MongoDB Associate Developer Practice Exam (Sample)

Study Guide

Everything you need from our exam experts!

Questions

1. How do you delete the default _id index?

- **A. By using the dropIndex method**
- **B. You cannot delete this index**
- **C. By manually updating the collection**
- **D. By running a specific command line operation**

2. Is hiding an index the same as deleting an index?

- **A. Yes, they are equivalent actions**
- **B. No, hiding does not remove the index**
- **C. Yes, both actions will free up resources**
- **D. No, deletion is permanent, while hiding can be reversed**

3. What will the following query return? db.routes.find({ $and: [{ $or: [{ dst_airport: "IST" }, { src_airport: "IST" }] }, { $or: [{ stops: 0 }, { airline.name: "Turkish Airlines"}] },] })

- **A. All nonstop flights from Istanbul airport**
- **B. All flights either departing or landing at Istanbul airport that are operated by Turkish Airlines or have zero stops**
- **C. All flights that have at least one stop**
- **D. All flights with no specified conditions**

4. In MongoDB, why should transactions be used when updating account balances across multiple collections?

- **A. They provide faster updates**
- **B. They ensure operations are atomic across multiple documents**
- **C. They eliminate the need for aggregation pipelines**
- **D. They reduce storage requirements**

5. Are indexes created automatically based on usage patterns in MongoDB?

- **A. Yes, always**
- **B. No, they must be created manually**
- **C. Only for compound indexes**
- **D. Only when specified by the user**

6. What parameters can be provided to the countDocuments() method?

- **A. Only a query document**
- **B. A query document and options document**
- **C. Only an options document**
- **D. No parameters are needed**

7. Which of the following queries correctly uses the $elemMatch operator?

- **A. { "items": { $elemMatch: { name: "item1", quantity: { $gt: 0 } } }}**
- **B. { "items": { $elemMatch: { name: "item1" }} }**
- **C. { "items": { $and: [{ name: "item1" }, { quantity: { $gt: 0 } }] }}**
- **D. { "items": { name: "item1" }}**

8. How does the Performance Advisor analyze collections in MongoDB?

- **A. By tracking user interactions**
- **B. By monitoring query performance**
- **C. By reviewing system resource usage**
- **D. By evaluating schema structures**

9. Data that is assessed together should be...

- **A. stored separately**
- **B. stored in different databases**
- **C. stored together**
- **D. archived for later use**

10. What is the function of the upsert option in MongoDB operations?

- **A. It updates all matching documents**
- **B. It inspects a document before making changes**
- **C. It inserts a document if none exist based on the query**
- **D. It creates a backup of the document before updating**

Answers

1. B
2. B
3. B
4. B
5. B
6. B
7. A
8. B
9. C
10. C

Explanations

1. How do you delete the default _id index?

A. By using the dropIndex method

B. You cannot delete this index

C. By manually updating the collection

D. By running a specific command line operation

The default _id index in MongoDB is a unique index created automatically for the _id field of every document in a collection. This index is crucial for the functioning of MongoDB because it ensures that each document has a unique identifier, which is fundamental for document retrieval and data integrity. Option B, stating that you cannot delete this index, is correct because MongoDB enforces the _id field as a unique key for each document in the collection. The system depends on this index to ensure that no two documents can share the same _id value. Consequently, any attempt to manually drop or alter this default index would not be supported by MongoDB, as it is integral to the structure and operational mechanics of the database. This characteristic of the _id index emphasizes the importance of having a unique identifier for documents in collections and helps maintain database consistency and performance. The other options are not valid given that they suggest ways to modify or delete the _id index, which goes against MongoDB's design principles.

2. Is hiding an index the same as deleting an index?

A. Yes, they are equivalent actions

B. No, hiding does not remove the index

C. Yes, both actions will free up resources

D. No, deletion is permanent, while hiding can be reversed

Hiding an index is fundamentally different from deleting an index. When an index is hidden, it is still present in the database but is not used by the query planner to execute queries. This means that while the hidden index can be retained for potential future use or for testing purposes, it does not contribute to the performance of query operations until it is made visible again. On the other hand, deleting an index completely removes it from the database, including all its associated resources. Once deleted, it cannot be reverted or reinstated without creating a new index. This permanence is a critical distinction: hiding maintains the structure of the index and its metadata, allowing for flexibility in managing query performance without the need to recreate the index later. Therefore, the statement that hiding does not remove the index captures the essence of how hiding and deleting differ. Hiding allows for temporary exclusion from query planning, while deletion is a final, irreversible action.

3. What will the following query return? db.routes.find({ $and: [{ $or: [{ dst_airport: "IST" }, { src_airport: "IST" }] }, { $or: [{ stops: 0 }, { airline.name: "Turkish Airlines"}] },] })

A. All nonstop flights from Istanbul airport

B. All flights either departing or landing at Istanbul airport that are operated by Turkish Airlines or have zero stops

C. All flights that have at least one stop

D. All flights with no specified conditions

The query presented is structured to find documents from the "routes" collection that satisfy a combination of conditions using the logical operators $and and $or. The first part of the query uses $or to look for routes where either the destination airport (dst_airport) is "IST" or the source airport (src_airport) is also "IST". This means it will retrieve all flights that either land at or depart from Istanbul airport. The second part of the query again employs $or, but focuses on flights that either have no stops (stops: 0) or are operated by "Turkish Airlines" (airline.name: "Turkish Airlines"). By combining these two conditions with $and, the query effectively captures all flights that either depart from or land at Istanbul airport and fulfills at least one of the criteria from the second part, which are either being nonstop flights or operated by Turkish Airlines. Thus, the overall result of the query is that it will return all flights that are either departing from or landing at Istanbul airport and are either nonstop or operated by Turkish Airlines. This aligns perfectly with the interpretation provided in the answer, making it the correct choice.

4. In MongoDB, why should transactions be used when updating account balances across multiple collections?

A. They provide faster updates

B. They ensure operations are atomic across multiple documents

C. They eliminate the need for aggregation pipelines

D. They reduce storage requirements

Using transactions in MongoDB to update account balances across multiple collections is crucial because they ensure operations are atomic across multiple documents. This means that either all the operations in the transaction are completed successfully, or none of them are applied at all. In the context of updating account balances, this atomicity is vital. For instance, if you are deducting an amount from one account and adding it to another, you want to make sure that both operations succeed or fail together. If one operation succeeds but the other fails, it could lead to data inconsistencies, such as an incorrect balance in either account. Transactions prevent this scenario by allowing you to maintain the integrity and consistency of your data throughout the entire process. The other options do not align with the primary purpose of transactions. Faster updates, elimination of aggregation pipelines, and reduced storage requirements are not guaranteed benefits of using transactions in MongoDB. Transactions focus on maintaining data consistency and integrity rather than performance enhancements or storage optimizations.

5. Are indexes created automatically based on usage patterns in MongoDB?

A. Yes, always

B. No, they must be created manually

C. Only for compound indexes

D. Only when specified by the user

In MongoDB, indexes are not created automatically based on usage patterns. Instead, they must be created manually by the developer or database administrator. This manual process allows for precise control over which fields are indexed, enabling optimized query performance tailored to the specific needs of the application. By requiring that indexes be defined explicitly, MongoDB empowers users to understand and manage the performance trade-offs associated with index creation, such as increased storage requirements and potentially slower write operations. The other options suggest varying degrees of automatic behavior for index creation, which does not align with MongoDB's operational model. Specifically, while it can analyze query performance and suggest indexes, it remains the user's responsibility to implement these suggestions and create indexes as deemed necessary for their application's performance.

6. What parameters can be provided to the countDocuments() method?

A. Only a query document

B. A query document and options document

C. Only an options document

D. No parameters are needed

The countDocuments() method in MongoDB provides a way to count the number of documents in a collection that match a specified query. The correct answer highlights that it can accept both a query document and an options document. When using countDocuments(), the query document allows you to specify criteria that filter the documents to count based on given conditions, such as matching a specific field value or applying multiple conditions with logical operators. This enables users to perform precise counts that align with their querying needs. Additionally, the options document can include several configurations that influence how the count is performed. For example, options can include the "collation" parameter, which allows the count to be case-insensitive or locale-specific, thereby enhancing the functionality and versatility of the method to cater to specific use cases. Thus, by accepting both a query document and an options document, countDocuments() empowers developers to efficiently obtain counts that are tailored to their specific requirements.

7. Which of the following queries correctly uses the $elemMatch operator?

A. { "items": { $elemMatch: { name: "item1", quantity: { $gt: 0 } } }}

B. { "items": { $elemMatch: { name: "item1" }} }

C. { "items": { $and: [{ name: "item1" }, { quantity: { $gt: 0 } }] }}

D. { "items": { name: "item1" }}

The use of the $elemMatch operator is specifically designed for querying arrays in a document to ensure that the elements within those arrays meet multiple criteria. In the context of the query given, the correct answer demonstrates this functionality effectively. In the query that selects documents where the "items" array contains an element that meets both of the specified conditions—having the name "item1" and having a quantity greater than zero—$elemMatch is the correct operator to utilize. This operator not only identifies the target element within the array but also asserts that both conditions apply to the same array element. The written structure, `{ "items": { $elemMatch: { name: "item1", quantity: { $gt: 0 } } }}`, ensures that the query only returns documents containing an "items" array member where both the specified name and the quantity condition are satisfied simultaneously, making it precise and powerful for such scenarios. In contrast, while the other options present various query expressions for filtering arrays or fields, they either do not consolidate the conditions to a single element within the array (as seen in options that utilize $and or straightforward field queries) or would result in less specificity for the desired outcome. Hence, the chosen query correctly represents the

8. How does the Performance Advisor analyze collections in MongoDB?

A. By tracking user interactions

B. By monitoring query performance

C. By reviewing system resource usage

D. By evaluating schema structures

The Performance Advisor in MongoDB primarily analyzes collections by monitoring query performance. This tool is specifically designed to enhance the efficiency of your database by identifying slow queries, which may lead to performance issues. It gathers data on how queries interact with the database, including which queries take the longest, their frequency, and whether they are utilizing indexes effectively. Through this monitoring process, the Performance Advisor can provide insights and recommendations for optimizing queries, such as suggesting appropriate indexes that could improve performance. By focusing on query performance, users can make informed decisions about how to enhance their application's speed and efficiency when interacting with the database, leading to overall better system responsiveness. While other options discuss factors that can influence performance, such as user interactions, resource usage, or schema structures, the distinctive focus of the Performance Advisor is on how queries are executed and how they can be improved. Thus, monitoring query performance is central to its function.

9. Data that is assessed together should be...

A. stored separately

B. stored in different databases

C. stored together

D. archived for later use

Data that is assessed together should be stored together because this approach aids in reducing complexity during data retrieval, enhances performance, and maintains the logical relationships between data elements. When data is stored together, it minimizes the need for complex joins or data lookups across multiple sources, which can slow down query performance. In many database designs, particularly in NoSQL databases like MongoDB, the idea is to store related information in close proximity to one another, often within the same document or collection. This not only simplifies the data model but also allows for more efficient access patterns, as the associated data can be retrieved in a single read operation. This practice supports efficient data access patterns and can improve application performance significantly, making it crucial to consider how data interrelations affect storage strategies. Storing related data together aligns with principles of data locality and optimal access in NoSQL databases.

10. What is the function of the upsert option in MongoDB operations?

A. It updates all matching documents

B. It inspects a document before making changes

C. It inserts a document if none exist based on the query

D. It creates a backup of the document before updating

The upsert option in MongoDB operations serves the important function of inserting a new document if no documents match the specified query criteria. When you perform an update operation with the upsert option set to true, MongoDB first attempts to find a document that meets the query's criteria. If such a document is found, it gets updated with the new data specified in the update operation. Conversely, if no matching document exists, MongoDB will create a new document, incorporating the fields from the update statement as well as any fields specified in the query. This feature is particularly useful in scenarios where you want to ensure that a document either exists (and is updated) or is created if it doesn't. The use of upsert can help streamline the logic in your application by reducing the need to perform separate check-and-insert operations, allowing for a more efficient interaction with the database. In this context, the other options do not capture the primary function of upsert. For instance, updating all matching documents is a standard behavior of the update operations regardless of the upsert option. Inspecting a document before making changes does not represent the upsert functionality, as upsert inherently focuses on adding a document if none are found. Additionally, creating a backup of a