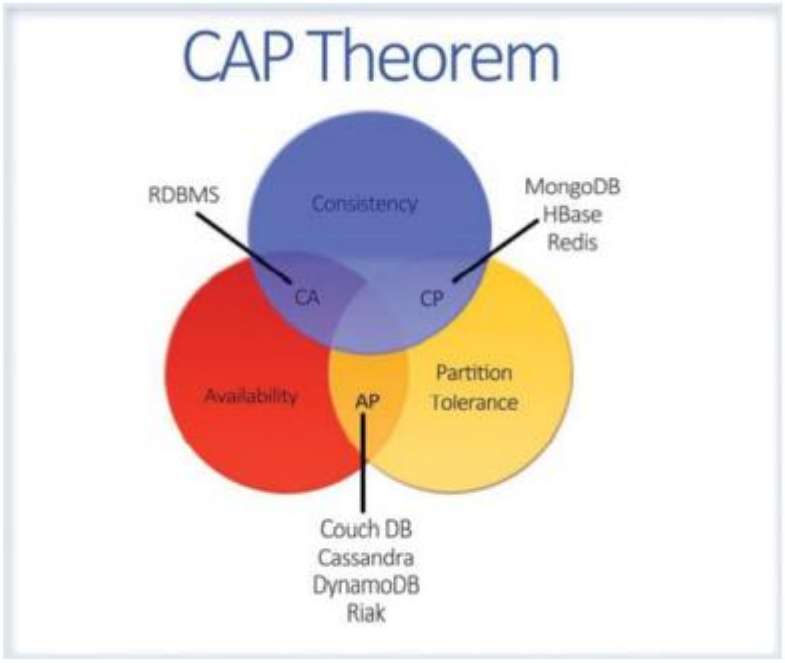
Next Generation Technologies Solution set

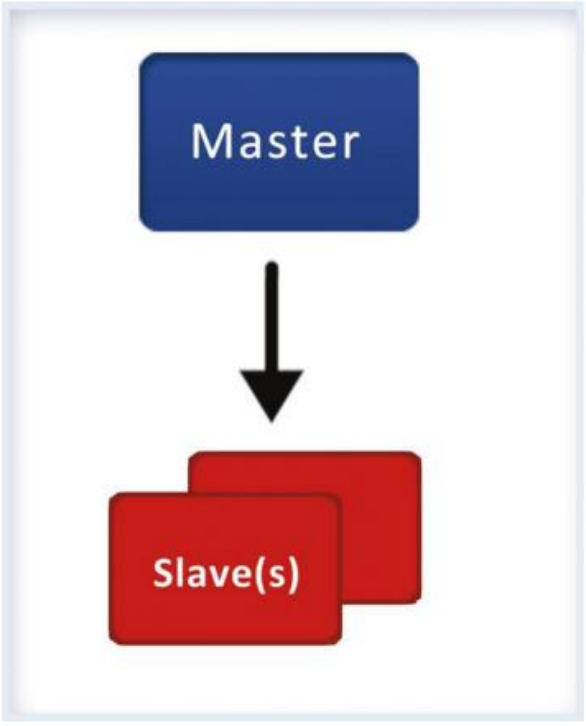
1.	**Attempt *<u>any three</u>* of the following:**	**15**
a.	Explain the three aspects of data i) Data at rest ii) Data in motion iii) Data in many forms. **Explanation: 5 Marks** **Data at rest:** Volume in big data means the size of the data or data at rest. As businesses are becoming more transaction-oriented, 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. **Data in Motion:** 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. **Data in many forms:** Also known as variety of data. 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. New sources and structures of data are being created at a rapid pace. So the onus is on technology to find a solution to analyze and visualize the huge variety of data that is out there.	
b.	List the Big data sources and also explain the challenges of big data. **List the Big data sources: 2 Marks** • Social media – Facebook, Twitter etc. • Banks(online transactions, ATM) • Instruments (Radio Frequency (RF) id card, CCTV cameras), sensors • E-Commerce Websites • Stock market • Smart phones • Cloud computing • Enterprises, which are collecting data with more granularities. • Increase in multimedia usage across industries such as health care, product companies, etc. **Challenges of Big data: 3 Marks** **Policies and Procedures**: As more and more data is gathered, digitized, and moved around the globe, the policy and compliance issues become increasingly important to protect private data. Data confidentiality, data security, intellectual property and protection of data become extremely important for organizations. Compliance with various statutory and legal requirements poses a challenge in data handling. Many big data projects leverage the scalability features of public cloud computing providers. This poses a challenge for compliance. Policy questions on who owns the data, what is defined as fair use of data, and who is responsible for accuracy and confidentiality of data also need to be answered **Access to Data**: Accessing data for consumption is a challenge for big data projects. Some of the data may be available to third parties, and gaining access can be a legal, contractual challenge. Data about a product or service is available on Facebook, Twitter feeds, reviews, and blogs, so how does the product owner access this data from various sources owned by various providers? Likewise, contractual clauses and economic incentives for accessing big data need to be tied in to enable the availability of data by the consumer. **Technology and Techniques:** New tools and technologies built specifically to address the needs of big data must be leveraged, rather than trying to address the aforementioned issues through legacy systems. The inadequacy of legacy systems to deal with big data on one hand and the lack of experienced resources in newer technologies is a challenge that any big data project has to manage.	

c.	Explain Brewer's theorem along with a neat diagram. **Brewer's Theorem: 3 Marks** CAP theorem is also known as Brewer's Theorem This 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. **Diagram: 2Marks**  	
d.	How consistency can be implemented at both read and write operation levels explain. **Write operation: 3 Marks** N=W implies that the write operation will update all data copies before returning the control to the client and marking the write operation as successful. This is similar to how the traditional RDBMS databases work when implementing synchronous replication. This setting will slow down the write performance. If write performance is a concern, which means you want the writes to be happening fast, you can set W=1, R=N. This implies that the write will just update any one copy and mark the write as successful, but whenever the user issues a read request, it will read all the copies to return the result. If either of the copies is not updated, it will ensure the same is updated, and then only the read will be successful. This implementation will slow down the read performance. Hence most NoSQL implementations use N>W>1. This implies that greater than one node needs to be updated successfully; however, not all nodes need to be updated at the same time. **Read operation: 2 Marks** If R=1, the read operation will read any data copy, which can be out dated. If R>1, more than one copy is read, and it will read most recent value. This can slow down the read operation. Using N<W+R always ensures that a read operation retrieves the latest value This is because the number of written copies and read copies are always greater than the	

	actual number of copies, ensuring that at least one read copy has the latest version. This is quorum assembly.	
e.	List the categories of NoSQL databases. Also explain the ways in which MongoDB is different from SQL. **List the categories of NoSQL databases: 2 Marks** Document based – MongoDB XML database – Mark Logic Graph Database – GraphDB Key value store – Cassandra, Redis **Explain the ways in which MongoDB is different from SQL: 3 Marks** MongoDB uses documents for storing its data, which offer a flexible schema (documents in same collection can have different fields). This enables the users to store nested or multi-value fields such as arrays, hashes, etc. In contrast, RDBMS systems offer a fixed schema where a column's value should have a similar data type. Also, it's not possible to store arrays or nested values in a cell. MongoDB doesn't provide support for JOIN operations, like in SQL. However, it enables the user to store all relevant data together in a single document, avoiding at the periphery the usage of JOINs. MongoDB doesn't provide support for transactions in the same way as SQL. However, it guarantees atomicity at the document level. Also, it uses an isolation operator to isolate write operations that affect multiple documents, but it does not provide "all-or-nothing" atomicity for multi-document write operations.	
f.	What is NoSQL? Explain the advantages of NoSQL databases. **NoSQL: 1 Mark** NoSQL stands for Not only SQL **Advantages: 4 Marks** High scalability: This scaling up approach fails when the transaction rates and fast response requirements increase. In contrast to this, the new generation of NoSQL databases is designed to scale out (i.e. to expand horizontally using low-end commodity servers). Manageability and administration: NoSQL databases are designed to mostly work with automated repairs, distributed data, and simpler data models, leading to low manageability and administration. Low cost: NoSQL databases are typically designed to work with a cluster of cheap commodity servers, enabling the users to store and process more data at a low cost. Flexible data models: NoSQL databases have a very flexible data model, enabling them to work with any type of data; they don't comply with the rigid RDBMS data models. As a result, any application changes that involve updating the database schema can be easily implemented.	
2.	**Attempt *any three* of the following:**	**15**
a.	Explain about _id, capped collection and BSON **The Identifier (_id): 1 Mark** A key that uniquely identifies each document within a collection is referred to as _id in MongoDB. **Capped collection : 2 Marks** MongoDB has a concept of capping the collection. This means it stores the documents in the collection in the inserted order. As the collection reaches its limit, the documents will be removed from the collection in FIFO (first in, first out) order. This means that the least recently inserted documents will be removed first. This is good for use cases where the order of insertion is required to be maintained automatically, and deletion of records after a fixed size is required. One such use cases is log files that get automatically truncated after a certain size. **BSON: 2 Marks** BSON stands for Binary Javascript Object Notation	

	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.	What is a polymorphic schema? Explain the various reasons for using a polymorphic schema. **What is a polymorphic schema: 1 Mark** A polymorphic schema is a schema where a collection has documents of different types or schemas. **Various reasons for using a polymorphic schema: 4 Marks** Object oriented programming Object-oriented programming enables you to have classes share data and behaviors using inheritance. It also lets you define functions in the parent class that can be overridden in the child class and thus will function differently in a different context. In other words, you can use the same function name to manipulate the child as well as the parent class, although under the hood the implementations might be different. This feature is referred to as polymorphism. The requirement in this case is the ability to have a schema wherein all of the related sets of objects or objects within a hierarchy can fit in together and can also be retrieved identically. // "Document collections" - "HTMLPage" document { _id: 1, title: "Hello", type: "HTMLpage", text: "Hi..Welcome to my world " } // Document collection also has a "Picture" document { _id: 3, title: "Family Photo", type: "JPEG", sizeInMB: 10,..... } This schema not only enables you to store related data with different structures together in a same collection, it also simplifies the querying. The same collection can be used to perform queries on common fields such as fetching all content uploaded on a particular date and time as well as queries on specific fields such as finding images with a size greater than X MB. Thus object-oriented programming is one of the use cases where having a polymorphic schema makes sense. Schema Evolution The design should be done in a way as to have minimal or no impact on the application, meaning no or minimal downtime, no or very minimal code changes, etc. schema evolution happens by executing a migration script that upgrades the database schema from the old version to the new one. If the database is not in production, the script can be simple drop and recreation of the database. However, if the database is in a production environment and contains live data, the migration script will be complex because the data will need to be preserved. The script should take this into consideration. Although MongoDB offers an Update option that can be used to update all the documents' structure within a collection if there's a new addition of a field, imagine the impact of doing this if you have thousands of documents in the collection. It would be very slow and would have a negative impact on the underlying application's performance. One of the ways of doing this is to include the new structure to the new documents being added to the collection and then gradually migrating the collection in the background while the application is still running. This is one of the many use cases where having a polymorphic schema will be advantageous.	
c.	How can you create a collection explicitly? Explain about selector and projector with example. **Explicitly creating a collection: 1 Mark** db.createCollection("users") **Selector: 2 Marks** A selector is like a where condition in SQL or a filter that is used to filter out the results.	

	Example: The following command will return all the female users: db.users.find({"Gender":"F"}) **Projector: 2 Marks** A projector is like the select condition or the selection list that is used to display the data fields. Selecting only the necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them. Example: db.users.find({"Gender":"F"}, {"Name":1,"Age":1}	
d.	What is the use of findOne() method? Briefly explain about explain() function **What is the use of findOne() method: 1 Mark** Similar to find() is the findOne() command. The findOne() method can take the same parameters as find() , but rather then returning a cursor, it returns a single document. **Explain about explain() function: 4 Marks** The explain() function can be used to see what steps the MongoDB database is running while executing a query. It takes an optional parameter called verbose, which determines what the explain output should look like. The following are the verbosity modes: allPlansExecution executionStats queryPlanner . The default verbosity mode is queryPlanner, which means if nothing is specified, it defaults to queryPlanner.	
e.	Explain about Master Slave Replication with a neat diagram **Explanation: 4 Marks** In this type of replication, there is one master and a number of slaves that replicate the data from the master. The only advantage with this type of replication is that there's no restriction on the number of slaves within a cluster. However, thousands of slaves will overburden the master node, so in practical scenarios it's better to have less than dozen slaves. In addition, this type of replication doesn't automate failover and provides less redundancy. In a basic master/slave setup , you have two types of mongod instances: one instance is in the master mode and the remaining are in the slave mode. Since the slaves are replicating from the master, all slaves need to be aware of the master's address. The master node maintains a capped collection (oplog) that stores an ordered history of logical writes to the database. 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, the replication will stop and manual intervention will be needed to re-establish the replication **Diagram: 1 Mark**	



f. Explain the components of a sharded cluster.

Explanation : 5 Marks

The following are the components of a sharded cluster:

- Shards
- mongos
- Config servers

The shard is the component where the actual data is stored. For the sharded cluster, it holds a subset of data and can either be a mongod or a replica set. All shard's data combined together forms the complete dataset for the sharded cluster

Sharding is enabled per collection basis, so there might be collections that are not sharded. In every sharded cluster there's a primary shard where all the unsharded collections are placed in addition to the sharded collection data.

The mongos act as the routers. They are responsible for routing the read and write request from the application to the shards. An application interacting with a mongo database need not worry about how the data is stored internally on the shards. For them, it's transparent because it's only the mongos they interact with. The mongos, in turn, route the reads and writes to the shards.

Config servers are special mongods that hold the sharded cluster's metadata. This metadata depicts the sharded system state and organization. The config server stores data for a single sharded cluster. The config servers should be available for the proper functioning of the cluster. One config server can lead to a cluster's single point of failure. For production deployment it's recommended to have at least three config servers, so that the cluster keeps functioning even if one config server is not accessible. A config server stores the data in the config database, which enables routing of the client requests to the respective data. This database should not be updated. MongoDB writes data to the config server only when the data distribution has changed for balancing the cluster.

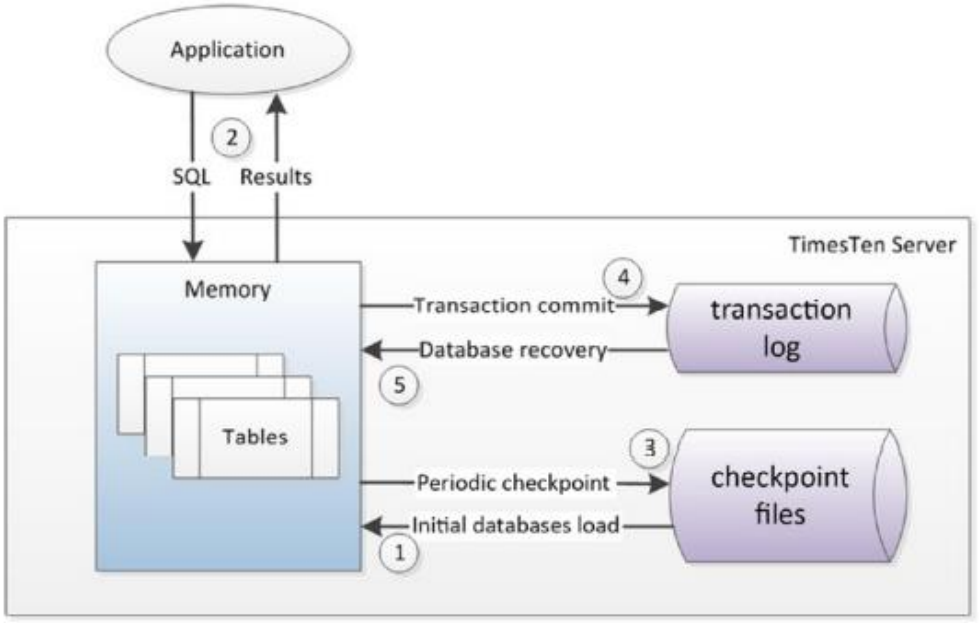
3.	**Attempt *any three* of the following:**	**15**
a.	Write a short note on WiredTiger storage engine. **Explanation : 5 Marks** WIredTiger stores d ata in compressed fomat on the disk. Compression reduces the data size by up to 70% (disk only) and index size by up to 50% (disk and memory both) depending on	

	the compression algorithm used. In addition to reduced storage space, compression enables much higher I/O scalability as fewer bits are read from disk. It provides significant benefits in the areas of greater hardware utilization, lower storage costs, and more predictable performance. The following compression algorithms are available to choose from: • Snappy is the default, which is used for documents and journals. It provides a good compression ratio with little CPU overhead. Depending on data types, the compression ratio is somewhere around 70%. • zlib provides extremely good compression but at the expense of extra CPU overhead. • Prefix compression is the default used for indexes, reducing the in-memory footprint of index storage by around 50% (workload dependent) and freeing up more of the working set for frequently accessed documents. Administrators can modify the default compression settings for all collections and indexes. Compression is also configurable on a per-collection and per-index basis during collection and index creation. WiredTiger also provides granular document-level concurrency. Writes are no longer blocked by other writes unless they are accessing the same document. Thus it supports concurrent access by readers and writers to the documents in a collection. Clients can read documents while write operations are in progress, and multiple threads can modify different documents in a collection at the same time. Thus it excels for write-intensive workloads	
b.	Explain the concept of GridFS – The MongoDB File System. **Explanation – 5 Marks** GridFS is MongoDB's specification for handling large files that exceed BSON's document size limit. By design, a MongoDB document (i.e. a BSON object) cannot be larger than 16MB. This is to keep performance at an optimum level, and the size is well suited for our needs. For example, 4MB of space might be sufficient for storing a sound clip or a profile picture. However, if the requirement is to store high quality audio or movie clips, or even files that are more than several hundred megabytes in size, MongoDB has covered by using GridFS. GridFS uses two collections for storing the file. One collection maintains the metadata of the file and the other collection stores the file's data by breaking it into small pieces called chunks. This means the file is divided into smaller chunks and each chunk is stored as a separate document. By default the chunk size is limited to 255KB. This approach not only makes the storing of data scalable and easy but also makes the range queries easier to use when a specific part of files are retrieved. Whenver a file is queried in GridFS, the chunks are reassembled as required by the client. This also provides the user with the capability to access arbitrary sections of the files. For example, the user can directly move to the middle of a video file. The GridFS specification is useful in cases where the file size exceeds the default 16MB limitation of MongoDB BSON document. It's also used for storing files that you need to access without loading the entire file in memory.	
c.	What is sharding? List and explain the sharding limitations. **What is sharding : 1 Mark** Sharding is the mechanism of splitting data across shards. **Limitations: 3 Marks** 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.	
d.	Explain MongoDB limitations from security perspective. Also give an overview about Read and Write limitations. **MongoDB limitations from security perspective: 2 Marks** Security is an important matter when it comes to databases. 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: 3 Marks** 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'}) . Type- Sensitive Fields Since there's no enforced schema for documents in MongoDB, it can't know you are making a mistake. You must make sure that the correct type is used for the data. 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. 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.	
e.	Write a short note on Deployment in MongoDB. **Explanation: 5 Marks** While deciding on the deployment strategy, keep the following tips in mind so that the hardware sizing is done 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 than your working set size. If the size exceeds the single node's memory, you should consider using sharding so that the amount of available memory can be increased. This maximizes the overall deployment's performance. • Disk Type: If speed is not a primary concern or if the data set is larger than what any in-memory strategy can support, it's very important to select a proper disk type. IOPS (input/output operations per second) is the key for selecting a disk type; the higher the IOPS, the better the MongoDB performance. If possible, local disks should be used because network storage can cause poor performance and high latency. • CPU: If you anticipate using map reducing, then the clock speed and the available processors become important considerations. Clock speed can also have a major impact on the overall performance when you are running a mongod with the majority of data in memory. In circumstances where you want to maximize the operations per second, you must consider including a CPU with a high clock/bus speed in your deployment strategy. • Replication is used if high availability is one of the requirements. In any MongoDB deployment it should be a standard to set up a replica set with at least three nodes. A 2x1 deployment is the most common configuration for replication with three nodes, where there are two nodes in one data center and a backup node in a secondary data center.	
f.	What are the tips need to be considered when coding with the MongoDB database. **Any 5 points: 5 Marks** • The first point is to think of the data model to be used for the given application requirement and to decide on embedding or referencing or a mix of both. • Avoid application patterns that lead to unbounded growth of document size. In MongoDB, the maximum size for a BSON document is 16MB. Application patterns that make the documents grow in an unbounded way should be avoided. For instance, an application should not update documents a way that leads them to grow significantly. When the document size exceeds the allocated size, MongoDB will relocate the document. This process is not only time consuming, but is also resource intensive and can unnecessarily slow down other database operations. In addition, it can lead to inefficient use of storage. • You can also design documents for the future. Although MongoDB provides the option of appending new fields within the documents as and when required, it has a drawback. When new fields are introduced, there might be a scenario where the document might not fit in the current space available, leading to MongoDB finding a new space for the document and moving it there, which might take time. So it is always efficient to create all the fields at the start if you are aware of the structure, irrespective of whether you have data available at that time or not. As highlighted above, the space will be allotted to the document and whenever value is there only needs to be updated. In doing so, MongoDB will not have to look for space; it merely updates the values entered, which is much faster. • You can also create documents with the anticipated size wherever applicable. This point is also to ensure that enough space is allotted to the document and any further growth doesn't lead to hopping here and there for space. This can be achieved by using a garbage field, which contains a string of the anticipated size while initially inserting the document and then immediately unsetting that field: > mydbcol.insert({"_id" : ObjectID(..),....., "tempField" : stringOfAnticipatedSize}) > mydbcol.update({"_id" : ...}, {"$unset" : {"tempField" : 1}}) Subdocuments should always be used in a scenario when you know and will always know the names of the fields that you are accessing. Otherwise, use arrays. • If you want to query for information that must be computed and is not explicitly present in the document, the best choice is to make the information explicit in the document. As MongoDB is designed to just store and retrieve the data, it does no computation. Any trivial computation is pushed to the client, leading to performance issues. • Also, avoid $Where as much as possible because it's an extremely time- and resourceintensive operation.	

	• Use the correct data types while designing documents. For example, a number should be stored as a number data type only and not as a string data type. Using strings takes more space to store data and has an impact on the operations that can be performed on the data. • Another thing to note is that strings in MongoDB are case sensitive. Hence a search for "practicalMongoDB" will not find "Practicalmongodb". Hence when doing a string search, you can do one of the following: • Store data in a normalized case format. • Use a regular expression with /I while searching. • Use $toUpper or $toLower in the aggregation framework. • Using your own unique key as a _id will save a bit of space and will be useful if you are planning to index on the key. However, you need to keep the following things in mind when deciding to use your own key as _id: • You must ensure the uniqueness of the key. • Also, consider the insertion order for your key because the insertion order will identify how much RAM will be used to maintain this index. • Retrieve fields as needed. When hundreds or thousands of requests are fulfilled per second, it's certainly advantageous to fetch only fields that are needed. • Use GridFS only for storing data that is larger than what can fit in a single document or is too big to load at once on the client, such as videos. Anything that will be streamed to a client is a good candidate for GridFS. • Use TTL to delete documents. If documents in a collection need to be deleted after a pre-defined time period, the TTL feature can be used to automatically delete the document after it reaches the predefined age. • Use capped collections if you require high throughput based on insertion orders. In some scenarios, based on data size you need to maintain a rolling window of data in the system. For example, a capped collection can be used to store a high-volume system's log information to quickly retrieve the most recent log entries. • Note that MongoDB's flexible schema can lead to inconsistent data if care is not taken. For example, the ability to duplicate data (embedded documents) if not updated properly can lead to data inconsistency, and so on. So it's very important to check for data consistency. • Although MongoDB handles seamless failover, per good coding practice, the application should be well written to handle any exception and to gracefully handle such a situation.	
4.	**Attempt *any three* of the following:**	**15**
a.	Explain about TimesTen In-Memory Database with a neat diagram. **TimesTen Explanation: 4 Marks** TimesTen is a relatively early in-memory database system that aspires to support workloads similar to a traditional relational system, but with better performance. TimesTen was founded in 1995 and acquired by Oracle in 2005. Oracle offers it as a standalone in-memory database or as a caching database supplementing the traditional disk-based Oracle RDBMS. In a TimesTen database, all data is memory resident. Persistence is achieved by writing periodic snapshots of memory to disk, as well as writing to a disk-based transaction log following a transaction commit. In the default configuration, all disk writes are asynchronous: a database operation would normally not need to wait on a disk IO operation. However, if the power fails between the transaction commit and the time the transaction log is written, then data could be lost. This behavior is not ACID compliant because transaction durability (the "D" in ACID) is not guaranteed. However, the user may choose to configure synchronous writes to the transaction log during commit operations. In this case, the database becomes ACID compliant, but some database operations will wait on disk IO. Figure illustrates the TimesTen architecture. When the database is started, all data is loaded from checkpoint files into main memory (1). The application interacts with TimesTen via SQL requests that are guaranteed to find all relevant data inside that main memory (2).	

	Periodically or when required database data is written to checkpoint files (3). An application commit triggers a write to the transaction log (4), though by default this write will be asynchronous so that the application will not need to wait on disk. The transaction log can be used to recover the database in the event of failure (5). **Diagram – 1 Mark**  ***Figure 7-5.*** *TimesTen Architecture*	
b.	Explain about Solid State Disk. **Explanation : 5 Marks** In contrast to a magnetic disk, solid state disks contain no moving parts and provide tremendously lower IO latencies. Commercial SSDs are currently implemented using either DDR RAM—effectively a battery-backed RAM device—or NAND flash. NAND flash is an inherently nonvolatile storage medium and almost completely dominates today's SSD market. Performance of flash SSD is on orders of magnitude superior to magnetic disk devices, especially for read operations. A random read from a high-end solid state disk may complete in as little as 25 microseconds, while a read from a magnetic disk may take up to 4,000 microseconds (4 milliseconds or 4/1000 of a second)—over 150 times slower. While SSDs are certainly faster than magnetic disks, the speed improvement is not proportionate for all workloads. In particular, it costs more—takes longer—to modify information in an SSD than to read from it. SSDs store bits of information in cells. A single-level cell (SLC) SSD contains one bit of information per cell, while a multi-level cell (MLC) SSD contains more than one bit— usually only two but sometimes three—in each cell. Cells are arranged in pages of about 4K and pages are arranged in blocks that typically contain 256 pages. Read operations, and initial write operations, require only a single-page IO. However, changing the contents of a page requires an erase and overwrites of a complete block. Even the initial write can be significantly slower than a read, but the block erase operation is particularly slow—around two milliseconds.	
c.	Discuss the oracle 12c In-Memory Database architecture with a neat diagram. Explanation: 3 Marks Oracle RDBMS version 12.1 introduced the "Oracle database in-memory" feature. This wording is potentially misleading, since the database as a whole is not held in memory. Rather, Oracle has implemented an in-memory column store to supplement its disk-based row store. Figure 7-9 illustrates the essential elements of the Oracle in-memory column store architecture. OLTP applications work with the database in the usual manner. Data is maintained in disk files (1), but cached in memory (2). An OLTP application primarily reads	

and writes from memory (3), but any committed transactions are written immediately to the transaction log on disk (4). When required or as configured, row data is loaded into a columnar representation for use by analytic applications (5). Any transactions that are committed once the data is loaded into columnar format are recorded in a journal (6), and analytic queries will consult the journal to determine if they need to read updated data from the row store (7) or possibly rebuild the columnar structure.

Diagram : 2 Marks

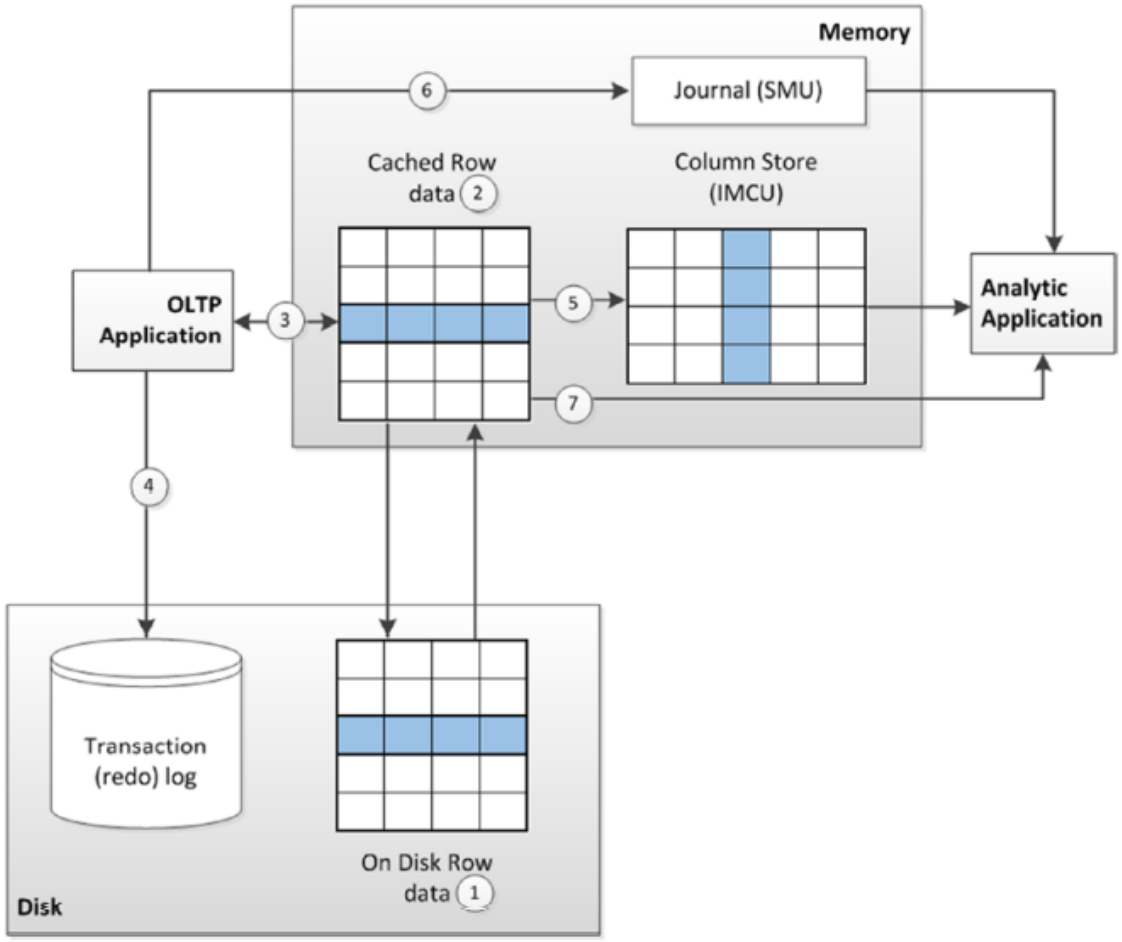


Figure 7-9. *Oracle 12c "in-memory" architecture*

d. What is JQuery? Explain jQuery element selector, id selector and class selector with example.

What is JQuery: 1 Mark

jQuery is a small and lightweight JavaScript library.

3 selectors: 3 Marks

Element Selector

The jQuery element selector selects elements based on the element name.You can select all <p> elements on a page like this: $("p")

Id selector

The jQuery #id selector uses the id attribute of an HTML tag to find the specific element. An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.To find an element with a specific id, write a hash character, followed by the id of the HTML element: $("#test)

Class Selector

The jQuery .class selector finds elements with a specific class.To find elements with a specific class, write a period character, followed by the name of the class: `$(".test")`

Example: 1 Mark

```
$(document).ready(function(){
  $("button").click(function(){
  $("p").hide();
   $("#test1").hide();
   $(".test").hide();
  });
});
</script>
</head>
<body>

<h2 class="test">This is a heading</h2>

<p class="test">This is a paragraph.</p>
<p>This is another paragraph.</p>
<p id="test1">This is other paragraph.</p>
<button>Click me</button>
```

e. What is an Event? Explain with syntax fadeIn() and fadeOut() jQuery methods.

What is an Event: 1 Mark

An event represents the precise moment when something happens.

fadeIn() and fadeOut(): 2 Marks each

fadeIn()

With jQuery you can fade an element in and out of visibility. The jQuery fadeIn() method is used to fade in a hidden element.

Syntax: $(selector).fadeIn(speed,callback);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

fadeOut():

The jQuery fadeOut() method is used to fade out a visible element.

Syntax: $(selector).fadeOut(speed,callback);

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

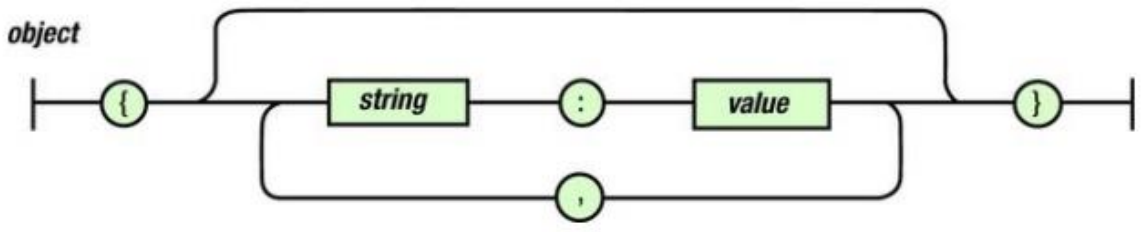
f. Explain the features supported by jQuery.

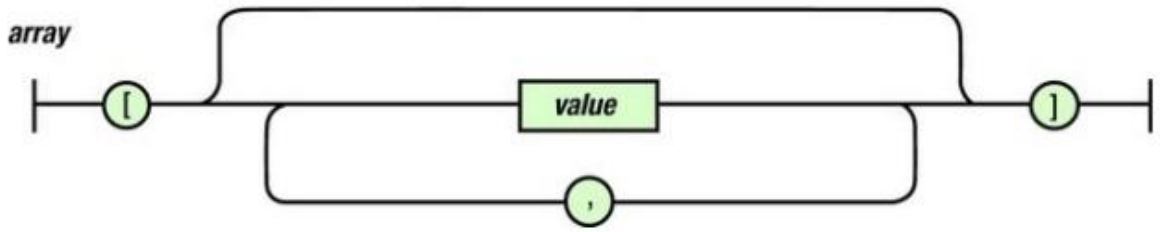
Any 5 features: 5 Marks

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.	
5.	**Attempt *any three* of the following:**	**15**
a.	What is the use of stringify method? Explain with syntax. **Stringify: 1 Mark** stringify is used for serializing JavaScript values into that of a valid JSON. **Syntax: 1 Mark** JSON.stringify(value[, replacer [, space]]); **Explanation: 3 Marks**. The method itself accepts three parameters, value, replacer, and space The **value** parameter of the stringify method is the only required parameter of the three parameters. The argument supplied to the method represents the JavaScript value intended to be serialized. This can be that of any object, primitive, or even a composite of the two. The second parameter, **replacer**, is optional, and when supplied, it can augment the default behavior of the serialization that would otherwise occur. The third parameter**, space**, is also optional and allows you to specify the amount of padding that separates each value from one another within the produced JSON text. This padding provides an added layer of readability to the produced string.	
b.	Explain the six members of the web storage Interface. **Six members: 5 Marks** **setItem** does not merely accept a singular string but, rather, requires two strings to be provided. The first string represents the name of the key, and the second string will represent the value to be held. Parameters : string (key), string (value) Return: void **getItem** It allows us to retrieve the persisted state that corresponds to the key provided to the method getItem(key) The key is the only expected parameter, will return the corresponding state for the supplied key. If, however, the name of the key supplied does not exist on the Storage Object, a value of null will be returned. **removeItem** The Storage Object method removeItem is the sole means of expiring the persistence of an individual key/value pair. Its signature is similar to that of getItem, in that it accepts one parameter. removeItem(key) **clear** This method clear does not require any parameters. This is because this method is simply used to instantly purge each and every key/value pair retained by the targeted Storage Object. clear( ) **key** The Storage Object method key is used to obtain the identities of all stored keys that possess	

	accompanying data retained by the given Storage Object. If a value does not exist for the provided index, the method will return a value of null. key(index) **length** The Storage Object provides us with access to the length of all values stored by the Storage Object in question. This total can be obtained via the length property.	
c.	Explain the structure of Hypertext Transfer Protocol (HTTP) – Request. **Structure: 1 mark** **Parts Required** 1 Request Line Yes 2 Headers No 3 Entity Body No Request Line The first component, known as the request line, is absolutely mandatory for any request. It alone is responsible for the type of request, the resource of the request, and, last, which version of the HTTP protocol the client is making use of. The request line itself is composed of three parts, separated from one another by whitespace. These three components are Method, Request-URI, and HTTP-Version. Method represents the action to be performed on the specified resource and can be one of the following: GET, POST, HEAD, PUT, LINK, UNLINK, DELETE, OPTIONS, and TRACE. Headers The second component of the request concerns the manner by which the request is able to provide supplemental meta-information. The meta-information is supplied within the request in the form of a header, whereas a header, at its most atomic unit, is simply a key/value pair separated by the colon (:) The HTTP protocol has formalized a plethora of headers that can be utilized to relay a variety of detail to the server. These headers fall under one of three categories: general headers, request headers, and entity headers Entity Body The final component of the request is the entity body. While the entity headers carry the meta-information, the entity body is strictly the nomenclature for the data being sent to the server. The syntax of the entity can reflect that of HTML, XML, or even JSON. However, if the Content-Type entity header is not supplied, the server, being the receiving party of the request, will have to guess the appropriate MIME type of the data provided.	
d.	Explain the JSON Grammar. **Explanation : 5 Marks** 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 The two structural representations of JSON through a series of syntax diagrams. Figure illustrates the grammatical representation for a collection of string/value pairs  As the diagram outlines, a collection begins with the use of the opening brace ({), and ends with the use of the closing brace (}). The content of the collection can be composed of any of	

	the following possible three designated paths: • The top path illustrates that the collection can remain devoid of any string/value pairs. • The middle path illustrates that our collection can be that of a single string/value pair. • 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 pair possessed by the collection must be delimited or separated from one another by way of a comma (,). An ordered list of values Now we can see the grammatical representation for that of an ordered list of values. 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 singular value. The bottom path illustrates that the length of our list can possess any number of values, which must be delimited, that is, separated, with the use of a comma (,).	
e.	Give an overview about JavaScript Object Notation (JSON). Also explain about JSON tokens. **JSON overview: 2 Marks** JSON is a text format for storing and transporting data JSON is "self-describing" and easy to understand JSON is plain text written in JavaScript object notation JSON is language independent **JSON Tokens: 3 Marks** When regarding the interchange of JSON and the many languages that do not natively possess Objects and Arrays, the tokens that make up the JSON text are all that is required to interpret if any collections or ordered lists exist and apply all values in a manner required of that language. This is accomplished with six structural characters, as listed below Token Literal Name Array Opening [Left square bracket Array Closing] Right square bracket Object opening { Left curly bracket Object closing } Right curly bracket Name/value separator : colon Value separator , comma	
f.	Explain about JSON parsing with syntax. **Explanation: 5 Marks** 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.parse Method **JSON.parse(text [, reviver]);** JSON.parse can accept two parameters, text and reviver. The name of the parameter text is indicative of the value it expects to receive. The parameter reviver is used similarly to the replacer parameter of stringify, 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's signature, only the provision of text is required.	