First Name:
Last Name:
Group:
Date:

INTRODUCTION

1. MongoDB is an open-source document-oriented database classified as a NoSQL database, and differs from traditional relational databases because:

    a. MongoDB doesn't enable to JOIN documents from different collections.
    b. MongoDB is only used for development while relational databases are used for production environments.
    c. MongoDB scales horizontally much better than traditional relational databases and keeps a fairly good amount of features that makes it a general purpose database.
    d. None of the previous options are valid.

2. How is the data organized within MongoDB?

    a. MongoDB can hold up to one database containing many collections where documents are stored.
    b. MongoDB is able to handle multiple databases holding many collections and the documents are finally stored within them.
    c. MongoDB doesn't have collections, but databases where the documents are persisted.
    d. MongoDB stores data in tables that belong to only one database.

3. JSON is the way documents are represented in MongoDB. The mongoshell, as a javascript interpreter, uses JSON documents intensively to accomplish many different tasks such as CRUD operations, server configuration and so on. Which of the following assertions are correct (many of them might apply)?:

    a. The JSON format is also used to persist data on disk, which makes more optimal to deal with data.
    b. JSON documents allow modeling our business entities easily as it enables storing complex data types such as arrays and nested documents.
    c. JSON documents don't need to follow a schema and hence different documents from the same collection might have different fields.
    d. JSON documents cannot be bigger than 16MB.

4. In regards to how MongoDB can be deployed, it happens that:

    a. Stand-alone MongoDB deployments are the only valid choice for production environments.
    b. Replica sets are used to scale horizontally when the database is running out of space.
    c. Stand-alone MongoDB servers are typically used in development stages while Replica Sets and Sharded Cluster are preferred for production environments.
    d. Sharded Clusters are only recommended if the application needs to store documents bigger than 16MB.

5. Choose the correct sentence (only one applies):

    a. If a document is inserted without the _id field, the mongod process creates it automatically with a unique ObjectId as the value.
    b. If a document is inserted without the _id field, the driver creates it automatically with a unique ObjectId as the value.
    c. Documents cannot have an _id field.
    d. None of the above is correct.

CRUD

6. Consider the following sentences being executed in a `mongoshell` session and guess which answer is the right one:

```
use mongorules
db.ilikemongodb.drop()
db.ilikemongodb.insert({firstName: "Raul", lastName: "Marin", role:"teacher"})
db.ilikemongodb.insert({firstName: "Raul", lastName: "Marin", role:"teacher"})
db.ilikemongodb.insert({firstName: "Ruben", lastName: "Terceno", role:"teacher"})
```

    a. The second insert operation will fail because MongoDB doesn't allow having two documents with the same fields.
    b. None of the operations will fail, but the second insert will overwrite the first one and only two documents will persist.
    c. All the documents will be added, and hence the collection will have three documents.
    d. All the inserts will fail. The _id is a mandatory field that every document must specify, otherwise will raise an error.

7. Inserts will fail if… (only one applies):

    a. There is already a document in the collection with that _id.
    b. You try to assign an array to the _id.
    c. The argument is not a well-formed document.
    d. All the answers above are correct.

8. Based on the following commands in the `mongoshell`, which affirmations are true? (more than one can apply):

```
db.ilikemongodb.insertMany( [ { _id: "6748FHJ", brand: "Open", model: "Corsa" },
                              { _id: "6749FHJ", brand: "Open", model: "Corsa" },
                              { _id: 4389, companyName: "MongoDB" },
                              { _id: [1,2], CPU: "Intel", brand: "Dell" },
                              { _id: "6750FHJ", brand: "Open", model: "Corsa" } ],
                            { ordered: true } )
```

    a. The documents are sorted by `_id` before insertion.
    b. The third and four inserts will fail because their `_id` data types differ from the `_id` data types in the other operations.
    c. The fourth insert will fail because arrays are not allowed in the `_id` field.
    d. The last insert won't be executed because of the option `{ ordered: true }` prevent it due to a previous error.

9. When it comes to document deletion, how many commands are available in MongoDB?

    a. MongoDB doesn't allow deleting documents because the NoSQL paradigm is meant to prevent data loss.
    b. Documents in MongoDB can only be deleted one by one with the `deleteOneByOne` command to avoid "fat finger" errors.
    c. Documents can be deleted in MongoDB by using `deleteOne` and `deleteMany` according to how many documents need to be removed.
    d. The `deleteOne` command is used to remove documents from a Stand-alone deployment while the `deleteMany` helps to delete documents in Replica sets and Sharded Clusters.

10. What are the benefits of dropping a collection rather than removing all the documents?

    a. It removes all the documents and metadata associated such as indexes.
    b. Prevent indexes of being removed and allow reusing them later if new documents are added.
    c. It doesn't have any benefit.

11. Given the following query, what's the expected output?

```
db.movies.find( { "year" : 1989, "title" : "Batman" } )
```

    a. Return all the documents having the field `year` equals 1989 and the field `title` equals Batman.
    b. Display a list of all the collections within the database `movies` containing documents where the field `year` is equal to 1989 and the field `title` is equal to Batman.
    c. Only pull out all the documents where the field `year` is equal to 1989 or the field `title` is equal to Batman.
    d. None of the above is valid as MongoDB always negate query's predicates and bring back documents fulfilling the opposite condition.

12. In regards to filtering out documents by field within embedded documents, which of the following sentences are correct:

    a. It's not possible to filter out by fields inside embedded documents.
    b. The dot notation enables to accomplish this type of queries.
    c. Embedded documents is a feature not supported by MongoDB.
    d. The `$within_embedded_document` must be provided within the filter criteria.

13. Queries' results can return the whole document or part of it via field projections. Taking the following MongoDB query as an input, choose which option represents the right output:

```
db.movies.insertOne(
{
   "title" : "Forrest Gump",
   "category" : [ "drama", "romance" ],
   "imdb_rating" : 8.8,
   "filming_locations" : [
     { "city" : "Savannah", "state" : "GA", "country" : "USA" },
     { "city" : "Monument Valley", "state" : "UT", "country" : "USA" },
     { "city" : "Los Angeles", "state" : "CA", "country" : "USA" }
```

```
   ],
   "box_office" : {
     "gross" : 557,
     "opening_weekend" : 24,
     "budget" : 55
   }
 })
db.movies.findOne( { "title" : "Forrest Gump" },
                   { "title" : 1, "imdb_rating" : 1 } )
```

a. `findOne` queries don't allow projecting fields.
b. It returns the following document:

```
{
   "_id" : ObjectId("5515942d31117f52a5122353"),
   "title" : "Forrest Gump",
   "imdb_rating" : 8.8
}
```

c. It returns the following document:

```
{
   "title" : "Forrest Gump",
   "imdb_rating" : 8.8
}
```

d. It returns the following document:

```
{
   "category" : [ "drama", "romance" ],
   "filming_locations" : [
     { "city" : "Savannah", "state" : "GA", "country" : "USA" },
     { "city" : "Monument Valley", "state" : "UT", "country" : "USA" },
     { "city" : "Los Angeles", "state" : "CA", "country" : "USA" }
   ],
   "box_office" : {
     "gross" : 557,
     "opening_weekend" : 24,
     "budget" : 55
   }
 }
```

14. What happens when a match criteria is specified on a field that represents an array?

```
db.sensordata.insertOne(
{
   "_id" : "building1_floor1_room1_sensor1",
   "manufacturer" : "Siemens",
   ...
   "last_10_measures" : [ 10.5, 12, 14, 17, 18, 25, 18, 15, 12, 10],
   ...
 })
db.sensordata.findOne( { "last_10_measures" : 25 } )
```

    a. MongoDB iterates the array and check every item against the value in the match criteria. If there is a match returns that element.
    b. MongoDB iterates the array and check every item against the value in the match criteria. If there is a match returns the whole document.
    c. MongoDB doesn't allow this type of queries.
    d. The query is not correct as we omitted the `$elemMatch` operator.

15. What happens when a match criteria is specified on a field that represents an array, and values inside the array are also arrays?

```
db.sensordata.insertOne(
{
   "_id" : "building1_floor1_room1_sensor2",
   "manufacturer" : "Hitachi",
   ...
   "last_3_measures" : [ [ 1.5, 50, -5 ], [ 1.6, 49, -4 ], [ 1.3, 52, -7] ],
   ...
 })
db.sensordata.findOne( { "last_3_measures" : 50 } )
```

    a. MongoDB iterates the first level array and if it founds a new array, it inspects every item inside to identify if there is a match with the elements inside. If there is a match returns that element.
    b. MongoDB iterates the first level array and if it founds a new array, it inspects every item inside to identify if there is a match with the elements inside. If there is a match returns the whole document.
    c. MongoDB only checks items in first level arrays; deeper levels are not checked.
    d. This type of models leads to segmentation faults in the server.

16. How are results from queries got back to the application? Choose the one that applies:

a. Results are delivered directly as a stream of documents.
b. Queries return cursors that are pointers to result sets, and allow to iterate over the documents using `next()`.
c. Results are actually pointers to collections, and it's applications' responsibility to filter out those documents matching queries' criteria.
d. MongoDB doesn't retrieve values; it's a write-only database.

17. Which of the following options are cursors' methods? Check all that apply:

a. .count()
b. .skip()
c. .goto()
d. .batchSize()

18. Choose the only option describing the meaning of the following query:

```
db.movies.find( { $or : [   { $or : [
     { "category" : "sci-fi", "imdb_rating" : { $gte : 8 } },
     { "category" : "family", "imdb_rating" : { $gte : 7 } }
   ] } ,
   { $or : [
     { "category" : "action", "imdb_rating" : { $gte : 6 } }
   ] }
 ] } )
```

a. This query will fail as it's not possible to have nested logical operators within the filter criteria. The aggregation framework is meant to run this type of queries.
b. It'll return all the documents whose `category` field is equal to any of the categories in the filter criteria and an `imdb_rating` greater to the value specified for that particular category.
c. It'll return nothing because the second `$or` will always evaluate false no matter what values are assigned to the `category` and `imbd_rating`.
d. It'll return the whole collection and the use of indexes should be avoided.

19. What's the output of the following query?

```
db.movies.insertOne( {
    "title" : "Raiders of the Lost Ark",
    "filming_locations" : [
      { "city" : "Los Angeles", "state" : "CA", "country" : "USA" },
      { "city" : "Rome", "state" : "Lazio", "country" : "Italy" },
      { "city" : "Florence", "state" : "SC", "country" : "USA" }
```

```
    ] } )

db.movies.find( {
    "filming_locations" : {
      $elemMatch : {
        "city" : "Florence",
        "country" : "Italy"
       } } } )
```

a. Empty output.
b. Two nested documents: `{ "city" : "Florence", "state" : "SC", "country" : "USA" }` and `{ "city" : "Rome", "state" : "Lazio", "country" : "Italy" }`.
c. The whole document because the condition is fully satisfied.
d. The `$elemMatch` doesn't exist and this query will raise a syntax error.

20. In regards to the `updateOne` and `updateMany` commands, choose all those options that are correct.

    a. Only `updateOne` allows to modifying the `_id` field from documents.
    b. Both operations have two parameters: the query parameter to locate those documents that will be modified, and the replacement parameter with the changes.
    c. Both allow adding new fields and removing existing ones.
    d. If the query parameter cannot use an index, the operation will do a COLLSCAN to match all the documents that need to be updated.

21. How many documents are modified by the following update operation? What are the contents of the array field `category` in the first document?

```
db.movies.insertMany( [
  {
    "title" : "Batman",
    "category" : [ "action", "adventure" ],
    "imdb_rating" : 7.6,
    "budget" : 35
  },
  {
    "title" : "Godzilla",
    "category" : [ "action", "adventure", "sci-fi" ],
    "imdb_rating" : 6.6
  },
  {
    "title" : "Home Alone",
```

```
    "category" : [ "family", "comedy" ],
    "imdb_rating" : 7.4
  }
] )

db.movies.updateMany( { "category": "action",  },
                      { $set: { "category.$" : "action-adventure" } } )
```

a. Two documents are modified. The category field in the first document is `...,"category" : [ "action-adventure", "adventure" ],... .`
b. All the documents are modified. The category field in the first document is `...,"category" : [ "action-adventure" ],... .`
c. No documents are modified. The category field in the first document will remain the same.
d. The last document is modified because is the only one that doesn't have the category `"action"`. The category field in the first document will remain the same.

22. What does an upsert do when it inserts a document? Check all that apply:

a. Creates a new document using equality conditions in the query document.
b. Adds an _id if the query did not specify one.
c. Performs the write on the new document.
d. The upsert operation is not implemented in MongoDB.

INDEXES

23. How many different type of indexes are available in MongoDB?

a. MongoDB doesn't support indexes. That's a unique feature of relational databases.
b. Only the `_id` field is indexed in a MongoDB collection.
c. There are many of them. For instance: single, compound, geospatial, text, multikey and partial.
d. Only three types: single, compound and unique.

24. What does it imply to have indexes in your collections?

a. Indexes improve read performance for queries that are supported by the index but inserts will be slower when there are indexes that MongoDB must also update.
b. Indexes impact negatively on performance all the time.
c. Indexes always make operations faster, even inserts.
d. Indexes don't have any implications on collections as indexes are related to databases and not collections.

25. How many fields can be included in a compound index?

a. As much fields as are available in the documents.
b. Compound indexes, by definition, cannot have more than one field.
c. Up to 31 fields.
d. Up to 64 fields.

26. Check which option better describe what happens after issuing the following sequence of commands:

```
db.testcol.drop()
db.testcol.createIndex( { x : 1, y : 1 } )
db.testcol.insertOne( { _id : 1, x : 1, y : 1 } )
db.testcol.insertOne( { _id : 2, x : [ 1, 2 ], y : 1 } )
db.testcol.insertOne( { _id : 3, x : 1, y : [ 1, 2 ] } )
db.testcol.insertOne( { _id : 4, x : [ 1, 2 ], y : [ 1, 2 ] } )
```

a. This is a compound index definition on fields `x` and `y`, and the four inserts will successfully persist the document in MongoDB.
b. A multikey index is created on fields `x` and `y` due to the existence of arrays. The last insert will fail because a multikey index cannot index two arrays in the same document.
c. This is a multikey index created on fields `x` and `y`, and hence it expects always an array in either two fields. That'll make the first insert operation fail.
d. This is a geospatial index because documents contain GPS coordinates.

27. Which mechanisms exist in MongoDB indexes to remove documents after some time:

a. Indexes cannot remove indexes by themselves; this has to be accomplished by `deleteOne` and `deleteMany` commands.
b. TTL indexes remove documents automatically based on a field containing dates and an expiration threshold.

c. Single and compound indexes have an option that enables to specify documents' lifetime.
d. MongoDB always remove documents after one year by default to save disk space.

28. How do you interpret the following output from the `explain` command:

```
..
"executionStats" : {
  "executionSuccess" : true,
  "nReturned" : 8,
  "executionTimeMillis" : 107,
  "totalKeysExamined" : 0,
  "totalDocsExamined" : 51428,
  "executionStages" : {
    "stage" : "COLLSCAN",
    "filter" : {
      "user.followers_count" : {
        "$eq" : 1000
      }
    },
```

a. This is an optimal query because the whole collection is returned and a COLLSCAN works faster.
b. This is not an optimal query but is very fast as it took 100ms to execute.
c. This is something to avoid as the whole collection is scanned and only 8 documents were returned.
d. The query seems fairly good and hence no action should be taken.

STORAGE ENGINES

29. What's the storage engine's journal?

a. It's the part of the storage engine that connects with the filesystem journal and ensure consistency at database level. It's only available in WiredTiger.
b. It's a specific component in MMAPv1 and WiredTiger that keeps track of all changes made to data files. If there is a power failure in the server, writes from journal can be replayed to data files.
c. It's something that is available in other databases but not in MongoDB.

d. The journal is a feature that is only available within the journalist edition of MongoDB.

30. How many official storage engines are available in MongoDB 3.4?

a. MMAPv1, WiredTiger, Encrypted, and PureMemory.
b. MMAPv1 and WiredTiger.
c. MMAPv1, WiredTiger Encrypted and In-Memory.
d. MMAPv1, WiredTiger and In-Memory.

31. What are they main differences between MMAPv1 and WiredTiger?

a. They're almost the same, but MMAPv1 is only compatible with MongoDB up to version 3.0. WiredTiger, in the other hand, is only compatible with MongoDB 3.2 and upcoming versions.
b. WiredTiger compresses data in disk and in memory, while MMAPv1 only compresses indexes in memory.
c. MMAPv1 has document-locking granularity and WiredTiger has collection-locking granularity. Both compress data in memory and disk, but only WiredTiger compresses indexes in memory.
d. WiredTiger has document-locking granularity while MMAPv1 has collection-level locking granularity (database-level locking in older versions). Aside from that, WiredTiger only compresses documents in disk but indexes are either compressed in disk and memory (key prefix-compression).

32. How does the WiredTiger journal works?

a. Data is flushed from the shared view to data files every 60 seconds. The operating system may force a flush at a higher frequency than 60 seconds if the system is low on free memory
b. WiredTiger will commit a checkpoint to disk every 60 seconds or when there are 2 gigabytes of data to write.
c. WiredTiger's journal is not open-source therefore it's not documented how its journal works.
d. WiredTiger doesn't have a journal file because it's always consistent no matter what might cause the server to fail.

AGGREGATION FRAMEWORK

33. What's the aggregation framework?

a. It's a connector that enables issuing SQL sentences on top of MongoDB. It translates those SQL sentences into MongoDB queries.
b. It's an extension that enables to use the Unix pipe operator to filter and process data.
c. The aggregation framework is a deprecated feature in MongoDB and it's superseded by the map-reduce command.
d. It's a framework to transform and analyze data in MongoDB, and it's based on the concept of a pipeline.

34. Choose the option displaying the output generated by the following command:

```
db.tweets.aggregate( [
  { "$match" : { "user.time_zone" : "Brasilia",
                 "user.statuses_count" : {"$gte" : 100} } },
  { "$project" : { "followers" : "$user.followers_count",
                   "tweets" : "$user.statuses_count",
                   "screen_name" : "$user.screen_name" } },
  { "$sort" : { "followers" : -1 } },
  { "$limit" : 1 } ] )
```

a.
```
{ _id        : ObjectId('52fd2490bac3fa1975477702'),
  followers  : 2597,
  screen_name: 'marbles',
  tweets     : 12334
}
```

b.
```
{ followers  : 2597,
  screen_name: 'marbles',
  tweets     : 12334
}
```

c.
```
{ screen_name: 'marbles',
  tweets     : 12334,
  followers  : 2597
}
```

d.
```
{ _id        : ObjectId('52fd2490bac3fa1975477702'),
  followers  : 2597,
  screen_name: 'marbles',
  tweets     : 5
}
```

35. Given the following documents and aggregation operation, choose what the expected output would be among the different options:

```
db.sales.insertMany( [
  { "_id" : 1, "item" : "abc", "price" : 10,
    "quantity" : 2, "date" : ISODate("2014-03-01T08:00:00Z") }
  { "_id" : 2, "item" : "jkl", "price" : 20,
    "quantity" : 1, "date" : ISODate("2014-03-01T09:00:00Z") }
  { "_id" : 3, "item" : "xyz", "price" : 5,
    "quantity" : 10, "date" : ISODate("2014-03-15T09:00:00Z") }
  { "_id" : 4, "item" : "xyz", "price" : 5,
    "quantity" : 20, "date" : ISODate("2014-04-04T11:21:39.736Z") }
  { "_id" : 5, "item" : "abc", "price" : 10,
    "quantity" : 10, "date" : ISODate("2014-04-04T21:23:13.331Z") } ] )
db.sales.aggregate(  [
      { $group : {
           _id : null,
           totalPrice: { $sum: { $multiply: [ "$price", "$quantity" ] } },
           averageQuantity: { $avg: "$quantity" },
           count: { $sum: 1 }
         } } ] )
```

a. { "_id" : null, "totalPrice" : 250, "averageQuantity" : 6.6, "count" : 6 }
b. { "_id" : null, "totalPrice" : 190, "averageQuantity" : 8.9, "count" : 15 }
c. { "_id" : null, "totalPrice" : 290, "averageQuantity" : 8.6, "count" : 5 }
d. { "_id" : null, "totalPrice" : 90, "averageQuantity" : 8, "count" : 1 }

36. What's the aggregation framework stage that enables combining documents from different collections?

a. `$group`
b. `$project`
c. `$join`
d. `$lookup`

37. Check all the sentences (many might apply) that are correct in regards to the aggregation framework limitation:

a. An aggregation pipeline cannot use more than 100 MB of RAM.
b. `allowDiskUse : true` allows you to get around this limit.
c. `$match`, `$skip`, `$limit`, `$unwind` and `$project` stages are not subject to the 100 MB limit.

d. `$group` and `$sort` might require all documents in memory at once.

38. Consider the following statement:

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

    Which of the following are true about the statement?

    a. This is incorrect. The aggregation framework can switch position of some stages if needed, so you can use `$skip`, `$limit` and `$sort` after the `$graphlookup`.
    b. This is correct because `$graphlookup` pipes out the result of a recursive search into a collection, similar to `$out` stage.
    c. This is incorrect, you can use `$graphlookup` in any position of the pipeline.
    d. This is incorrect. `$graphlookup` must be used in the first stage of the pipeline to be able to use indexes.

SCHEMA DESIGN IN MONGODB

39. Which of the following affirmations are recommendations for a good schema design in MongoDB?

    a. Schema has nothing to do with the application.
    b. Application's read usage of the data is only considered for defining indexes but not for designing the schemas of documents.
    c. Application's write usage of the data is indifferent when it comes to schema design.
    d. None of the above is correct.

40. Is it to embed documents within a document a good idea in MongoDB? Choose all that apply:

    a. To embed documents is never a good idea. Indexes cannot be defined in fields within nested documents, and hence queries by these fields are always inefficient.
    b. Embedded documents are handy to model complex applications' business entities.
    c. Embedded documents can be considered to model relationships between entities.
    d. Improve read performance as all the needed information is in one single document.

41. What are the pros of data denormalization?

    a. Data denormalization doesn't have any advantage and it should be avoided.
    b. It allows optimizing applications' data access as everything (or most of the information) is persisted all together.
    c. Denormalization means to store main entity data in different collections and use identifiers to point out to other entities to model relationships. This optimize access to data because one single query pull all the data out from the disk.
    d. Denormalization is a non-existent concept, the right term is normalization.

42. What's modeling the schema of the following documents?

```
db.subjects.find()
{   _id: "American Literature" }

{   _id : "1920s",
    ancestors: ["American Literature"],
    parent: "American Literature"
}

{   _id: "Jazz Age",
    ancestors: ["American Literature", "1920s"],
    parent: "1920s"
}

{   _id: "Jazz Age in New York",
    ancestors: ["American Literature", "1920s", "Jazz Age"],
    parent: "Jazz Age"
}
```

    a. Linked lists.
    b. Stacks.
    c. Trees Structures.
    d. American libraries.

REPLICATION

43. What are the typical use cases for MongoDB replication? Choose all that are true:

    a. High Availability.
    b. Disaster Recovery.
    c. Functional Segregation

d. Scaling Horizontally.

44. How can a network partition affect to a Replica Set?

    a. If it's not properly configured and designed can lead to a service outage because the replica set will lack of a primary node, and hence the application won't be able to write data.
    b. Replica Set is network partition proof, nothing will happen.
    c. Nothing will happen if the replica set is composed by an even number of nodes.
    d. Arbiters are meant to fix any network partition as act as referees and have the authority to choose the primary at any time.

45. What's the recommendation to prevent a MongoDB outage in the situation of having one data center down? (several can apply))

    a. Reconfigure the replicaset to give more voting power to the available nodes.
    b. It's not possible to avoid a MongoDB outage if only one data center goes down, and eventually will happen.
    c. Have a heterogeneous replica set with different operating systems (Linux, Windows and Solaris). This gives 100% of data center availability.
    d. Deploy 3 full-node MongoDB replica sets across three different data centers, and avoid arbiters if writeconcern `{ w: majority }` is going to be used.

46. Given the following Replica Set configuration, choose the only valid option:

```
MongoDB Enterprise drplan:PRIMARY> rs.conf()
{ "_id" : "drplan",
  "version" : 4,
  "protocolVersion" : NumberLong(1),
  "members" : [ { "_id" : 0,
                  "host" : "node1:27017",
                  "arbiterOnly" : false,
                  "buildIndexes" : true,
                  "hidden" : false,
                  "priority" : 10,
                  "tags" : {     },
                  "slaveDelay" : NumberLong(0),
                  "votes" : 1 },
                { "_id" : 1,
                  "host" : "node1:27018",
                  "arbiterOnly" : false,
                  "buildIndexes" : true,
                  "hidden" : false,
```

```
                    "priority" : 5,
                    "tags" : {      },
                    "slaveDelay" : NumberLong(0),
                    "votes" : 1 },
                  { "_id" : 2,
                    "host" : "node2:27019",
                    "arbiterOnly" : false,
                    "buildIndexes" : true,
                    "hidden" : false,
                    "priority" : 1,
                    "tags" : {      },
                    "slaveDelay" : NumberLong(0),
                    "votes" : 1 } ],
    "settings" : { "chainingAllowed" : true,
                    "heartbeatIntervalMillis" : 2000,
                    "heartbeatTimeoutSecs" : 10,
                    "electionTimeoutMillis" : 10000,
                    "getLastErrorModes" : { },
                    "getLastErrorDefaults" : {   "w" : 1, "wtimeout" : 0 },
                    "replicaSetId" : ObjectId("57fb70585ddfacd502de56c8") }
}
```

a. `node2` will always be the primary no matter what the current status for the rest of the nodes is.
b. `node1:27017` will be only primary if it connects to the majority of the nodes and it has the most recent optime. Otherwise it'll act as secondary until it sees a majority of members and catches up with the rest of the nodes.
c. `node2` will be the primary as it has the highest priority (lower values mean higher priority).
d. In protocol version 1 all the nodes act as primaries.

47. How are write operations replicated over the different replica set members?

a. Keeping track of the data files, offsets and how many bytes were written for each change.
b. Replica sets rely on the `rsync` Unix utility to synchronize the nodes' data paths. Replica sets on Windows need Cygwin to be installed to use `rsync`.
c. Operations are stored in chunks and distributed by the MongoDB balancer (`mongos`). Jumbo chunks prevent replicating data optimally and should be avoided.
d. Storing a statement for every operation in a capped collection called the oplog. Secondaries apply those statements that were issued on the primary.

48. Write concern `{ w: "majority" }` will prevent application's rollback and will guarantee that the application is able to write while there is an available primary node.

    a. This is always true.
    b. This is true unless the replica set contains arbiters. It might happen that the majority of the nodes (including arbiters) are available and hence a primary is elected, but writes cannot be committed to the majority of the writable nodes (arbiters don't persist data by definition) and hence write operations get stuck waiting for its ACK.
    c. Write concern `{ w: "majority" }` only works in stand alone deployments.
    d. None of the above is correct.

49. Which read preference setup would be the right choice for doing analytics from an application?

    a. `secondary`
    b. `primary`
    c. `nearest`
    d. `analytics`

SHARDING

50. What are the factors to consider evolving a MongoDB deployment to sharded cluster setup? Choose all that apply:

    a. If you have more data than one machine can hold on its drives
    b. If you want to show off your MongoDB skills.
    c. If your application is write heavy and you are experiencing too much latency.
    d. If your working set outgrows the memory you can allocate to a single machine.

51. Using the default configuration, what's the biggest size that a chunk can have before splitting up?

    a. 64MB
    b. 32MB
    c. 16MB
    d. 128MB

52. Which of the following define the characteristics of a good shard key? More than one can apply:

a. Reads hit only 1 or 2 shards per query.
b. Writes are distributed across all servers.
c. It has high cardinality.
d. It should be consistent with the application's query patterns.

53. Which of the following statements are applicable to Zone Sharding?

a. Zone ranges are immutable.
b. Zone ranges cannot overlap.
c. One shard can only be added to one zone.
d. Zone ranges can be updated.

SECURITY

54. What are the options offered by MongoDB in terms of database users authorization?

a. MongoDB doesn't provide any security mechanism in order to achieve a better performance.
b. MongoDB enforces a role-based authorization model, and a user is granted roles that determine the user's access to database resources and operations.
c. SCRAM-SHA-1, MONGODB-CR, X509 Certificates, LDAP and Kerberos.
d. `a` and `c` are true.

55. What does a MongoDB role define?

a. All the options below are correct.
b. A role defines personal data about people and the Human Resources departments mainly use it when they're applications use MongoDB as a backend.
c. Roles are sets of privileges that users may require to use MongoDB resources (Databases, Collections and Cluster).
d. A role is a set of users that share the same credentials to access MongoDB from applications.

56. How do you remove a view? Check all that apply.
a. Use the `db.dropView( <view_name> )` command.
b. Delete it from `system.views` using `db.system.views.deleteOne({ _id : <view name> })`
c. Drop it using `db.<view_name>.drop()` as if it were a collection.
d. Use the `db.deleteView( <view_name> )` command

DIAGNOSTICS AND BACKUP

57. Which of the following options are steps that you should perform to identify the cause of a performance issue?

    a. Check MongoDB log files to identify slow queries.
    b. Ensure that every query uses an index (unless that more than 40-50% of documents are retrieved from a collection).
    c. Hardware monitoring to identify bottlenecks.
    d. All the above is correct.

58. In which situations the `db.<COLLECTION>.stats()` command is useful?

    a. When you want to do a sizing exercise on top of your MongoDB deployment (storage size, index size, average object size, …).
    b. To monitor how many users are connected to the database.
    c. To display the queries that are running on that collection.
    d. That commands cannot be applied to collections but databases.

59. What's the fastest backup strategy in terms of snapshot creation and restore?

    a. Querying all the collections from the mongoshell and redirecting the output to an external file. Running a script inserting the data that was previously pulled out will restore data.
    b. Doing filesystem and virtual machine snapshots or copying the data path to an external drive.
    c. Using mongodump and mongorestore.
    d. Avoiding backing up data.

60. How would you export the data and indexes inside a small MongoDB deployment into another MongoDB deployment? (many can apply)

    a. Copying the data path into the new server.
    b. Using mongoexport and mongoimport.
    c. Using mongodump and mongorestore.
    d. Copying the journal file and the oplog from the source deployment into the target one.