Department of Computer Science & Information Systems

Advanced Database Systems (CS G516)

Objectives:

1. To understand the fundamental concepts of NoSQL databases and their necessity.
2. To differentiate between Relational (SQL) and Non-Relational (NoSQL) systems.
3. To learn the installation and basic configuration of MongoDB.
4. To learn CRUD operations and commands in the MongoDB shell.

---

1. Introduction to NoSQL

1.1 What is NoSQL and Why is it Needed?

NoSQL (often interpreted as **“Not Only SQL”**) represents a broad class of database management systems that differ from traditional Relational Database Management Systems (RDBMS). Unlike RDBMS, which rely on **fixed schemas**, **tables**, and **Structured Query Language (SQL)**, NoSQL databases are designed to store and retrieve data in formats other than tabular relations.

NoSQL databases support **flexible or schema-less data models**, allowing data to be stored as **documents, key-value pairs, column families, or graphs**. This flexibility makes NoSQL suitable for handling **semi-structured and unstructured data**, such as JSON documents, logs, multimedia data, and real-time sensor data.

The need for NoSQL databases arose due to the limitations of relational databases when dealing with **big data**, **high user concurrency**, and **rapidly changing data structures**. Modern applications such as social media platforms, e-commerce systems, and real-time analytics require databases that can **scale horizontally**, provide **high availability**, and deliver **fast read/write performance** across distributed systems.

1.2 Difference Between SQL and NoSQL

The following table highlights the key distinctions:

Feature	**SQL (Relational)**	**NoSQL (Non-Relational)**
Data Structure	Table-based (Rows & Columns)	Key-value, Document, Graph, or Column-family
Schema	Pre-defined, rigid schema	Dynamic, flexible schema
Scalability	Vertical (Scale-up)	Horizontal (Scale-out)
Query Language	Structured Query Language (SQL)	Focused on collection/documents; varies by DB
Transactions	ACID (Atomicity, Consistency, Isolation, Durability)	Generally follows BASE (Basically Available, Soft state, Eventual consistency)

1.3 What Problems Does It Solve?

NoSQL addresses several bottlenecks inherent in traditional systems:

- **Handling Unstructured Data:** NoSQL databases natively handle semi-structured and unstructured data such as **JSON, XML, log files, and binary data** without forcing them into rigid tables, reducing the need for complex schema design.

- **High Velocity:** NoSQL systems support **high write and read throughput**, making them ideal for **write-heavy and real-time applications** such as **IoT sensor streams, social media feeds, and event logging systems**, where data is generated continuously at high speed.

- **Agile Development:** Developers can store application objects directly in the database without complex **Object-Relational Mapping (ORM)** layers. This reduces the mismatch between application code and database structure, **simplifies data modeling**, and significantly **speeds up development and iteration cycles**.

- **Horizontal Scalability & Availability:** NoSQL databases are designed to **scale horizontally** by distributing data across multiple nodes. This allows systems to handle growing workloads efficiently while maintaining **high availability and fault tolerance**.

1.4 Types of NoSQL Databases

There are four primary categories of NoSQL databases, each optimized for different types of data and application requirements:

- **Document Stores:** Store data as JSON-like documents, where each document can have a flexible structure. This model closely matches how data is represented in modern applications, making it easy to store complex objects. Document stores are commonly used in **content management systems, user profiles, and e-commerce applications** *(e.g., **MongoDB**, CouchDB).*

- **Key-Value Stores:** Store data as simple key-value pairs, similar to a dictionary or hash map. They offer **very fast read and write operations** but limited querying capability. These databases are ideal for **caching, session management, and real-time data access** *(e.g., Redis, DynamoDB).*

- **Column-Oriented Databases:** Store data in columns rather than rows, allowing efficient access to large volumes of related data. This design is highly optimized for **analytical queries, data warehousing, and time-series data** where operations are performed on specific columns *(e.g., Cassandra, HBase).*

- **Graph Databases:** Store data as nodes (entities) and edges (relationships), making relationship traversal fast and intuitive. Graph databases are well suited for **social networks, recommendation engines, and network topology analysis** *(e.g., Neo4j).*

To read more on NoSQL :-
https://www.mongodb.com/resources/basics/databases/nosql-explained

---

2. Introduction to MongoDB

2.1 What is MongoDB?

MongoDB is a source-available, cross-platform, document-oriented database management system. It is classified as a NoSQL database and stores data in **JSON-like documents** with **optional (flexible) schemas**, allowing different documents in the same collection to have different structures.

Working Principle

Instead of storing data in rows and columns as in relational databases, MongoDB stores data in **BSON (Binary JSON)** documents. BSON is a binary-encoded format that extends JSON by supporting additional data types such as dates and binary data, enabling efficient storage and faster data processing.

These BSON documents are grouped into **collections**, which are analogous to tables in relational databases. Since MongoDB does not enforce a fixed schema, fields can vary from one document to another, and the data model can evolve over time without requiring costly schema migrations or downtime.

MongoDB also supports **indexing, replication, and sharding**, which together provide fast query performance, data redundancy, and horizontal scalability across multiple servers.

Real-Life Usages

- **Content Management Systems (CMS):** Efficiently stores diverse and evolving content types such as articles, images, and metadata.
- **Product Catalogs:** Used by e-commerce platforms where different products may have different attributes like size, color, or specifications.
- **Real-Time Analytics:** Handles high-velocity data such as logs, clickstreams, and sensor data for near real-time analysis.

To read more on mongoDB visit :- https://www.mongodb.com/docs/

2.2 Terminology in MongoDB

It is helpful to map MongoDB terms to the SQL terms we already know:

SQL Term	MongoDB Term	Description
Database	**Database**	Container for collections.
Table	**Collection**	A grouping of documents.
Row	**Document**	A single record, stored as BSON.
Column	**Field**	A key-value pair within a document.
Join	**Embedding/Linking**	Documents can contain sub-documents (Embedding) or references (Linking).

2.3 Steps to Download and Install

- **Download:** Visit the MongoDB Community Server Download center. Select the version appropriate for your OS (e.g., Windows x64) and download the MSI package.
- **Install:** Run th
- e installer.
    - Choose "Complete" setup.
    - **Important:** Select "Install MongoDB as a Service" to have it run automatically.
    - **Environment Variable:** Add the MongoDB bin path (usually C:\Program Files\MongoDB\Server\X.X\bin) to your system's PATH environment variable so you can run commands globally.
    - **Verification:** Open your terminal or command prompt and type:

    ```
    mongod --version
    ```

    This should display the installed version details.

- Go to **MongoDB Shell Download**
    - Use this Link: https://www.mongodb.com/try/download/shell

- Download **Windows x64 MSI**
- Install → keep default options
- Reopen Command Prompt
- **Verification:** Open your terminal or command prompt and type:

  ```
  mongosh --version
  ```

  This should open MongoDB Shell without any error

- Go to **MongoDB Command Line Tools**
  - Use this Link: https://www.mongodb.com/try/download/database-tools
  - Download **Windows x64 MSI**
  - Install → keep default options
  - **Environment Variable:** Add the MongoDB bin path (usually C:\Program Files\MongoDB\Tools\100\bin) to your system's PATH environment variable so you can run commands globally.
  - Reopen Command Prompt

---

3. MongoDB Queries and Operations

3.1 Basic Run Commands

The following commands allow us to switch between databases in MongoDB and perform operations on collections stored within a database:

1. **List Databases:**

   The following command will list the all available databases in current MongoDB server that we are working:
   ```
   show dbs
   ```

2. **Display the database you are using:**

   The following command will display the current database you are working with:
   ```
   db
   ```

3. **Create new database or use existing database:**

   The following command will create new database if no database with given name is found and will use the old database if there is an existing database with the given name.

```
use <database>
```

The above given command will create a new database with name <database> if not already present and will start using it any subsequent query will target this database. If a database with the name <database> is already present it will simply use this and any subsequent query will target this database.

4. **Create new collection:**

   The following command will create the collection inside the specified database:

   ```
   <database>.createCollection(“<collection_name>”)
   ```

   The above command will create a new collection inside the <database> if we use db at the place of <database> we can create a collection at the current selected database. <collection_name> can be any name without any space and must start with character letters, numbers or underscore(_).

   To read more visit :- https://www.mongodb.com/docs/mongodb-shell/run-commands/

3.2 CRUD Operation : Insert Document Operations

The following queries allow us to create and insert the document inside the collection. Based on the command used we can add one or more documents simultaneously:

1. **Insert a single document**

   The following query adds one document to the specified collection with the provided key–value data.

   ```
   <database>.<collection_name>.insert_one({
           key1 : value1,
           key2 : value2,
           .
           .
           keyN : valueN
   })
   ```

   The above command inserts all the specified key–value pairs as a single document into the selected collection of the specified database. We can use this query to create the collection inside the database. Also remember key names are case sensitive. Following is an example of this query:

```
db.student.insert_one({
        name : ”Kalp”,
        subjects : [“COA”, “CN”],
        age : 22
})
```

2. **Insert multiple documents**

The following query adds multiple documents to the specified collection at once. Each document is defined using key–value pairs and is enclosed within an array:

```
<database>.<collection_name>.insertMany([
  {
    key1 : value1,
    key2 : value2,
    ...
  },
  {
    key1 : value1,
    key2 : value2,
    ...
  }
])
```

The above command inserts all the provided documents into the selected collection of the specified database. Each object inside the array represents one separate document. If the collection does not already exist, MongoDB will automatically create the collection during insertion. As with single insertion, key names are case-sensitive.

This method is useful for bulk data insertion, as it reduces the number of database operations and improves performance.

```
db.student.insertMany([

  {

    name : "Kalp",

    subjects : ["COA", "CN"],
  age : 22
 },
 {
  name : "Amit",
  subjects : ["DBMS", "OS"],
  age : 21
 },
 {
  name : "Neha",
  subjects : ["CN", "AI"],
  age : 23
 }
])
```

To read more visit :- https://www.mongodb.com/docs/mongodb-shell/crud/insert/

3.3 CRUD Operation : Read Document Operations

The following queries allow us to read the document inside the collection. Based on the conditions used in the query used we can read one or group of documents:

1. **find() Query:**

   The find() query is used to retrieve documents from a specified collection.
   It returns a cursor pointing to all matching documents:

   ```
   <database>.<collection_name>.find({ query })
   ```

   The above command fetches all documents that satisfy the given condition.
   If no condition is specified, all documents in the collection are returned.

```
db.student.find()
```

2. **findOne() Query:**

   The findOne() query is used to retrieve only one matching document from a collection:

   ```
   <database>.<collection_name>.findOne({ query })
   ```

   This command returns the first matching document found.
   If multiple documents satisfy the condition, only one document is returned.

   ```
   db.student.findOne({ name: "Kalp" })
   ```

3. **Filters in Query:**

   Filters are used to limit the documents returned by a query based on conditions:

   ```
   db.student.find({ age: 22 })
   ```

   The above filter returns documents where age is 22.

4. **Logical Operators($and, $or)**

   AND:

   Logical operators are used to combine multiple conditions.

   ```
   db.student.find({
           $and: [
              { age: 22 },
              { name: "Kalp" }
           ]
   })
   ```

   Returns documents that satisfy all conditions.

   OR:

   ```
           db.student.find({
             $or: [
                   { age: 22 },
                   { age: 21 }
             ]
   ```

```
        })
```

Returns documents that satisfy at least one condition.

5. **Relational Operators**

   Relational operators are used to compare field values.

Operator	Meaning
$gt	Greater than
$lt	Less than
$gte	Greater than or equal
$lte	Less than or equal
$ne	Not equal

```
db.student.find({ age: { $gt: 21 } })
```

6. **Projection**

   Projection is used to display only selected fields from documents.

```
db.student.find(
        { age: 22 },
        { name: 1, age: 1, _id: 0 }
)
```

1 → include field
0 → exclude field

7. **Querying null values**

   To find documents where a field has a null value:

```
db.student.find({ marks: null })
```

To find documents where the field does not exist:

```
db.student.find({ marks: { $exists: false } })
```

8. **Querying array fields**

   MongoDB allows querying directly on array fields.

   Match a Value Inside an Array

```
db.student.find({ subjects: "CN" })
```

   Returns documents where the subjects array contains “CN”.

9. **Match multiple values in array**

```
db.student.find({
        subjects: { $all: ["CN", "COA"] }
})
```

   Returns documents where the array contains both values.

   To read more visit :- https://www.mongodb.com/docs/mongodb-shell/crud/read/

3.4 CRUD Operation : Update Document Operations

The following queries allow us to modify existing documents inside a collection. Based on the command used, we can update one document, multiple documents, or completely replace a document.

1. **Update single document:**

   The following query updates only one document that matches the specified condition. By default, MongoDB requires update operators such as $set to modify fields.

```
<database>.<collection_name>.updateOne(
  { condition },
  { $set: { key1 : value1, key2 : value2 } }
)
```

The above command updates the first matching document found in the collection.
Only the specified fields are modified, while the remaining fields remain unchanged.

```
db.student.updateOne(
  { name: "Kalp" },
  { $set: { age: 23 } }
)
```

2. **Update multiple document:**

   The following query updates all documents that satisfy the given condition.

```
<database>.<collection_name>.updateMany(
  { condition },
  { $set: { key1 : value1 } }
)
```

   The above command modifies multiple documents simultaneously.
   This method is useful when the same update needs to be applied to many documents.

```
db.student.updateMany(
  { subjects: "CN" },
  { $set: { department: "Computer" } }
)
```

3. **Replace document:**

   The replaceOne() query is used to completely replace an existing document with a new document.

```
<database>.<collection_name>.replaceOne(
  { condition },
  { new_document }
)
```

   The above command removes the existing document and replaces it entirely with the new document provided.
   All previous fields are removed except the `_id` field, which is preserved.

```
db.student.replaceOne(
  { name: "Amit" },
  {
    name: "Amit",
    subjects: ["DBMS", "CN"],
    age: 22,
    semester: 6
  }
)
```

To read more visit :- https://www.mongodb.com/docs/mongodb-shell/crud/update/

3.5 CRUD Operation : Delete Document Operations

The following queries allow us to remove documents from a collection. Based on the command used, we can delete a single document, multiple documents, or all documents from a collection.

1. **Delete single document:**

   The following query deletes only one document that matches the specified condition.

```
<database>.<collection_name>.deleteOne(
  { condition }
)
```

   The above command removes the first matching document found in the collection.
   If no document matches the condition, no deletion is performed.

```
db.student.deleteOne({ name: "Neha" })
```

2. **Delete multiple documents:**

   The following query deletes all documents that satisfy the given condition.

```
<database>.<collection_name>.deleteMany(
  { condition }
)
```

   The above command removes multiple documents simultaneously from the collection.
   This method is useful when a group of documents needs to be deleted at once.

```
db.student.deleteMany({ age: { $lt: 22 } })
```

3. **Delete all documents from collection:**

   To remove all documents from a collection without deleting the collection itself, use an empty condition.

```
<database>.<collection_name>.deleteMany({})
```

```
db.student.deleteMany({})
```

To read more visit :- https://www.mongodb.com/docs/mongodb-shell/crud/delete/

3.6 Aggregation Pipelines (Joins)

Aggregation in MongoDB is used to process and transform documents from a collection to produce summarized and computed results.

It is commonly used for grouping, filtering, sorting, and calculating values from multiple documents.

Aggregation works using a pipeline, where each stage performs an operation on the documents and passes the result to the next stage.

```
db.<collection_name>.aggregate([

  { <stage1> },

  { <stage2> },

  ...

])
```

Each stage begins with a $ symbol and modifies the documents flowing through the pipeline.

Common Uses of Aggregation

- Grouping documents by a field
- Calculating averages, sums, or counts
- Filtering documents based on conditions

- Sorting aggregated results

Example: Student Collection

Consider the student collection containing the following fields:

- name
- age
- subjects
- Marks

Aggregation Example: Average Age per Subject

```
db.student.aggregate([
 { $unwind: "$subjects" },
 { $group:
   {
     _id: "$subjects",
     averageAge: { $avg: "$age" }
   }
 },
 { $sort: { averageAge: -1 } }
])
```

Explanation of Stages

- $unwind: Converts each element of the subjects array into a separate document
- $group: Groups documents by subject and calculates the average age
- $sort: Sorts the results in descending order

Output : The result displays each subject along with the average age of students enrolled in that subject.

To read more visit :- *https://www.mongodb.com/docs/mongodb-shell/run-agg-pipelines/*

To learn MongoDB shell from download to executing complex queries refer to this :- https://www.mongodb.com/docs/mongodb-shell/

3.6 Importing Data from CSV into MongoDB

MongoDB provides a utility called mongoimport that allows users to import data from CSV files directly into a collection. This method is useful when working with large datasets or data exported from spreadsheet applications.

```
mongoimport --type csv --db <database_name> --collection <collection_name> --headerline
--file <file_path>
```

Explanation of Parameters

- --type csv : Specifies the file type as CSV
- --db : Target database name
- --collection : Target collection name
- --headerline : Uses the first row of the CSV file as field names
- --file : Path to the CSV file

```
mongoimport --type csv --db collegeDB --collection student --headerline --file students.csv
```

The above command imports all records from students.csv into the student collection of the collegeDB database.

If the collection does not exist, MongoDB creates it automatically.

To learn more :- https://www.mongodb.com/docs/database-tools/mongoimport/

3.7 Connecting to remote MongoDB Database

MongoDB allows users to connect to a remote database server using a connection string (URI). This is commonly used when working with cloud databases or MongoDB installed on another machine.

To connect to a remote MongoDB database, use the following command:

```
mongosh "mongodb://<username>:<password>@<host>:<port>/<database_name>"
```

Example:-

```
mongosh "mongodb://student:student123@192.168.1.100:27017/collegeDB"
```

To learn more :- https://www.mongodb.com/docs/mongodb-shell/connect/

---

4. Exercise

4.1 Setting up database environment

Follow the below given steps and setup database accordingly to practice queries given in **4.2.** Use hints given and also use the materials discussed in above sections.

1. Download Formula 1 dataset from given link :- *https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020*
2. Extract following CSV files from the downloaded zip file :-
    a. drivers.csv
    b. circuits.csv
    c. races.csv
    d. results.csv
3. Now import the data from this CSVs into the MongoDB (**hint :-** Use mongoimport with appropriate parameters)
4. Check that data is perfectly inserted into the database.
5. Learn the relationship between these collections.

4.2 Practicing queries

First Setup the database form **4.1** then practice the below given queries. The queries are divided into three sections **Easy**, **Medium** and **Hard** according to their difficulty. Use hints wherever given and also refer to the materials discussed in above sections and also to the official documents given for reference:

Q1 Perform following **Easy** queries:

A. Retrieve the complete details of the driver whose first name is "Max" and last name is "Verstappen" from the driver collection.
B. Find all races that were conducted between the years 1990 and 2005 (both inclusive) from the races collection.
C. Display all drivers whose nationality begins with either "British" or "German" from the drivers collection.
D. List all circuits from the circuits collection, displaying only the circuit name, location, and country, while excluding the _id and url fields.
E. Retrieve the top 5 oldest drivers from the drivers collection by sorting them in ascending order of their date of birth (dob). (**hint** :- Explore MongoDB commands that allow you to change the order of documents and control how many documents are displayed in the output.)

Q2 Perform following **Medium** queries:

A. Find how many drivers belong to each nationality in the drivers collection. Display the nationalities in such a way that the most common nationality appears first.
B. Calculate the total number of career points scored by Lewis Hamilton using the results collection. (Assume his driverId is 1.)
C. Identify the constructor that has won the maximum number of races. A race win is indicated when a driver finishes in position 1. Display the constructor ID and the number of wins.
D. Calculate the average finishing position of Michael Schumacher using the results collection. Consider only the races he actually finished, and ignore abnormal or missing finishing positions. (Assume his driverId is 30.)
E. Determine how many races were held in each year using the races collection. Display the results in descending order of year (latest year first).
F. List all the distinct countries where Formula 1 races have been conducted using the circuits collection.
G. Find the fastest lap speed recorded in raceId = 1000 from the results collection. Ignore entries where the lap speed is missing or invalid, and display only the highest speed recorded. (Note: The dataset uses "\N" to represent missing values.)

Q3 Perform following **Hard** queries:

Students are encouraged to refer to the official MongoDB documentation to understand aggregation stages, accumulator operators, and join operations used in complex queries.

.

References :-

- Aggregation Framework (Core Concept) :- https://www.mongodb.com/docs/manual/aggregation/
- $lookup (Join Collections) :- https://www.mongodb.com/docs/manual/reference/operator/aggregation/lookup/
- $group and Accumulators ($sum, $avg, $count) :- https://www.mongodb.com/docs/manual/reference/operator/aggregation/group/
- $match (Filtering in Aggregation) :- https://www.mongodb.com/docs/manual/reference/operator/aggregation/match/
- $project (Field Selection & Computed Fields) :- https://www.mongodb.com/docs/manual/reference/operator/aggregation/project/
- $unwind (Array Decomposition) :- https://www.mongodb.com/docs/manual/reference/operator/aggregation/unwind/
- Conditional Logic ($cond) :- https://www.mongodb.com/docs/manual/reference/operator/aggregation/cond/
- Arithmetic Operators ($subtract) :- https://www.mongodb.com/docs/manual/reference/operator/aggregation/subtract/

Questions :-

A. For the 2021 Formula 1 season, list all race winners.
 For each race, display Race name, Race round and Winner's full name (first name + last name) (Note: Race winners are drivers who finished in position 1.)
B. Calculate the total points scored by each constructor during the 2010 season. Display the constructor name along with the total points, and show the results in descending order of points.
C. Find the driver who has won the Monaco Grand Prix the maximum number of times. Display the driver ID and the total number of wins at Monaco.

D. Create a career summary for a specific driver (assume driverId = 1). The summary should include Total number of races participated, Total number of wins and Total career points
E. Identify the single race performance where a driver gained the maximum number of positions during a race. (Positions gained = starting grid position − finishing position.) Consider only races where the driver finished the race successfully.
F. Find how many times a driver started from pole position (grid = 1) and won the race (position 1).
G. List all unique drivers who have raced for the constructor Ferrari at any point in their career. Display the driver's first name and last name.
H. Find all race instances where a driver achieved all three of the following in the same race Started from pole position, Won the race and Set the fastest lap