Database - The Magical Marvels of MongoDB Answers

Conjuring MongoDB (Level 1)

01. How would you set the current database to “tapePlayer”?
    Answer: **use wandRecorder**

02. How would you find all the documents inside the “tapes” collection?
    Answer: **db.tapes.find()**

03. How would you add a document to the “tapes” collection whose “name” is “Rockin 70s” and whose “creator” is “Big Rig Joe”?
    Answer: **db.tapes.insert({“name”: “Rockin 70s”, “creator”: “Big Rig Joe”})**

04. How would you query for documents whose “name” is “Cool 60s” in the “tapes” collection?
    Answer: **db.tapes.find({“name”: “Cool 60s”})**

05. How would you query for documents whose “creator” is “Slim Jim Johnson” in the “tapes” collection?
    Answer: **db.tapes.find({“creator”: “Slim Jim Johnson”})**

06. How would you insert the following tape into the “tapes” collection after adding the following name-value pairs?
    Answer: **db.tapes.insert({**
    “name”: “Best of 90s”,
    “creator”: “Sweet Kim Kahn”,
    “age”: 20,
    “price”: 19.99,
    “themes”: [“Rock”, “Jazz”, “Hip Hop”],
    “critics”: {“official”: 6, “unofficial”: 9}
    })

07. How would you query your “tapes” collection to find documents who have “Rock” in their “themes”?
    Answer: **db.tapes.find({“themes”: “Rock”})**

08. Which validation does the following bad data break?
    Answer: **A. Unique_id**

Mystical Modifications (Level 2)

09. How would you remove the tape with “name” of “Rockin 70s” from our “tapes” collection?
    Answer: **db.tapes.remove({“name”: “Rockin 70s”})**

10. How would you remove all tapes that have “Rock” in their “themes” from our “tapes” collection?
    Answer: **db.tapes.remove({“themes”: “Rock”})**

11. How would you the “price” of the tape with “name” of “Rockin 70s” in the “tapes” collection to 14.99?

Answer: **db.tapes.update({"name": "Rockin 70s"}, {"$set": { price": 14.99}})**

12. How would you increase the "age" by 2 for all tapes with "Jazz" in their "themes" in the "tapes" collection?
    Answer: **db.tapes.update({"themes": "Jazz"}, {"$inc": {"age": 2}}, {"multi": true})**

13. How would you increase the view "count" of the tape with "name" of "Cool 60s" in the "logs" collection, and create the document if it doesn't exist?
    Answer: **db.logs.update({"name": "Cool 60s"}, {"$inc": {"count": 1}}, {"upsert": true})**

14. How would you remove the "taste" field from all documents in the "tapes" collection?
    Answer: **db.tapes.update({}, {"$unset": {"taste": ""}}, {"multi": true})**

15. How would you change the "creator" field to "writer" for all documents in the "tapes" collection?
    Answer: **db.tapes.update({}, {"$rename": {"creator": "writer"}}, {"multi": true})**

16. In the following document, how would you update the value of "Rock" to "Rock and Roll" in the "themes" array, assuming this document is in the "tapes" collection.
    Answer: **db.tapes.update({"name": "Best of 90s"}, {"$set": {"themes.0": "Rock and Roll"}})**

17. Using the updated document from question 16, how would you update the value of "Jazz" to "Smooth Jazz" in the "themes" field using the positional operator?
    Answer: **db.tapes.update({"themes": "Jazz"}, {"$set": {"themes.$": "Smooth Jazz"}})**

18. Using the updated document from question 17, how would you add "Disco" to the end of the "themes" field?
    Answer: **db.tapes.update({"name": "Best of 90s"}, {"$push": {"themes": "Disco"}})**

19. How would you add "Disco" to "themes" field in every document in the "tapes" collection if it isn't present already?
    Answer: **db.tapes.update({}, {"$addToSet": {"themes": "Disco"}}, {"multi": true")**

20. Using the updated document from question 18, how would you multiply the "official" value by 2?
    Answer: **db.tapes.update({}, {"$mul": {"critics.official": 2}})**

Materializing Potions (Level 3)

21. How would you find all tapes where the "maker" is "Slim Jim Johnson" and the "age" is 10 that are in the "tapes" collection?
    Answer: **db.tapes.find({"maker": "Slim Jim Johnson", "age": 10})**

22. How would you find all tapes in the "tapes" collection where "age" is less than or equal to 15?
    Answer: **db.tapes.find({"age": {"$lte": 15}})**

23. How would you find all tapes in the "tapes" collection that don't have "Rock and Roll" in the "themes" field?
    Answer: **db.tapes.find({"themes": {"$ne": "Rock and Roll"}})**

24. How would you find tapes in the “tapes” collection that have “critics.official” value greater than or equal to 2 and less than or equal to 5?
    Answer: **db.tapes.find({“critics.official”: {“$gte”: 2, “$lte”: 5}})**

25. All tapes now have a “speeds” field showing a list of speeds the tapes play in, as shown below. How would you find all tapes that contains “speeds” that are greater than or equal to 1.5 but less than 2.5?
    Answer: **db.tapes.find({“speeds”: {“$elemMatch”: {“$gte”: 1.5, “$lt”: 2.5}}})**

26. How would you find tapes in the “tapes” collection” whose “maker” is not “Slim Jim Johnson”, whose “age” is less than or equal to 10, whose “price” is less than 14.99, and whose “speeds” are 1.5 or 2?
    Answer: **db.tapes.find({“maker”: {“$ne”: “Slim Jim Johnson},**
    “age”: {“$lte”: 10}, “price”: {“$lt”: 14.99},
    “speeds”: {“$elemMatch”: {“$gte”: 1.5, “$lte”: 2}}})

27. How would you query all tapes in the “tapes” collection, but project on the “_id” and “name” fields, and sort them alphabetically?
    Answer: **db.tapes.find({}, {“name”: true}).sort({“name”: 1})**

28. How would you query all tapes in the “tapes” collection, and project everything except the “_id”, “price”, and “speeds” fields?
    Answer: **db.find({}, {“price”: false, “speeds”: false, “_id”: false})**

29. How would you query all tapes in the “tapes” collection, and project the “name” and “themes” fields, but exclude the “_id” field?
    Answer: **db.tapes.find({}, {“name”: true, “themes”: true, “_id”: false})**

30. How would you count the number of tapes in the “tapes” collection that have an “age” of 5?
    Answer: **db.tapes.find({“age”: 15}).count()**

31. Finish the following query so that only 5 tapes are returned by the cursor for each page from the “tapes” collection:
    Answer: **db.find({}).skip(0).limit(5)**

32. How would you write a query to match all tapes in the “tapes” collection, then sort them in ascending order by their “price”, and add a cursor method to limit the results to 4 documents?
    Answer: **db.find({}).sort({“price”: 1}).limit(4)**

Morphing Models (Level 4)

33. Fill in the blank. When we take the data from one document and place it inside another one, that’s called an _____ document.
    Answer: **embedded**

34. Fill in the blank. If we take a unique value like an _id from one document and store it as a value within a related document, we have just created a _____ document.
    Answer: **referenced**

35. Consider the following documents in the “tapes” collection. For what reason might we want to consider referencing “maker” information instead of embedding it within each wand?

    Answer: B. Duplicate Data

    With the information in “maker” being repeated as much as it is, we want to reduce the possibility that an inconsistency forms. We can accomplish that by making “maker” a reference so that it only changes in 1 place, and all the references will receive the updated information.

36. What’s the minimum number of queries we’d have to write in order to retrieve a document and its referenced data?

    Answer: B. 2

    The first query will return data that contains the reference. The second query will be used to get the information about the reference. While there could be layers upon layers of references, you need at least 2 queries get referenced data.

37. Which modeling option would give us all the data we need with a single query, support for atomic writes, and is great for data that is strongly related?

    Answer: A. Embedding

    While using references is useful related data, it takes more than 1 query to get all the data. Embedding your data will return all the data with a single query, and ensures that the related data applies in full.

38. Which data modeling decision doesn’t have default support for atomic writes across multiple document and be utilized with care?

    Answer: B. Referenced

    As discussed previously, MongoDB doesn’t recognize a relationship between document. It only sees field value pairs. So when you make an update to multiple related documents, and one fails, then that causes an error with the other.

39. In general, what’s the best option to first consider for modeling your related data?

    Answer: A. Embedding

    In general, a document-oriented database like MondoDB will have strongly related data, small to medium sized collections, and data that that doesn’t change often. These are all characteristics of a document-orientated database, in general.

40. Which data modeling option would be the best fit for storing users and their addresses when we know that the data is used together often, won’t be changing regularly, and each user will only be storing a few addresses?

    Answer: A. Embedding

    Again, knowing our data won’t be changing regularly and storing a small amount of data are characteristics of embedded data.

41. We’d like to store information about computers, and each computer can have a few hundred parts. Most of the time, we won’t be needing specific information about each part. Which data modeling route should we take?

    Answer: B. Referencing

In this case, we will be working with a larger data set, but won't need the specific information about each part in the computer. So its better to put a reference to the part rather than embedding it.

42. Which modeling route is best when we have data that is constantly changing and will help prevent data inconsistencies from arising?
    Answer: B. Referencing
    Again, if we will have data that changes constantly, we want to make these changes in only one place, so that a reference to it will be guaranteed to be unaffected.

43. Which data modeling route allows us to access our data independently instead of having to use something like dot notation to get information?
    Answer: B. Referencing
    Again, its important to remember that dot notation applies to embedded objects. Its also worth remembering that using documents independently is a staple of referencing.

Aggregation Apparitions (Level 5)

44. How would you write an aggregate to group "tapes" by their "maker" so we get a list of unique makers?
    Answer: db.tapes.aggregate([{"$group": {"_id": "$maker"}}])

45. How would you write an aggregate that groups "tapes" by their "critics.unofficial", and add an accumulator with a "tapes_critiqued" to count the number of critiques per "critics.unofficial" score?
    Answer:
    ```
    db.tapes.aggregate([
      {"$group": {"_id": "$critics.unofficial", "wand_count": {"$sum": 1}}}
    ])
    ```

46. How would you write an aggregate that groups our "tapes" by the "maker" field, and add an accumulator with the "total_cost" field that sums the "price" for each tape per "maker"?
    Answer:
    ```
    db.tapes.aggregate([
      {"$group": {"_id": "$maker", "total_cost": {"$sum": "$price"}}}
    ])
    ```

47. How would you write an aggregate that groups "tapes" by their "age", and add an accumulator with a field "price_averaged" to get the average "price" for the tapes per age?
    Answer:
    ```
    db.tapes.aggregate([
      {"$group": {"_id": "$age", "price_averaged": {"$avg": "$price"}}}
    ])
    ```

48. How would you write an aggregate that groups “tapes” by their “maker”, add an accumulator with the field “total_tapes” to sum the number of tapes each “maker” has, add an accumulator with the field “max_critique” that finds the greatest “critics.official” per “maker”, and add an accumulator with the field “lowest_price” that finds the lowest tapes “price” per maker?

Answer:
```
db.tapes.aggregate([
  {“$group”: {“_id”: “$maker”, “total_tapes”: {“$sum”:1},
              “max_critique”: {“$max”: “critics.official”},
              “lowest_price”: {“$min”: “$price”}}}
])
```

49. How would you write an aggregate that will only match tapes that have “Rock and Roll” in their “themes”, then another stage to group the data by their “maker”, then an accumulator with a field name “lowest_age” that finds the lowest “age” per “maker”?

Answer:
```
db.tapes.aggregate([
  {“$match”: {“themes”: “Rock and Roll”}},
  {“$group”: {“_id”: “$maker”, “lowest_age”: {“$min”: “$age”}}}
])
```

50. How would you write an aggregate that matches tapes that have a “price” that is less than 50, then a stage to group the tapes by their “maker”, then a accumulator with a field name “average_critique” to find the average “critics.official” per “maker”, then after the existing stages, add another match stage that retrieves results with an “average_critique” that is greater than 40?

Answer:
```
db.tapes.aggregate([
  {“$match”: {“price”: {“$lt”: 50}}},
  {“$group”: {“_id”: “$maker”, “average_critique”: {“$avg”: “$critics.official”}}},
  {“$match”: {“average_critique”: {“$gt”: 40}}}
])
```

51. How would you write an aggregate that finds tapes that have an “age” that’s less than or equal to 5, then add a stage to group tapes by their “maker”, then add an accumulator with the field “max_critique” that finds the max “critics.unofficial” per “maker”, then add a stage to sort the “max_critique” in descending order, the add a stage to limit only the first 4 results, then add a stage in the correct place to project only the “maker” and “critics.unofficial” fields?

Answer:
```
db.tapes.aggregate([
  {“$match”: {“age”: {“$lte”: 5}}},
  {“$project”: {“maker”: true, “critics.unofficial”: true, “_id”: false}},
  {“$group”: {“_id”: “$maker”, “max_critique”: {“$max”: “$critics.unofficial”}}},
  {“$sort”: {“max_critique”: -1}},
  {“$limit”: 4}
])
```