MongoDB

Study online at quizlet.com/_8ccuky

1. **What is MongoDB and Why we need MongoDB?**	MongoDB is one of the most popular open source NoSQL database management system. MongoDB is a document oriented database, it stores data in BSON format.
2. **What is BSON?**	BSON is a binary representation of JSON. It has more data types than JSON.
3. **What data types exist in BSON but not in JSON?**	Date TimeStamp Int Double (JSON only has Number)
4. **What does table in SQL correspond in MongoDB?**	Collection
5. **What does row in SQL correspond in MongoDB?**	Document
6. **What does column in SQL correspond in MongoDB?**	Field
7. **What does join in SQL correspond in MongoDB?**	Linking
8. **What is the difference between MongoDB and SQL?**	1. MongoDB is schemaless, which makes it easy to change and extend objects stored in DB, very suitable for a growing business 2. MongoDB is very horizontally extendable, because data is stored across multiple clusters. This makes it easy to add extra clusters when there is need to extend storage. 3. MongoDB is highly available because the design of MongoDB puts replication at high priority. Each MognoDB cluster consists of several nodes that are replications of each other. In the case of primary node failure, MongoDB can automatically detect the secondary node that is running and direct the request to that node 4. MongoDB is easy for development, especially for web based API because each document is readily wrapped into JSON object, which is also what most modern object oriented languages use. 5. MongoDB is fast, because you don't have to query different tables in SQL DB to gather different pieces of data, all information related to the same document is stored in the same document in one collection.
9. **MySQL can also return query results in JSON. What is the advantage of MongoDB over MySQL?**	MySQL always puts query results in JSON strings, the application needs to parse them later. This is because MySQL is still a relational database. MongoDB can return the query results respecting the true JSON types: dates are dates, times are times, doubles are doubles.

10. **High availability often comes with high latency, because a query may need to wait for all replications. This could be a concern in highly concurrent scenario. How does MongoDB overcome this?**	MongoDB allows you specify readConcern and writeConcern in the request. They specify the level of acknowledgement we require from MongoDB for reading and writing operations. A low readConcern will not wait for confirming replication in other replica nodes. It returns with lower latency at the cost of potential stale. A low writeConcern will not wait for confirmation that the written result can be recovered from shutdown or has been replicated to other nodes. It returns with lower latency at the cost of not being able to confirm the possibility of recover. For applications that are very concerned about data lost, write request can specify a high writeConcern.
11. **What is the _id field in MongoDB?**	Every collection has a default index "_id". Every document has a unique id. Most MongoDB driver fills an _id before sending it to the database, but even if the driver doesn't, the database would autogenerate an id for the document.
12. **How do you create or select a db in MongoDB?**	use [db name] If db already exists, all the following requests will be treated as done in this db If db doesn't exist, then this db will be created
13. **How to find a document?**	db.[collection].find([query])
14. **How to update a document?**	db.[collection].update([query], [update])

15. **What is the syntax of a MongoDB query?**	It is a JSON object
	
	{
	...
	}

16. **How to query by match a field exactly?**	[field] : [value]
17. **How to query by matching an array field exactly?**	[field] : [[value]]
18. **How to query by matching an array field ignoring order?**	[field] : { $all: [[value]] }

19. **How to query by matching an array field that contains any element that satisfies a combination of conditions?**	[field] : { [conditions] }
	
	e.g.
	
	c : {$gt: 10, $lt: 100, $ne: 20}
	
	returns any document whose field 'c' is greater than 10, less than 100, and not equal to 20

20. **How to query by matching an array field whose all elements satisfy a combination of conditions?**

[field] : {$not: { [the negative of the conditions] }

e.g.

c : {$not: {$lte: 1, $gte: 2}}

returns any document whose field 'c' is has no element <= 1 or >= 2, i.e., all elements between 1 and 2 (exclusive)

21. **How to query by either of the specified conditions?**

{$or: [{[condition1}, {[condition2}]}

e.g.

{$or: [{c: {$gt: 1}}, {d: {$lt: 2}}] }
returns any document whose field c > 1 or d < 2

Note: you can specify different conditions in different fields

22. **How to query by all of the specified conditions?**

{$and: [{[condition1]}, {[condition2]}]

or

{$elemMatch: [{[condition1}, {condition2}]}

E.g.

{$elemMatch: [
{a: 1},
{b: 2}
]}

23. **Why would you want to use $elemMatch?**

db.[collection].find({
{a:1},
{b:2}
})

will return documents that EITHER a=1 OR b=2

db.[collection].find({
$elemMatch: [
{a:1},
{b:2}
]
})

will return documents that BOTH a=1 AND b=2

24. **When would you prefer $elemMatch to $and, or vice versa?**

$elemMatch iterates through documents in the outer loop and iterates through queries in the inner loop

$and iterates through queries in the outer loop and iterates through documents in the inner loop

so if there a restrictive condition, you should put it more upfront in the query and use $and, which should be faster than $elemMatch.

25. **How to specify a date?**

ISODate("yyyy-mm-dd")

26. **How to group by?**	db.[collection].aggregate([aggregation pipeline])
27. **What are the stages of aggregation pipeline?**	$match: [query] $sort: [sorting order] $group: [grouping]
28. **What does the sorting order look like?**	$sort: {[col1]: 1, [col2]: -1} means sort in ascending order of col1, descending order of col2.

29. **What does the grouping look like?**	$group: {
	_id: { [col1] : $[col1 alias], [col2] : $[col2 alias]
	},
	[agg col alias] : { $[op] : $[col] }
	[agg col alias] : { $[op] : [constant] }
	}
	
	E.g.
	
	$group: {
	_id: {grouped_A : "$a"},
	sum_B : {$sum: "$b"},
	row_ct : {$sum: 1}
	}

30. **How to do SQL-style "having"?**	Just append another layer of match after group stage

31. **I don't want to group by any column, I just want the biggest value in column b and a row count, what should I do?**	db.[collection].aggregate(
	$group: {
	_id : null,
	max_b : {$max: "$b"},
	row_ct: {$sum: 1},
	}
	)

32. **How should I count the number of unique values in column c?**	db.[collection].aggregate(
	$group: {
	_id: "$c"
	}
	$group: {
	uniq_count_c: {$sum : 1}
	}
	)

33. **How do I query all documents that have/do not have a certain field?**	[col] : {exists : true/false}
34. **How do I query by comparing values between fields?**	$expr: {$[op]: [$[col1], $[col2]] } E.g. $expr: {$gt: ["$a", "$b"] } means query for documents where field a is greater than field b
35. **How do I query text field by regex?**	$regex: {$[text field] : [reg ex]}

36. **I want more complex logic in my query, it cannot be easily written into the native syntax, what can I do?**	$where: [javascript function]
	
	You can use "this" inside the javascript function to refer to the document
	
	E.g.
	
	$where: function() {
	this.a % this.b === 0;
	}

37. **I have embedded field inside a field, how can I query by the embedded field?**	use dot notation [field].[embedded field] to refer to the embedded field E.g. db.[collection].insert({a: {b:1}}) db.[collection].query({"a.b": 1})
38. **I have a field that is an array of objects, how can I query by the object key inside this array field?**	use dot notation [field].[embedded array key name] E.g. db.[collection].insert({a:[{b:1, c:2}, {b:10, c:20}]}) db.[collection].query({"a.b": 1})
39. **I have a field that is an array of elements, how can I query by the index of elements in this array field?**	use dot notation [field].[embedded array index] E.g. db.[collection].insert({a: [1,2]}) db.[collection].find({"a.1" : 2})
40. **How to insert a document?**	db.[collection].insert([document])
41. **How to insert many documents?**	db. [collection].insertMany([[documents]])
42. **How to update many documents?**	db.[collection].update([query], [update], multi : true)
43. **How to do these two operations together: first try to find and update, if not exist, then insert?**	db.[collection].upsert([query], [update], multi : true/false)
44. **I want to find a document and update its certain field, but not by merging, but by changing the existing value by a certain amount. What should I do?**	The [update] stage in db. [collection].update can have operator $inc : {[col] : [delta]} $dec: {[col] : [delta]}

45. **I want to find a document and replace it, what should I do?**	Use a plain document in the [update] stage in db.[collection].update, without any $ operator, then it means to replace the document entirely.
46. **I want to find a document and replace some fields only while keeping other fields unchanged, what should I do?**	In the [update] stage in db.[collection].update, use $set operator $set: {[col to replace] : [new value]}
47. **I want to find a document and remove some fields, what should I do?**	In the [update] stage in db.[collection].update, use $unset operator $unset: {[col to remove] : null} the actual value assigned to col to remove doesn't impact the operation result
48. **I want to find a document and add more embedded documents to an array field, what should I do?**	In the [update] stage in db.[collection].update, use $push operator $push: [[values to add]]
49. **How to only project some fields from query results?**	In the find command, you can have a second argument in addition to query db.[collection].find([query], [project]) The second [project] argument takes the form of {[col1 to keep] : 1, [col2 to keep] : 1, ...}
50. **How to project all but some fields from query results?**	Use {[col1 not to keep] : 0, [col2 not to keep] : 0} in the second [project] argument of the find command
51. **Can I have a mixture of col: 1 and col: 0 in the second [project] argument of the find command?**	No, you can't, the only exception being {[col1 to keep] : 1, _id : 0} You can only choose not to keep the _id column while keeping some other columns. You cannot have {[col1 to keep] : 1, [col2 not to keep] : 0}

52. **I want to find a document and remove some embedded documents from an array field, what should I do?**	In the [update] stage in db.[collection].update, use $pull operator
	
	$pull: {[conditions]}
	
	E.g.
	
	db.[collection].insert({a:[{b: 1, c: 2}, {b: 10, c: 20}]})
	db.[collection].update(
	{"a.b": 1},
	{
	$pull: {$elemMatch: {b:10, c:20}}
	}
	)

53. **How to delete documents?**	db.[collection].deleteMany([query])
54. **How to delete collection?**	db.[collection].drop()
55. **How to delete database?**	db.dropDatabase()

56. **How to create collection?**

    db.createCollection()

57. **How to create index?**

    ```
    db.[collection].createIndex(
    {[col1] : 1, [col2] : -1}
    )
    ```

    +1 and -1 specifies sorting order
    such an index can facilitate query using both col1 and col2, or just col1, but cannot facilitate query using just col2.

58. **How can I drop an index?**

    ```
    db.[collection].dropIndex(
    {[col1] : 1, [col2] : -1}
    )
    ```

    the order of columns and the sorting order must match exactly with the existing indices

59. **How can I see the existing indexes?**

    db.[collection].getIndexes()

60. **Can I require an index to be unique and reject insertions that could cause duplicates?**

    ```
    db.[collection].createIndex(
    [index],
    {unique : true}
    )
    ```

    It will fail to create such a unique index if there are already duplicates

61. **What is a name space?**

    The concatenation of a database and a collection

62. **Is there transaction locking in MongoDB?**

    No, MongoDB doesn't lock transactions to improve performance

63. **What are the disadvantages of MongoDB?**

    1. NoSQL database is difficult to join.

    2. Asynchronous and batch commit. MongoDB default installation turns on asynchronous and batch commit, i.e., when you write data to DB, it will store the data into a batch and write the batch to DB later. If there is shutdown before the batch is written into DB, then the data can be lost. This is what MongoDB does to improve performance. You can require confirmation in writing operations by specifying writeConcern level, but then MongoDB won't perform that well.

64. **What functionalities are MongoDB most suited for?**

    Analytics and caching, where small loss can be tolerated