_______Mongo DB Assignment______

Theoretical Questions

---

1. What are the key differences between SQL and NoSQL databases?

The primary differences lie in their data model, schema, scalability, and query language.

- **Data Model: SQL databases are relational (like MySQL, PostgreSQL) and store data in tables with rows and columns. NoSQL databases are non-relational and can have various data models, such as document (MongoDB), key-value (Redis), column-family (Cassandra), or graph (Neo4j).**
- **Schema: SQL databases have a predefined, rigid schema. You must define the table structure before inserting data. NoSQL databases typically have a dynamic schema, allowing you to insert documents without a predefined structure, which offers great flexibility.**
- **Scalability: SQL databases are designed to scale vertically, meaning you increase performance by adding more resources (CPU, RAM) to a single server. NoSQL databases are built to scale horizontally, meaning you add more servers to distribute the load, which is often more cost-effective for large-scale applications.**
- **Query Language: SQL databases use the powerful and standard Structured Query Language (SQL). NoSQL databases have their own unique query languages or APIs, which can vary significantly between different databases.**

---

2. What makes MongoDB a good choice for modern applications?

MongoDB is well-suited for modern applications due to its:

- **Flexible Document Model: It stores data in JSON-like BSON documents, which can be easily mapped to objects in modern programming languages. This flexibility allows for rapid development and iteration as application requirements change.**
- **High Scalability: MongoDB is designed for horizontal scaling through a process called sharding, allowing it to handle massive amounts of data and high traffic loads.**
- **High Availability: Through replica sets, MongoDB provides automatic failover and data redundancy, ensuring that applications remain online even if a server fails.**
- **Rich Query Language: It supports a powerful query language, indexing, and real-time aggregation capabilities that allow for sophisticated data analysis.**

---

3. Explain the concept of collections in MongoDB.

A collection in MongoDB is a grouping of BSON documents. It is the equivalent of a table in a relational (SQL) database.

- **Collections exist within a database.**
- **Unlike SQL tables, collections do not enforce a schema by default. This means documents within the same collection can have different fields and structures.**
- **For example, you could have an `users` collection where some documents have a `middle_name` field and others do not.**

---

4. How does MongoDB ensure high availability using replication?

MongoDB ensures high availability through replica sets. A replica set is a group of connected MongoDB instances (`mongod` processes) that maintain the same dataset.

A replica set consists of:

- One Primary Node: This is the main node that receives all write operations.
- Multiple Secondary Nodes: These nodes replicate the data from the primary node. They can be used for read operations to distribute the load.

If the primary node becomes unavailable, the remaining secondary nodes will automatically elect a new primary from among themselves. This failover process ensures that the database remains operational with minimal downtime. 🛡️

---

5. What are the main benefits of MongoDB Atlas?

MongoDB Atlas is the official cloud database-as-a-service (DBaaS) for MongoDB. Its main benefits are:

- Fully Managed: Atlas automates time-consuming administrative tasks like database setup, configuration, patching, and backups.
- Multi-Cloud Deployment: It allows you to deploy your database on major cloud providers like AWS, Google Cloud, and Azure, preventing vendor lock-in.
- Global Distribution: You can easily deploy clusters across different geographic regions to reduce latency for users worldwide.
- Built-in Security: It provides robust security features out-of-the-box, including network isolation, encryption at rest and in transit, and role-based access control.
- Scalability: You can easily scale your database up or down with just a few clicks or via an API call, without any downtime.

---

6. What is the role of indexes in MongoDB, and how do they improve performance?

Indexes in MongoDB store a small portion of the collection's data in an easy-to-traverse form. The role of an index is to make queries faster. ⚡

Without an index, MongoDB must perform a collection scan, meaning it has to look through every single document in a collection to find the ones that match the query. As the collection grows, this becomes very slow.

When you create an index on a specific field, MongoDB creates a sorted data structure. When you query that field, MongoDB can use the index to quickly locate the relevant documents without scanning the entire collection, dramatically improving query performance.

---

7. Describe the stages of the MongoDB aggregation pipeline.

The aggregation pipeline is a framework for data analysis in MongoDB. It consists of a sequence of stages, where each stage transforms the documents as they pass through it. The output of one stage becomes the input for the next.

Common stages include:

- `$match`: Filters documents, similar to a `find()` query, allowing only matching documents to pass to the next stage.
- `$group`: Groups documents by a specified identifier and applies accumulator expressions (e.g., `$sum`, `$avg`, `$max`) to the grouped data.
- `$project`: Reshapes documents by adding new fields, removing existing fields, or renaming fields.
- `$sort`: Sorts the documents based on a specified field.
- `$limit`: Restricts the number of documents passed to the next stage.
- `$lookup`: Performs a left outer join to another collection in the same database.

---

8. What is sharding in MongoDB?

How does it differ from replication?

Sharding is MongoDB's method for horizontal scaling. It involves distributing data across multiple servers or clusters, known as shards. Each shard holds a subset of the total data, allowing the database to handle larger datasets and higher throughput than a single server could manage. Sharding is used to solve problems of data size and traffic volume.

The key difference between sharding and replication is their purpose:

- **Replication is for high availability and redundancy. It copies the *entire* dataset to multiple servers. Its goal is to prevent data loss and ensure the system remains online if a server fails.**
- **Sharding is for scalability. It splits the dataset into *parts* and distributes those parts across multiple servers. Its goal is to handle large amounts of data and high loads by distributing the work.**

A production environment typically uses both sharding and replication: each shard is itself a replica set to ensure both scalability and high availability.

---

9. What is PyMongo, and why is it used?

PyMongo is the official and most widely used Python driver for MongoDB. It's a library that allows Python applications to connect to and interact with a MongoDB database.

It is used to perform all database operations from a Python script, including:

- **Connecting to a MongoDB instance or cluster.**
- **Creating, reading, updating, and deleting (CRUD) documents.**

- Executing aggregation pipelines.
- Managing indexes and collections.

---

10. What are the ACID properties in the context of MongoDB transactions?

Starting from version 4.0, MongoDB supports multi-document ACID transactions on replica sets. The ACID properties ensure data integrity:

- Atomicity: Transactions are "all or nothing." If any operation within the transaction fails, the entire transaction is aborted, and the database is left unchanged.
- Consistency: A transaction brings the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including schema validation.
- Isolation: Transactions that run concurrently produce the same result as if they were executed serially (one after another). MongoDB achieves this using snapshot isolation.
- Durability: Once a transaction is successfully committed, the changes are permanent and will survive any subsequent system failures, such as a power outage or crash.

---

11. What is the purpose of MongoDB's explain() function?

The `explain()` function is a diagnostic tool used to get detailed information about how MongoDB executes a query. When you append `.explain()` to a query (like `find()` or `aggregate()`), it doesn't return the query results but instead provides a report.

This report includes details like:

- Which index, if any, was used for the query.
- Whether a collection scan was performed.

- The number of documents scanned versus the number of documents returned.
- The execution time and other performance metrics.

Developers use `explain()` to diagnose and optimize slow queries, for instance, by seeing if an index is being used effectively.

---

12. How does MongoDB handle schema validation?

While MongoDB is known for its flexible schema, you can enforce a specific structure using schema validation. This feature allows you to define rules that documents must follow to be inserted or updated in a collection.

Schema validation is defined on a per-collection basis using the `$jsonSchema` operator. You can specify:

- The data type of fields (e.g., `bsonType: "string"`).
- Required fields in a document.
- Ranges for numerical values.
- Regular expressions for string patterns.

You can also set the validation level to `strict` (apply rules to all inserts/updates) or `moderate` (apply rules only to valid documents), and choose whether to `error` or just `warn` when a document fails validation.

---

13. What is the difference between a primary and a secondary node in a replica set?

In a MongoDB replica set:

- Primary Node: This is the master node. There can only be one primary node at any given time. It is the only node that can accept write operations (inserts, updates, deletes). All changes are recorded in its operation log (oplog).

- **Secondary Node(s): These are the slave nodes. They asynchronously replicate data from the primary's oplog to maintain an identical copy of the dataset. Secondaries cannot accept write operations but can be used to serve read traffic, helping to distribute the load. They also participate in elections to choose a new primary if the current one fails.**

---

14. What security mechanisms does MongoDB provide for data protection?

MongoDB provides a comprehensive set of security features to protect data:

- **Authentication: Verifies the identity of users connecting to the database. MongoDB supports mechanisms like SCRAM (default), x.509 certificates, and LDAP.**
- **Authorization (Role-Based Access Control): Restricts what authenticated users are allowed to do. You can assign built-in or custom roles to users that grant specific privileges (e.g., read-only, read-write) on certain databases or collections.**
- **Encryption:**
    - **In-Transit Encryption: Uses TLS/SSL to encrypt all data moving between the client and the server and between nodes in a cluster.**
    - **At-Rest Encryption: Encrypts the database files on disk, available in the Enterprise version or through cloud providers like Atlas.**
    - **Client-Side Field Level Encryption: Allows applications to encrypt specific fields in a document before sending them to the database, so the server never sees the sensitive data in plaintext.**
- **Auditing: Records administrative and data access events on the database for security analysis.**

---

15. Explain the concept of embedded documents and when they should be used.

An embedded document (or sub-document) is a document that is nested inside another document within a MongoDB collection. This creates a "one-to-many" or "one-to-few" relationship where the "many" or "few" items are stored within the "one" parent document.

Example: Instead of separate `user` and `address` collections, you can embed address information directly within the user document:

JSON

```json
{
  "_id": 1,
  "name": "John Doe",
  "email": "john@example.com",
  "addresses": [
    { "type": "home", "street": "123 Main St", "city": "Anytown" },
    { "type": "work", "street": "456 Business Rd", "city":
"Businesstown" }
  ]
}
```

When to use them:

- **For "contains" or "has-a" relationships where the embedded data doesn't have a reason to exist on its own.**
- **When you need to retrieve the related data in a single database query. Embedding avoids the need for joins and can improve read performance significantly.**
- **When the "many" side is not excessively large and doesn't grow unboundedly. If an array of embedded documents grows too large, it can hit the 16MB BSON document size limit and hurt performance.**

---

16. What is the purpose of MongoDB's $lookup stage in aggregation?

The `$lookup` stage is used in an aggregation pipeline to perform a left outer join with another collection in the same database.

Its purpose is to de-normalize data by pulling in documents from another collection and adding them to the input documents. For example, if you have an `orders` collection and a `products` collection, you could use `$lookup` to join `orders` with `products` based on a `product_id` field to get the full product details for each order. 🔗

---

17. What are some common use cases for MongoDB?

MongoDB's flexibility and scalability make it suitable for a wide range of use cases, including:

- Content Management Systems (CMS): Easily store articles, blog posts, and user comments with varying fields.
- E-commerce Applications: Manage product catalogs where different products have different attributes, and handle user profiles and shopping carts.
- Real-time Analytics: Use the aggregation framework to process and analyze large volumes of data for dashboards and reports.
- Internet of Things (IoT): Ingest and process high-velocity time-series data from sensors and devices.
- Single View Applications: Aggregate data from multiple systems into a single, comprehensive view of a customer or entity.

---

18. What are the advantages of using MongoDB for horizontal scaling?

The main advantages of using MongoDB's horizontal scaling (sharding) are:

- Increased Storage Capacity: By distributing data across multiple machines, you can store much larger datasets than would be possible on a single server.
- Increased Throughput: Both read and write operations can be distributed across the shards, allowing the database to handle a higher load of concurrent requests.
- High Availability: When combined with replica sets (each shard is a replica set), you get both the scalability of sharding and the redundancy of replication.
- Cost-Effectiveness: Scaling out by adding more commodity servers is often cheaper than scaling up by purchasing a single, more powerful and expensive server.

---

19. How do MongoDB transactions differ from SQL transactions?

While both provide ACID guarantees, there are key differences:

- Scope: In MongoDB, the need for multi-document transactions is less frequent due to the document model, which often allows related data to be updated atomically within a single document. In SQL, related data is typically spread across multiple tables, making transactions essential for most operations.
- Syntax and Implementation: The syntax is different. MongoDB transactions are initiated on a session object (e.g., `with session.start_transaction():`) and have specific API calls. SQL transactions use commands like `BEGIN TRANSACTION`, `COMMIT`, and `ROLLBACK`.
- Performance: Because of the distributed nature of MongoDB, multi-document transactions can introduce higher latency compared to single-document operations or typical SQL transactions on a single node. It's recommended to use them only when necessary.

---

20. What are the main differences between capped collections and regular collections?

- **Size:** A capped collection has a fixed maximum size in bytes. A regular collection has no size limit other than available disk space.
- **Behavior:** Once a capped collection reaches its maximum size, it starts behaving like a circular buffer. The oldest documents are automatically overwritten to make space for new ones (First-In, First-Out). Regular collections simply grow as new data is added.
- **Insertion Order:** Capped collections maintain the insertion order of documents. Queries on a capped collection return documents in this order by default.
- **Use Case:** Capped collections are ideal for high-throughput scenarios where you only need to store recent data, such as logging, caching, or real-time event streaming. Regular collections are used for general-purpose data storage.

---

21. What is the purpose of the $match stage in MongoDB's aggregation pipeline?

The `$match` stage is used to filter documents in an aggregation pipeline. It works just like a `find()` query. Only the documents that satisfy the specified conditions are passed to the next stage in the pipeline.

Placing `$match` at the beginning of a pipeline is a crucial performance optimization, as it reduces the amount of data that subsequent stages need to process.

---

22. How can you secure access to a MongoDB database?

Securing access involves a layered approach:

1. Enable Authentication: Turn on access control so that only authenticated users can connect.
2. Enforce Authorization: Use Role-Based Access Control (RBAC) to give users only the permissions they need (Principle of Least Privilege).
3. Encrypt Communication: Use TLS/SSL to encrypt all network traffic between clients and the database.
4. Limit Network Exposure: Configure firewalls to allow connections only from trusted IP addresses. Avoid exposing your database directly to the public internet.
5. Keep MongoDB Updated: Regularly apply security patches and updates to protect against known vulnerabilities.

---

23. What is MongoDB's WiredTiger storage engine, and why is it important?

WiredTiger is the default storage engine for MongoDB. A storage engine is the component of the database responsible for managing how data is stored, retrieved, and managed on disk and in memory.

WiredTiger is important because it provides several key features that are critical for modern, high-performance applications:

- Document-Level Concurrency: It uses optimistic concurrency control, allowing multiple clients to read and write to different documents in a collection simultaneously without conflict, which significantly improves performance for concurrent workloads.
- Compression: It supports compression libraries like Snappy (default) and zlib, which reduce the amount of disk space needed to store data and can improve I/O performance.
- In-Memory Caching: It has a highly efficient caching mechanism that keeps the most frequently accessed data in RAM for faster retrieval.
- Snapshots and Checkpoints: It supports stable checkpoints, making it durable and allowing for recovery from crashes.

MONGO DB PRACTICAL QUESTION

```
#1. Write a Python script to load the Superstore dataset from a CSV file
into MongoDB.

#ans:-

import pymongo

import csv


# 1. Connect to MongoDB

client = pymongo.MongoClient("mongodb://localhost:27017/")

db = client["superstore_db"]

collection = db["orders"]


# 2. Clear old data

collection.delete_many({})


# 3. Read CSV and insert into MongoDB

csv_file_path = 'superstore.csv'

data_to_insert = []


# Open file with the correct encoding

with open(csv_file_path, mode='r', encoding='latin-1') as file:

    csv_reader = csv.DictReader(file)

    for row in csv_reader:

        # Convert numeric fields

        try:

            row['Sales'] = float(row.get('Sales', 0))

            row['Quantity'] = int(row.get('Quantity', 0))
            row['Discount'] = float(row.get('Discount', 0))

            row['Profit'] = float(row.get('Profit', 0))

        except (ValueError, TypeError):

            pass

        data_to_insert.append(row)


if data_to_insert:

    collection.insert_many(data_to_insert)

    print(f"Successfully loaded {len(data_to_insert)} documents.")




#2. Retrieve and print all documents from the Orders collection.

#ans:-

# find({}) with an empty dictionary retrieves all documents

all_orders = collection.find({})


# The result is a cursor, which we can iterate over

for order in all_orders:

    print(order)



#3. Count and display the total number of documents in the Orders
collection

#ans:-

# count_documents({}) with an empty filter counts all documents
```

```
total_documents = collection.count_documents({})


print(f"Total number of documents in the Orders collection:
{total_documents}")



#4. Write a query to fetch all orders from the "West" region.

#ans:-

# The filter {"Region": "West"} specifies the condition

west_region_orders = collection.find({"Region": "West"})


print("Orders from the West region:")

for order in west_region_orders:

    print(order)



#5. Write a query to find orders where Sales is greater than 500.

#ans:-

# The $gt operator stands for "greater than"

high_sales_orders = collection.find({"Sales": {"$gt": 500}})


print("Orders with Sales greater than 500:")

for order in high_sales_orders:

    print(order)



#6. Fetch the top 3 orders with the highest Profit.

#ans:-
```

```
# We use sort() to order the results and limit() to get the top N

# pymongo.DESCENDING or -1 is used for descending order

top_3_profit_orders = collection.find({}).sort("Profit",
pymongo.DESCENDING).limit(3)


print("Top 3 orders with the highest profit:")

for order in top_3_profit_orders:

    print(order)



#7. Update all orders with Ship Mode as "First Class" to "Premium Class."

#ans:-

# The first dictionary is the filter, the second is the update operation

# $set is used to update the value of a field

update_result = collection.update_many(

    {"Ship Mode": "First Class"},

    {"$set": {"Ship Mode": "Premium Class"}}

)


print(f"Matched {update_result.matched_count} documents and modified
{update_result.modified_count} documents.")




#8. Delete all orders where Sales is less than 50

#ans:-

# The $lt operator stands for "less than"

delete_result = collection.delete_many({"Sales": {"$lt": 50}})
```

```
print(f"Deleted {delete_result.deleted_count} documents where Sales was
less than 50.")



#9. Use aggregation to group orders by Region and calculate total sales per
region.

#ans:-

# The aggregation pipeline is a list of stages

pipeline = [

    {

        "$group": {

            "_id": "$Region",  # Group by the "Region" field

            "TotalSales": {"$sum": "$Sales"}  # Calculate the sum of
"Sales" for each group

        }

    },

    {

        "$sort": {"TotalSales": -1} # Optional: sort by total sales

    }

]


sales_by_region = collection.aggregate(pipeline)


print("Total sales per region:")

for region_data in sales_by_region:

    print(region_data)



#10. Fetch all distinct values for Ship Mode from the collection
```

```
#ans:-

# The distinct() method returns a list of unique values for a given field

distinct_ship_modes = collection.distinct("Ship Mode")


print("Distinct Ship Modes:")

print(distinct_ship_modes)
```

```
#11. Count the number of orders for each category.

#ans:-

# This aggregation pipeline groups by "Category" and counts the documents
in each group

pipeline = [

    {

        "$group": {

            "_id": "$Category",

            "OrderCount": {"$sum": 1} # For each document in the group, add
1 to the count

        }

    }

]


orders_per_category = collection.aggregate(pipeline)


print("Number of orders per category:")

for category_count in orders_per_category:

    print(category_count)
```

```
#  Close the connection

client.close()

print("Connection closed.")
```