Experiment-1

Title: Introduction to NoSQL and MongoDB.

Outcome: Understanding of NoSQL is and how MongoDB stores and manages data.

Objective: To introduce the concept of NoSQL databases and provide a simple overview of MongoDB.

Theory:

NoSQL Database:

- **Introduction:**
  NoSQL stands for "Not Only SQL." It represents a broad class of database systems that are non-relational and do not use the traditional table-based structure. NoSQL databases are designed to store, retrieve, and manage unstructured or semi-structured data, especially in scenarios involving large volumes, high velocity, and high variety of data (Big Data).

  NoSQL databases became popular as companies like Facebook, Google, Amazon, and Twitter started dealing with massive data that relational databases couldn't handle efficiently in terms of scalability and performance.

- **Types of NoSQL Databases:**
  1. **Document-based:**
     Stores data as JSON-like documents (BSON in MongoDB). Each document is self-contained and can contain nested structures.
  2. **Key-Value Stores:**
     Data is stored in key-value pairs, where keys are unique and values can be anything (strings, blobs, etc.).
  3. **Column-based Stores:**
     Data in columns rather than rows. Efficient for queries on large datasets with lots of columns.
  4. **Graph-based:**
     Uses graph structures with nodes, edges, and properties to represent data and relationships.

- **Features of NoSQL:**
  - **Non-relational:**
    Unlike SQL databases, NoSQL databases don't rely on the relational model with tables, rows, and columns. They offer flexible data structures, such as key-value pairs, documents, graphs, or column-family stores.
  - **Flexible Schema:**
    NoSQL databases typically have either a schema-free or relaxed schema, allowing for dynamic and evolving data models without requiring pre-defined schemas. This makes them ideal for handling data with varying structures.

- **Horizontal Scalability:**
  NoSQL databases are designed to scale horizontally by adding more nodes to the database cluster, allowing them to handle large volumes of data and high traffic without downtime.
- **High Performance and Low Latency:**
  NoSQL databases are optimized for high performance and low latency, especially for applications that require real-time data access and processing.
- **Support for Diverse Data Models:**
  NoSQL encompasses a variety of data models, including document stores, key-value stores, graph databases, and column-family stores, each suited for specific types of data and use cases.
- **Distributed Architecture:**
  Many NoSQL databases are built on a distributed architecture, enabling them to handle large amounts of data across multiple machines and locations.
- **BASE Properties:**
  Instead of ACID properties, NoSQL databases often follow BASE (Basically Available, Soft state, Eventual consistency) principles, which prioritize availability and eventual consistency over strict transactional consistency, particularly in distributed environments.
- **Agile Development:**
  NoSQL databases are well-suited for agile development environments, as their flexible schemas and scalability allow for rapid changes and iterations.
- **Cost-Effectiveness:**
  Horizontal scalability often makes NoSQL databases more cost-effective than traditional databases when dealing with large datasets.

- **Advantages of NoSQL:**
  - Structured Data Management: SQL databases excel at managing structured data with well-defined schemas. They enforce data integrity through constraints such as primary keys, foreign keys, and data types, ensuring consistency and reliability.
  - ACID Compliance: SQL databases typically adhere to the ACID (Atomicity, Consistency, Isolation, Durability) properties, guaranteeing transactional consistency and reliability. This makes them suitable for applications that require strict data consistency, such as financial systems and e-commerce platforms.
  - Mature Ecosystem: SQL databases have a mature ecosystem with robust tools, documentation, and community support. Developers have access to a wide range of frameworks, libraries, and resources for building and maintaining SQL-based applications.
  - Standardized Language: SQL is a standardized language recognized by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). This standardization ensures portability and interoperability across different database systems,

allowing developers to work with various SQL databases with minimal effort.

- Strong Data Integrity: SQL databases provide mechanisms for enforcing data integrity, such as constraints, triggers, and stored procedures. These features help maintain the quality and consistency of the data, reducing the risk of errors and inconsistencies.

- **Difference Between NoSQL and SQL:**

Feature	SQL (Relational)	NoSQL (Non-Relational)
Data Model	Table-based (Rows & Columns)	Document, Key-Value, Column, Graph
Schema	Fixed schema	Dynamic schema
Scalability	Vertical (adding more resources to one server)	Horizontal (adding more servers)
ACID Compliance	Strong support	Eventual consistency, BASE model
Complex Queries	Excellent support using JOINs	Limited JOIN support, but optimized for fast access
Examples	MySQL, PostgreSQL, Oracle	MongoDB, Cassandra, Redis, Neo4j

- **MongoDB:**
  - **Introduction**
    MongoDB is a document-oriented NoSQL database that uses BSON (Binary JSON) to store data. It is open-source, scalable, and offers high performance. MongoDB is widely used in modern web applications, especially those using Node.js and Express.js in the MEAN/MERN stack.
  - **Key Components:**
    Database – A physical container for collections.
    Collection – Similar to a table in SQL; stores documents**.**
    Document – A single record in BSON format.
  - **MongoDB Features:**
    Flexible document model.
    Powerful query language and indexing.
    Aggregation framework.
    Replication and sharding for high availability and scalability.
    Integrated support for geospatial data.

Viva Questions:

- **What is NoSQL? Why is it needed?**
- **List different types of NoSQL databases.**
- **What is the difference between SQL and NoSQL?**

Experiment - 2

Title: Installation of MongoDB Server.

Outcome:

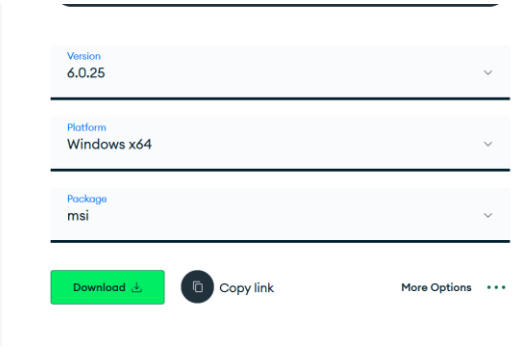
- Successfully installed MongoDB on the desired operating system (Windows, Linux, or macOS).
- Configured MongoDB to start its **server (mongod)** and connect through its **client shell (mongosh)**.
- Understood how to create the default **data directory** for MongoDB to store its database files.

Objective:

The objective is to understand how to install MongoDB, configure MongoDB, start the MongoDB server, connect to it via the MongoDB shell, and perform basic database operations.

Theory:

- **Introduction to MongoDB**
  - MongoDB is an open-source, document-oriented NoSQL database that uses a flexible, JSON-like format called BSON (Binary JSON).
  - It is designed to handle large volumes of structured, semi-structured, and unstructured data.
  - MongoDB is widely used in modern applications such as web development, cloud-based applications, and big data analytics.
- **System Requirements**
  - Before installation, ensure your system meets these requirements:
    - **Windows**: Windows 7 or later (64-bit).
    - **Linux**: RHEL, Ubuntu, CentOS, SUSE (64-bit).
    - **macOS**: Version 10.13 or later.
    - Minimum **2 GB RAM**, though **4 GB+** is recommended.
    - Disk space depending on data size
- **Installation Steps**
  - **Installation on Windows**
  - Go to the official MongoDB website: https://www.mongodb.com/try/download/community.



- Select your operating system and download the **MSI installer**.
- Run the installer and choose:
  - **Complete Setup** (recommended for beginners).
  - Install MongoDB Compass (GUI for MongoDB).
- After installation, MongoDB is installed at:
  - C:\Program Files\MongoDB\Server\<version>\
- Add the **bin** folder path to the system **Environment Variables** (so that mongod and mongo commands can be run from anywhere).
- Create a data directory for MongoDB:
  - C:\data\db
- Run the MongoDB server:
  - **mongod**
- Open a new command prompt and start the MongoDB shell:
  - **Mongo**

Viva Questions

1. What is **MongoDB** and how is it different from traditional **RDBMS**?
2. Why do we need to create a **data\db** folder in Windows for MongoDB?
3. What is the difference between **mongod** and **mongo/mongosh** commands?

Experiment 3

Title:

To implement and study **CRUD (Create, Read, Update, Delete)** operations in MongoDB, a NoSQL document oriented database, using Mongo Shell or MongoDB Compass.

Objectives:

1. To understand how MongoDB manages data using collections and documents.
2. To learn CRUD operations for inserting, retrieving, modifying, and deleting data.
3. To differentiate between relational (SQL) and non-relational (NoSQL) database operations.
4. To explore the flexibility of schema-less databases.
5. To gain practical skills for real-world database-driven applications.

Outcome:

- Create and manage a MongoDB database and collections.
- Perform insertions of single and multiple documents.
- Retrieve data using queries and projections.
- Update documents selectively or in bulk.
- Delete documents with different conditions.

Theory:

Introduction to MongoDB

MongoDB is a **NoSQL, open-source, cross-platform, document-oriented database** that uses a **flexible schema**. Unlike traditional relational databases (like MySQL or Oracle) that use rows and tables, MongoDB stores data as **documents** inside **collections**. Each document is represented in a format similar to JSON, called **BSON (Binary JSON),** which allows storage of complex data types, arrays, and nested objects.

CRUD in MongoDB

CRUD operations represent the four basic interactions with persistent storage:

1. Create (Insert Data)
   - Adds new documents into a collection.
   - MongoDB provides insertOne() for a single document and insertMany() for multiple documents.
   - Example: db.students.insertOne({ name: "Rahul", age: 20, course: "B.Tech" });
   - db.students.insertMany([
   - { name: "Priya", age: 21, course: "MBA" },
   - { name: "Amit", age: 22, course: "BCA" }]);
2. Read (Query Data)
   - Retrieves documents from a collection using queries and filters.
   - MongoDB provides find() and findOne().

- Example:
- db.students.find();
- db.students.find({ age: { $gt: 20 } });
- db.students.find({}, { name: 1, course: 1 }); // Projection

3. Update (Modify Data)
   - Modifies existing documents.
   - Operations include updateOne(), updateMany(), and replaceOne().
   - Example:
   - db.students.updateOne({ name: "Rahul" }, { $set: { age: 21 } });
   - db.students.updateMany({ course: "B.Tech" }, { $set: { course: "B.E" } });
4. Delete (Remove Data)
   - Removes one or more documents from a collection.
   - Operations include deleteOne() and deleteMany().
   - Example:
   - db.students.deleteOne({ name: "Amit" });
   - db.students.deleteMany({ course: "MBA" });

Advantages of CRUD in MongoDB

- **Schema-less structure:** No fixed schema required.
- **High flexibility:** Can store complex, nested data directly.
- **Scalability:** Easily scales horizontally using sharding.
- **Performance:** Fast read and write operations compared to traditional SQL in many use cases.
- **JSON-like format:** Easy integration with programming languages like JavaScript, Python, and Node.js.

Thus, CRUD operations form the foundation of MongoDB applications, enabling developers to perform essential data handling in real-time systems such as e-commerce platforms, social media apps, and content management systems.

```
>_MONGOSH

> use Demodb
< switched to db Demodb
> db.students.insertOne({
    name: "Harry",
    age: 22,
    course: "Computer Science"
  })
< {
    acknowledged: true,
    insertedId: ObjectId('68bf26bfd3cc3d23ceb0c135')
  }
> db.students.insertMany([
    { name: "Ron", age: 23, course: "IT" },
    { name: "Joe", age: 21, course: "Electronics" },
    { name: "luis", age: 24, course: "Mechanical" }
  ])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('68bf26f3d3cc3d23ceb0c136'),
      '1': ObjectId('68bf26f3d3cc3d23ceb0c137'),
      '2': ObjectId('68bf26f3d3cc3d23ceb0c138')
    }
  }
Demodb >
```

```
>_MONGOSH
> db.students.find()
< {
    _id: ObjectId('68bf26bfd3cc3d23ceb0c135'),
    name: 'Harry',
    age: 22,
    course: 'Computer Science'
  }
  {
    _id: ObjectId('68bf26f3d3cc3d23ceb0c136'),
    name: 'Ron',
    age: 23,
    course: 'IT'
  }
  {
    _id: ObjectId('68bf26f3d3cc3d23ceb0c137'),
    name: 'Joe',
    age: 21,
    course: 'Electronics'
  }
  {
    _id: ObjectId('68bf26f3d3cc3d23ceb0c138'),
    name: 'luis',
    age: 24,
    course: 'Mechanical'
  }
```

```
>_MONGOSH
> db.students.find({age: {$gt: 22}})
< {
    _id: ObjectId('68bf26f3d3cc3d23ceb0c136'),
    name: 'Ron',
    age: 23,
    course: 'IT'
  }
  {
    _id: ObjectId('68bf26f3d3cc3d23ceb0c138'),
    name: 'luis',
    age: 24,
    course: 'Mechanical'
  }
> db.students.find({}, {name: 1, course: 1, _id: 0})
< {
    name: 'Harry',
    course: 'Computer Science'
  }
  {
    name: 'Ron',
    course: 'IT'
  }
  {
    name: 'Joe',
    course: 'Electronics'
  }
```

```
>_MONGOSH
> db.students.updateOne(
    { name: "Harry" },
    { $set: { course: "Data Science" } } )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
> db.students.updateMany(
    { age: { $gt: 22 } },
    { $set: { course: "Engineering" } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 2,
    modifiedCount: 2,
    upsertedCount: 0
  }
> db.students.find()
< {
    _id: ObjectId('68bf26bfd3cc3d23ceb0c135'),
    name: 'Harry',
    age: 22,
    course: 'Data Science'
  }
  {
```

```
>_MONGOSH
    age: 24,
    course: 'Engineering'
  }
> db.students.deleteOne({ name: "luis" })
< {
    acknowledged: true,
    deletedCount: 1
  }
> db.students.find({name: "luis"})
<
> db.students.deleteMany({ course: "Engineering" })
< {
    acknowledged: true,
    deletedCount: 1
  }
> db.students.find()
< {
    _id: ObjectId('68bf26bfd3cc3d23ceb0c135'),
    name: 'Harry',
    age: 22,
    course: 'Data Science'
  }
  {
    _id: ObjectId('68bf26f3d3cc3d23ceb0c137'),
    name: 'Joe',
    age: 21,
    course: 'Electronics'
  }
Demodb >
```

Viva Questions

1. What is MongoDB, and how does it differ from relational databases?
2. Explain CRUD operations in MongoDB with examples.
3. What is the purpose of the _id field in a MongoDB document?
4. Difference between updateOne() and updateMany() in MongoDB.
5. Why is MongoDB considered schema-less, and what are its advantages?

EXPERIMENT 4

Title:

Implementation of Comparison Query Operators in MongoDB

Objective:

1. To understand and implement comparison operators in MongoDB.
2. To retrieve documents based on conditions using $eq, $ne, $gt, $lt, $gte, and $lte.
3. To differentiate between various operators through practical examples.

Outcome:

1. Successfully implemented queries using comparison operators.
2. Retrieved filtered results from the collection based on conditions.
3. Understood how operators help in flexible and efficient querying.

Theory:

MongoDB is a NoSQL database that stores data in BSON (Binary JSON) format. Querying in MongoDB uses operators to filter data.

Comparison Query Operators are used to compare field values with specified conditions:

- **$eq** → Matches values equal to a specified value.
- **$ne** → Matches values not equal to a specified value.
- **$gt** → Matches values greater than a specified value.
- **$lt** → Matches values less than a specified value.
- **$gte** → Matches values greater than or equal to a specified value.
- **$lte** → Matches values less than or equal to a specified value.

These operators are typically used in find() queries to filter documents.

Steps:

Step 1: Create a collection

```
db.students.insertMany([
    { name: "Alice", age: 21, marks: 82 },
    { name: "Bob", age: 23, marks: 75 },
    { name: "Charlie", age: 20, marks: 90 },
    { name: "David", age: 22, marks: 60 },
    { name: "Eva", age: 21, marks: 95 }
])
```

Step 2: Apply comparison operators

- **Equal to ($eq)**

```
db.students.find({ age: { $eq: 21 } })
```

- **Not equal to ($ne)**

```
db.students.find({ marks: { $ne: 75 } })
```

- **Greater than ($gt)**

```
db.students.find({ marks: { $gt: 80 } })
```

- **Less than ($lt)**

```
db.students.find({ age: { $lt: 22 } })
```

- **Greater than or equal to ($gte)**

```
db.students.find({ marks: { $gte: 90 } })
```

- **Less than or equal to ($lte)**

```
db.students.find({ marks: { $lte: 75 } })
```

Result:

- Documents are filtered successfully based on the applied comparison operators.
- Example: Query with { marks: { $gt: 80 } } retrieved Alice, Charlie, and Eva.
- This confirms that MongoDB's comparison operators function similarly to traditional relational operators but work on JSON-like documents.

```
>_MONGOSH
> db.students.insertMany([
  { name: "Aditya", age: 21, marks: 82 },
  { name: "Abeer", age: 20, marks: 75 },
  { name: "Akshat", age: 22, marks: 90 },
  { name: "Aakash", age: 20, marks: 60 },
  { name: "Ansh", age: 23, marks: 95 }
  ])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('68bf2b2cd3cc3d23ceb0c139'),
      '1': ObjectId('68bf2b2cd3cc3d23ceb0c13a'),
      '2': ObjectId('68bf2b2cd3cc3d23ceb0c13b'),
      '3': ObjectId('68bf2b2cd3cc3d23ceb0c13c'),
      '4': ObjectId('68bf2b2cd3cc3d23ceb0c13d')
    }
  }
> db.students.find({ age: { $eq: 21 } })
< {
    _id: ObjectId('68bf26f3d3cc3d23ceb0c137'),
    name: 'Joe',
    age: 21,
    course: 'Electronics'
  }
  {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c139'),
    name: 'Aditya',
    age: 21,
    marks: 82
```

```
>_MONGOSH
  }
> db.students.find({ marks: { $ne: 75 } })
< {
    _id: ObjectId('68bf26bfd3cc3d23ceb0c135'),
    name: 'Harry',
    age: 22,
    course: 'Data Science'
  }
  {
    _id: ObjectId('68bf26f3d3cc3d23ceb0c137'),
    name: 'Joe',
    age: 21,
    course: 'Electronics'
  }
  {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c139'),
    name: 'Aditya',
    age: 21,
    marks: 82
  }
  {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c13b'),
    name: 'Akshat',
    age: 22,
    marks: 90
  }
```

```
> db.students.find({ marks: { $gt: 80 } })
< {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c139'),
    name: 'Aditya',
    age: 21,
    marks: 82
  }
  {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c13b'),
    name: 'Akshat',
    age: 22,
    marks: 90
  }
  {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c13d'),
    name: 'Ansh',
    age: 23,
    marks: 95
  }
Demodb >
```

```
> db.students.find({ age: { $lt: 22 } })
< {
    _id: ObjectId('68bf26f3d3cc3d23ceb0c137'),
    name: 'Joe',
    age: 21,
    course: 'Electronics'
  }
  {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c139'),
    name: 'Aditya',
    age: 21,
    marks: 82
  }
  {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c13a'),
    name: 'Abeer',
    age: 20,
    marks: 75
  }
  {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c13c'),
    name: 'Aakash',
    age: 20,
    marks: 60
  }
```

```
> db.students.find({ marks: { $gte: 90 } })
< {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c13b'),
    name: 'Akshat',
    age: 22,
    marks: 90
  }
  {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c13d'),
    name: 'Ansh',
    age: 23,
    marks: 95
  }
> db.students.find({ marks: { $lte: 75 } })
< {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c13a'),
    name: 'Abeer',
    age: 20,
    marks: 75
  }
  {
    _id: ObjectId('68bf2b2cd3cc3d23ceb0c13c'),
    name: 'Aakash',
    age: 20,
    marks: 60
  }
Demodb >
```

Viva Questions:

1. What is the purpose of comparison operators in MongoDB?
2. Differentiate between $gt and $gte with an example.
3. Can $ne be combined with other operators? Explain with an example.
4. How is querying in MongoDB different from SQL WHERE clauses?
5. What data types can MongoDB comparison operators be applied to?