**Section 1: Introduction to MongoDB (10 points)**

**1. Define MongoDB and explain its significance in modern database management.**

A) MongoDB was founded by Dwight Merriman, Eliot Horowitz, and a team of developers which is a NoSQL database system. The history of MongoDB dates to the early 2000s when Merriman and Horowitz encountered challenges with traditional relational database systems while working at DoubleClick, an online advertising firm now owned by Google Inc

The name "MongoDB" was derived from the term "humongous" to reflect its capacity for handling vast amounts of data. MongoDB was released as an open-source project in 2009. It gained attention for its document-oriented data model, which offered greater flexibility compared to traditional relational database.

MongoDB is a widely used, open-source NoSQL database system designed to handle large volumes of data across distributed systems. Unlike traditional relational databases, which store data in tables with rows and columns, MongoDB stores data in a flexible, JSON-like format called BSON (Binary JSON), making it well-suited for modern application development.

Key features of MongoDB include:

1. **Document-Oriented**: MongoDB stores data in documents, which are JSON-like data structures. Each document can have its own structure, and fields within documents can vary from one document to another, providing flexibility in data modeling.
2. **Scalability**: MongoDB is designed to scale horizontally across multiple servers, allowing for high availability and performance. It supports automatic sharding, which distributes data across multiple servers to handle large data sets and high throughput.
3. **High Performance**: MongoDB's architecture and indexing capabilities enable efficient data retrieval and query processing. It supports various types of indexes, including single-field, compound, and geospatial indexes, to optimize query performance.
4. **Rich Query Language**: MongoDB provides a powerful query language that supports a wide range of operations, including CRUD (Create, Read, Update, Delete) operations, aggregation, text search, and geospatial queries.
5. **Built-in Replication and Failover**: MongoDB supports replica sets, which are groups of MongoDB servers that maintain copies of the same data. Replica sets provide high availability and data redundancy by automatically electing a primary server and handling failover in case of node failures.
6. **Flexible Data Model**: MongoDB's flexible schema allows developers to evolve their data models over time without requiring schema migrations. This flexibility is particularly useful in agile development environments where requirements may change frequently.
7. **Rich Ecosystem**: MongoDB has a vibrant ecosystem of drivers, libraries, and tools for various programming languages and platforms. These include official MongoDB drivers for popular programming languages like Python, Java, Node.js, and others, as well as integrations with frameworks and cloud platforms.

**2. Discuss the advantages of using MongoDB Atlas over traditional self-hosted MongoDB installations.**

A) MongoDB Atlas simplifies database management with automated updates, maintenance, and scaling. It ensures high availability, fault tolerance, and security with managed clustering and replication. Comprehensive monitoring, global load balancing, and compliance support make MongoDB Atlas an ideal choice for modern applications, offering seamless scalability and hassle-free operation.

MongoDB offers several potential benefits:

* **Aggregation.** The DBMS also has built-in aggregation capabilities, which lets users run code directly on the database rather than running MapReduce on Hadoop. The use of the file system is primarily for storing files larger than BSON's size limit of 16 MB per document.
* **Automated Updates and Maintenance**: MongoDB Atlas manages routine updates, patches, and maintenance tasks automatically. This ensures that the database infrastructure stays up to date with the latest features, bug fixes, and security patches without requiring manual intervention from the user.
* **Performance Optimization**: MongoDB Atlas offers performance optimization features such as query profiling, performance monitoring, and real-time analytics. Users can use these tools to identify bottlenecks, optimize query performance, and improve the overall responsiveness of their applications.
* **Managed Clustering and Replication**: MongoDB Atlas simplifies the setup and management of database clusters and replication configurations. Users can easily configure replica sets and sharded clusters to distribute data across multiple nodes for improved scalability, fault tolerance, and high availability.
* **Resource Efficiency**: MongoDB Atlas perfects resource use by automatically adjusting resource allocation based on workload patterns and usage trends. This helps to minimize resource wastage and ensures that the database infrastructure still is cost-effective and efficient.
* **Comprehensive Monitoring and Alerting**: MongoDB Atlas provides comprehensive monitoring and alerting capabilities, allowing users to track key performance metrics, watch system health, and receive real-time alerts for critical events or anomalies. This enables initiative-taking monitoring and troubleshooting, helping to prevent downtime and ensure continuous availability.
* **Global Load Balancing and Traffic Management**: MongoDB Atlas offers global load balancing and traffic management features that enable users to distribute incoming traffic across multiple geographic regions or data centers. This helps to reduce latency, improve responsiveness, and enhance the overall user experience for globally distributed applications.
* **Compliance and Regulatory Support**: MongoDB Atlas follows various industry standards and regulatory requirements, including GDPR, HIPAA, and SOC 2. The platform provides built-in security controls, encryption mechanisms, and audit trails to help organizations meet their compliance obligations and data protection requirements.

**Section 3: MongoDB and the Document Model (10 points)**

**3. Explain the document model used in MongoDB. How does it differ from the relational model used in traditional databases? Can provide examples**.

A relational database management system is a collection of programs and capabilities that let IT teams and others create, update, administer and otherwise interact with a relational database. Database store data in the form of tables and rows. Although it is not necessary, RDBMS most commonly uses SQL.

One of the main differences between MongoDB and RDBMS is that RDBMS is a relational database while MongoDB is nonrelational. Likewise, while most RDBMS systems use SQL to manage stored data, MongoDB uses BSON for data storage -- a type of NoSQL database. While RDBMS uses tables and rows, MongoDB uses documents and collections. In RDBMS a table -- the equivalent to a MongoDB collection -- stores data as columns and rows. Likewise, a row in RDBMS is the equivalent of a MongoDB document but stores data as structured data items in a table. A column denotes sets of data values, which is the equivalent to a field in MongoDB.

For instance, consider a document representing a user profile in MongoDB:

*{*

*"\_id": ObjectId("61f1ed834dbedda3b7d5429c"),*

*"title": "Friends",*

*"director": "Matthew Perry",*

*"genre": ["Drama", "Comedy"],*

*"release\_year": 1994,*

*"actors": ["Jennifer Aniston", "Lisa kudrow", “Courtney cox”.”Mathew Perry”,”Matt LeBlance”,”David”],*

*"ratings": {*

*"IMDb": 9.2,*

*"Rotten Tomatoes": 91*

*}*

*}*

Relational Model:

Movie Table:

| **id** | **title** | **director** | **release\_year** |
| --- | --- | --- | --- |
| 1 | Friends | Matthew Perry | 1994 |

Genre Table:

| **id** | **movie\_id** | **genre** |
| --- | --- | --- |
| 1 | 1 | Drama |
| 2 | 1 | Comedy |

Actor Table:

| **id** | **movie\_id** | **actor** |
| --- | --- | --- |
| 1 | 1 | Jennifer Aniston |
| 2 | 1 | Lisa kudrow |
| 3 | 1 | Courtney cox |
| 4 | 1 | Matthew Perry |
| 5 | 1 | Matt LeBlance |
| 6 | 1 | David |

Ratings Table:

| **id** | **movie\_id** | **source** | **rating** |
| --- | --- | --- | --- |
| 1 | 1 | IMDb | 9.2 |
| 2 | 1 | Rotten Tomatoes | 91 |

In the relational model, the movie's information is split across multiple tables: Movie, Genre, Actor, and Ratings. The Movie table contains details about the movie, while the Genre, Actor, and Ratings tables store related information such as genres, actors, and ratings respectively. The movie\_id serves as a foreign key linking the records in the related tables back to the original movie record.

**==========================================================================**

**Section 4: MongoDB Data Modelling Intro (10 points)**

**4. Describe the process of data modelling in MongoDB. How does it differ from data modelling in relational databases? Can provide examples.**

A) A data model is like a blueprint for our data. A good data model can provide structure and organization to what might be a diverse and complex set of information. **Data modeling** is the process of building a structure for a database system. The goal of data modeling is to ensure that data is organized in a way that is efficient, easy to maintain, and meets the application criteria.

MongoDB's document-oriented data model offers great flexibility compared to traditional relational databases, allowing for easier handling of semi-structured and unstructured data. However, modeling data requires designing the structure of the documents and collections in a MongoDB database. In the following sections, we will discuss the concept of document-based data modeling, the importance of denormalization, and the advantages of MongoDB's flexible schema.

| **Aspect** | **MongoDB Data Modeling** | **Relational Database Data Modeling** |
| --- | --- | --- |
| Data Structure | Document-oriented, flexible schema | Table-oriented, fixed schema |
| Hierarchical Data Storage | Suitable for hierarchical data storage | Not inherently suitable for hierarchical data |
| Scalability | Horizontally scalable, adding more servers | Vertically scalable, increasing RAM |
| Schema | Dynamic schema, no predefined structure | Predefined schema, requires defined structure |
| Vulnerability to SQL Injection | Not affected by SQL injection | Vulnerable to SQL injection attacks |
| Data Integrity and Transactions | Centers around the CAP theorem | Centers around ACID properties (Atomicity, Consistency, Isolation, Durability) |
| Data Representation | Document-based | Row-based |
| Performance | Typically faster than relational databases | May be slower compared to MongoDB |
| Joins | No support for complex joins | Supports complex joins |
| Storage Structure | Field-based | Column-based |
| Client Query Language Support | Provides a JavaScript client for querying | May not provide a JavaScript client for querying |
| Query Language Support | Supports JSON query language along with SQL | Supports SQL query language only |

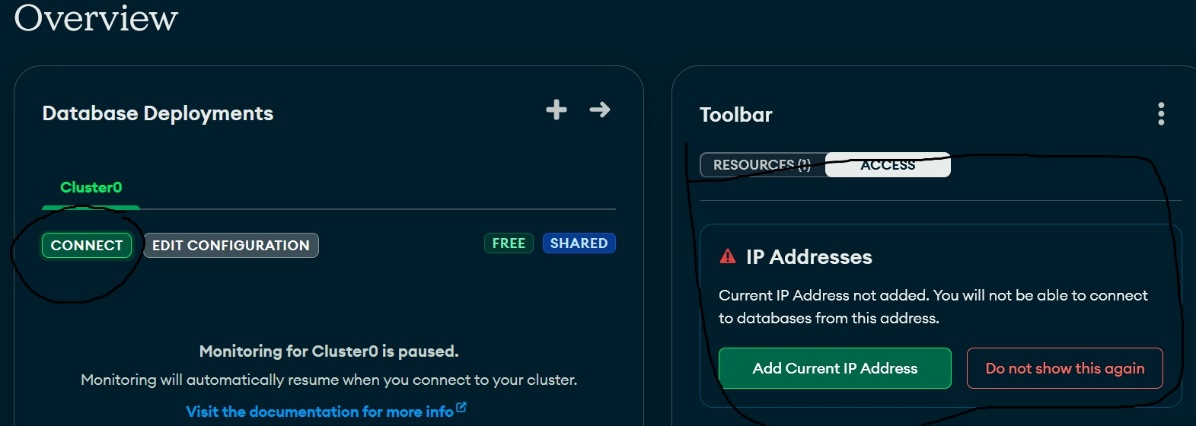
===========================================================================

**Section 5: Connecting to a MongoDB Database (10 points)**

**5. Explain the steps involved in connecting to a MongoDB database using the MongoDB shell. Can provide screenshots**.

* Install **MongoDB**: Download and install MongoDB from the official MongoDB website: <https://www.mongodb.com/try/download/community>
* **Start MongoDB Server**: Launch the MongoDB server by executing the mongod command in your terminal. This action initiates the MongoDB server process.
* **Open MongoDB Shell**: Access the MongoDB shell by running the mongo command in your terminal. This opens the MongoDB shell interface.
* **Connect to Database**: Within the MongoDB shell, utilize the use command followed by the name of the database to connect to a specific database.

**Add current device IP Address before we connect and click to connect, which connect the applications**



This connects to Shell

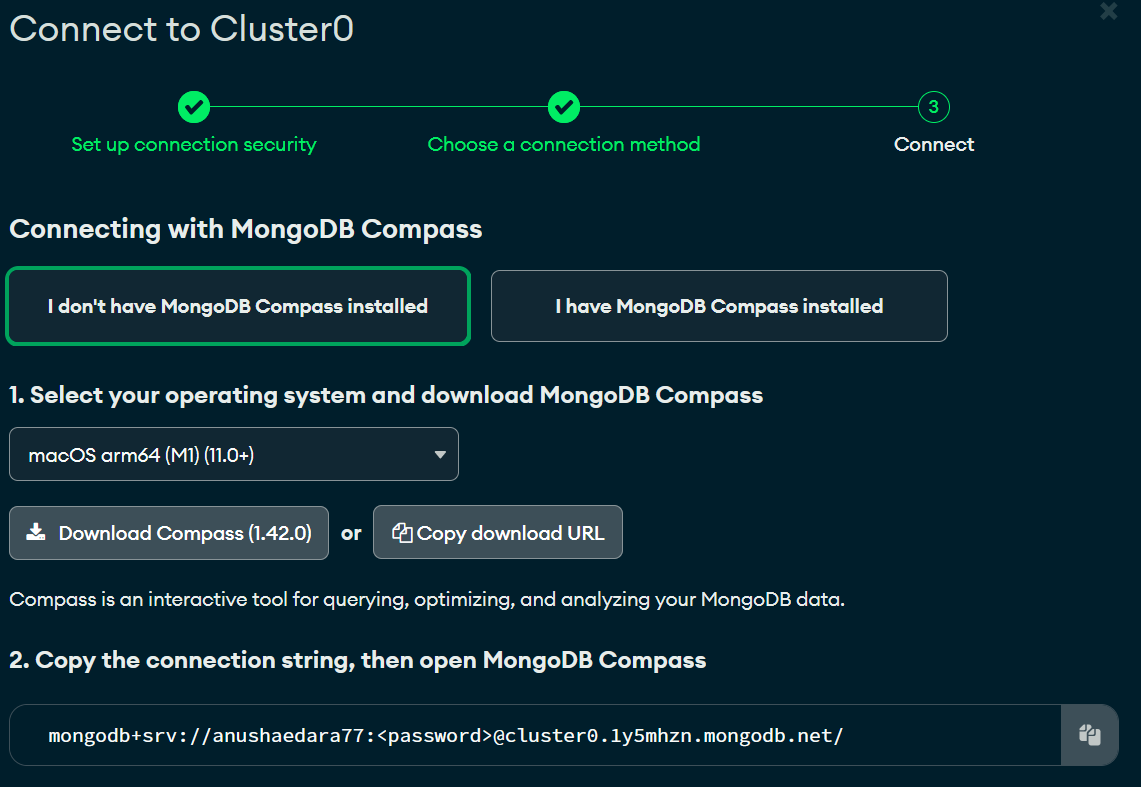
A screenshot of a computer

Description automatically generated

A screenshot of a computer program

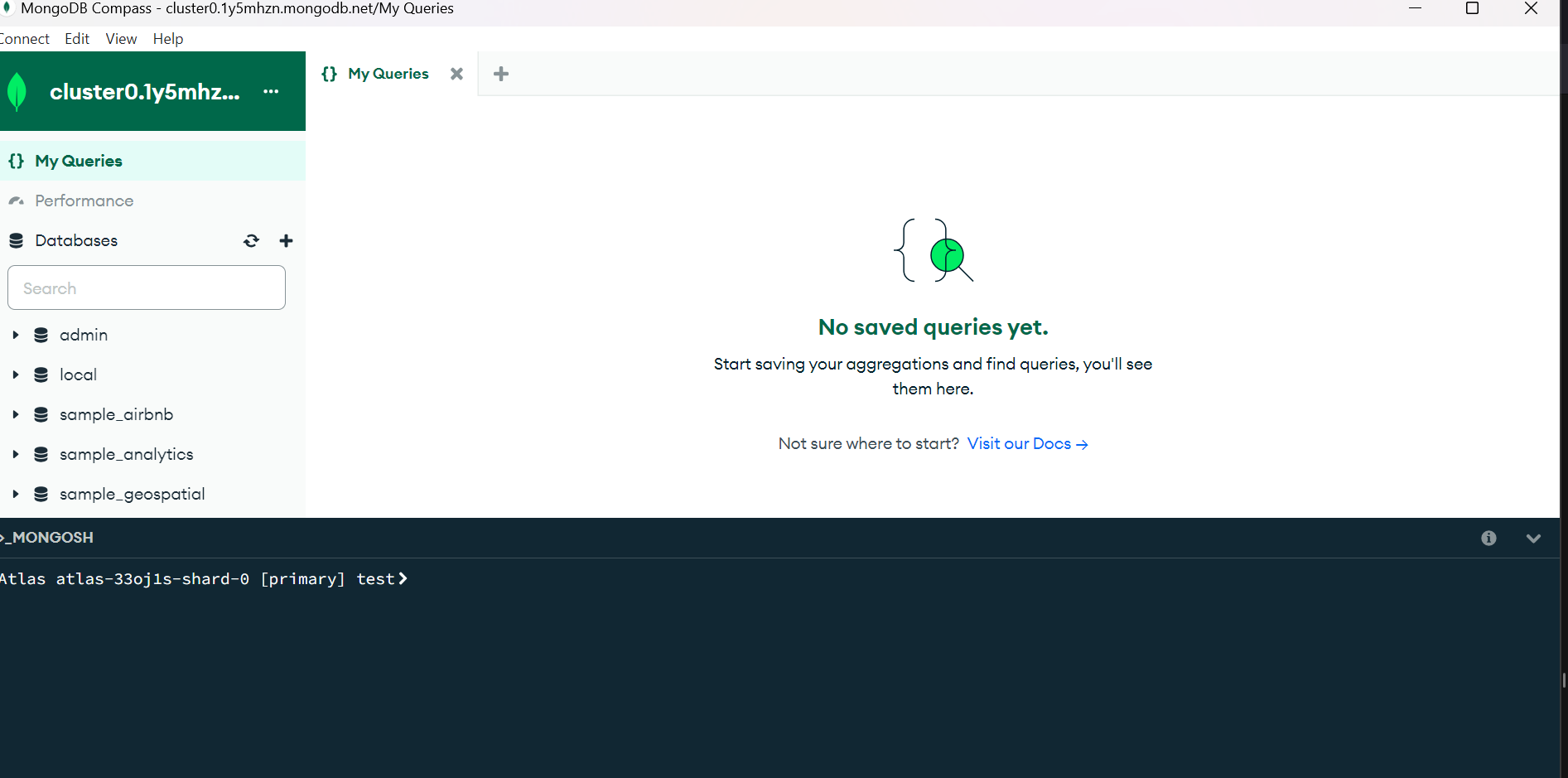
Description automatically generated

This connects to compass



Copy the MongoDB compass link and paste the Compass interface

mongodb+srv://anushaedara77:<password>@cluster0.1y5mhzn.mongodb.net/ (give password and connect the server)



* **Execute Database Operations**: With the successful connection, you can now perform various operations like inserting, querying, updating, or deleting documents using MongoDB shell commands.

**Section 6: Connecting to MongoDB in Python (10 points)**

6. **Discuss the methods available for connecting to MongoDB databases using Python. Provide code examples to illustrate your points.**

A) In Python, there are several methods available for connecting to MongoDB databases, each offering different levels of abstraction and flexibility. Here are some common methods along with code examples:

1. **Using the pymongo Library**:

pymongo is the official Python driver for MongoDB and provides a high-level interface for interacting with MongoDB databases.

Code:

*from pymongo import MongoClient*

*# Connect to MongoDB*

*client = MongoClient('mongodb://localhost:27017/')*

*# Access a specific database*

*db = client['mydatabase']*

*# Access a specific collection*

*collection = db['mycollection']*

*# Perform database operations*

*result = collection.find\_one({'key': 'value'})*

*print(result)*

**Using MongoDB Atlas and pymongo**:

If you are using MongoDB Atlas, you can connect to your Atlas cluster using the connection string provided in the Atlas dashboard.

Code:

*from pymongo import MongoClient*

*# Replace <password> with your MongoDB Atlas password*

*client = MongoClient('mongodb+srv://username:<password>@cluster0.mongodb.net/mydatabase?retryWrites=true&w=majority')*

*# Access a specific database*

*db = client['mydatabase']*

*# Access a specific collection*

*collection = db['mycollection']*

*# Perform database operations*

*result = collection.find\_one({'key': 'value'})*

*print(result)*

**Using Object-Document Mapper (ODM) Libraries**:

ODM libraries like mongoengine and pymodm provide higher-level abstractions over pymongo for defining document schemas and performing CRUD operations.

Example using mongoengine:

Code:

*from mongoengine import connect, Document, StringField*

*# Connect to MongoDB*

*connect('mydatabase')*

*# Define a document schema*

*class MyDocument(Document):*

*key = StringField(required=True)*

*# Perform database operations*

*result = MyDocument.objects(key='value').first()*

*print(result.to\_json())*

These methods provide flexible ways to connect to MongoDB databases using Python. Depending on your requirements and preferences, you can choose the most suitable method for your application. Each method allows you to perform CRUD operations and interact with MongoDB databases seamlessly from your Python codebase.

**==========================================================================Section 7: MongoDB CRUD Operations: Insert and Find Documents (10 points)**

1. **Explain the CRUD operations in MongoDB, focusing on the insertion and retrieval of documents. Provide examples to demonstrate each operation**.

A)

A CRUD operation in MongoDB describes a user interface that allows users to view, search, and modify data in a database.Modifications to MongoDB documents are performed by connecting to a server, querying the correct documents, and changing the settings before sending them back to the database. CRUD is a data-oriented, HTTP action verb-based process.

**CRUD stands for Create, Read, Update, and Delete**.

* **Create** - inserts new documents into the MongoDB database.

| **Method** |
| --- |
| **db.collection.updateOne()** |
| **db.collection.updateMany()** |
| **db.collection.replaceOne()** |

| **Method** |
| --- |
| **db.collection.find()** |

* **Read** operation is used to retrieve documents from a database.
* **Update** operation modifies existing documents in the database.

| **Method** |
| --- |
| **db.collection.insertOne()** |
| **db.collection.insertMany()** |
| **db.createCollection()** |

* **Delete** operation removes documents from a database.

| **Method** |
| --- |
| **db.collection.deleteOne()** |
| **db.collection.deleteMany()** |

**Insert Documents**: In MongoDB, the insert\_one() and insert\_many() methods are used to insert documents into collections.

Example of inserting a single document:

***# Define a list of documents***

***documents = [***

***{'name': ‘Anusha’, 'age': 26},***

***{'name': ‘Sam’, 'age': 33},***

***{'name': ‘Ema’, 'age': 40}***

***]***

***# Insert multiple documents***

***result = collection.insert\_many(documents)***

***print(result.inserted\_ids)***

**Find Documents with Query Conditions**: MongoDB allows you to specify query conditions to retrieve documents that match specific criteria

Example

*# Find documents with age greater than 30*

*result = collection.find({'age': {'$gt': 30}})*

*# Print documents found*

*for document in result:*

*print(document)*

**Find One Document**: find\_one() retrieves a single document that matches the specified query criteria.

*# Find the first document with name ‘Anusha’*

*result = collection.find\_one({'name': ‘Anusha’})*

*print(result)*

*==============================================================*

**Section 8: MongoDB CRUD Operations: Replace and Delete Documents (10 points)**

8. Discuss the replace and delete operations in MongoDB. How do these operations differ from traditional update and delete operations in relational databases? Can provide examples.

Replace:

The **replace** method **replaces a single document** in a collection. The replace\_one() method replaces a document’s entire contents, so fields not found in the new document will be lost.

| **Method** |
| --- |
| The **replace\_one()** |

*# Replace a document with a new one*

*result = collection.replace\_one({'name': ‘Sam’}, {'name': ‘Sam’, 'age': 28})*

*print(result.modified\_count)*

* **Delete** operation removes documents from a database.

| **Method** |
| --- |
| **db.collection.deleteOne()** |
| **db.collection.deleteMany()** |

***# Delete a single document***

***result = collection.delete\_one({'name': 'Emy'})***

***print(result.deleted\_count)***

***# Delete multiple documents***

***result = collection.delete\_many({'age': {'$gte': 40}})***

***print(result.deleted\_count)***

Difference from Relational Databases:

* Replace Operation: In relational databases, updates are typically performed using SQL's UPDATE statement, where specific fields are modified. Unlike MongoDB's replace operation, which completely replaces the document, relational databases offer more granular control over data modifications.
* Delete Operation: While both MongoDB and relational databases offer delete operations, relational databases often support cascading deletes, where related records are automatically deleted when the parent record is deleted. MongoDB requires explicit deletion of related documents.

**Section 9: MongoDB CRUD Operations: Modifying Query Results (10 points)**

9) Describe how to modify query results in MongoDB using projection, sorting, and limiting techniques. Can provide examples.

A) In MongoDB, you can modify query results using projection, sorting, and limiting techniques to refine the output based on specific requirements.

1. **Projection**:
   * Projection allows you to specify which fields to include or exclude from the query results.
   * You can include fields by specifying them with a value of 1 and exclude them with a value of 0.
2. **Sorting**:

* Sorting allows you to order query results based on one or more fields, either in ascending or descending order.
* You can use the sort() method and specify the field(s) to sort by along with the direction (ASCENDING or DESCENDING).

1. **Limiting**:

* Limiting allows you to restrict the number of documents returned by a query.
* You can use the limit() method and specify the maximum number of documents to return.

1. Combining Techniques: You can combine projection, sorting, and limiting techniques to fine-tune query results based on specific criteria.

Example

*from pymongo import MongoClient*

*# Connect to MongoDB*

*client = MongoClient('mongodb://localhost:27017/')*

*db = client['mydatabase']*

*collection = db['mycollection']*

*# Projection: Include 'name' and 'age', exclude '\_id'*

*projection = {'name': 1, 'age': 1, '\_id': 0}*

*# Sorting: Sort by 'age' in ascending order*

*sort\_criteria = [('age', 1)]*

*# Limiting: Limit to 3 documents*

*limit\_count = 3*

*# Execute query with projection, sorting, and limiting*

*result = collection.find({}, projection).sort(sort\_criteria).limit(limit\_count)*

*# Print modified query results*

*for document in result:*

*print(document)*

**Section 10: MongoDB CRUD Operations in Python (10 points)**

10. Write Python code to perform CRUD operations in MongoDB. Provide explanations for each operation and any challenges you encountered during implementation. Can provide examples.

A) Here is the code for CRUD operations.

*from pymongo import MongoClient # Import MongoClient from pymongo library*

*# Connect to MongoDB*

*client = MongoClient('mongodb://localhost:27017/') # Connect to MongoDB server running on localhost*

*# Access 'mydatabase' database*

*db = client['mydatabase'] # Access 'mydatabase' database within MongoDB*

*# Access 'mycollection' collection within 'mydatabase'*

*collection = db['mycollection'] # Access 'mycollection' collection within 'mydatabase'*

*# Create Operation: Insert multiple documents*

*documents = [ # Define a list of documents to insert*

*{'name': 'Anusha', 'age': 25}, # Document 1*

*{'name': 'Abhi', 'age': 30}, # Document 2*

*{'name': 'Sam', 'age': 35} # Document 3*

*]*

*collection.insert\_many(documents) # Insert multiple documents into the collection*

*print("Create operation - Documents inserted") # Print confirmation message*

*# Read Operation: Find documents*

*cursor = collection.find({'age': {'$gte': 30}}) # Find documents where 'age' is greater than or equal to 30*

*print("Read operation - Documents found:") # Print header for documents found*

*for document in cursor: # Iterate over documents found*

*print(document) # Print each document*

*# Update Operation: Update a document*

*query = {'name': 'Anusha'} # Define query to identify the document to update*

*new\_values = {'$set': {'age': 27}} # Define new values to set for the document*

*collection.update\_one(query, new\_values) # Update the document that matches the query*

*print("Update operation - Document updated") # Print confirmation message*

*# Read Operation after update*

*updated\_document = collection.find\_one({'name': 'Anusha'}) # Find the updated document*

*print("Read operation after update - Updated document:", updated\_document) # Print the updated document*

*# Delete Operation: Delete a document*

*delete\_query = {'name': 'Abhi'} # Define query to identify the document to delete*

*collection.delete\_one(delete\_query) # Delete the document that matches the query*

*print("Delete operation - Document deleted") # Print confirmation message*

*# Read Operation after delete*

*cursor = collection.find({'name': 'Abhi'}) # Attempt to find the deleted document*

*if cursor.count() == 0: # Check if no documents were found*

*print("Read operation after delete - Document not found") # Print message indicating document not found*

1. **Create Operation**: We insert multiple documents into the MongoDB collection using insert\_many() method.
2. **Read Operation**: We retrieve documents from the collection using find() method. We print out the documents found.
3. **Update Operation**: We update a document using update\_one() method. We specify the query to identify the document to update and the new values to set.
4. **Delete Operation**: We delete a document using delete\_one() method. We specify the query to identify the document to delete.
5. We perform read operations after update and delete operations to demonstrate the changes made.

11. Choose a real-world scenario where MongoDB could be beneficially applied. Describe the scenario and explain how MongoDB's features align with the requirements of the scenario. Can provide examples.

A) Consider a news website that publishes articles, blog posts, videos, and user comments. The CMS needs to efficiently manage a large volume of diverse content, support user authentication and authorization, enable content editors to perform CRUD operations, and provide real-time analytics for audience engagement.

How MongoDB's features align with the requirements:

1. Schema Flexibility: MongoDB's document-oriented data model allows the CMS to store content in a flexible schema format, accommodating various types of content such as articles, blog posts, and videos, each with different attributes and structures.
2. Scalability: MongoDB's horizontal scalability enables the CMS to handle increasing content volumes and user traffic by distributing data across multiple servers in a cluster. This ensures high availability and performance, even during peak periods of website traffic.
3. Indexing and Querying: MongoDB's indexing capabilities facilitate fast and efficient retrieval of content based on attributes like publication date, author, category, and keywords. This allows users to quickly search for relevant articles and videos, improving the overall user experience.
4. User Authentication and Authorization: MongoDB's support for role-based access control (RBAC) enables the CMS to authenticate users and enforce access permissions based on user roles and privileges. This ensures that only authorized users can create, edit, or delete content within the CMS.
5. Real-time Analytics: MongoDB's aggregation framework and real-time data processing capabilities enable the CMS to analyze user interactions, track article views, monitor audience engagement, and generate real-time insights to optimize content delivery and user engagement strategies.

Example: Suppose a content editor logs into the CMS dashboard to publish a breaking news article about a recent event. Using MongoDB, the editor can easily create a new document with relevant metadata such as the article title, author name, publication date, and article content. The CMS can then store the article as a MongoDB document and index it for fast retrieval. As users visit the news website, MongoDB tracks user interactions, such as page views, likes, and comments, allowing the CMS to generate real-time analytics reports and personalized content recommendations based on user interests and preferences.

In summary, MongoDB's flexible data model, scalability, indexing capabilities, user authentication, and real-time analytics features make it an ideal choice for building content management systems that require agility, performance, and scalability to manage and deliver diverse content to a global audience.