Programme No 1S00255, QP Code: 75440, Next Generation Technologies, Solution Set

Answer: 1

(a) What is Big Data? List the different uses of Big Data.

"Big data is a term used to describe data that has massive volume, comes in a variety of structures, and is generated at high velocity. This kind of data poses challenges to the traditional RDBMS systems used for storing and processing data. Bid data is paving way for newer approaches of processing and storing data."

This can be broadly categorized into five ways of usage of big data.

Visibility

Accessibility to data in a timely fashion to relevant stakeholders generates a tremendous amount of value. Example:-Consider a manufacturing company that has R&D, engineering, and manufacturing departments dispersed geographically. If the data is accessible across all these departments and can be readily integrated, it can not only reduce the search and processing time but will also help in improving the product quality according to the present needs.

Discover and Analyse Information

Most of the value of big data comes from when the data collected from outside sources can be merged with the organization's internal data. Organizations are capturing detailed data on inventories, employees, and customers. Using all of this data, they can discover and analyse new information and patterns; as a result, this information and knowledge can be used to improve processes and performance.

<u>Segmentation and Customizations</u>

Big data enables organizations to create tailor-made products and services to meet specific segment needs. This can also be used in the social sector to accurately segment populations and target benefit schemes for specific needs. Segmentation of customers based on various parameters can aid in targeted marketing campaigns and tailoring of products to suit the needs of customers.

<u>Aiding Decision Making</u>

Big data can substantially minimize risks, improve decision making, and uncover valuable insights. Automated fraud alert systems in credit card processing and automatic fine-tuning of inventory are examples of systems that aid or automate decision-making based on big data analytics.

<u>Innovation</u>

Big data enables innovation of new ideas in the form of products and services. It enables innovation in the existing ones in order to reach out to large segments of people. Using data gathered for actual products, the manufacturers can not only innovate to create the next generation product but they can also innovate sales offerings.

(b) Briefly explain how is MongoDB different from SQL.

The differentiation is based on the following concepts :

1) **Type of database**

   SQL is called ***a relational database*** as it organizes structured data into defined rows and columns, with each table being related to the other tables in the database.

NoSQL, on the other hand, is known ***as a non-relational database***. This is because data is stored in the form of collections with no or few relations between them.

2) Schema

SQL needs a ***predefined schema*** for structured data. So, before you start using SQL to extract and manipulate data, you need to make sure that your data structure is pre-defined in the form of tables.

However, NoSQL, have a ***dynamic schema*** for unstructured data. So, if you are using a NoSQL database, then there is no pre-defined schema present, and the complete schema of your data completely depends upon how you wish to store data. i.e. which fields do you want to store in documents and collections.

3)Complex Queries

SQL is a ***better fit for complex query environment*** when compared to NoSQL as the schema in SQL databases is structured and has data stored in a tabular format. So, even if you wish to apply nested queries with many subqueries inside the outer query, you can easily do by using the proper table and column names.

Now, the reason why ***NoSQL databases isn't a good fit for complex queries*** is because the NoSQL databases aren't queried in a standard language like SQL.

4) Hierarchical Data Storage

Well, when we compare the databases on this factor, ***NoSQL fits better for hierarchical storage*** when compared to SQL databases.

This is because as the number of tables increases, the complexity of maintaining relations between them also keeps increasing. So, in such a scenario, you cannot relate the humongous amount of tables with many columns in them to each other. But, when you consider a NoSQL database, this kind of database fits better for the hierarchical data storage as it follows the key-value pair way of storing data which is similar to JSON data.

5) Scalability

The SQL databases are ***vertically scalable***. You can load balance the data servers by optimizing hardware such as increasing CPU, RAM, SSD, etc.

On the other hand, NoSQL databases are ***horizontally scalable***. You can perform load balancing by adding more servers to your cluster to handle a large amount of traffic

(c) What is MongoDB? Explain the features of MongoDB.

MongoDB is a NoSQL database which stores the data in form of key-value pairs. It is an **Open Source**, **Document Database** which provides high performance and scalability along with data modelling and data management of huge sets of data in an enterprise application. MongoDB also provides the feature of Auto-Scaling.

The features of Mongodb are as following:

- MongoDB is an open-source, cross-platform, document-oriented database that provides high performance, high availability, and automatic scaling. It is also classified as a NoSQL database.
- The document is the unit of storing data in a MongoDB database.
- MongoDB stores documents in collections.
- A collection may store a number of documents. A collection is similar to a table of an RDBMS.
- MongoDB documents are similar to JSON objects. MongoDB stores data in the form of BSON -Binary encoded JSON documents.
- It can store any type of data: structured, semi-structured and polymorphic.

- Supporting a different set of fields for each document in a collection is one of MongoDB’s features. It allows you to store similar data, but with different properties in the same collection.
- The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
- Indexing – Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.
- MongoDB supports searching by arena, range of queries, and daily face searches. Queries can be complete to return exact fields within documents.
- MongoDB provides high performance. Input/Output operations are lesser than relational databases due to support of embedded documents(data models) and Select queries are also faster as Indexes in MongoDB supports faster queries.
- MongoDB has a rich Query Language, supporting all the major CRUD operations. The Query Language also provides good Text Search and Aggregation features.
- **Auto Replication** feature of MongoDB leads to High Availability. It provides an automatic failover mechanism, as data is restored through backup(replica) copy if server fails.
- Sharding is a major feature of MongoDB. Horizontal Scalability is possible due to sharding.
- MongoDB supports multiple Storage Engines. When we save data in form of documents(NoSQL) or tables(RDBMS) who saves the data? It's the Storage Engine. Storage Engines manages how data is saved in memory and on disk.

(d) **Explain how Volume, Velocity and Variety are important components of Big Data.**

Big data can be defined as data with three Vs: volume, velocity, and variety, as shown in below figure.

Volume

Volume in big data means the size of the data. Various factors contribute to the size of big data: as businesses are becoming more transaction-oriented, we see ever increasing numbers of transactions; more devices are getting connected to the Internet, which is adding to the volume; there is an increased usage of the Internet; and there is an increase in the digitization of content.

Variety

The data generated from various devices and sources follows no fixed format or structure. Compared to text, CSV or RDBMS data varies from text files, log files, streaming videos, photos, meter readings, stock ticker data, PDFs, audio, and various other unstructured formats. There is no control over the structure of the data these days.

Velocity

Velocity in big data is the speed at which data is created and the speed at which it is required to be processed. If data cannot be processed at the required speed, it loses its significance. Due to data streaming in from social media sites, sensors, tickers, metering, and monitoring, it is important for the organizations to speedily process data both when it is on move and when it is static. Reacting and processing quickly enough to deal with the velocity of data is one more challenge for big data technology. Real-time insight is essential in many big data use cases.

e) Write a short note on Cap theorem.

Eric Brewer outlined the CAP theorem in 2000. This is an important concept that needs to be well understood by developers and architects dealing with distributed databases. The theorem states that when designing an application in a distributed environment there are three basic requirements that exist, namely consistency, availability, and partition tolerance.

• Consistency means that the data remains consistent after any operation is performed that changes the data, and that all users or clients accessing the application see the same updated data.

• Availability means that the system is always available.

• Partition Tolerance means that the system will continue to function even if it is partitioned into groups of servers that are not able to communicate with one another.

(f) Discuss the various categories of NoSQL Databases.

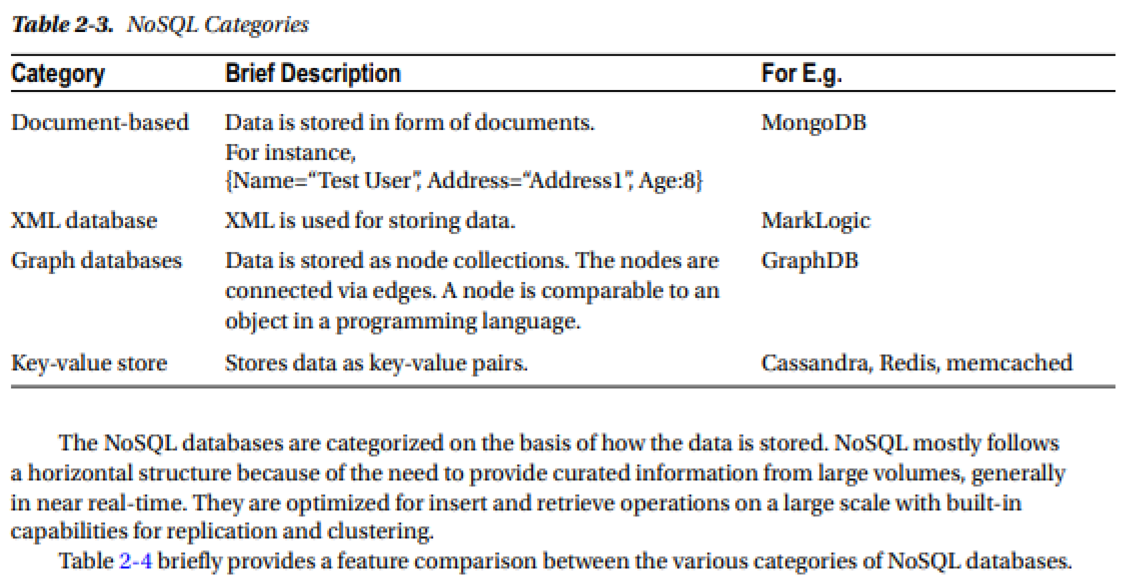


Table 2-3. *NoSQL Categories*

Category	Brief Description	For E.g.
Document-based	Data is stored in form of documents. For instance, {Name="Test User", Address="Address1", Age:8}	MongoDB
XML database	XML is used for storing data.	MarkLogic
Graph databases	Data is stored as node collections. The nodes are connected via edges. A node is comparable to an object in a programming language.	GraphDB
Key-value store	Stores data as key-value pairs.	Cassandra, Redis, memcached

The NoSQL databases are categorized on the basis of how the data is stored. NoSQL mostly follows a horizontal structure because of the need to provide curated information from large volumes, generally in near real-time. They are optimized for insert and retrieve operations on a large scale with built-in capabilities for replication and clustering.

Table 2-4 briefly provides a feature comparison between the various categories of NoSQL databases.

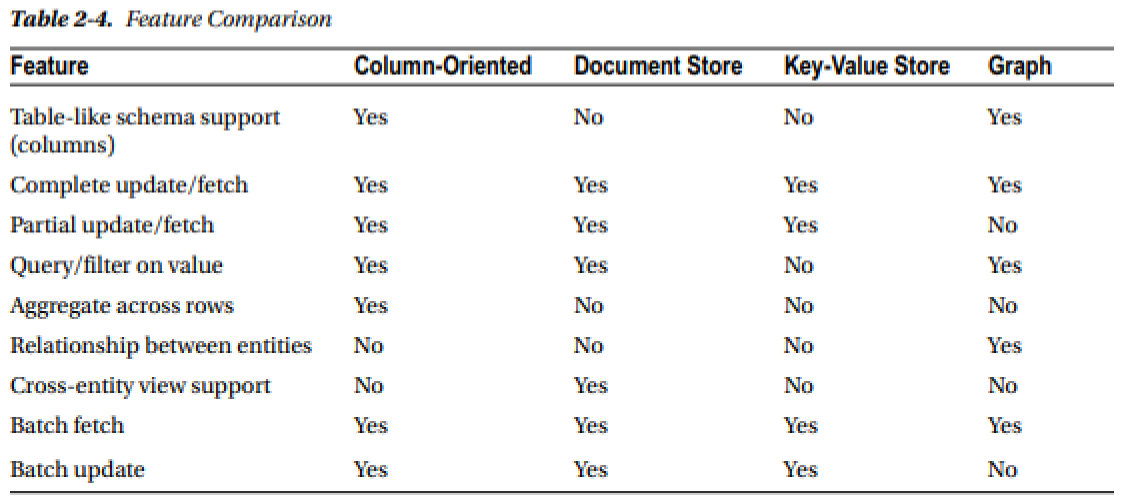


Table 2-4. *Feature Comparison*

Feature	Column-Oriented	Document Store	Key-Value Store	Graph
Table-like schema support (columns)	Yes	No	No	Yes
Complete update/fetch	Yes	Yes	Yes	Yes
Partial update/fetch	Yes	Yes	Yes	No
Query/filter on value	Yes	Yes	No	Yes
Aggregate across rows	Yes	No	No	No
Relationship between entities	No	No	No	Yes
Cross-entity view support	No	Yes	No	No
Batch fetch	Yes	Yes	Yes	Yes
Batch update	Yes	Yes	Yes	No

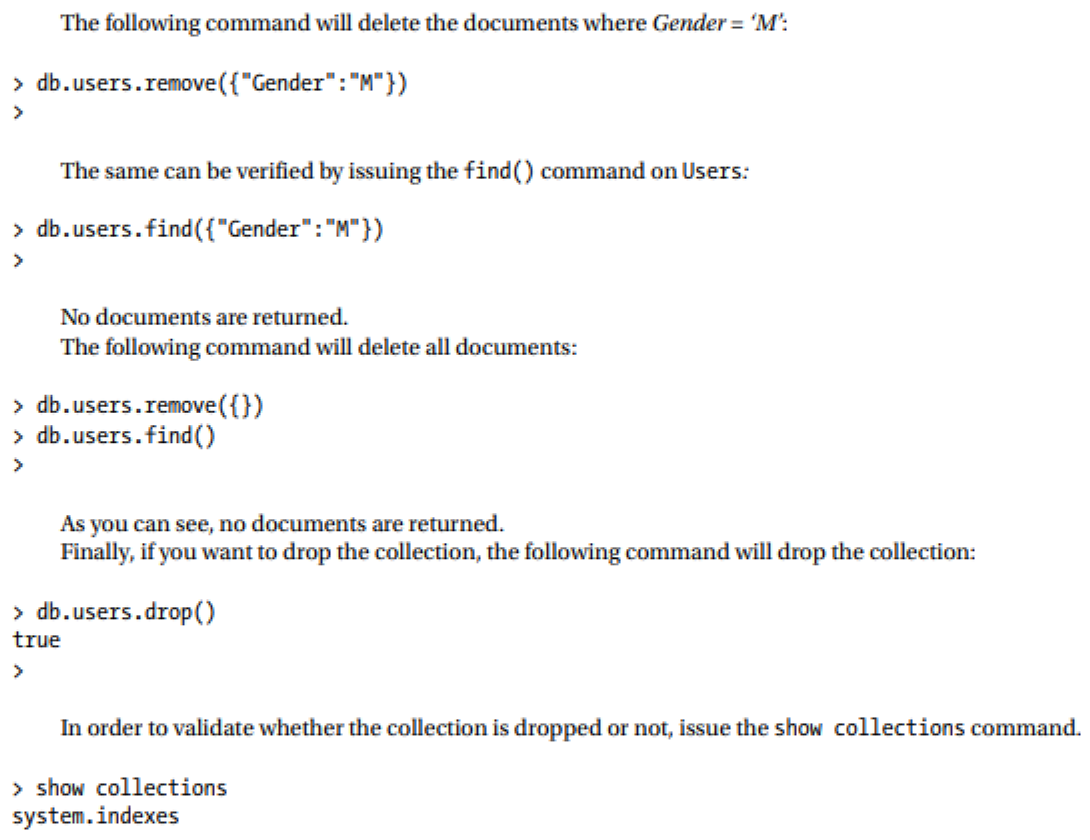
Answer 2

(a) Explain Binary JSON(BSON).

MongoDB stores the JSON document in a binary-encoded format. This is termed as BSON. The BSON data model is an extended form of the JSON data model. MongoDB's implementation of a BSON document is fast, highly traversable, and lightweight. It supports embedding of arrays and objects within other arrays, and also enables MongoDB to reach inside the objects to build indexes and match objects against queried expressions, both on top-level and nested BSON keys.

(b) Explain with an example the process of deleting documents in a Collection.

To delete documents in a collection, use the remove () method . If you specify a selection criterion, only the documents meeting the criteria will be deleted. If no criteria is specified, all of the documents will be deleted.



The following command will delete the documents where *Gender* = *'M'*:

```
> db.users.remove({"Gender":"M"})
>
```

The same can be verified by issuing the `find()` command on `Users`:

```
> db.users.find({"Gender":"M"})
>
```

No documents are returned.
The following command will delete all documents:

```
> db.users.remove({})
> db.users.find()
>
```

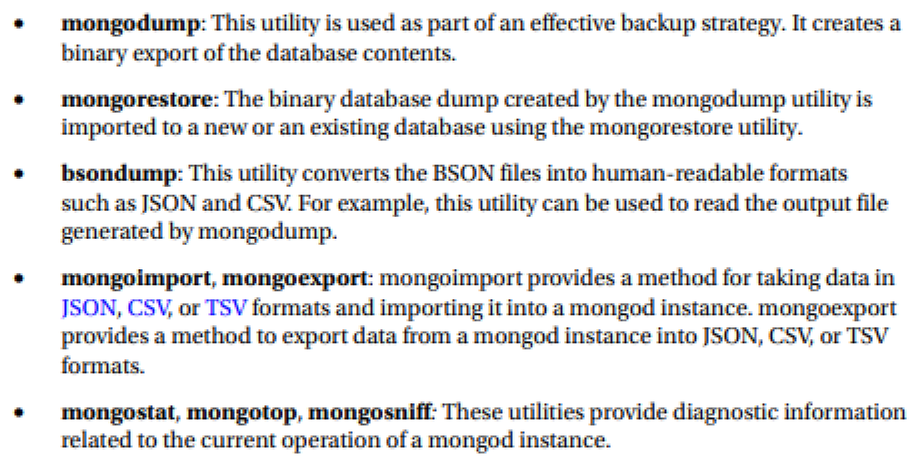
As you can see, no documents are returned.
Finally, if you want to drop the collection, the following command will drop the collection:

```
> db.users.drop()
true
>
```

In order to validate whether the collection is dropped or not, issue the `show collections` command.

```
> show collections
system.indexes
```

(c) Discuss the various tools in MongoDB.

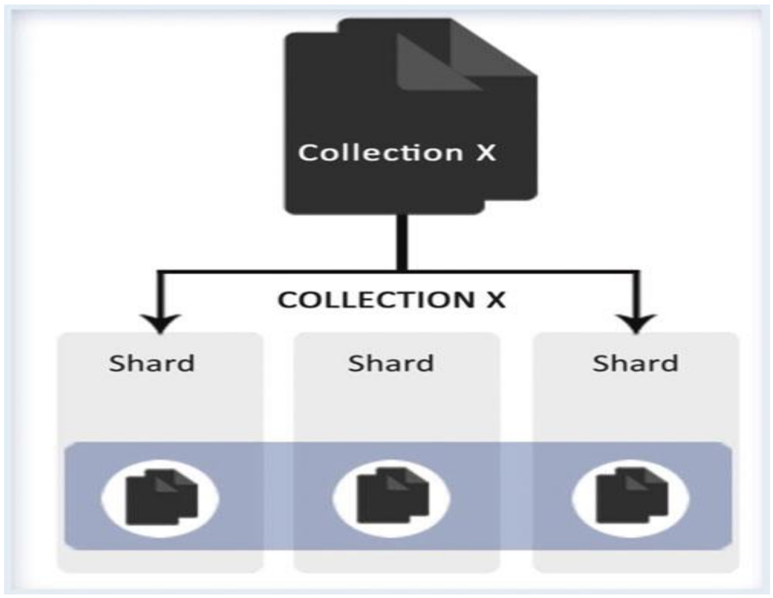


- **mongodump**: This utility is used as part of an effective backup strategy. It creates a binary export of the database contents.
- **mongorestore**: The binary database dump created by the mongodump utility is imported to a new or an existing database using the mongorestore utility.
- **bsondump**: This utility converts the BSON files into human-readable formats such as JSON and CSV. For example, this utility can be used to read the output file generated by mongodump.
- **mongoimport**, **mongoexport**: mongoimport provides a method for taking data in JSON, CSV, or TSV formats and importing it into a mongod instance. mongoexport provides a method to export data from a mongod instance into JSON, CSV, or TSV formats.
- **mongostat**, **mongotop**, **mongosniff**: These utilities provide diagnostic information related to the current operation of a mongod instance.

(d) Explain the concept of Sharding in detail.

In MongoDB, the scaling is handled by scaling out the data horizontally (i.e. partitioning the data acrossmultiple commodity servers), which is also called sharding (horizontal scaling).
Sharding addresses the challenges of scaling to support large data sets and high throughput by horizontally dividing the datasets across servers where each server is responsible for handling its part of dataand no one server is burdened. These servers are also called shards.Every shard is an independent database. All the shards collectively make up a single logical database .Sharding reduces the operations count handled by each shard. For example, when data is inserted, onlythe shards responsible for storing those records need to be accessed.The processes that need to be handled by each shard reduce as the cluster grows because the subset ofdata that the shard holds reduces. This leads to an increase in the throughput and capacity horizontally.Let’s assume you have a database that is 1TB in size. If the number of shards is 4, you will haveapproximately 265GB of data handled by each shard, whereas if the number of shards is increased to 40, only25GB of data will be held on each shard.

Figure depicts how a collection that is sharded will appear when distributed across three shards.



Although sharding is a compelling and powerful feature, it has significant infrastructure requirementsand it increases the complexity of the overall deployment. So you need to understand the scenarios whereyou might consider using sharding.
Use sharding in the following instances:

• The size of the dataset is huge and it has started challenging the capacity of a
single system.

• Since memory is used by MongoDB for quickly fetching data, it becomes
important to scale out when the active work set limits are set to reach.

• If the application is write-intensive, sharding can be used to spread the writesacross multiple servers.

(e) Differentiate between Single Key and Compound Index.

Single Key Index

An index on the Name field of the document. Use ensureIndex() to create the index.

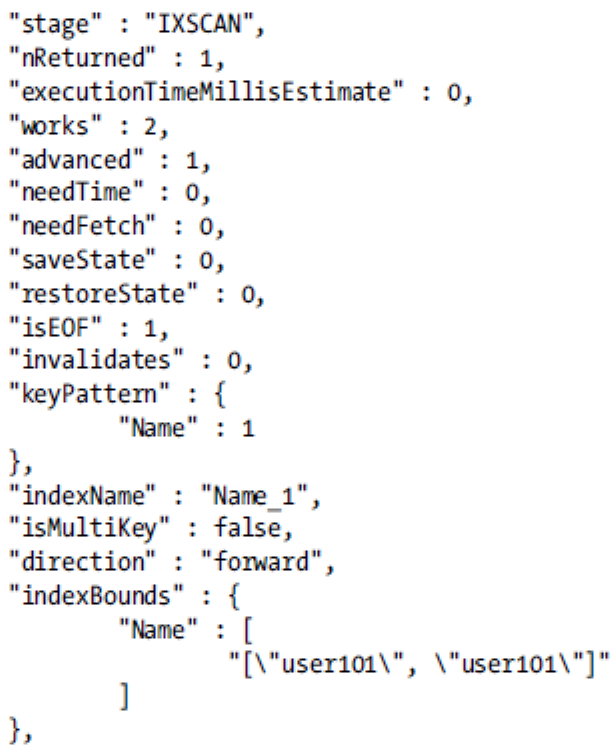
```
> db.testindx.ensureIndex({"Name":1})
```

The index creation will take few minutes depending on the server and the collection size.

Let's run the same query that you ran earlier with `explain()` to check what the steps the database is executing post index creation. Check the `n`, `nscanned`, and `millis` fields in the output.

```
>db.testindx.find({"Name":"user101"}).explain("allPathsExecution")

{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "mydbproc.testindx",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "Name" : {
                                "$eq" : "user101"
                        }
                },
                "winningPlan" : {
                        "stage" : "FETCH",
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "keyPattern" : {
                                        "Name" : 1
                                },
                                "indexName" : "Name_1",
                                "isMultiKey" : false,
                                "direction" : "forward",
                                        "indexBounds" : {
                                                "Name" : [
                                                        "[\"user101\", \"user101\"]"
                                                ]
                                        }
                                }
                        },
                        "rejectedPlans" : [ ]
                },
                "executionStats" : {
                        "executionSuccess" : true,
                        "nReturned" : 1,
                        "executionTimeMillis" : 0,
                        "totalKeysExamined" : 1,
                        "totalDocsExamined" : 1,
                        "executionStages" : {
                                "stage" : "FETCH",
                                "nReturned" : 1,
                                "executionTimeMillisEstimate" : 0,
                                "works" : 2,
                                "advanced" : 1,
                                "needTime" : 0,
                                "needFetch" : 0,
                                "saveState" : 0,
                                "restoreState" : 0,
                                "isEOF" : 1,
                                "invalidates" : 0,
                                "docsExamined" : 1,
                                "alreadyHasObj" : 0,
                                "inputStage" : {
```



```
"stage" : "IXSCAN",
"nReturned" : 1,
"executionTimeMillisEstimate" : 0,
"works" : 2,
"advanced" : 1,
"needTime" : 0,
"needFetch" : 0,
"saveState" : 0,
"restoreState" : 0,
"isEOF" : 1,
"invalidates" : 0,
"keyPattern" : {
        "Name" : 1
},
"indexName" : "Name_1",
"isMultiKey" : false,
"direction" : "forward",
"indexBounds" : {
        "Name" : [
                "[\"user101\", \"user101\"]"
        ]
},
```

Compound Index

When creating an index, you should keep in mind that the index covers most of your queries. If yousometimes query only the Name field and at times you query both the Name and the Age field, creating acompound index on the Name and Age fields will be more beneficial than an index that is created on either ofthe fields because the compound index will cover both queries.

The following command creates a compound index on fields Name and Age of the collection testindx .

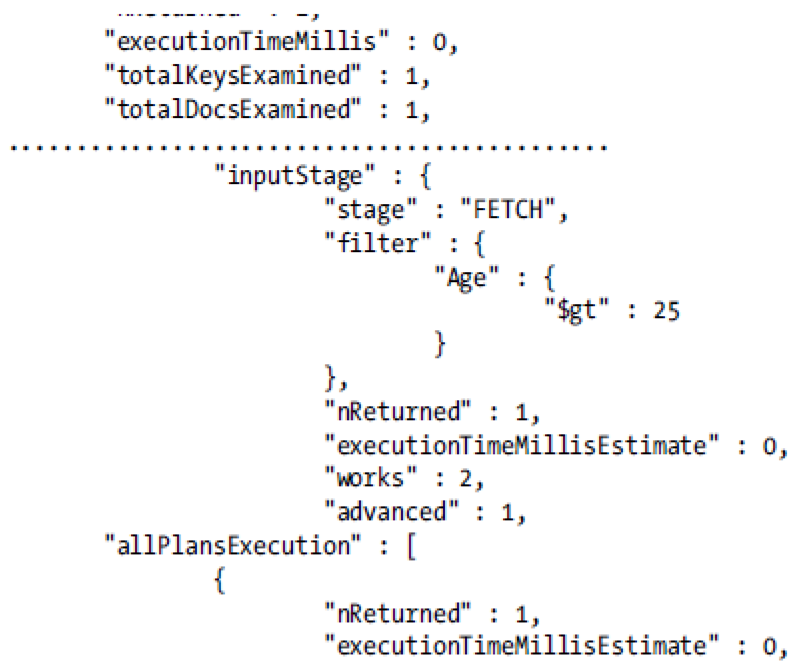
```
> db.testindx.ensureIndex({"Name":1, "Age": 1})
```

Compound indexes help MongoDB execute queries with multiple clauses more efficiently. When creating a compound index, it is also very important to keep in mind that the fields that will be used for exact matches (e.g. `Name: "S1"`) come first, followed by fields that are used in ranges (e.g. `Age: {"$gt":20}`).

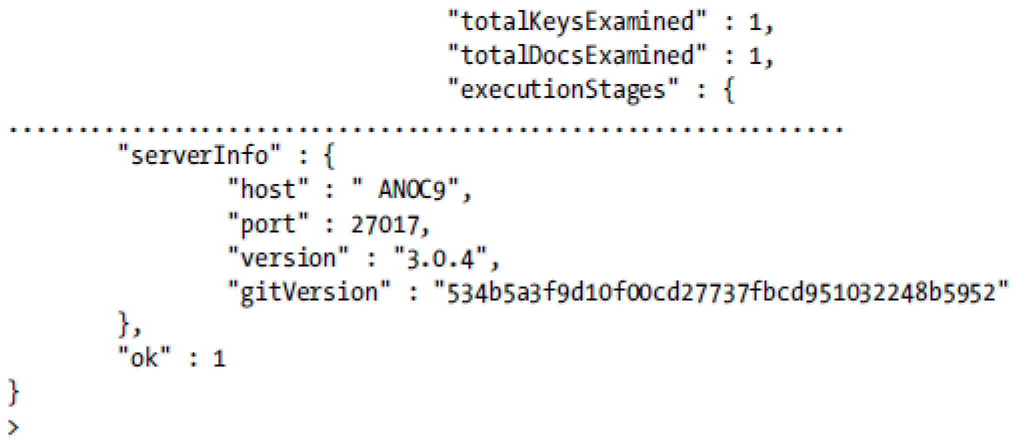
Hence the above index will be beneficial for the following query:

```
>db.testindx.find({"Name": "user5","Age":{"$gt":25}}).explain("allPlansExecution")

{
        "queryPlanner" : {
                "plannerVersion" : 1,
                "namespace" : "mydbproc.testindx",
                "indexFilterSet" : false,
                "parsedQuery" : {
                        "$and" : [
                                {
                                        "Name" : {
                                                "$eq" : "user5"
                                        }
                                },
                {
                    "Age" : {
                        "$gt" : 25
                    }
                }
            ]
        },
        "winningPlan" : {
            "stage" : "KEEP_MUTATIONS",
            "inputStage" : {
                "stage" : "FETCH",
                "filter" : {
                    "Age" : {
                        "$gt" : 25
                    }
                },
                ...........................
                    "indexBounds" : {
                        "Name" : [
                            "[\"user5\", \"user5\"
        },
        "rejectedPlans" : [
            {
                "stage" : "FETCH",
....................................................
                    "indexName" : "Name_1_Age_1",
                    "isMultiKey" : false,
                    "direction" : "forward",
...................................................
    "executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 1,
```



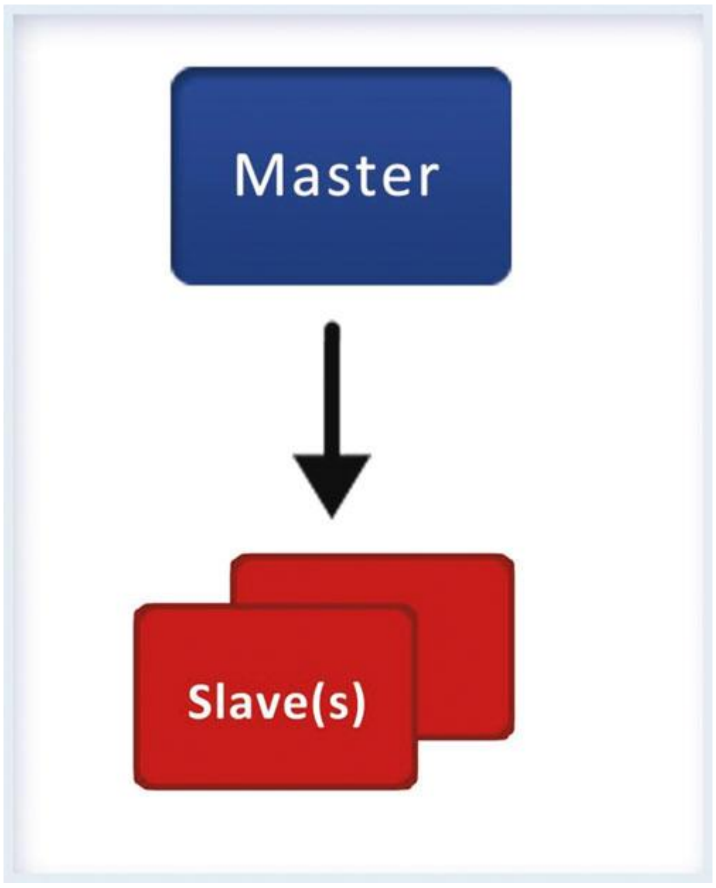
```
        "executionTimeMillis" : 0,
        "totalKeysExamined" : 1,
        "totalDocsExamined" : 1,
.........................................
                "inputStage" : {
                        "stage" : "FETCH",
                        "filter" : {
                                "Age" : {
                                        "$gt" : 25
                                }
                        },
                        "nReturned" : 1,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 2,
                        "advanced" : 1,
        "allPlansExecution" : [
                {
                        "nReturned" : 1,
                        "executionTimeMillisEstimate" : 0,
```



```
                                "totalKeysExamined" : 1,
                                "totalDocsExamined" : 1,
                                "executionStages" : {
...........................................................
        "serverInfo" : {
                "host" : " ANOC9",
                "port" : 27017,
                "version" : "3.0.4",
                "gitVersion" : "534b5a3f9d10f00cd27737fbcd951032248b5952"
        },
        "ok" : 1
}
>
```

(f) Write a short note on Master/Slave replication of MongoDB.

In MongoDB, the traditional master/slave replication is available but it is recommended only for more than50 node replications. The preferred replication approach is replica sets, which we will explain later. In thistype of replication, there is one master and a number of slaves that replicate the data from the master. Theonly advantage with this type of replication is that there's no restriction on the number of slaves within acluster. However, thousands of slaves will overburden the master node, so in practical scenarios it’s better tohave less than dozen slaves. In addition, this type of replication doesn’t automate failover and provides lessredundancy.In a basic master/slave setup , you have two types of mongod instances: one instance is in the mastermode and the remaining are in the slave mode, as shown in diagram. Since the slaves are replicating fromthe master, all slaves need to be aware of the master’s address



.

The slaves replicate the data using this oplog collection. Since the oplog is a capped collection, if the slave's state is far behind the master's state, the slave may become out of sync. In that scenario, thereplication will stop and manual intervention will be needed to re-establish the replication.There are two main reasons behind a slave becoming out of sync:

• The slave shuts down or stops and restarts later. During this time, the oplog may have deleted the log of operations required to be applied on the slave.

• The slave is slow in executing the updates that are available from the master.

Answer : 3

(a)Discuss the fields used for Sharding.

When the data no longer fits on one node, sharding can be used to ensure that the data is distributed evenlyacross the cluster and the operations are not affected due to resource constraints.

• Select a good shard key.
• You must use three config servers in production deployments to provide redundancy.
• Shard collections before they reach 256GB.

The fields used for sharding.

1. **Time field** : In choosing this option, although the data will be distributed evenly among the shards, neither the inserts nor the reads will be balanced.
As in the case of performance data, the time field is in an upward direction, so all the inserts will end up going to a single shard and the write throughput will end up being same as in a standalone instance.
Most reads will also end up on the same shard, assuming you are interested in viewing the most recent data frequently.

2. Hashes : You can also consider using a random value to cater to the above situations; a hash of the id field can be considered the shard key.
Although this will cater to the write situation of the above (that is, the writes will be distributed), it will affect querying. In this case, the queries must be broadcasted to all the shards and will not be routable.

3. Use the key, which is evenly distributed, such as Host .
This has following advantages: if the query selects the host field, the reads will be selective and local to a single shard, and the writes will be balanced.
However, the biggest potential drawback is that all data collected for a single host must go to the same chunk since all the documents in it have the same shard key.
This will not be a problem if the data is getting collected across all the hosts, but if the monitoring collects a disproportionate amount of data for one host, you can end up with a large chunk that will be completely unsplittable, causing an unbalanced load on one shard.

4. Combining the best of options 2 and 3, you can have a compound shard key, such as *{host:1, ssk: 1}* where *host* is the host field of the document and *ssk*is _id field's hash value.
In this case, the data is distributed largely by the host field making queries, accessing the host field local to either one shard or group of shards. At the same time, using ssk ensures that data is distributed evenly across the cluster.
In most of the cases, such keys not only ensure ideal distribution of writes across the cluster but also ensure that queries access only the number of shards that are specific to them.

(b) List and explain the limitations of Indexes.

The limitations are as following :

Index size : Indexed items cannot be greater than 1024 bytes.
• **Number of indexes per collection** : At the most 64 indexes are allowed per collection.

• **Index name length** : By default the index name is made up of the field names and the index directions. The index name including the namespace (which is the database and the collection name) cannot be greater than 128 bytes.
If the default index name is becoming too long, you can explicitly specify an index name to the ensureIndex() helper.
• **Unique indexes in shardedcollections** : Only when the full shard key is contained as a prefix of the unique index is it supported across shards; otherwise, the unique index is not supported across shards. In this case, the uniqueness is enforced across the full key and not a single field.
• **Number of indexed fields in a compound index** : This can't be more than 31 fields.

(c) Explain the 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.

(d) Write a short note on Deployment.

While deciding on the deployment strategy , keep the following tips in mind so that the hardware sizing isdone appropriately. These tips will also help you decide whether to use sharding and replication .

• **Data set size** : The most important thing is to determine the current and anticipated data set size. This not only lets you choose resources for individual physical nodes, but it also helps when planning your sharding plans (if any).

• **Data importance** : The second most important thing is to determine data importance, to determine how important the data is and how tolerant you can be to any data loss or data lagging (especially in case of replication) .

• **Memory sizing** : The next step is to identify memory needs and accordingly take care of the RAM.

Like other data-oriented applications, MongoDB also works best when the entire data set can reside in memory, thereby avoiding any kind of disk I/O.
Page faults indicate that you may exceed the available deployment's memory and should consider increasing it. Page fault is a metric that can be measured using monitoring tools like MongoDB Cloud Manager.If possible, you should always select a platform that has memory greater thanyour working set size.If the size exceeds the single node's memory, you should consider using shardingso that the amount of available memory can be increased. This maximizes theoverall deployment's performance.

(e) Define Monitoring. Explain the factors to be considered while using Monitoring Services.

MongoDB system should be proactively monitored to detect unusual behaviors so that necessary actionscan be taken to resolve issues. Several tools are available for monitoring the MongoDB deployment.A free hosted monitoring service named MongoDB Cloud Manager is provided by MongoDBdevelopers. MongoDB Cloud Manager offers a dashboard view of the entire cluster metrics. Alternatively,you can use nagios, SNMP, or munin to build your own tool.
MongoDB also provides several tools such as mongostat and mongotop to gain insights into the performance. When using monitoring services, the following should be watched closely:

• **Op counters** : Includes inserts, delete, reads, updates and cursor usage.

• **Resident memory** : An eye should always be kept on the allocated memory. This counter value should always be lower than the physical memory. If you run out of memory, you will experience slowness in the performance due to page faults and index misses.

• **Working set size** : The active working set should fit into memory for a good performance, so a close eye needs to be kept on the working set. You can either optimize the queries so that the working set fits inside the memory or increase the memory when the working set is expected to increase.

Queues : Prior to the release of MongoDB 3.0, a reader-writer lock was used for simultaneous reads and exclusive access was used for writes. In such scenario, you might end up with queues behind a single writer, which may contain read/write queries. Queue metrics need to be monitored along with the lock percentage. If the queues and the lock percentage are trending upwards, that implies that you have contention within the database. Changing the operation to batch mode or changing the data model can have a significant, positive impact on the concurrency. Starting from Version 3.0, collection level locking (in the MMAPv1 storage engine) and document level locking (in the WiredTiger storage engine) have been introduced. This leads to an improvement in concurrency wherein no write lock with exclusive access will be required at the database level. So starting from this version you just need to measure the Queue metric.

• Whenever there’s a hiccup in the application, the CRUD behavior, indexing patterns, and indexes can help you better understand the application’s flow.

• It’s recommended to run the entire performance test against a full-size database, such as the production database copy, because performance characteristic are often highlighted when dealing with the actual data. This also lets you to avoid unpleasant surprises that might crop up when dealing with the actual performance database

(f) What is Data Storage Engine? Differentiate between MMAP and Wired storage engines.

The component that has been an integral part of MongoDB database is a Storage Engine, which defines how data is stored in the disk. There may be different storage engines for a particular database and each one is productive in different aspects.

Memory Mapping(MMAP) has been the only storage engine but in version 3.0 and more, another storage engine, WiredTiger has been introduced which has its own benefits.

The differentiatiom between these two are on basis of certain characteristics..

Concurrency

MMAP uses Collection-level locking. In MongoDB 3.0 and more,if a client acquires a lock on a document to modify its content,then no other client can access the collection which holds the document currently. Whereas in earlier versions a single write operation acquires the lock to the database.

WiredTiger uses Document-Level Locking. Multiple clients can access the same collection, since the lock is acquired for that particular document.

In Case of parallel inserts and updates,WiredTiger would be extremely advantageous.

Consistency

Journaling is one feature that helps your database from recovery after a failure. MongoDB uses write ahead logging to on-disk journal files.

MMAP uses this feature to recover from failure.

In WiredTiger,a consistent view of the data is given by means of checkpoints. so that in case of failure it can rollback to the previous checkpoint. But journaling is required if the changes made after checkpoint is also needed.

It’s left to the user’s choice to enable or disable journaling.

Compression

Data compression is needed in places where the data growth is extremely fast which can be used to reduce the disk space consumed.

Data compression facilities is not present in MMAP storage engine.

In WiredTiger Storage Engine. Data compression is achieved using two methods

1.Snappy compression

2.Zlib

In Snappy method,compression rate is slow whereas in Zlib its high, And again, it's the user's preference to have it or not.

MemoryConstraints

Wired Tiger can make use of multithreading and hence multi core CPU's can be made use of it.whereas in MMAP, increasing the size of the RAM decreases the number of page faults which in turn increases the performance.

Answer 4

(a) Define In-Memory Database. What are the techniques used in In-Memory Database to ensure that data is not lost?

An **in-memory database** (**IMDB**, also **main memory database system** or **MMDB** or **memory resident database**) is a database management system that primarily relies on main memory for computer data storage. It is contrasted with database management systems that employ a disk storage mechanism. In-memory databases are faster than disk-optimized databases because disk access is slower than memory access, the internal optimization algorithms are simpler and execute fewer CPU instructions. Accessing data in memory eliminates seek time when querying the data, which provides faster and more predictable performance than disk.

In-memory databases generally use some combination of techniques to ensure they don't lose data. These include:

• Replicating data to other members of a cluster.

• Writing complete database images (called *snapshots* or *checkpoints*) to disk files.

• Writing out transaction/operation records to an append-only disk file

(b) Explain how does Redis uses disk files for persistence.

Redis uses disk files for persistence:
The Snapshot files store copies of the entire Redis system at a point in time.Snapshots can be created on demand or can be configured to occur at scheduledintervals or after a threshold of writes has been reached. A snapshot also occurs
when the server is shut down

The **Append Only File** (AOF) keeps a journal of changes that can be used to "roll forward" the database from a snapshot in the event of a failure. Configurationoptions allow the user to configure writes to the AOF after every operation, at
one-second intervals, or based on operating-system-determined flush intervals.

(c) What is Berkeley Analytics Data Stack? Explain its components.

Big Data stack by providing a flexible, scalable, and
economic framework for processing massive amounts of structured, unstructured, and semi-structured data.

BDAS consists of a few core components:

• **Spark** is an in-memory, distributed, fault-tolerant processing framework.Implemented in the Java virtual-machine-compatible programming language Scala,it provides higher-level abstractions than MapReduce and thus improves developer productivity. As an in-memory solution, Spark excels at tasks that cause bottleneckson disk IO in MapReduce. In particular, tasks that iterate repeatedly over adataset—typical of many machine-learning workloads—show significant improvements.

• **Mesos**is a cluster management layer somewhat analogous to Hadoop's YARN.However, Mesos is specifically intended to allow multiple frameworks, includingBDAS and Hadoop, to share a cluster.

• **Tachyon** is a fault-tolerant, Hadoop-compatible, memory-centric distributed filesystem. The file system allows for disk storage of large datasets, but promotes aggressivecaching to provide memory-level response times for frequently accessed data.

(d) What is an Event ?State the different types of Events in jQuery.

An event represents the precise moment when something happens.

click: Clicking an elements such as a button

• hover: Interacting with an element via the mouse; in pure JavaScript, known as mouseenter or mouseleave

• submit: Submitting a form

• trigger: Making an event happen

• off: Removing an event

The most popular is the click event.

A lot of older tutorials you'll see on the Web will tell you to use methods such as click() to bind code toa click event, like so:

```
$("div").click(function() {
alert("clicked");
});
```

(e) Write a short note on jQuery CSS method.

jQuery’s css() method is very powerful. There are actually three primary ways that you’ll work with it.The first is when determining the value of an element’s property. Simply pass it one parameter—theproperty whose value you want to know:

```
$("div").css("width");
$("div").css("margin-right");
$("div").css("color");
```

It’s important to note that if you have a set of more than one element and you call css(), you’ll get theresult as if css() was called on just the first element. Another important note is that you can’t use shorthand.For example, this won’t work:

```
$("div").css("margin");
```

You can also use CSS to set values. To set just one value, pass in a property and a value as separate parameters.

```
$("div").css("color", "red");
$("div").css("border", "1px solid red");
```

What’s more useful is that the css() method also accepts an object of key-value pairs that map CSS properties to the values you want to set. For example:

```
$("div").css({
"background" : "red",
"margin-left": "200px",
"color": "black"
});
```

(f) State the features of jQuery.

jQuery is a fast and concise JavaScript Library created by John Resig in 2006 with a nice motto: **Write less, do more**. jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code. Here is the list of important core features supported by jQuery −

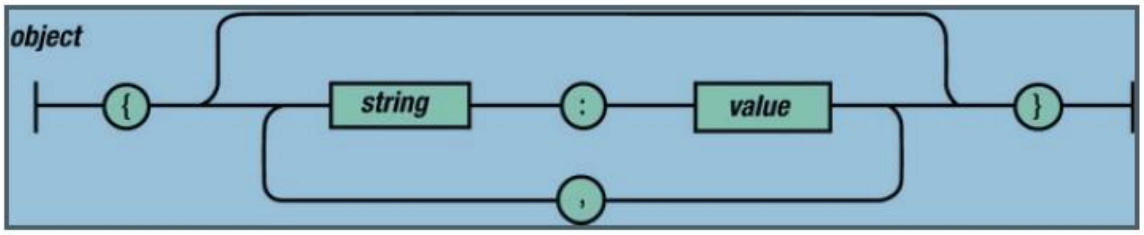
- **DOM manipulation** − The jQuery made it easy to select DOM elements, negotiate them and modifying their content by using cross-browser open source selector engine called **Sizzle**.
- **Event handling** − The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- **AJAX Support** − The jQuery helps you a lot to develop a responsive and featurerich site using AJAX technology.
- **Animations** − The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- **Lightweight** − The jQuery is very lightweight library - about 19KB in size (Minified and gzipped).
- **Cross Browser Support** − The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- **Latest Technology** − The jQuery supports CSS3 selectors and basic XPath syntax.

Answer: 5

(a) Explain the JSON Grammar

JSON, in a nutshell, is a textual representation defined by a small set of governing rules in which data is structured. The JSON specification states that data can be structured in either of the two following compositions:
1. A collection of name/value pairs
2. An ordered list of values

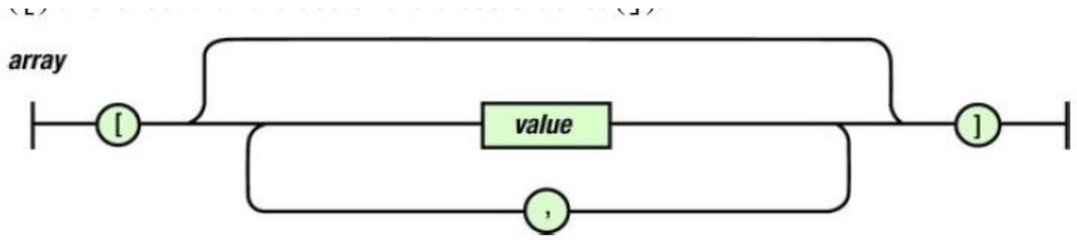
JSON stem from the ECMAScript standardization, the implementations of the two structures are represented in the forms of the object and array.



a collection begins with the use of the opening brace ({), andends with the use of the closing brace (}). The content of the collection can be composedof any of the following possible three designated paths:

1. The top path illustrates that the collection can remain devoid of anystring/value pairs.
2. The middle path illustrates that our collection can be that of a singlestring/value pair.
3. The bottom path illustrates that after a single string/value pair is supplied, the collection needn't end but, rather, allow for any number of string/value pairs, before reaching the end. Each string/value pairpossessed by the collection must be delimited or separated from oneanother by way of a comma (,).

Now we can see the grammatical representation for that of an ordered list ofvalues. Here we can witness that an ordered list begins with the use of the open bracket([) and ends with the use of the close bracket (]).



The values that can be held within each index are outlined by the following three
"railroad" paths:The top path illustrates that our list can remain devoid of any value(s).
The middle path illustrates that our ordered list can possess a singularvalue. The bottom path illustrates that the length of our list can possess anynumber of values, which must be delimited, that is, separated, with theuse of a comma (,).

(b) Difference between XML and JSON

JSON	XML
It is JavaScript Object Notation	It is Extensible markup language
It is based on JavaScript language.	It is derived from SGML.
It is a way of representing objects.	It is a markup language and uses tag structure to represent data items.
It does not provides any support for namespaces.	It supports namespaces.
It supports array.	It doesn't supports array.
Its files are very easy to read as compared to XML.	Its documents are comparatively difficult to read and interpret.
It doesn't use end tag.	It has start and end tags.
It is less secured.	It is more secured than JSON.
It doesn't supports comments.	It supports comments.
It supports only UTF-8 encoding.	It supports various encoding.

(C) Explain Request Headers.

These headers can besupplied with the request to provide the server with preferential information that will assistin the request. Additionally, they outline the configurations of the client making therequest. Such headers may reveal information about the user-agent making the request or the preferred data type that the response should provide. By utilizing the headers within this category, we can potentially influence the response from the server. For this reason,therequest headers are the most commonly configured headers.One very useful header is the Accept header. It can be used to inform the server as towhat MIME type or data type the client can properly handle. This can often be set to aparticular MIME type, such as application/json, or text/plain. It can even be set to */*,which informs the server that the client can accept all MIME types. The response providedby the server is expected to reflect one of the MIME types the client can handle. Thefollowing are request headers:

Accept
Accept-Charset
Accept-Encoding
Accept-Language
Authorization
Expect
From
Host
If-Match
If-Modified-Since
If-None-Match

(d) Write a short note on JSON Parsing.

Parsing is the process of analyzing a string of symbols, either in natural language or in computer languages, according to the rules of a formal grammar. As the grammar of JSON is a subset of JavaScript, the analysis of its tokens by the parser occurs indifferently from how the Engine parses source code. Because of this, the data produced from the analysis of the JSON grammar will be that of objects, arrays, strings, and numbers

JSON.parse

JSON.parse converts serialized JSON into usable JavaScript values.

Syntax of the JSON.parseMethod

JSON.parse(text [, reviver]);

JSON.parse can accept two parameters, text andreviver. The name of the parameter text is indicative of the value it expects toreceive. The parameter reviver is used similarly to the replacer parameter ofstringify, in that it offers the ability for custom logic to be supplied for necessary parsing that would otherwise not be possible by default. As indicated in the method'ssignature, only the provision of text is required.

(e) Explain the stringify object for JSON Object.

stringify is used for serializing JavaScript values into that of avalid JSON. The method itself accepts three parameters, value, replacer, and space. the JSON Object is aglobal object that does not offer the ability to create any instances of the JSON Object. one must simply access the stringify method via the global JSON Object.

Syntax of the JSON stringify Method

JSON.stringify(value[, replacer [, space]]);

The brackets surrounding the two parameters, replacer and space, is just a
way to illustrate in a method definition what is optional. However, while an argument
supplied to the method may be optional, you must follow the proper parameter order, as
outlined by the method. In other words, to specify an argument for space but not
replacer, you must supply null as the second argument to the stringify method.

(f) Discuss the JSON data values.

JSON (JavaScript Object Notation) is most widely used data format for data interchange on the web. **JSON** is a lightweight text based, data-interchange format and it completely language independent. It is based on a subset of the JavaScript programming language and it is easy to understandandgenerate.

JSON supports mainly 6 data types:

1. string
2. number
3. boolean
4. null
5. object

6. array