UNIT -III MongoDB Limitations

MongoDB Limitations: MongoDB Space Is Too Large (Applicable for MMAPv1), Memory Issues (Applicable for Storage Engine MMAPv1), 32-bit vs. 64-bit, BSON Documents, Namespaces Limits, Indexes Limit, Capped Collections Limit - Maximum Number of Documents in a Capped Collection, Sharding Limitations, Shard Early to Avoid Any Issues, Shard Key Can't Be Updated, Shard Collection Limit, Select the Correct Shard Key, Security Limitations, No Authentication by Default, Traffi c to and from MongoDB Isn't Encrypted, Write and Read Limitations, Case-Sensitive Queries, Type-Sensitive Fields, No JOIN, Transactions, MongoDB Not Applicable Range

PYQ : (Previous Year Mumbai University Question)

Nov – 18

1 : List and explain limitation of sharding

Apr -19

1 : List and explain limitation of mongodb

Nov – 19

1. List and explain limitation of Indexes
2. Explain the MongoDB limitation from Security Perspective

Nov – 22

1. What is sharding . List and explain the limitations of sharding .
2. Explain MongoDB Limitations from security perspective .Also give an overview about read and write limitations

Dec – 23

1. Outline limitation of i) Sharding ii) 32 bit – 64 bit

Nov – 24

1. Explain the limitations of sharding in MongoDB

MongoDB Space Is Too Large (Applicable for MMAPv1) :

#### **Introduction**
In MongoDB, when using the MMAPv1 storage engine (the default before MongoDB 3.2), you might encounter a situation where the database space on the disk seems much larger than the actual data it stores. This is a common issue related to how the MMAPv1 engine manages space.

#### **Why Does This Happen?**
The MMAPv1 storage engine handles data by using memory-mapped files. These files allocate a fixed amount of space on disk, which can lead to inefficient storage use. Here's why this happens:

1. **Preallocation of Space:**
  - MMAPv1 preallocates disk space to avoid frequent resizing of files, which can be costly in terms of performance.

- For example, if MongoDB anticipates that your database will grow, it might allocate more space on disk than you actually need at the moment.
- This preallocated space remains reserved on disk, even if it's not filled with actual data.

2. **Document Deletion:**
- When you delete documents in MongoDB, the space they occupied isn't immediately released.
- Instead, MongoDB marks this space as free within the existing files, which can be reused for new data.
- However, if the new data doesn't perfectly fit into the freed space, MongoDB might need to allocate additional space on disk, leading to "holes" or fragmentation.

3. **Fragmentation:**
- Over time, as documents are added, deleted, or updated, the storage files can become fragmented, meaning that free space is scattered in small, unusable chunks.
- This fragmentation makes it difficult for MongoDB to efficiently use the preallocated space, leading to more space being used on the disk than the actual data size.

#### **Example Explanation**
Imagine you have a notebook where you jot down notes on every page. Each page has 100 lines. Initially, you write on 10 pages, but you skip some lines here and there. Later, you realize that the notes on pages 2, 5, and 7 are no longer needed, so you cross them out. However, when you want to write new notes, you find that the empty lines left after crossing out aren't enough to write a complete new note, so you start using new pages instead.

After some time, even though you've only written notes worth 10 pages, your notebook might have 20 pages used, with scattered empty lines here and there. This is similar to how MongoDB ends up using more disk space than necessary due to preallocation and fragmentation.

#### **How to Address This Issue?**

1. **Compact Command:**

   - MongoDB provides a `compact` command that attempts to defragment the data, reorganizing the storage files to reduce the amount of wasted space.
   - However, this command locks the database, making it unavailable for other operations while it's running, so it should be used with caution.

2. **Repair Database:**
   - The `repairDatabase` command can also be used to reclaim space, but it creates a copy of the database, requiring sufficient free disk space to run.
   - This command is more comprehensive but can be time-consuming.

3. **Switch to WiredTiger:**
   - Since MongoDB 3.2, the WiredTiger storage engine has become the default, offering more efficient space management.
   - WiredTiger doesn't preallocate space in the same way as MMAPv1 and handles fragmentation better, reducing the chances of excessive disk space usage.

#### **Conclusion**
When using the MMAPv1 storage engine, the disk space used by MongoDB might appear too large compared to the actual data size due to preallocation, document deletion, and fragmentation. While MongoDB provides tools to manage and reduce this space, switching to the WiredTiger engine in newer versions can help avoid these issues altogether.

Memory Issues (Applicable for Storage Engine MMAPv1) :

MMAPv1 is one of MongoDB's older storage engines. It uses memory-mapped files, meaning the database data is directly mapped to the server's memory.

### Common Memory Issues:

1. **High Memory Usage:**
   - **Reason:** MMAPv1 loads large portions of data into memory for faster access.
   - **Impact:** Can cause the server to run out of memory, slowing down or crashing.

2. **Memory Fragmentation:**
   - **Reason:** Frequent updates and deletes can create gaps in memory, leading to inefficient use of space.
   - **Impact:** Over time, this reduces available memory and performance.

3. **Page Faults:**
   - **Reason:** When MongoDB needs data that isn't in memory, it retrieves it from disk, causing delays.
   - **Impact:** High page faults slow down database operations.

### Example Explanation:

Imagine your computer has a desk (memory) to work on. If the desk is too cluttered (high memory usage), it's hard to find things (data), slowing you down. If you keep shuffling papers around without organizing them (memory fragmentation), the clutter gets worse. Sometimes, you may need to get a paper from a drawer (disk), which takes time (page faults).

In MMAPv1, these issues can make your MongoDB database slower and less efficient.

32Bit vs 64 Bit :

### MongoDB Versions: 32-bit vs. 64-bit

#### Overview
MongoDB is available in two versions based on the underlying architecture of the system: **32-bit** and **64-bit**. The choice between these versions depends on the operating system and hardware architecture of the machine you are using.

#### 1. **32-bit Version**

- **Memory Limitation**: The 32-bit version of MongoDB has a significant limitation. It can only address up to 2 GB of data. This means that your MongoDB database, including its data, indexes, and other associated structures, cannot exceed 2 GB.
- **Use Case**: The 32-bit version is primarily intended for testing, development, or small-scale applications that don't require large data storage. It's generally not recommended for production environments due to its memory constraint.
- **Example**: Suppose you are building a small application that stores a limited amount of data, like a personal project or a small website. The 32-bit version could be sufficient in such cases since you don't expect to exceed the 2 GB limit.

#### 2. **64-bit Version**
- **No Memory Limitation**: The 64-bit version of MongoDB doesn't have the 2 GB limitation. It can handle much larger datasets, limited only by the available disk space and memory of the system.
- **Performance**: In addition to supporting larger datasets, the 64-bit version also offers better performance and scalability. It's designed to handle large-scale applications and production environments.
- **Use Case**: The 64-bit version is recommended for production environments and applications that require the storage and processing of large amounts of data, such as big data applications, enterprise-level systems, or any scenario where the database size is expected to grow significantly.
- **Example**: Imagine you are building a social networking site where users can upload photos, videos, and interact with each other. The data will grow rapidly as the number of users increases. In this case, you would choose the 64-bit version of MongoDB to ensure your system can handle the large amount of data without running into memory limitations.

#### 3. **Why It Matters**
- **System Compatibility**: The choice between 32-bit and 64-bit depends on the architecture of your system. If your operating system and hardware support 64-bit, it's generally better to choose the 64-bit version for better performance and scalability.
- **Future Growth**: Even if you start with a small dataset, it's wise to consider future growth. The 64-bit version provides the flexibility to scale as your data grows.

#### Conclusion

In summary, the 32-bit version of MongoDB is limited in terms of memory and is best suited for small-scale or testing environments. The 64-bit version, on the other hand, is more powerful, capable of handling large datasets, and is ideal for production environments. Choosing the right version depends on your specific needs, but for most practical purposes, the 64-bit version is the better choice.

This explanation covers the topic in simple terms, providing a clear understanding of the differences between the two versions and when to use each.

## BSON Document :

### BSON Document in MongoDB

****1. What is BSON?****

- BSON (Binary JSON) is a binary-encoded serialization format used to store documents in MongoDB. It is similar to JSON but is designed to be more efficient in terms of space and speed.

- BSON extends the JSON model to provide additional data types like int, long, date, and binary, which are not available in JSON.

****2. Structure of a BSON Document****

- A BSON document is a collection of field-value pairs, where each field (or key) is a string, and the value can be of various data types like string, integer, array, or even another document (nested document).

- Example:

```
{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "name": "Alice",
  "age": 29,
  "address": {
    "street": "123 Main St",
    "city": "Springfield"
  },
  "hobbies": ["reading", "traveling", "swimming"]
}
```

- In this example:
  - The `_id` field is of type `ObjectId`.
  - The `name` field is a string.
  - The `age` field is an integer.
  - The `address` field is a nested document (another BSON document).
  - The `hobbies` field is an array of strings.

3. BSON Data Types

- BSON supports a variety of data types, some of which are:
  - **String**: Stores textual data.
  - **Integer**: Stores numbers without a decimal point.
  - **Double**: Stores numbers with a decimal point.
  - **Boolean**: Stores `true` or `false`.
  - **Date**: Stores date and time.
  - **ObjectId**: A unique identifier generated by MongoDB.
  - **Array**: Stores a list of values, which can be of different types.
  - **Embedded Document**: A document within another document.
  - **Binary Data**: Stores binary data like images, files, etc.

Example:

```
{
  "product": "Laptop",
  "price": 1200.99,
  "available": true,
  "releaseDate": ISODate("2023-01-01T00:00:00Z"),
  "specs": {
    "processor": "Intel i7",
    "memory": "16GB"
  }
}
```

- Here:
  - `product` is a string.
  - `price` is a double.
  - `available` is a boolean.
  - `releaseDate` is a date.
  - `specs` is an embedded document.

****4. Why Use BSON in MongoDB?****

- **Efficiency**: BSON is more efficient in terms of space and speed compared to JSON, which is important for large-scale applications.

- **Flexibility**: BSON allows for the storage of complex data structures, such as nested documents and arrays, making it suitable for handling real-world data.

- **Native Support**: MongoDB natively supports BSON, making it the ideal format for storing and retrieving data from the database.

****5. Converting JSON to BSON****

- When you insert a document into MongoDB, it is automatically converted from JSON to BSON.

- For example, if you insert the following JSON document:

```
{
  "name": "John Doe",
  "age": 35,
  "email": "john@example.com"
}
```

- MongoDB converts it to BSON and stores it in the database.
- The `_id` field is automatically added by MongoDB to uniquely identify the document.

6. Working with BSON in MongoDB

- When you retrieve a document from MongoDB, it is automatically converted from BSON to JSON format, which can be easily used in your application.
- Example:

```
# Python code to retrieve a document
document = collection.find_one({"name": "John Doe"})
print(document)
```

- This will output the document in JSON format, even though it was stored in BSON.

7. Example Explanation

- Let's say you are building an e-commerce website. Each product is stored in a MongoDB collection as a BSON document.
- Example product document:

```
{
  "_id": ObjectId("5f1f77bcf86cd799439011"),
  "productName": "Smartphone",
  "brand": "TechBrand",
  "price": 699.99,
  "specifications": {
    "screenSize": "6.5 inches",
    "battery": "4000mAh",
    "camera": "12MP"
  },
  "inStock": true
}
```

- Here:
  - `productName` is a string representing the name of the product.
  - `brand` is a string representing the brand.
  - `price` is a double representing the price of the product.
  - `specifications` is an embedded document with details about the product.
  - `inStock` is a boolean indicating whether the product is in stock.

This document is stored as BSON in MongoDB, making it compact and efficient for storage and retrieval.

In summary, BSON is a powerful and efficient way to store complex data structures in MongoDB, making it ideal for modern applications that require flexible data handling.

Namespace limit :

#### What is a Namespace in MongoDB?

In MongoDB, a **namespace** is a combination of the database name and the collection name. For example, if you have a database named `school` and a collection named `students`, the namespace for this collection would be `school.students`.

#### Understanding the Namespace Limit

MongoDB has a limitation on the number of namespaces that can exist in a single database. This is primarily due to the way namespaces are stored and managed internally.

- **Namespace Storage**: Each namespace in MongoDB (like `school.students`) is stored in a special file on disk called the **namespace file**. This file has a fixed size.

- **Maximum Number of Namespaces**: The maximum number of namespaces in a MongoDB database depends on the version and the storage engine used. In earlier versions, using the MMAPv1 storage engine, this limit was more strict (around 24,000 namespaces). However, with the WiredTiger storage engine, this limit is less of a concern, but it's still something to be aware of.

- **Namespace Limit in WiredTiger**: In the WiredTiger storage engine, each collection and index counts as a namespace. The limit is generally much higher, but creating an excessive number of collections and indexes can still lead to performance issues.

#### Example Explanation

Let's consider a scenario:

- **Scenario:** You have a database named `company` where you store employee information. Each department has its own collection, like `company.hr`, `company.finance`, and `company.engineering`.

- **Collections and Indexes:** If you have 10 departments, and each department's collection has 3 indexes, you would have a total of 10 collections + (10 * 3) indexes = 40 namespaces.

- **Impact of Namespace Limit:** If you try to create too many collections or indexes, especially in older versions or under specific conditions, you might hit the namespace limit. When this happens, MongoDB will prevent you from creating new collections or indexes, which could disrupt your application.

## Indexes Limit :

### Indexes Limit in MongoDB

Indexes are critical in MongoDB to speed up query operations. However, there are certain limits and considerations when working with indexes. Below is a detailed explanation of the various aspects of index limits:

#### 1. **Index Size**

- **Explanation**: The size of an index in MongoDB is limited by the memory and storage available. Indexes are stored in memory, and the total index size must fit within the available RAM. If the index size exceeds the available memory, it may result in performance issues.

- **Example**: Imagine you have a large collection with millions of documents. If you create an index on a large text field, the index size might become very large. If the index can't fit in memory, queries that use this index will slow down significantly.

#### 2. **Number of Indexes Per Collection**

- **Explanation**: MongoDB allows you to create multiple indexes on a collection to support various types of queries. However, there is a limit on the number of indexes you can create per collection, which is 64.

- **Example**: Suppose you have a collection that stores information about books. You might create indexes on the `title`, `author`, and `ISBN` fields to speed up searches. If you try to create more than 64 indexes on this collection, MongoDB will not allow it.

#### 3. **Index Name Length**

- **Explanation**: The name of an index in MongoDB has a maximum length of 128 characters. The index name is used internally by MongoDB to identify the index, and exceeding this length can cause errors.

- **Example**: If you try to create an index with a very long name, such as `index_on_title_author_publisher_year_edition`, and the name exceeds 128 characters, MongoDB will return an error, and you'll need to shorten the name.

#### 4. **Unique Indexes in Sharded Collections**

- **Explanation**: In a sharded MongoDB cluster, you can create unique indexes, but they must include the shard key. This ensures that the uniqueness constraint is enforced across all shards.

- **Example**: If you have a sharded collection where the shard key is `user_id`, and you want to create a unique index on the `email` field, you must include the `user_id` in the index like this: `{user_id: 1, email: 1}`. This way, the uniqueness of `email` is maintained across the entire cluster.

#### 5. **Number of Indexed Fields in a Compound Index**

- **Explanation**: A compound index in MongoDB can include up to 32 fields. Compound indexes allow you to create an index that supports queries on multiple fields.

- **Example**: Suppose you have a collection of orders, and you want to create an index to support queries on `customer_id`, `order_date`, and `status`. You can create a compound index like `{customer_id: 1, order_date:

1, status: 1}`. If you try to include more than 32 fields in this index, MongoDB will not allow it.

### Summary

MongoDB imposes certain limits on indexes to ensure optimal performance and efficient use of resources. Understanding these limits helps in designing indexes that are both effective and within the constraints of the system.

## Capped Collections Limit - Maximum Number of Documents in a Capped Collection :

****Capped collections**** in MongoDB are special types of collections that maintain a fixed size. When the collection reaches its maximum size, it automatically overwrites the oldest documents with the newest ones. This behavior makes capped collections ideal for scenarios like logging, where you always want to keep the most recent data and discard older entries.

### Capped Collections Limit: Maximum Number of Documents

In MongoDB, a capped collection has two main constraints:

1. **Maximum Size (in bytes):** This is the total space allocated for the collection. Once the collection reaches this size, MongoDB starts overwriting the oldest documents.

2. **Maximum Number of Documents:** Unlike regular collections, capped collections can also be configured to have a maximum number of documents. This means that once the collection reaches this document limit, it will start to overwrite the oldest documents with the new ones, even if the size limit has not been reached.

### Detailed Explanation

- **Size Limit**: The size limit is defined in bytes, and you set it when you create the capped collection. For example, you might create a capped collection with a size limit of 100 MB.

- **Document Count Limit**: The maximum number of documents is another constraint that you can set when creating a capped collection. For example, you might specify that the collection can hold a maximum of 1,000 documents.

- **Behavior**: When either the size limit or the document count limit is reached, MongoDB starts removing the oldest documents to make space for new ones. This ensures that the collection remains within its defined limits.

### Example Explanation

Imagine you are building a system to store logs from a web server. You want to keep only the last 1,000 log entries, regardless of how much space they take up. Here’s how you could do it:

1. **Creating the Capped Collection**:

```
  db.createCollection("serverLogs", { capped: true, size: 104857600, max:
1000 })
```

  - `capped: true` tells MongoDB to create a capped collection.

  - `size: 104857600` sets the maximum size to 100 MB.

  - `max: 1000` limits the collection to 1,000 documents.

2. **Behavior**:

- As new logs are added, they are inserted into the `serverLogs` collection.

- Once the collection has 1,000 documents, adding a new log entry will remove the oldest log entry, keeping the total number of documents at 1,000.

- If the collection reaches 100 MB before reaching 1,000 documents, it will start overwriting the oldest documents based on the size limit.

Sharding Limitations :

#### Question 1 : List and explain limitation of sharding

Sharding is the mechanism of splitting data across shards.

The following sections talk about the limitations that you need to be aware of when dealing with sharding.

#### Shard Early to Avoid Any Issues :

- Using the shard key, the data is split into chunks, which are then automatically distributed amongst the shards.
- However, if sharding is implemented late, it can cause slowdowns of the servers because the splitting and migration of chunks takes time and resources.
- A simple solution is to monitor your MongoDB instance capacity using tools such as MongoDB Cloud Manager (flush time, lock percentages, queue lengths, and faults are good measures) and shard before reaching 80% of the estimated capacity.

#### Shard Key Can't Be Updated :

- The shard key can't be updated once the document is inserted in the collection because MongoDB uses shard keys to determine to which shard the document should be routed.
- If you want to change the shard key of a document, the suggested solution is to remove the document and reinsert the document when he change has been made.

#### shard Collection Limit :

- The collection should be sharded before it reaches 256GB.

#### Select the Correct Shard Key :

- It’s very important to choose a correct shard key because once the key is chosen it’s not easy to correct it

----

Security Limitations :

Security is an important matter when it comes to databases. Let’s look at MongoDB limitations from security perspective

#### No Authentication by Default

Although authentication is not enabled by default, it’s fully supported and can be enabled easily.

#### Traffic to and from MongoDB Isn’t Encrypted :

By default the connections to and from MongoDB are not encrypted. When running on a public network, consider encrypting the communication; otherwise it can pose a threat to your data. Communications on a public network can be encrypted using the SSL-supported build of MongoDB, which is available in the 64-bit version only.

Write and Read Limitations :

Case-Sensitive Queries By default :

- MongoDB is case sensitive

- For example, the following two commands will return different results: db.books.find({name: 'PracticalMongoDB'}) and db.books.find({name: 'practicalmongodb'}) . You should ensure that you know in which case the data is stored. Although regex searches like db.books.find({name: /practicalmongodb/i}) can be used, they aren't ideal because they are relatively slow.

Type- Sensitive Fields :

- Since there's no enforced schema for documents in MongoDB, it can't know you are making a

No JOIN :

- Joins are not supported in MongoDB. If you need to retrieve data from more than one collection, you must do more than one query.
- However, you can redesign the schema to keep the related data together so that the information can be retrieved in a single query.mistake. You must make sure that the correct type is used for the data.

Replica Set Limitations - Number of Replica Set Members :

- A replica set is used to ensure data redundancy in MongoDB. One member acts as a primary member and the rest act as secondary members. Due to the way voting works with MongoDB, you must use an odd number of members.

Transactions MongoDB only supports single document atomicity. Since a write operation can modify multiple documents, this operation is not atomic. However, you can isolate write operations that affect multiple documents using the isolation operator.

MongoDB Not Applicable Range :

MongoDB is not suitable for the following:

• Highly transactional systems such as accounting or banking systems. Traditional RDBMS are still more suitable for such applications, which require a large number of atomic complex matters.

• Traditional business intelligence applications, where an issue-specific BI database would generate highly optimized queries. For such applications, the data warehouse may be a more appropriate choice.

• Applications requiring complex SQL queries.

• MongoDB does not support transactional operations, so a banking system certainly cannot use it.