are true of the $project stage?

1. Once we specify a field to retain or perform some computation in a $project stage, we must specify all fields we wish to retain. The only exception to this is the \_id field.
2. Beyond simply removing and retaining fields, $project lets us add new fields.
3. $project can only be used once within an Aggregation pipeline.
4. $project cannot be used to assign new values to existing fields.

---------------------------------------------------------------------------------------

87. We'd like to calculate how many movies every cast member has been in and get an average imdb.rating for each cast member.

What is the name, number of movies, and average rating (truncated to one decimal) for the cast member that has been in the most number of movies with English as an available language?

Provide the input in the following order and format-

{ "\_id": "First Last", "numFilms": 1, "average": 1.1 }

We start with a familiar $match stage, looking for movies that include "English" as a language

{

$match: {

languages: "English"

}

},

Next, we use a $project stage, keeping only the data necessary for the aggregation stages that follow

{$project: { \_id: 0, cast: 1, "imdb.rating": 1 }}

$unwind follows next, extracting every entry in the cast array and creating a document for each one

{$unwind: "$cast" }

Our $group stage groups cast members together by their name, totals the number of documents, and gets the average imdb.rating

{

$group: {

\_id: "$cast",

numFilms: { $sum: 1 },

average: { $avg: "$imdb.rating" }

}

}

We then use a $project stage to truncate the imdb.rating to one decimal. This is done by first multiplying by 10, truncating the number, then dividing by 10

{

$project: {

numFilms: 1,

average: {

$divide: [

{ $trunc: { $multiply: ["$average", 10] } }

, 10

]

}

}

}

Lastly, we $sort in descending order so the result with the greatest number of movies comes first, and then $limit our result to 1 document, giving the expected answer

{ "\_id" : "John Wayne", "numFilms" : 107, "average" : 6.4 }

------------------------------------------------------------------------------------------------------------

88. Which of the following statements is true about the $lookup stage?

1. Specifying an existing field name to as will overwrite the the existing field
2. $lookup matches between localField and foreignField with an equality match
3. You can specify a collection in another database to from
4. The collection specified in from cannot be sharded

-------------------------------------------------------------------------------------------------------------

89. Which alliance from air\_alliances flies the most routes with either a Boeing 747 or an Airbus A380 (abbreviated 747 and 380 in air\_routes)?

We begin by aggregating over our air\_routes collection to allow for filtering of documents containing the string "747" or "380". If we started from air\_alliances we would have to do this after the lookup!

{

$match: {

airplane: /747|380/

}

},

Next, we use the $lookup stage to match documents from air\_alliances on the value of their airlines field against the current document's airline.name field

{

$lookup: {

from: "air\_alliances",

foreignField: "airlines",

localField: "airline.name",

as: "alliance"

}

},

We then use $unwind on the alliance field we created in $lookup, creating a document with each entry in alliance

{$unwind: "$alliance"},

We end with a $group and $sort stage, grouping on the name of the alliance and counting how many times it appeared

{

$group: {

\_id: "$alliance.name",

count: { $sum: 1 }

}

},

{

$sort: { count: -1 }

}

This produces the following output

1. { "\_id" : "SkyTeam", "count" : 16 }
2. { "\_id" : "Star Alliance", "count" : 11 }
3. { "\_id" : "OneWorld", "count" : 11 }

--------------------------------------------------------------------------------------------------------------

90. Which of the following statements apply to $graphLookup operator? check all that apply

1. - $graphLookup is a new stage of the aggregation pipeline introduced in MongoDB 3.2 [MongoDB 3.4 provide $graphLookup]
2. $lookup and $graphLookup stages require the exact same fields in their specification.
3. $graphLookup depends on $lookup operator. Cannot be used without $lookup
4. - $graphLookup provides MongoDB a transitive closure implementation [True]
5. - Provides MongoDB with graph or graph-like capabilities [True]

--------------------------------------------------------------------------------------------------------------

91. Which of the following statements is/are correct? Check all that apply.

1. as determines a collection where $graphLookup will store the stage results
2. connectToField will be used on recursive find operations [True]
3. connectFromField value will be use to match connectToField in a recursive match [True]
4. startWith indicates the index that should be use to execute the recursive match

-----------------------------------------------------------------------------------------------------------

92. Which of the following statements are incorrect? Check all that apply

1. depthField determines a field, in the result document, which specifies the number of recursive lookup needed to reach that document
2. depthField determines a field, which contains the value number of documents matched by the recursive lookup [TRUE]
3. maxDepth only takes $long values [TRUE]
4. maxDepth allows to specify the number of recursive lookups

------------------------------------------------------------------------------------------------------------

93. ``$graphLookup`` is required to be the last element on the pipeline.

1. This is correct because $graphLookup pipes out the results of recursive search into a collection, similar to $out stage.
2. This is incorrect. $graphLookup can be used in any position of the pipeline and acts in the same way as a regular $lookup. [TRUE]
3. This is correct because of the recursive nature of $graphLookup we want to save resources for last.
4. This is incorrect. graphLookup needs to be the first element of the pipeline, regardless of other stages needed to perform the desired query.

-----------------------------------------------------------------------------------------------------------

94. Now that you have been introduced to $graphLookup, let's use it to solve an interesting need. You are working for a travel agency and would like to find routes for a client! For this exercise, we'll be using the air\_airlines, air\_alliances, and air\_routes collections in the aggregations database.

The air\_airlines collection will use the following schema:

{

"\_id" : ObjectId("56e9b497732b6122f8790280"),

"airline" : 4,

"name" : "2 Sqn No 1 Elementary Flying Training School",

"alias" : "",

"iata" : "WYT",

"icao" : "",

"active" : "N",

"country" : "United Kingdom",

"base" : "HGH"

}

The air\_routes collection will use this schema:

{

"\_id" : ObjectId("56e9b39b732b6122f877fa31"),

"airline" : {

"id" : 410,

"name" : "Aerocondor",

"alias" : "2B",

"iata" : "ARD"

},

"src\_airport" : "CEK",

"dst\_airport" : "KZN",

"codeshare" : "",

"stops" : 0,

"airplane" : "CR2"

}

Finally, the air\_alliances collection will show the airlines that are in each alliance, with this schema:

{

"\_id" : ObjectId("581288b9f374076da2e36fe5"),

"name" : "Star Alliance",

"airlines" : [

"Air Canada",

"Adria Airways",

"Avianca",

"Scandinavian Airlines",

"All Nippon Airways",

"Brussels Airlines",

"Shenzhen Airlines",

"Air China",

"Air New Zealand",

"Asiana Airlines",

"Brussels Airlines",

"Copa Airlines",

"Croatia Airlines",

"EgyptAir",

"TAP Portugal",

"United Airlines",

"Turkish Airlines",

"Swiss International Air Lines",

"Lufthansa",

"EVA Air",

"South African Airways",

"Singapore Airlines"

]

}

Determine the approach that satisfies the following question in the most efficient manner:

Find the list of all possible distinct destinations, with at most one layover, departing from the base airports of airlines that make part of the "OneWorld" alliance. The airlines should be national carriers from Germany, Spain or Canada only. Include both the destination and which airline services that location. As a small hint, you should find 158 destinations.

Answer - db.air\_alliances.aggregate([{

$match: { name: "OneWorld" }

}

It then iterates, with maxDepth 0 on the air\_airlines collection to collect the details on the airlines, specially their base airport, but restricting that $lookup to airlines of the requested countries [Spain, Germany, Canada], using restrictSearchWithMatch.

{

$graphLookup: {

startWith: "$airlines",

from: "air\_airlines",

connectFromField: "name",

connectToField: "name",

as: "airlines",

maxDepth: 0,

restrictSearchWithMatch: {

country: { $in: ["Germany", "Spain", "Canada"] }

}

}

}

We then iterate over all routes up to maximum of one layover by setting our maxDepth to 1. We find all possible destinations when departing from the base airport of each carrier by specify $airlines.base in startWith

{

$graphLookup: {

startWith: "$airlines.base",

from: "air\_routes",

connectFromField: "dst\_airport",

connectToField: "src\_airport",

as: "connections",

maxDepth: 1

}

}

We now have a document with a field named connections that is an array of all routes that are within 1 layover. We use a $project here to remove unnecessary information from the documents. We also need to include information about valid airlines that match our initial restriction and the name of the current airline.

{

$project: {

validAirlines: "$airlines.name",

"connections.dst\_airport": 1,

"connections.airline.name": 1

}

}

After this, we'll unwind our connections array, and then use $project to add a field representing whether this particular route is valid, meaning it is a route flown by one of our desired carriers.

{ $unwind: "$connections" },

{

$project: {

isValid: {

$in: ["$connections.airline.name", "$validAirlines"]

},

"connections.dst\_airport": 1

}

}

Lastly, we use $match to filter out invalid routes, and then $group them on the destination.

{ $match: { isValid: true } },

{

$group: {

\_id: "$connections.dst\_airport"

}

}

An important aspect to this pipeline is that the first $graphLookup will act as a regular $lookup since we are setting a maxDepth to zero. The reason why we are taking this approach is due to the match restriction that $graphLookup allows, which can make this stage more efficient. Think back to the earlier lab on $lookup, can you think of a way to simplify the aggregation using $graphLookup instead?

----------------------------------------------------------------------------------------

95. Which of the following aggregation pipelines are single facet queries?

[

{"$match": { "$text": {"$search": "network"}}},

{"$sortByCount": "$offices.city"}

] [TRUE]

[

{"$unwind": "$offices"},

{"$project": { "\_id": "$name", "hq": "$offices.city"}},

{"$sortByCount": "$hq"},

{"$sort": {"\_id":-1}},

{"$limit": 100}

] [TRUE]

[

{"$match": { "$text": {"$search": "network"}}},

{"$unwind": "$offices"},

{"$sort": {"\_id":-1}}

]

----------------------------------------------------------------------------------

96. Assuming that field1 is composed of double values(decimal form), ranging between 0 and Infinity, and field2 is of type string, which of the following stages are correct?

1. {'$bucket': { 'groupBy': '$field1', 'boundaries': [ "a", 3, 5.5 ]}} [FALSE] will generate inconsistent boundary type error. Boundaries are required to have the same type.
2. {'$bucket': { 'groupBy': '$field1', 'boundaries': [ 0.4, Infinity ]}} [FALSE] will generate a not matching branch, bucket, to place non matching documents. The default stage option would prevent such errors.
3. {'$bucket': { 'groupBy': '$field2', 'boundaries': [ "a", "asdas", "z" ], 'default': 'Others'}} [TRUE]

----------------------------------------------------------------------------------

97. Auto Bucketing will ...

1. given a number of buckets, try to distribute documents evenly accross buckets. [TRUE]
2. adhere bucket boundaries to a numerical series set by the granularity option. [TRUE]
3. randomly distributed documents accross arbitrarily defined bucket boundaries.
4. count only documents that contain the groupBy field defined in the documents.

----------------------------------------------------------------------------------------

98. Which of the following statement(s) apply to the $facet stage?

1. The $facet stage allows several sub-pipelines to be executed to produce multiple facets. [TRUE]
2. The $facet stage allows the application to generate several different facets with one single database request. [TRUE]
3. The output of the individual $facet sub-pipelines can be shared using the expression $$FACET.$.
4. We can only use facets stages ($sortByCount, $bucket and $bucketAuto) as sub-pipelines of $facet stage. [FALSE] individual also can use

-----------------------------------------------------------------------------------------

99. How many movies are in both the top ten highest rated movies according to the imdb.rating and the metacritic fields? We should get these results with exactly one access to the database.

Hint: What is the intersection?

We begin with a $match and $project stage to only look at documents with the relevant fields, and project away needless information

{

$match: {

metacritic: { $gte: 0 },

"imdb.rating": { $gte: 0 }

}

},

{

$project: {

\_id: 0,

metacritic: 1,

imdb: 1,

title: 1

}

},

Next follows our $facet stage. Within each facet, we need sort in descending order for metacritic and imdb.ratting and ascending for title, limit to 10 documents, then only retain the title

{

$facet: {

top\_metacritic: [

{

$sort: {

metacritic: -1,

title: 1

}

},

{

$limit: 10

},

{

$project: {

title: 1

}

}

],

top\_imdb: [

{

$sort: {

"imdb.rating": -1,

title: 1

}

},

{

$limit: 10

},

{

$project: { title: 1 }

}

]

}

},

Lastly, we use a $project stage to find the intersection of top\_metacritic and top\_imdb, producing the titles of movies in both categories

{

$project: {

movies\_in\_both: {

$setIntersection: ["$top\_metacritic", "$top\_imdb"]

}

}

}

This results in the following output

{ "movies\_in\_both" : [ { "title" : "The Godfather" } ] }

------------------------------------------------------------------------------------

100. With regards to aggregation performance, which of the following are true?

1. You can increase index usage by moving $match stages to the end of your pipeline
2. Passing allowDiskUsage to your aggregation queries will seriously increase their performance
3. - When $limit and $sort are close together a very performant top-k sort can be performed [TRUE]
4. Transforming data in a pipeline stage prevents us from using indexes in the stages that follow [TRUE] Yes, this is true. That's why it's important to put all your index using operators at the front of your pipelines!

-------------------------------------------------------------------------------------

101. Which of the following statements is true regarding the $out stage?

1. $out removes all indexes when it overwrites a collection. [FALSE]
2. $out will overwrite an existing collection if specified. [TRUE]
3. Using $out within many sub-piplines of a $facet stage is a quick way to generate many differently shaped collections. [FALSE] can not use with $facet
4. If a pipeline with $out errors, you must delete the collection specified to the $out stage. [FALSE] no new collection will be created so no deletion

--------------------------------------------------------------------------------------

102. Which of the following statements are true regarding MongoDB Views?

1. Views should be used cautiously because the documents they contain can grow incredibly large. [Views contain no documents, they are stored aggregations that run when queried.] [FALE]
2. A view cannot be created that contains both horizontal and vertical slices. [Views are read-only and contain no information themselves. The documents "in" a view are simply the result of the definining pipeline being executed.] [FALSE]
3. View performance can be increased by creating the appropriate indexes on the source collection. [TRUE]
4. - Inserting data into a view is slow because MongoDB must perform the pipeline in reverse. [FALSE]

--------------------------------------------------------------------------------------

103. What operators will cause a merge stage on the primary shard for a database?

1. - $lookup [TRUE]
2. $group
3. - $out [TRUE]

---------------------------------------------------------------------------------------

104. Which of the following statements is/are true?

1. The Aggregation Framework will automatically reorder stages in certain conditions.
2. Causing a merge in a sharded deployment will cause all subsequent pipeline stages to be performed in the same location as the merge.
3. The Aggregation Framework can automatically project fields if the shape of the final document is only dependent upon those fields in the input document. [TRUE]
4. The query in a $match stage can be entirely covered by an index. [TRUE]

----------------------------------------------------------------------------------------

105. Consider the following aggregation pipelines:

Pipeline 1

db.coll.aggregate([

{"$match": {"field\_a": {"$gt": 1983}}},

{"$project": { "field\_a": "$field\_a.1", "field\_b": 1, "field\_c": 1 }},

{"$replaceRoot":{"newRoot": {"\_id": "$field\_c", "field\_b": "$field\_b"}}},

{"$out": "coll2"},

{"$match": {"\_id.field\_f": {"$gt": 1}}},

{"$replaceRoot":{"newRoot": {"\_id": "$field\_b", "field\_c": "$\_id"}}}

])

Pipeline 2

db.coll.aggregate([

{"$match": {"field\_a": {"$gt": 111}}},

{"$geoNear": {

"near": { "type": "Point", "coordinates": [ -73.99279 , 40.719296 ] },

"distanceField": "distance"}},

{"$project": { "distance": "$distance", "name": 1, "\_id": 0 }}

])

Pipeline 3

db.coll.aggregate([

{

"$facet": {

"averageCount": [

{"$unwind": "$array\_field"},

{"$group": {"\_id": "$array\_field", "count": {"$sum": 1}}}

],

"categorized": [{"$sortByCount": "$arrayField"}]

},

},

{

"$facet": {

"new\_shape": [{"$project": {"range": "$categorized.\_id"}}],

"stats": [{"$match": {"range": 1}}, {"$indexStats": {}}]

}

}

])

1. - Pipeline 1 fails since $out is required to be the last stage of the pipeline [TRUE]
2. Pipeline 1 is incorrect because you can only have one $replaceRoot stage in your pipeline
3. Pipeline 3 fails since you can only have one $facet stage per pipeline
4. Pipeline 2 is incorrect because $geoNear needs to be the first stage of our pipeline [TRUE]
5. Pipeline 2 fails because we cannot project distance field
6. Pipeline 3 executes correctly
7. Pipeline 3 fails because $indexStats must be the first stage in a pipeline and may not be used within a $facet [TRUE]

Answer - are below

The correct statements are the following:

Pipeline 3 fails because $indexStats must be the first stage in a pipeline and may not be used within a $facet

$indexStats must be the first stage in an aggregation pipeline and cannot be used within a $facet stage.

Pipeline 1 fails since $out is required to be the last stage of the pipeline

$out is required to be the last stage of the pipeline.

Pipeline 2 is incorrect because $geoNear needs to be the first stage of our pipeline

$geoNear is required to be the first stage of a pipeline.

All other statements are incorrect.

---------------------------------------------------------------------------------------------------------------------

106. Consider the following collection:

db.collection.find()

{ "a": [1, 34, 13] }

Pipeline 1

db.collection.aggregate([

{"$match": { "a" : {"$sum": 1} }},

{"$project": { "\_id" : {"$addToSet": "$a"} }},

{"$group": { "\_id" : "", "max\_a": {"$max": "$\_id"} }}

])

Pipeline 2

db.collection.aggregate([

{"$project": { "a\_divided" : {"$divide": ["$a", 1]} }}

])

Pipeline 3

db.collection.aggregate([

{"$project": {"a": {"$max": "$a"}}},

{"$group": {"\_id": "$$ROOT.\_id", "all\_as": {"$sum": "$a"}}}

])

1. Pipeline 2 fails because the $divide operator only supports numeric types [TRUE]
2. Pipeline 1 will fail because $max can not operator on \_id field
3. Pipeline 1 is incorrect because you cannot use an accumulator expression in a $match stage. [TRUE]
4. Pipeline 3 is correct and will execute with no error. [TRUE]
5. Pipeline 2 is incorrect since $divide cannot operate over field expressions

The correct answers are the following:

**Pipeline 1** is incorrect because you cannot use an accumulator expression on $match stage.

We cannot use accumulator expressions within $match. Only query expressions are allowed within $match

**Pipeline 3** is correct and will execute with no error

This is correct. Although we may argue that $ROOT variable is totally unnecessary, since \_id field will be projected by default from the first $project stage of this pipeline, there are no observable errors with the use of this expression variable

**Pipeline 2** fails because $divide operator only supports numeric types

This is true, $divide operator will only supports expressions that represent numeric value types.

All the other statements are not true.

-----------------------------------------------------------------------------

107. Consider the following collection documents:

db.people.find()

* { "\_id" : 0, "name" : "Bernice Pope", "age" : 69, "date" : ISODate("2017-10-04T18:35:44.011Z") }
* { "\_id" : 1, "name" : "Eric Malone", "age" : 57, "date" : ISODate("2017-10-04T18:35:44.014Z") }
* { "\_id" : 2, "name" : "Blanche Miller", "age" : 35, "date" : ISODate("2017-10-04T18:35:44.015Z") }
* { "\_id" : 3, "name" : "Sue Perez", "age" : 64, "date" : ISODate("2017-10-04T18:35:44.016Z") }
* { "\_id" : 4, "name" : "Ryan White", "age" : 39, "date" : ISODate("2017-10-04T18:35:44.019Z") }
* { "\_id" : 5, "name" : "Grace Payne", "age" : 56, "date" : ISODate("2017-10-04T18:35:44.020Z") }
* { "\_id" : 6, "name" : "Jessie Yates", "age" : 53, "date" : ISODate("2017-10-04T18:35:44.020Z") }
* { "\_id" : 7, "name" : "Herbert Mason", "age" : 37, "date" : ISODate("2017-10-04T18:35:44.020Z") }
* { "\_id" : 8, "name" : "Jesse Jordan", "age" : 47, "date" : ISODate("2017-10-04T18:35:44.020Z") }
* { "\_id" : 9, "name" : "Hulda Fuller", "age" : 25, "date" : ISODate("2017-10-04T18:35:44.020Z") }

And the aggregation pipeline execution result:

db.people.aggregate(pipeline)

1. { "\_id" : 8, "names" : [ "Sue Perez" ], "word" : "P" }
2. { "\_id" : 9, "names" : [ "Ryan White" ], "word" : "W" }
3. { "\_id" : 10, "names" : [ "Eric Malone", "Grace Payne" ], "word" : "MP" }
4. { "\_id" : 11, "names" : [ "Bernice Pope", "Jessie Yates", "Jesse Jordan", "Hulda Fuller" ], "word" : "PYJF" }
5. { "\_id" : 12, "names" : [ "Herbert Mason" ], "word" : "M" }
6. { "\_id" : 13, "names" : [ "Blanche Miller" ], "word" : "M" }

Which of the following pipelines generates the output result?

var pipeline = [{

"$sort": { "date": 1 }

},

{

"$group": {

"\_id": { "$size": { "$split": ["$name", " "]} },

"names": {"$push": "$name"}

}

},

{

"$project": {

"word": {

"$zip": {

"inputs": ["$names"],

"useLongestLength": false,

}

},

"names": 1

}

}]

--

var pipeline = [{

"$project": {

"surname": { "$arrayElemAt": [ {"$split": [ "$name", " " ] }, 1]},

"name\_size": { "$add" : [{"$strLenCP": "$name"}, -1]},

"name":1

}

},

{

"$group": {

"\_id": "$name\_size",

"word": { "$addToSet": {"$substr": [{"$toUpper":"$name"}, 3, 2]} },

"names": {"$push": "$surname"}

}

},

{

"$sort": {"\_id": -1}

}

]

--

var pipeline = [{

"$project": {

"surname\_capital": { "$substr": [{"$arrayElemAt": [ {"$split": [ "$name", " " ] }, 1]}, 0, 1 ] },

"name\_size": { "$add" : [{"$strLenCP": "$name"}, -1]},

"name": 1

}

},

{

"$group": {

"\_id": "$name\_size",

"word": { "$push": "$surname\_capital" },

"names": {"$push": "$name"}

}

},

{

"$project": {

"word": {

"$reduce": {

"input": "$word",

"initialValue": "",

"in": { "$concat": ["$$value", "$$this"] }

}

},

"names": 1

}

},

{

"$sort": { "\_id": 1}

}

] [TRUE]

Answer Explained Below

For this lab we picked the first letter of each person surname, surname\_capital, by splitting the name into an array

{"$split": [ "$name", " " ] }

And by gathering the first letter of the surname using $substr and $arrayElemAt:

{ "$substr": [{"$arrayElemAt": [ {"$split": [ "$name", " " ] }, 1]}, 0, 1 ] }

We've also captured the number of all alphanumeric characters of the name field, except " ":

"name\_size": { "$add" : [{"$strLenCP": "$name"}, -1]}

After grouping all first capital letters into word array, and all name into names values by the name\_size:

{

"$group": {

"\_id": "$name\_size",

"word": { "$push": "$surname\_capital" },

"names": {"$push": "$name"}

}

},

We then $reduced the resulting word array into a single string:

{

"$project": {

"word": {

"$reduce": {

"input": "$word",

"initialValue": "",

"in": { "$concat": ["$$value", "$$this"] }

}

},

"names": 1

}

}

And finally sort the result:

{ "$sort": { "\_id": 1} }

-------------------------------------------------------------------------------------

108. $facet is an aggregation stage that allows for sub-pipelines to be executed.

var pipeline = [

{

$match: { a: { $type: "int" } }

},

{

$project: {

\_id: 0,

a\_times\_b: { $multiply: ["$a", "$b"] }

}

},

{

$facet: {

facet\_1: [{ $sortByCount: "a\_times\_b" }],

facet\_2: [{ $project: { abs\_facet1: { $abs: "$facet\_1.\_id" } } }],

facet\_3: [

{

$facet: {

facet\_3\_1: [{ $bucketAuto: { groupBy: "$\_id", buckets: 2 } }]

}

}

]

}

}

]

In the above pipeline, which uses $facet, there are some incorrect stages or/and expressions being used.

Which of the following statements point out errors in the pipeline?

1. $sortByCount cannot be used within $facet stage.
2. - can not nest a $facet stage as a sub-pipeline. [TRUE]
3. a $type expression does not take a string as its value; only the BSON numeric values can be specified to identify the types.
4. facet\_2 uses the output of a parallel sub-pipeline, facet\_1, to compute an expression [TRUE]
5. a $multiply expression takes a document as input, not an array.

Detail - The following options are not true:

**a $multiply expression takes a document as input, not an array.**

This is not true, a $multiply expression does take as input an array of expressions.

**a $type expression does not take a string as its value; only the BSON numeric values can be specified to identify the types**.

We can use either the numeric BSON representation, as well as a string alias to evaluate a field type.

**$sortByCount cannot be used within $facet stage.**

$facet does accept $sortByCount as a sub-pipeline stage.

The correct answers, that reflect problems with the pipeline, are the following:

**can not nest a $facet stage as a sub-pipeline.**

This is correct. $facet does not accept all sub-pipelines that include other $facet stages

facet\_2 uses the output of a parallel sub-pipeline, facet\_1, to compute an expression

Each sub-pipeline are completely independent of one another. The output of one sub-pipeline cannot be used as the input for different sub-pipelines.

-----------------------------------------------------------------------------------------------------------

109. Consider a company producing solar panels and looking for the next markets they want to target in the USA.

We have a collection with all the major cities (more than 100,000 inhabitants) from all over the World with recorded number of sunny days for some of the last years.

A sample document looks like the following:

db.cities.findOne()

{

"\_id": 10,

"city": "San Diego",

"region": "CA",

"country": "USA",

"sunnydays": [220, 232, 205, 211, 242, 270]

}

The collection also has these indexes:

db.cities.getIndexes()

[

{

"v": 2,

"key": {

"\_id": 1

},

"name": "\_id\_",

"ns": "test.cities"

},

{

"v": 2,

"key": {

"city": 1

},

"name": "city\_1",

"ns": "test.cities"

},

{

"v": 2,

"key": {

"country": 1

},

"name": "country\_1",

"ns": "test.cities"

}

]

We would like to find the cities in the USA where the minimum number of sunny days is 200 and the average number of sunny days is at least 220. Lastly, we'd like to have the results sorted by the city's name.

The matching documents may or may not have a different shape than the initial one.

We have the following query:

var pipeline = [

{"$addFields": { "min": {"$min": "$sunnydays"}}},

{"$addFields": { "mean": {"$avg": "$sunnydays" }}},

{"$sort": {"city": 1}},

{"$match": { "country": "USA", "min": {"$gte": 200}, "mean": {"$gte": 220}}}

]

db.cities.aggregate(pipeline)

However, this pipeline execution can be optimized!

Which of the following choices is still going to produce the expected results and likely improve the most the execution of this aggregation pipeline?

var pipeline = [

{"$sort": {"city": 1}},

{"$addFields": { "min": {"$min": "$sunnydays"}}},

{"$addFields": { "mean": {"$avg": "$sunnydays" }}},

{"$match": { "country": "USA", "min": {"$gte": 200}, "mean": {"$gte": 220}}}

]

var pipeline = [

{"$sort": {"city": 1}},

{"$addFields": { "min": {"$min": "$sunnydays"}}},

{"$match": { "country": "USA", "min": {"$gte": 200}}}

]

var pipeline = [

{"$sort": {"city": 1}},

{"$match": { "country": "USA"}},

{"$addFields": { "min": {"$min": "$sunnydays"}}},

{"$match": { "min": {"$gte": 200}, "mean": {"$gte": 220}}},

{"$addFields": { "mean": {"$avg": "$sunnydays" }}}

]

var pipeline = [

{"$match": { "country": "USA"}},

{"$sort": {"city": 1}},

{"$addFields": { "min": {"$min": "$sunnydays"}}},

{"$match": { "min": {"$gte": 200}, "mean": {"$gte": 220}}},

{"$addFields": { "mean": {"$avg": "$sunnydays" }}}

]

var pipeline = [

{"$match": { "country": "USA"}},

{"$addFields": { "mean": {"$avg": "$sunnydays"}}},

{"$match": { "mean": {"$gte": 220}, "sunnydays": {"$not": {"$lt": 200 }}}},

{"$sort": {"city": 1}}

] [TRUE]

In this case, we try to remove as much data as possible upfront, all cities not matching the right country, using the available index.

We then calculate the mean number of sunny days.

The $match stage then filters out documents where the mean isn't greater than or equal to 220, and there are no entries in the sunnydays vector less than 200.

We are left with a sort in memory, however the number should be small enough to not take much resources. There are 285 cities with 100,000 habitants in the USA, and some are likely not to match the number of sunny days criteria.

-------------------------------------------------------------------------------------------------------------------------

110.Consider the following people collection:

db.people.find().limit(5)

* { "\_id" : 0, "name" : "Iva Estrada", "age" : 95, "state" : "WA", "phone" : "(739) 557-2576", "ssn" : "901-34-4492" }
* { "\_id" : 1, "name" : "Roger Walton", "age" : 92, "state" : "ID", "phone" : "(948) 527-2370", "ssn" : "498-61-9106" }
* { "\_id" : 2, "name" : "Isaiah Norton", "age" : 26, "state" : "FL", "phone" : "(344) 479-5646", "ssn" : "052-49-6049" }
* { "\_id" : 3, "name" : "Tillie Salazar", "age" : 88, "state" : "ND", "phone" : "(216) 414-5981", "ssn" : "708-26-3486" }
* { "\_id" : 4, "name" : "Cecelia Wells", "age" : 16, "state" : "SD", "phone" : "(669) 809-9128", "ssn" : "977-00-7372" }

And the corresponding people\_contacts view:

db.people\_contacts.find().limit(5)

1. { "\_id" : 6585, "name" : "Aaron Alvarado", "phone" : "(631)\*\*\*\*\*\*\*\*\*", "ssn" : "\*\*\*\*\*\*\*\*8014" }
2. { "\_id" : 8510, "name" : "Aaron Barnes", "phone" : "(944)\*\*\*\*\*\*\*\*\*", "ssn" : "\*\*\*\*\*\*\*\*6820" }
3. { "\_id" : 6441, "name" : "Aaron Barton", "phone" : "(234)\*\*\*\*\*\*\*\*\*", "ssn" : "\*\*\*\*\*\*\*\*1937" }
4. { "\_id" : 8180, "name" : "Aaron Coleman", "phone" : "(431)\*\*\*\*\*\*\*\*\*", "ssn" : "\*\*\*\*\*\*\*\*7559" }
5. { "\_id" : 9738, "name" : "Aaron Fernandez", "phone" : "(578)\*\*\*\*\*\*\*\*\*", "ssn" : "\*\*\*\*\*\*\*\*0211" }

Which of the of the following commands generates this people\_contacts view?

var pipeline = [

{

"$project": {"name":1,

"phone": {

"$concat": [

{"$arrayElemAt": [{"$split": ["$phone", " "]}, 0]} ,

"\*\*\*\*\*\*\*\*\*" ]

},

"ssn": {

"$concat": [

"\*\*\*\*\*\*\*\*",

{"$arrayElemAt": [{"$split": ["$ssn", "-"]}, 2]}

]

}

}

}

];

db.runCommand({

"create": "people\_contacts",

"viewOn":"people",

"pipeline": pipeline})

--------------------------

var pipeline = [

{

"$sort": {"name": 1}

},

{

"$project": {"name":1,

"phone": {

"$concat": [

{"$arrayElemAt": [{"$split": ["$phone", " "]}, 0]} ,

"\*\*\*\*\*\*\*\*\*" ]

},

"ssn": {

"$concat": [

"\*\*\*\*\*\*\*\*",

{"$arrayElemAt": [{"$split": ["$ssn", "-"]}, 2]}

]

}

}

}

];

db.createView("people", "people\_contacts" pipeline);

-------------------------------------------

var pipeline = [

{

"$sort": {"state": 1}

},

{

"$project": {"name":1,

"phone": {

"$concat": [

{"$arrayElemAt": [{"$split": ["$phone", " "]}, 0]} ,

"\*\*\*\*\*\*\*\*\*" ]

},

"ssn": {

"$concat": [

"\*\*\*\*\*\*\*\*",

{"$arrayElemAt": [{"$split": ["$ssn", "-"]}, 2]}

]

}

}

}

];

db.runCommand({

"create": "people",

"viewOn":"people",

"pipeline": pipeline})

-----------------------------

var pipeline = [

{

"$sort": {"name": 1}

},

{

"$project": {"name":1,

"phone": {

"$concat": [

{"$arrayElemAt": [{"$split": ["$phone", " "]}, 0]} ,

"\*\*\*\*\*\*\*\*\*" ]

},

"ssn": {

"$concat": [

"\*\*\*\*\*\*\*\*",

{"$arrayElemAt": [{"$split": ["$ssn", "-"]}, 2]}

]

}

}

}

];

db.createView("people\_contacts", "people", pipeline); [TRUE]

-----------------------------------------------------------------------------------------------

111. Using the air\_alliances and air\_routes collections, find which alliance has the most unique carriers(airlines)

operating between the airports JFK and LHR, in either directions.

Names are distinct, i.e. Delta != Delta Air Lines

src\_airport and dst\_airport contain the originating and terminating airport information.

* Star Alliance, with 6 carriers
* SkyTeam, with 4 carriers
* OneWorld, with 4 carriers [TRUE]
* OneWorld, with 8 carriers

db.air\_routes.aggregate([

{

$match: {

src\_airport: { $in: ["LHR", "JFK"] },

dst\_airport: { $in: ["LHR", "JFK"] }

}

},

{

$lookup: {

from: "air\_alliances",

foreignField: "airlines",

localField: "airline.name",

as: "alliance"

}

},

{

$match: { alliance: { $ne: [] } }

},

{

$addFields: {

alliance: { $arrayElemAt: ["$alliance.name", 0] }

}

},

{

$group: {

\_id: "$airline.id",

alliance: { $first: "$alliance" }

}

},

{

$sortByCount: "$alliance"

}

])

This produces the following output

{ "\_id": "OneWorld", "count": 4 }

{ "\_id": "SkyTeam", "count": 2 }

----------------------------------------------------------------------

112. Which of these are default configurations for mongod?

1. database files are stored in the directory /data/db/ [TRUE]
2. mongod listens on port 27017 [TRUE]
3. authentication is enabled [FALSE]
4. mongod can connect to local and remote clients [FALSE]

By default, mongod is only bound to localhost, or 127.0.0.1. This means that only local clients can connect - to change this, use the --bind\_ip flag.

-------------------------------------------------------------------------------------------------------------

113. Which of the following files in the MongoDB data directory can you access to view collection data?

1. The collection.wt file
2. The storage.bson file
3. The WiredTiger.wt file
4. None of the above [TRUE]

---------------------------------------------------------------------------------------------------------------

114. Which of the following methods executes a database command?

1. db.runCommand( { <COMMAND> } ) [TRUE]
2. db.executeCommand( { <COMMAND> } )
3. db.runThisCommand( { <COMMAND> } )
4. db.command( { <COMMAND> } )

----------------------------------------------------------------------------------------------------------------

115. Which of the following operations can be used to access the logs?

1. Running tail -f <path-to-log-file> from the command line. [TRUE]
2. Running db.adminCommand({ "getLog": "global" }) from the Mongo shell. [TRUE]
3. Running db.getLogComponents() from the Mongo shell. [This operation does not access the logs; instead it returns the current verbosity settings. The verbosity settings determine the amount of Log Messages that MongoDB produces for each log message component.]

------------------------------------------------------------------------------------------------------------------

116. Which of the following events are captured by the profiler?

1. Network timeouts
2. CRUD operations [TRUE]
3. WiredTiger storage operations [FALSE] [However, Network timeouts and WiredTiger storage data are not captured by the profiler - this data is stored in the logs instead.]
4. Administrative operations [TRUE]
5. Configuration operations [TRUE]

117. When should you deploy a MongoDB deployment with security enabled?

1. When deploying your staging environment. [TRUE]
2. When deploying a development environment. [TRUE]
3. When deploying an evaluation environment. [TRUE]
4. When deploying your production environment. [TRUE]

------------------------------------------------------------------------------------------------------------------

118. Which of the following actions are granted to the userAdmin built-in role?

1. viewUser [TRUE]
2. createIndex
3. dropRole [TRUE]
4. createRole [TRUE]
5. dropCollection

-------------------------------------------------------------------------------------------------------------------

119. Which of the following are true differences between mongoexport and mongodump?

1. Mongodump can create a data file and a metadata file, but mongoexport just creates a data file. [TRUE]
2. Mongoexport outputs BSON, but mongodump outputs JSON.
3. By default, mongoexport sends output to standard output, but mongodump writes to a file.[TRUE]
4. Mongodump outputs BSON, but mongoexport outputs JSON. [TRUE]
5. Mongoexport is typically faster than mongodump.

**Mongoexport outputs BSON, but mongodump outputs JSON.**

The reverse is true - mongoexport outputs JSON, and mongodump outputs BSON.

**Mongoexport is typically faster than mongodump.**

Mongoexport must convert every document from BSON to JSON. This takes much longer than mongodump, which simply outputs BSON.

-------------------------------------------------------------------------------------------------------------------------------

120. Which of the following are true about binary replication and statement-based replication?

1. MongoDB uses statement-based replication, not binary replication. [TRUE]
2. Statement-based replication is platform independent. [TRUE]
3. Binary replication is more accurate than statement-based replication. [FALSE]

Both methods of replication are accurate; however they do vary in speed and variability across operating systems.

--------------------------------------------------------------------------------------------------------

121. Which of the following are true for replica sets in MongoDB?

1. Replica sets provide high availability. [TRUE]
2. We should always use arbiters.
3. Replica set members have a fixed role assigned.
4. We can have up to 50 voting members in a replica set.

We can have up to 50 replica set members, but only 7 of those will be voting members.

122. Which of the following is/are true about setting up a replica set?

1. When connecting to a replica set, the mongo shell will redirect the connection to the primary node.
2. rs.initiate() must be run on every node in the replica set.
3. All nodes in a replica set must be run on the same port.
4. Enabling internal authentication in a replica set implicitly enables client authentication.

This is incorrect; rs.initiate() should only be run on one node in the replica set.

123. Which of the following fields are included in the replica set configuration document?

1. Members [TRUE]
2. \_id [TRUE]
3. Version [TRUE]

124. What information can be obtained from running rs.printReplicationInfo()?

1. The earliest statement entered in the oplog.
2. The time of the earliest entry in the oplog. [TRUE]
3. The last statement entered in the oplog.
4. The time of the latest entry in the oplog. [TRUE]
5. The current primary in the replica set.

Correct answers:

The time of the earliest entry in the oplog.

The time of the latest entry in the oplog.

rs.printReplicationInfo() gives us the times of the earliest and latest entries in the oplog.

Incorrect answers:

The earliest statement entered in the oplog.

The last statement entered in the oplog.

rs.printReplicationInfo() will only return timestamps for oplog statements; the statements themselves can be found in local.oplog.rs.

The current primary in the replica set.

rs.printReplicationInfo() only contains information pertaining to the node where the command was run.

125. Which of the following is true?

1. The oplog.rs collection contains all operations that will be replicated.
2. The local database will not be replicated.
3. The local database does not allow the creation of other collections.
4. You cannot write to the local database.
5. We should drop the oplog.rs collection from time to time to avoid it becoming too big.

126. Which of the following is true?

1. The oplog.rs collection contains all operations that will be replicated.
2. The local database will not be replicated.
3. The local database does not allow the creation of other collections.
4. You cannot write to the local database.
5. We should drop the oplog.rs collection from time to time to avoid it becoming too big.

We cap the oplog.rs collection instead of dropping it entirely.

127. Which of the following is true about reconfiguring a replica set with rs.reconfig()?

1. It does not require any of the configuration files to be updated. [TRUE]
2. It does not require any of the nodes to restarted. [TRUE]
3. It does not require the entire configuration document.

Correct answers:

It does not require any of the nodes to restarted.

When we reconfigure a replica set with rs.reconfig(), we do not need to restart any of the individual nodes.

It does not require any of the configuration files to be updated.

**When we reconfigure a replica set with rs.reconfig(), we do not need to update any of the nodes' configuration files.**

Incorrect answers:

It does not require the entire configuration document.

When issuing an updated configuration to rs.reconfig(), the entire configuration document is required.

128. Which of the following is true about reading and writing from secondaries?

1. Running rs.slaveOk() allows us to read and write from secondaries.
2. We have to run rs.slaveOk() before we can read from secondary nodes.
3. Connecting to the replica set will automatically connect to a secondary node.

129. Which of the following is true about elections?

1. Nodes with higher priority are more likely to be elected primary. [TRUE]
2. Elections can take place anytime while the primary is available.
3. All nodes have an equal chance to become primary.
4. Nodes with priority 0 cannot be elected primary. [TRUE]

130. Consider a 3-member replica set, where one secondary is offline. Which of the following write concern levels can still return successfully?

Attempts Remaining:∞Unlimited Attempts

1. Majority [TRUE]
2. all
3. online
4. 3

131. Evaluate the effect of using a write concern with a replica set where one node has failed.

Consider a 3-node replica set with only 2 healthy nodes, that receives the following insert() operation:

use payroll

db.employees.insert(

{ "name": "Aditya", "salary\_USD": 50000 },

{ "writeConcern": { "w": 3, "wtimeout": 1000 } }

)

Which of the following is true about this write operation?

1. The unhealthy node will receive the new document when it is brought back online. [TRUE]
2. If a writeConcernError occurs, the document is still written to the healthy nodes.[TRUE]
3. The write operation will always return with an error, even if wtimeout is not specified. [FALSE]
4. w: "majority" would also cause this write operation to return with an error. [FALSE]

132. Which of the following read concerns only return data from write operations that have been committed to a majority of nodes?

1. linearizable
2. local
3. majority
4. available

133. Which of the following read preference options may result in stale data?

1. secondary [TRUE]
2. primary
3. primaryPreferred [TRUE]
4. secondaryPreferred [TRUE]
5. nearest [TRUE]

The key concept to understand here is that when two nodes go down in a three-node replica set, the third node becomes a secondary regardless of whether it started as a primary.

Therefore, connecting to the third node is the same as connecting to a secondary node, and any readPreference will work except for primary, which requires all operations to read from the primary node.

-----------------------------------------------------------------------------------------------------------------

134.We should consider sharding when:

1. our organization outgrows the most powerful servers available, limiting our vertical scaling options. [TRUE]
2. our server disks are full.
3. we are holding more than 5TB per server and operational costs increase dramatically. [TRUE]
4. we start a new project with MongoDB.
5. government regulations require data to be located in a specific geography. [TRUE]

135. In a sharded cluster, collection metadata is stored in:

1. a random shard.
2. the configuration servers. [TRUE]
3. mongos.
4. the primary shard.

136. What is true about the mongos?

1. The mongos configuration file doesn't need to have a dbpath. [TRUE]
2. The mongos configuration file needs to specify the config servers. [TRUE]
3. Users must be created on mongos when auth is enabled.
4. The config server configuration files need to specify mongos.
5. The mongos configuration file doesn't need to have a port.

Mongos uses the data from the config servers, so it cannot function without communicating with the CSRS. Users must be created on mongos when auth is enabled [FALSE]. Mongos inherits its users from the config servers.

137. When should you manually write data to the Config DB?

1. When importing a new dataset
2. When removing a shard
3. When adding a shard
4. When directed to by MongoDB documentation or Support Engineers. [TRUE]
5. When sharding a collection

138. True or False: Shard keys are mutable.

1. False [TRUE]
2. True

139. Which of the following are indicators that a field or fields are a good shard key choice?

1. High Cardinality [TRUE]
2. Monotonic change
3. Low Frequency [TRUE]
4. Indexed
5. Non-monotonic change [TRUE]

Shard key fields must be indexed, so that doesn't necessarily mean that they are a good shard key choice.

140. Which of the following functions does Hashed Sharding support?

1. Fast sorts on the shard key
2. Targeted queries on a range of shard key values
3. Even distribution of a monotonically changing shard key field in a compound index
4. Even distribution of a monotonically changing shard key field [TRUE]

Sorts are slower on a hashed index than a normal index.

141. Which of the following is true about chunks?

1. Chunk ranges can never change once they are set.
2. Chunk ranges have an inclusive minimum and an exclusive maximum. [TRUE]
3. Documents in the same chunk may live on different shards.

142. Consider the following document:

{

"\_id" : ObjectId("573f7197f29313caab89b3a4"),

"sku" : 20005012,

"name" : "Complete Hit Singles A's & B's - CD",

"type" : "Music",

"regularPrice" : 14.99,

"salePrice" : 14.99,

"shippingWeight" : "0.25"

}

Which of the following chunks would contain this document?

1. Choose the best answer:

{

"\_id" : "m103.products-sku\_MinKey",

"shard" : "shard1",

"min" : {

"sku" : 0

},

"max" : {

"sku" : 5000000

}

}

{

"\_id" : "m103.products-sku\_5000000",

"shard" : "shard1",

"min" : {

"sku" : 5000000

},

"max" : {

"sku" : 10000000

}

}

{

"\_id" : "m103.products-sku\_10000000",

"shard" : "shard1",

"min" : {

"sku" : 10000000

},

"max" : {

"sku" : 15000000

}

}

{

"\_id" : "m103.products-sku\_15000000",

"shard" : "shard2",

"min" : {

"sku" : 15000000

},

"max" : {

"sku" : 20000000

}

}

1. [TRUE]

{

"\_id" : "m103.products-sku\_20000000",

"shard" : "shard2",

"min" : {

"sku" : 20000000

},

"max" : {

"sku" : 25000000

}

}

143. Given a sharded cluster running MongoDB 3.6, which of the shard components is responsible for running the Balancer process?

* Mongos
* Secondary of the Config Server Replica Set
* Primary of each Shard Replica Set
* Primary node of the Config Server Replica Set [TRUE]

144. For a find() operation, which cluster component is responsible for merging the query results?

* The primary member of the config server replica set
* None, the results are coming out in the right order from the shards
* The mongos that issued the query [TRUE]
* The primary member of each shard
* A randomly chosen shard in the cluster

145. Given a collection that is sharded on the following shard key: { "sku" : 1, "name" : 1 } Which of the following queries results in a targeted query?

* db.products.find( { "sku" : 1337 } ) [TRUE]
* db.products.find( { "name" : "MongoHacker" } ) [TRUE]
* db.products.find( { "sku" : 1337, "name" : "MongoHacker" } )
* db.products.find( { "name" : "MongoHacker", "sku" : 1337 } ) [TRUE]

146. Which of the following is required in order for a query to be targeted to a subset of shards?

* The query uses the shard key [TRUE]
* An index exists on the shard key [TRUE]
* The shards are running on same data center

**The query uses the shard key**

This is correct - in order for a query to be targeted to a subset of shards, the query must use the shard key. This is because the data itself is divided on the shard key, so without that parameter the server cannot locate data without doing a Scatter Gather query.

**An index exists on the shard key**

This is correct - in order for a query to be targeted to a subset of shards, an index must exist on the shard key. This is required before the collection can be sharded.

**The shards are running on same data center**

This is incorrect - shards are designed such that they can be distributed throughout the world.

147. Which of the following are valid command line instructions to start a mongod? You may assume that all specified files already exist.

* mongod --logpath /var/log/mongo/mongod.log --dbpath /data/db –fork [TRUE]
* mongod --dbpath /data/db --fork
* mongod --log /var/log/mongo/mongod.log --authentication
* mongod -f /etc/mongod.conf [TRUE]

--------------------------------------------------------------------------------------------------------------------------------

148. Given the following config file:

storage:

dbPath: /data/db

systemLog:

destination: file

path: /var/log/mongod.log

net:

bindIp: localhost,192.168.0.100

security:

keyFile: /var/pki/keyfile

processManagement:

fork: true

How many directories must MongoDB have access to? Disregard the path to the configuration file itself.

Choose the best answer:

3 [TRUE]

4

2

1

148. Given the following output from rs.status().members:

[

{

"\_id": 0,

"name": "localhost:27017",

"health": 1,

"state": 1,

"stateStr": "PRIMARY",

"uptime": 548,

"optime": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDate": ISODate("2018-03-14T14:47:51Z"),

"electionTime": Timestamp(1521038358, 2),

"electionDate": ISODate("2018-03-14T14:39:18Z"),

"configVersion": 2,

"self": true

},

{

"\_id": 1,

"name": "localhost:27018",

"health": 1,

"state": 2,

"stateStr": "SECONDARY",

"uptime": 289,

"optime": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDurable": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDate": ISODate("2018-03-14T14:47:51Z"),

"optimeDurableDate": ISODate("2018-03-14T14:47:51Z"),

"lastHeartbeat": ISODate("2018-03-14T14:47:56.558Z"),

"lastHeartbeatRecv": ISODate("2018-03-14T14:47:56.517Z"),

"pingMs": NumberLong("0"),

"syncingTo": "localhost:27022",

"configVersion": 2

},

{

"\_id": 2,

"name": "localhost:27019",

"health": 1,

"state": 2,

"stateStr": "SECONDARY",

"uptime": 289,

"optime": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDurable": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDate": ISODate("2018-03-14T14:47:51Z"),

"optimeDurableDate": ISODate("2018-03-14T14:47:51Z"),

"lastHeartbeat": ISODate("2018-03-14T14:47:56.558Z"),

"lastHeartbeatRecv": ISODate("2018-03-14T14:47:56.654Z"),

"pingMs": NumberLong("0"),

"syncingTo": "localhost:27022",

"configVersion": 2

},

{

"\_id": 3,

"name": "localhost:27020",

"health": 1,

"state": 2,

"stateStr": "SECONDARY",

"uptime": 289,

"optime": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDurable": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDate": ISODate("2018-03-14T14:47:51Z"),

"optimeDurableDate": ISODate("2018-03-14T14:47:51Z"),

"lastHeartbeat": ISODate("2018-03-14T14:47:56.558Z"),

"lastHeartbeatRecv": ISODate("2018-03-14T14:47:56.726Z"),

"pingMs": NumberLong("0"),

"syncingTo": "localhost:27022",

"configVersion": 2

},

{

"\_id": 4,

"name": "localhost:27021",

"health": 0,

"state": 8,

"stateStr": "(not reachable/healthy)",

"uptime": 0,

"optime": {

"ts": Timestamp(0, 0),

"t": NumberLong("-1")

},

"optimeDurable": {

"ts": Timestamp(0, 0),

"t": NumberLong("-1")

},

"optimeDate": ISODate("1970-01-01T00:00:00Z"),

"optimeDurableDate": ISODate("1970-01-01T00:00:00Z"),

"lastHeartbeat": ISODate("2018-03-14T14:47:56.656Z"),

"lastHeartbeatRecv": ISODate("2018-03-14T14:47:12.668Z"),

"pingMs": NumberLong("0"),

"lastHeartbeatMessage": "Connection refused",

"configVersion": -1

},

{

"\_id": 5,

"name": "localhost:27022",

"health": 1,

"state": 2,

"stateStr": "SECONDARY",

"uptime": 289,

"optime": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDurable": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDate": ISODate("2018-03-14T14:47:51Z"),

"optimeDurableDate": ISODate("2018-03-14T14:47:51Z"),

"lastHeartbeat": ISODate("2018-03-14T14:47:56.558Z"),

"lastHeartbeatRecv": ISODate("2018-03-14T14:47:55.974Z"),

"pingMs": NumberLong("0"),

"syncingTo": "localhost:27017",

"configVersion": 2

},

{

"\_id": 6,

"name": "localhost:27023",

"health": 1,

"state": 2,

"stateStr": "SECONDARY",

"uptime": 289,

"optime": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDurable": {

"ts": Timestamp(1521038871, 1),

"t": NumberLong("1")

},

"optimeDate": ISODate("2018-03-14T14:47:51Z"),

"optimeDurableDate": ISODate("2018-03-14T14:47:51Z"),

"lastHeartbeat": ISODate("2018-03-14T14:47:56.558Z"),

"lastHeartbeatRecv": ISODate("2018-03-14T14:47:56.801Z"),

"pingMs": NumberLong("0"),

"syncingTo": "localhost:27022",

"configVersion": 2

}

]

At this moment, how many replica set members are eligible to become primary in the event of the current Primary crashing or stepping down?

4, 7, 6 ,5 [TRUE]

149 . Given the following replica set configuration:

conf = {

"\_id": "replset",

"version": 1,

"protocolVersion": 1,

"members": [

{

"\_id": 0,

"host": "192.168.103.100:27017",

"priority": 2,

"votes": 1

},

{

"\_id": 0,

"host": "192.168.103.100:27018",

"priority": 1,

"votes": 1

},

{

"\_id": 2,

"host": "192.168.103.100:27018",

"priority": 1,

"votes": 1

}

]

}

What errors are present in the above replica set configuration?

* You cannot have three members in a replica set.
* You cannot specify the same host information among multiple members.
* You cannot specify two members with the same \_id.
* You can only specify a priority of 0 or 1, member "\_id": 0 is incorrectly configured.

150. Given the following replica set configuration:

conf = {

"\_id": "replset",

"version": 1,

"protocolVersion": 1,

"members": [

{

"\_id": 0,

"host": "localhost:27017",

"priority": 1,

"votes": 1

},

{

"\_id": 1,

"host": "localhost:27018",

"priority": 1,

"votes": 1

},

{

"\_id": 2,

"host": "localhost:27019",

"priority": 1,

"votes": 1

},

{

"\_id": 3,

"host": "localhost:27020",

"priority": 0,

"votes": 0,

"slaveDelay": 3600

}

]

}

What is the most likely role served by the node with "\_id": 3?

* It serves as a hidden secondary available to use for non-critical analysis operations.
* It serves as a "hot" backup of data in case of accidental data loss on the other members, like a DBA accidentally dropping the database.
* It serves as a reference to perform analytics on how data is changing over time.
* It serves to service reads and writes for people in the same geographic region as the host machine.

151. Given the following shard key: { "country": 1, "\_id": 1 }

Which of the following queries will be routed (targeted)? Remember that queries may be routed to more than one shard.

* db.customers.find({"\_id": 914, "country": "Sweden"})
* db.customers.find({"country": "Norway", "\_id": 54})
* db.customers.find({"country": { $gte: "Portugal", $lte: "Spain" }})
* db.customers.find({"\_id": 455})

**The correct answers are:**

db.customers.find({"country": "Norway", "\_id": 54})

This specifies both indexes used in the shard key.

db.customers.find({"country": { $gte: "Portugal", $lte: "Spain" }})

This specifies a prefix of the indexes used in the shard key, "country", and will be routed to shards containing the necessary information.

db.customers.find({"\_id": 914, "country": "Sweden"})

Although the indexes are specified in reverse order, this is a routed query. Any document matching {"\_id": 914, "country": "Sweden"} must be identical to {"country": "Sweden", "\_id": 914}. The query planner will take advantage of this and reorder the fields.

**The incorrect answer is:**

db.customers.find({"\_id": 455})

Because the neither a prefix nor the full shard key is provided, mongos has no way to determine how to appropriately route this query. Instead, it will send this query to all shards in the cluster in a scatter-gather operation.