# Experiment Number 3 : To install and configure MongoDB to execute NoSQL commands

**Date of Performance :**

**Date of Submission :**

## 1. Statement of problems:

### 1.1 Aim: To install and configure MongoDB on a computer system and demonstrate the execution of basic NoSQL commands using the MongoDB shell environment.

### 1.2 Objectives:

1. To understand the system requirements and installation steps for MongoDB on various platforms.
2. To configure the MongoDB environment and verify successful installation.
3. To access and use the MongoDB shell (mongosh) for database operations.
4. To demonstrate the creation of databases and collections in MongoDB.
5. To perform CRUD (Create, Read, Update, Delete) operations using NoSQL commands.
6. To understand the advantages of NoSQL databases compared to traditional relational databases.

### 1.3 Software Used: MongoDB

**1.4 SCOPE:**

This experiment covers the end-to-end process of installing MongoDB on different operating systems (Windows, Linux), configuring system paths and services, and launching the MongoDB shell interface. It includes performing fundamental NoSQL operations like creating databases and collections, inserting and retrieving documents, and updating and deleting data. The scope is limited to standalone server installation and does not include advanced MongoDB topics such as sharding, replication, or cloud deployment. This experiment aims to provide foundational knowledge of NoSQL database systems using MongoDB for academic and development purposes.

**Scope 1: INSTALLING AND CONFIGURING MONGODB on Windows OS**

The experiment covers the complete lifecycle of installing and configuring MongoDB Community Edition on a Windows-based system. Scope 1: Installing and Configuring MongoDB on Windows OS This scope involves the complete process of installing and configuring MongoDB on a Windows-based operating system. It includes understanding the basic requirements needed for a successful setup and familiarizing oneself with the installation steps using the official installer. The experiment guides learners through selecting the appropriate installation options and setting up the system to run MongoDB effectively. It also includes configuring the environment so that MongoDB can be accessed easily through command-line interfaces. Once the setup is complete, learners will verify the installation by connecting to the database shell and executing introductory database operations. Through this process, they will gain practical experience in performing essential NoSQL commands, such as creating databases and collections, as well as inserting, retrieving, updating, and deleting data. The focus remains on providing a fundamental understanding of MongoDB usage in a Windows environment without involving advanced configurations or distributed system features.

**Scope 2: Installing and Configuring MongoDB on Linux OS (Ubuntu/Debian-Based Distributions)**

This scope focuses on setting up MongoDB on Linux systems, particularly those based on Ubuntu or Debian distributions. The experiment guides learners through the end-to-end process of preparing the Linux environment for MongoDB installation. It includes ensuring the operating system is suitable and that necessary package management tools are available. The process involves integrating MongoDB's official repository into the system and carrying out the installation through standard package management commands. Learners are then introduced to the steps involved in configuring MongoDB for optimal operation in a Linux setting, which may include adjusting configuration files and managing services. Once the installation is complete, the experiment concludes by testing the setup through the database shell, where basic database operations are executed. This exercise helps students become familiar with fundamental NoSQL operations in a Linux environment while reinforcing core concepts related to installation, configuration, and data handling in MongoDB.

**Scope 3: Study of NoSQL Database Management System**

This scope is centered on the study, installation, and configuration of MongoDB, a leading NoSQL database system, on both Windows and Linux operating systems. The primary aim is to provide students with a foundational understanding of NoSQL databases and hands-on experience in executing basic operations within a standalone server setup. By engaging in this experiment, learners are introduced to the core principles and practical skills required for managing NoSQL environments in modern application development.

The exercise includes downloading the appropriate MongoDB packages for the respective operating systems, configuring system-specific settings, and validating successful installations using command-line interactions. On Windows, the process involves installing MongoDB via the graphical .msi installer, setting up data directories, and managing services through PowerShell or the Services Manager. On Linux (especially Ubuntu/Debian-based systems), the steps include adding MongoDB's repository, importing cryptographic keys, installing packages, and controlling MongoDB as a system service.

Once the environment is fully configured, the scope extends to performing essential NoSQL database operations. These include creating databases and collections, inserting and retrieving documents, updating existing data, and deleting content or entire structures. Through these tasks, students learn the document-oriented nature of MongoDB, its schema-less design, and how it utilizes JSON-like BSON formats for storing data.

The experiment is limited to standalone deployments, focusing strictly on foundational elements of NoSQL usage. Advanced features such as replication, sharding, clustering, and cloud-based deployments (e.g., MongoDB Atlas) are excluded to maintain a clear focus on core learning outcomes appropriate for introductory-level coursework.

This scope is specifically tailored for academic settings, targeting students in computer science, IT, and software engineering programs. It serves as a stepping stone to deeper exploration in backend development, large-scale data management, and distributed systems.

**scope4 classified domains of nosql for study:**

NoSQL databases are broadly classified into four main types based on their data storage and retrieval models.

Document-oriented databases, such as MongoDB, store data in structured documents (typically JSON or BSON), making them ideal for handling hierarchical or nested data structures. This type of database is well-suited for applications that require flexible schemas and complex data representations.

Key-value stores, like Redis and Riak, represent the simplest form of NoSQL databases. They store data as pairs of unique keys and their corresponding values, allowing for extremely fast read and write operations. These databases are often used for caching, session management, and real-time applications.

Column-family stores, such as Apache Cassandra and HBase, organize data into columns instead of traditional rows. This model excels at handling large volumes of structured data and is commonly

used in analytical applications and scenarios that require efficient reading and writing of wide tables. Lastly, graph databases, including Neo4j and ArangoDB, focus on representing and querying relationships between data entities. They use nodes to represent entities and edges to define the relationships between them, making them particularly effective for use cases like social networking, fraud detection, and recommendation engines where understanding complex interconnections is crucial.

In this experiment, the emphasis is on Document-Oriented Databases, particularly MongoDB, as an entry point into the broader NoSQL ecosystem.

## 2. Theory:

### 2.1 MongoDB Architecture

MongoDB is a popular **NoSQL document-oriented** database management system, known for its flexibility, high performance, high availability, and multi-storage engines. Unlike traditional relational databases (RDBMS), MongoDB does not store data in tables and rows but instead uses a document-based structure with **BSON** (Binary JSON) format.

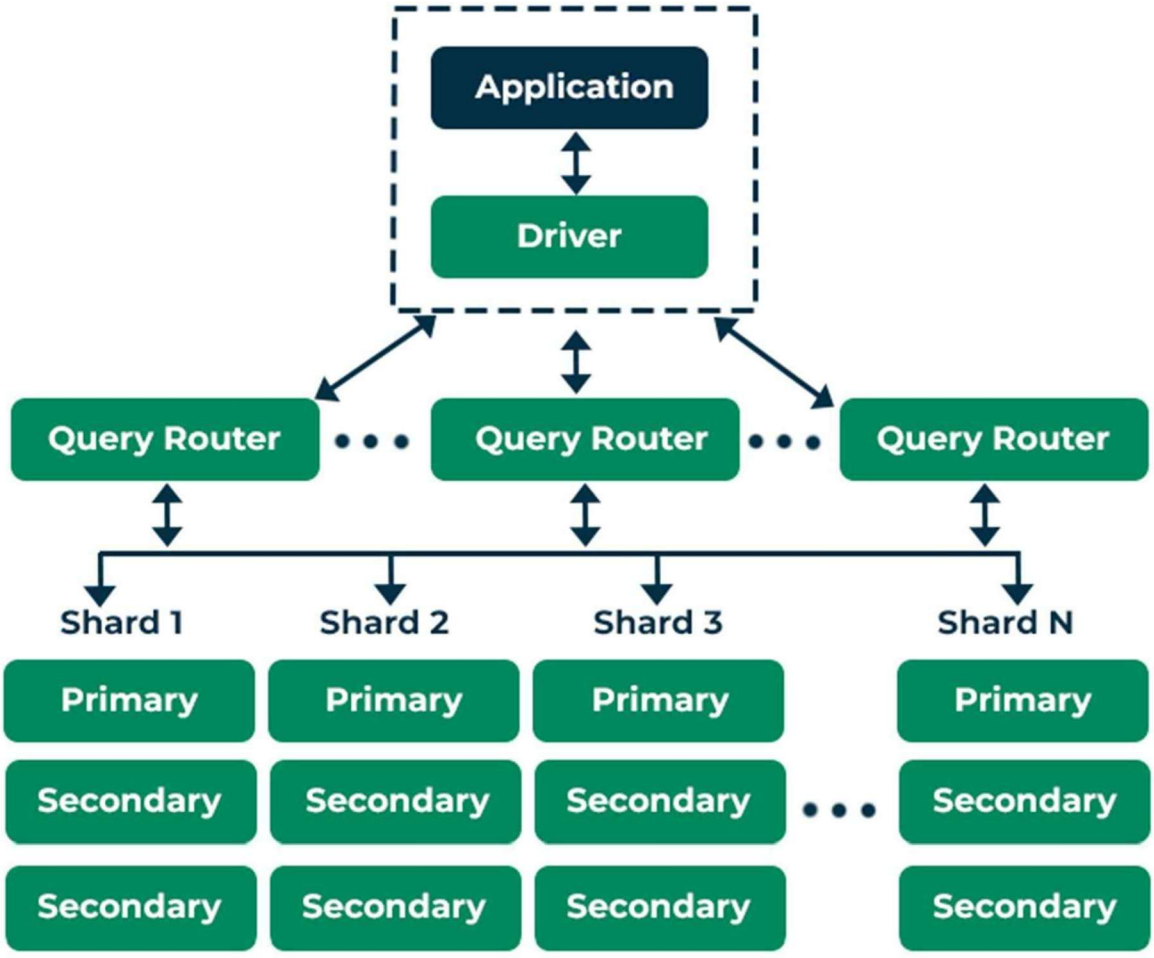
Companies like **Adobe, Uber, IBM, and Google** leverage MongoDB for big data applications, real-time analytics, and cloud computing solutions. In this article, we will explore the architecture of MongoDB, including its components, working, scalability mechanisms (replication & sharding), and indexing strategies that optimize performance.

**Key Features of MongoDB**

- **Document-Oriented Storage** – Stores data in JSON-like BSON documents instead of tables.
- **Schema-Less Database** – Collections can store documents with different structures.
- **Horizontal Scalability** – Uses sharding to distribute data across multiple nodes.
- **High Availability** – Ensures data redundancy using replication.
- **Aggregation Framework** – Supports complex queries and real-time analytics.
- **Fast Read/Write Operations** – Optimized with indexing and in-memory caching.

**MongoDB Architecture and its Components**

MongoDB's architecture consists of several core components that work together to provide efficient data storage, retrieval and processing.




**1. Drivers & Storage Engine**

MongoDB store the data on the server but that data we will try to retrieve from our application. So that time how the communication is happening between our application and MongoDB server. Any application which is written in python, .net and java or any kind of frontend application, these application are trying to access the data from these physical storage in server.

First they will interact with driver which will communicate with MongoDB server. What happen is once the request is going from the frontend application through the driver then driver will change appropriate query by using query engine and then the query will get executed in MongoDB data model. Left side is security which provides security to the database that who will access the data and right side is management this management will manage all these things.

**Drivers**

Drivers are client libraries that offer interfaces and methods for applications to communicate with MongoDB databases. Drivers will handle the translation of documents between BSON objects and mapping application structures. Java,.NET, JavaScript, Node.js, Python, etc are some of the widely used drives supported by MongoDB.

**Storage Engine**

The storage engine significantly influences the performance of applications, serving as an intermediary between the MongoDB database and persistent storage, typically disks. MongoDB supports different storage engines:

| Storage Engine | Description | Use Case |
|---|---|---|
| **WiredTiger** | Default storage engine since MongoDB 3.0. Supports document-level concurrency and data compression. | High-performance, write-intensive applications |
| **In-Memory Engine** | Stores data in RAM instead of disk, ensuring ultra-fast access. | Real-time data processing, caching |
| **MMAPv1** | Based on memory-mapped files. Supports high read workloads. | Read-heavy applications (deprecated after MongoDB 4.0) |

**2. Security in MongoDB**

Mongodb provides various security mechanisms to protect data from unauthorized access and breaches

- Authentication - Verifies user credentials
- Authorization - Assigns role -based access controls (RBAC).
- Encryption - Supports TLS/SSL encryption for data in transit
- Hardening - Ensures only trusted hosts can access the database

**3. MongoDB Server**

It serves as the central element and is in charge of maintaining, storing, and retrieving data from the database through a number of interfaces. The system's heart is the MongoDB server. Each mongod server instance is in charge of handling client requests, maintaining data storage, and performing database operations. Several mongod instances work together to form a cluster in a typical MongoDB setup.

**4. MongoDB Shell**

For dealing with MongoDB databases, MongoDB provides the MongoDB Shell command-line interface (CLI) tool. The ability to handle and query MongoDB data straight from the terminal is robust and flexible.

After installing MongoDB, you may access the MongoDB Shell, often known as mongo. It interacts with the database using JavaScript-based syntax. Additionally, it has built-in help that shows details about possible commands and how to use them.

**5. Data Storage in MongoDB**

MongoDB organizes data into databases, collections, and documents.

**Collections**

A database can contain as many collections as it wishes, and MongoDB stores data inside collections.As an example, a database might contain three collections a user's collection, a blog post collection, and a comments collection.

The user collection would hold user data and documents, the blog post collection would hold blog posts and documents, and the comments collection would hold documents related to comments. This would allow for the easy retrieval of all the documents from a single collection.

**Documents**

Documents themselves represent the individual records in a specific collection. For example inside the blog posts collection we'd store a lot of blog post documents and each one represents a single blog post

now the way that data is structured inside a document looks very much like a JSON object with key value pairs but actually it's being stored as something called BSON which is just binary JSON.

**6. Indexes**

Indexes are data structures that make it simple to navigate across the collection's data set. They help to execute queries and find documents that match the query criteria without a collection scan. These are the following different types of indexes in MongoDB:

**Single field**

MongoDB can traverse the indexes either in the ascending or descending order for single-field index. In this example, we are creating a single index on the item field and 1 here represents the filed is in ascending order.

db.students.createIndex({"item":1})

A compound index in MongoDB contains multiple single filed indexes separated by a comma. MongoDB restricts the number of fields in a compound index to a maximum of 31.

db.students.createIndex({"item": 1, "stock":1})

Here, we create a compound index on item: 1, stock:1

**Multi-Key**

When indexing a filed containing an array value, MongoDB creates separate index entries for each array component. MongoDB allows us to create multi-key indexes for arrays containing scalar values, including strings, numbers, and nested documents.

db.students.createIndex({<filed>: <1 or -1>})

**Geo Spatial**

Two geospatial indexes offered by MongoDB are called 2d indexes and 2d sphere indexes. These indexes allow us to query geospatial data. On this case, queries intended to locate data stored on a two-dimensional plane are supported by the 2d indexes. On the other hand, queries that are used to locate data stored in spherical geometry are supported by 2D sphere indexes.

**Hashed**

To maintain the entries with hashes of the values of the indexed field we use Hash Index. MongoDB supports hash based sharding and provides hashed indexes.

db.<collection>.createIndex( { item: "hashed" } )

**7. Replication**

Within a MongoDB cluster, data replication entails keeping several copies of the same data on various servers or nodes. Enhancing data availability and dependability is the main objective of data replication. A replica may seamlessly replace a failing server in the cluster to maintain service continuity and data integrity.

- **Primary Node (Primary Replica):** In a replica set, the primary node serves as the main source for all write operations. It's the only node that accepts write requests. The main node is where all data modifications begin and are implemented initially.
- **Secondary Nodes:** Secondary nodes duplicate data from the primary node (also known as secondary replicas). They are useful for dispersing read workloads and load balancing since they are read-only and mostly utilized for read activities.

**8. Sharding**

Sharding is horizontal scaling in MongoDB, where large datasets are divided into smaller chunks and distributed across multiple servers (shards) to improve performance and scalability. Instead of vertical scaling (adding more CPU/RAM to a single server), MongoDB distributes data across multiple machines.

For example, a 200 million document database can be split across 4 servers (S1, S2, S3, S4), each holding 50 million documents, reducing the load on a single machine. Sharding uses a shard key to

determine how data is partitioned, ensuring efficient query distribution and fault tolerance in large-scale applications

**Conclusion**

The architecture of MongoDB has been thoroughly examined in this extensive article, including its essential parts, data storage, replication, sharding, high availability, security, scalability, and performance optimization. MongoDB is a top choice for a wide range of applications, from small-scale initiatives to massive, data-intensive systems, due to its adaptable and potent design. To fully utilize MongoDB and create reliable, scalable, and secure solutions, developers and administrators must have a thorough understanding of the database's architecture.

# 3. Implementation:

## 3.1 Environment setup:

## Installation of MongoDB on Windows :

**Step-by-Step Installation of MongoDB on Windows**

- **Step 1: Download MongoDB**
  - Visit: https://www.mongodb.com/try/download/community
  - Choose:
    - **Platform**: Windows
    - **Version**: Current (e.g., MongoDB 8.0)
    - **Package**: MSI
  - Click **Download** and run the installer.

- **Step 2: Complete the Setup**
  - During installation:
    - Select **Complete Setup**
    - (Optional) Check **Install MongoDB Compass** if you want a GUI.
  - MongoDB will be installed at:
    C:\Program Files\MongoDB\Server\<version>\bin

- **Step 3: Create Data Directory**
  - MongoDB needs a folder to store database files.
  - Open **File Explorer** and create the folder:
    C:\data\db
  - This is the default data directory MongoDB uses.

- **Step 4: Start the MongoDB Server**
  - Open **Command Prompt** and run:
    "C:\Program Files\MongoDB\Server\8.0\bin\mongod.exe"
  - If successful, it will show:
    waiting for connections on port 27017

- **Step 5: Install and Run MongoDB Shell (mongosh)**
  - From MongoDB v6+, mongosh replaces mongo.exe.
  - Download mongosh from: https://www.mongodb.com/try/download/shell
  - Extract the ZIP file.
  - Go to the bin folder and double-click mongosh.exe
    - If Windows shows a security warning: Click **More info → Run anyway**
  - Or, open **Command Prompt** and run: "C:\Path\To\mongosh.exe"
- **Step 6: Connect and Start Using MongoDB**
  - When mongosh opens, it prompts: Please enter a MongoDB connection string (Default: mongodb://localhost/)
  - Just **press Enter** to connect locally.

**3.2 Explain statement of that problem :**

Traditional relational databases face limitations in handling large, dynamic, and unstructured data efficiently. In inventory management systems, where data about products, suppliers, stock levels, and transactions changes frequently, there is a need for a scalable and flexible solution. This experiment addresses the problem by using MongoDB, a NoSQL database, to install, configure, and execute queries for effective inventory data management.

## 3.3 Execute and Output
## MongoDB Inventory Database

1. **Database Overview:**

Database Name: inventory_db
Database Type: MongoDB (NoSQL document-oriented database)
Collections:

```
> use inventoryDB
< switched to db inventoryDB
> show collections
< categories
  inventory
  locations
  products
  suppliers
  transactions
```

```
> db.categories.countDocuments()
< 10
> db.suppliers.countDocuments()
< 10
> db.locations.countDocuments()
< 5
> db.products.countDocuments()
< 100
> db.inventory.countDocuments()
< 100
> db.transactions.countDocuments()
< 200
```

1. **products** – Stores product details including category and supplier references.

```
> db.products.findOne()
< {
    _id: ObjectId('6890b27fd2775cee751d93fb'),
    product_id: 1,
    name: 'Product 1',
    category_id: 3,
    supplier_id: 9,
    price: 691.15,
    unit: 'pack'
  }
```

**2. categories** – Stores product category details.

```
> db.categories.findOne()
< {
    _id: ObjectId('6890b21cd2775cee751d93de'),
    category_id: 1,
    category_name: 'Electronics'
  }
```

**3. suppliers** – Stores supplier information.

```
> db.suppliers.findOne()
< {
    _id: ObjectId('6890b247d2775cee751d93e9'),
    supplier_id: 1,
    name: 'Supplier A',
    email: 'suppliera@example.com',
    phone: '+91-9880079243',
    address: '62 Main St, City A'
  }
```

**4. locations** – Stores storage location details.

```
> db.locations.findOne()
< {
    _id: ObjectId('6890b262d2775cee751d93f5'),
    location_id: 1,
    location_name: 'Warehouse 1',
    address: '704 Industrial Area, Zone A'
  }
```

5. **inventory** – Tracks product stock levels at various locations.

```
> db.inventory.findOne()
< {
    _id: ObjectId('6890b297d2775cee751d9460'),
    inventory_id: 1,
    product_id: 1,
    location_id: 5,
    quantity: 48,
    last_updated: '2025-07-12'
  }
```

6. **transactions** – Records stock movements (IN/OUT).

```
> db.transactions.findOne()
< {
    _id: ObjectId('6890b2b4d2775cee751d94c5'),
    transaction_id: 1,
    product_id: 36,
    transaction_type: 'IN',
    quantity: 31,
    transaction_date: '2025-06-14',
    location_id: 2
  }
```

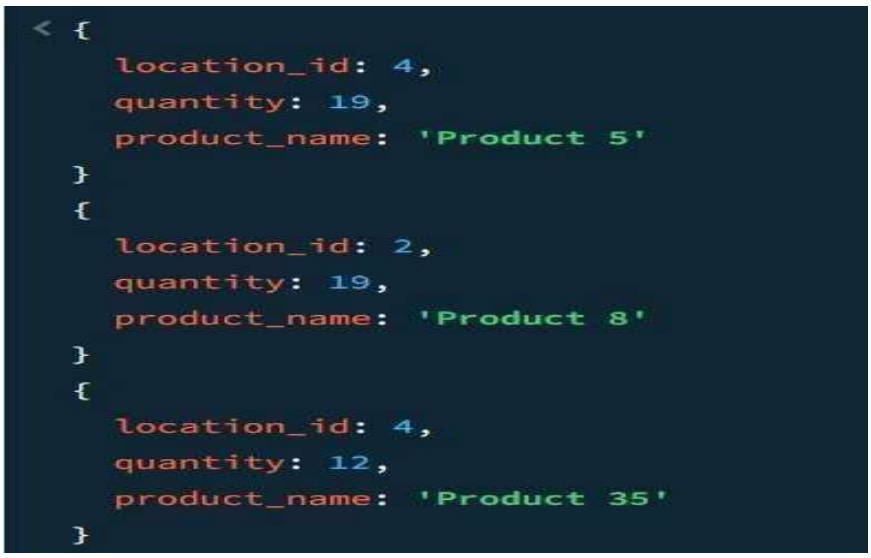
Purpose:
This database is designed to manage inventory, suppliers, products, stock quantities, and transaction histories for an inventory management system. It supports reporting, stock tracking, and supplier management.

## Command:

1. **Products with quantity less than 20**

```
db.inventory.aggregate([
  { $match: { quantity: { $lt: 20 } } },
  {
   $lookup: {
     from: "products",
     localField: "product_id",
     foreignField: "product_id",
     as: "product_details"
   }
  },
  { $unwind: "$product_details" },
  {
   $project: {
     _id: 0,
     product_name: "$product_details.name",
     quantity: 1,
     location_id: 1
   }
  }
])
```

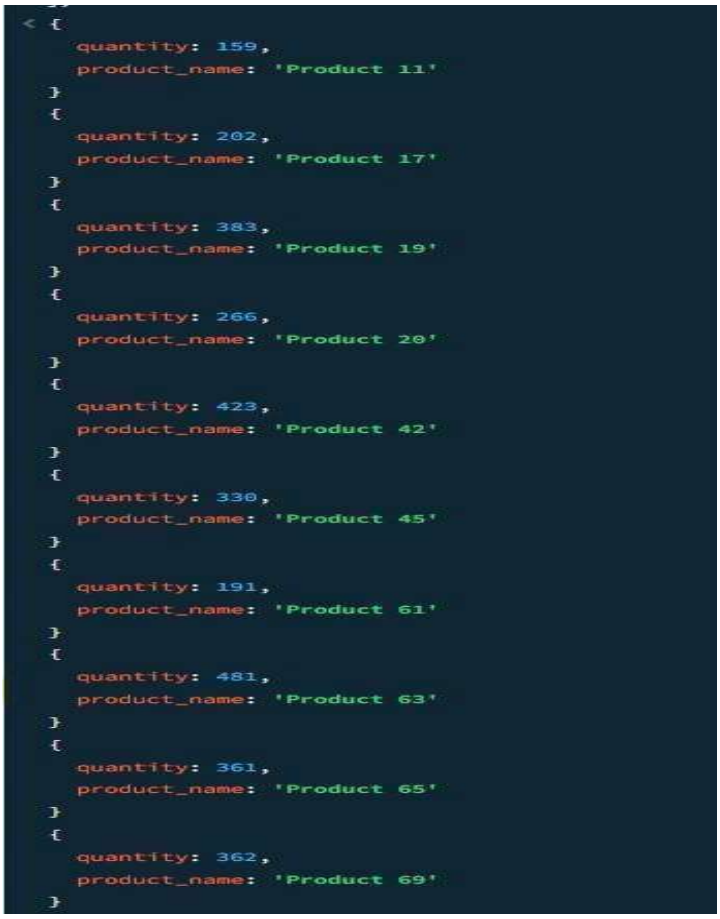
**Output:**



**2. Show product names with current inventory count at a single location**

```
db.inventory.aggregate([
  { $match: { location_id: 1 } },
  { $lookup: {
      from: "products",
      localField: "product_id",
      foreignField: "product_id",
      as: "product"
  }},
  { $unwind: "$product" },
  { $project: {
      _id: 0,
      product_name: "$product.name",
      quantity: 1
  }},
  { $limit: 10 }
])
```

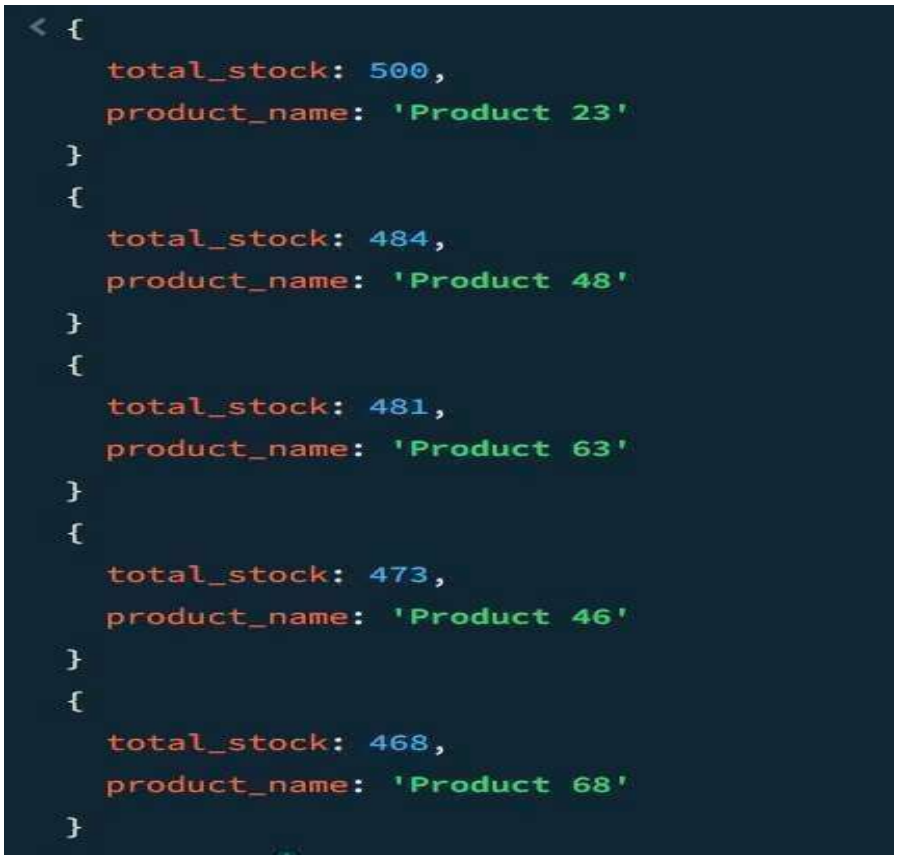
**Output:**



**3. Top 5 most stocked products**

```
db.inventory.aggregate([
  { $group: { _id: "$product_id", total_stock: { $sum: "$quantity" } } },
  { $sort: { total_stock: -1 } },
  { $limit: 5 },
  { $lookup: {
     from: "products",
     localField: "_id",
     foreignField: "product_id",
     as: "product"
  }},
  { $unwind: "$product" },
  { $project: { _id: 0, product_name: "$product.name", total_stock: 1 } }
])
```

**Output:**



**4. Average price per category**

```
db.products.aggregate([
  { $group: { _id: "$category_id", avg_price: { $avg: "$price" } } },
  { $lookup: {
      from: "categories",
      localField: "_id",
      foreignField: "category_id",
      as: "category"
  }},
  { $unwind: "$category" },
  { $project: {
      _id: 0,
      category: "$category.category_name",
      avg_price: { $round: ["$avg_price", 2] }
  }}
])
```

**Output:**

```
< {
    category: 'Cosmetics',
    avg_price: 676.16
  }
  {
    category: 'Furniture',
    avg_price: 435.7
  }
  {
    category: 'Pharmaceuticals',
    avg_price: 456.54
  }
  {
    category: 'Electronics',
    avg_price: 449.47
  }
  {
    category: 'Toys',
    avg_price: 556.87
  }
  {
    category: 'Automotive',
    avg_price: 557.1
  }
  {
    category: 'Stationery',
    avg_price: 564.89
  }
  {
    category: 'Groceries',
    avg_price: 458.81
  }
  {
    category: 'Footwear',
    avg_price: 727.97
  }
  {
    category: 'Clothing',
    avg_price: 662.37
  }
```

**5. Transactions for a specific product**

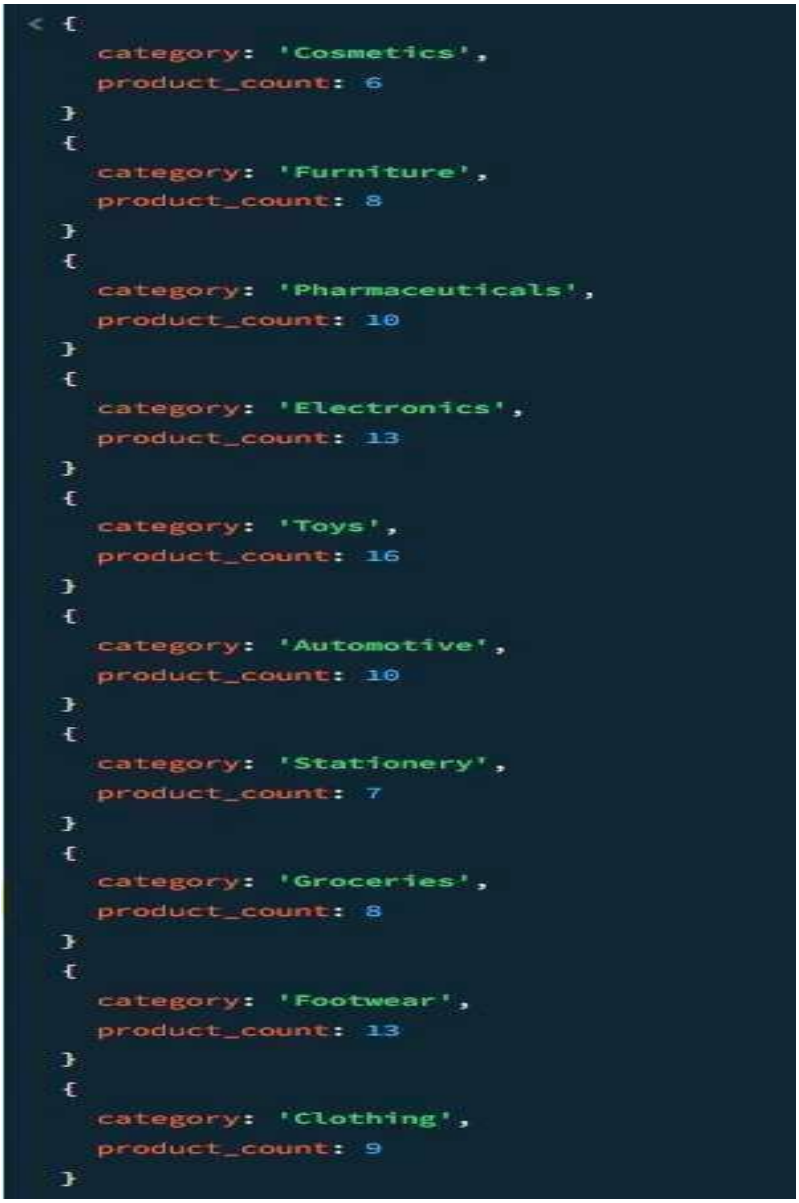
```
db.transactions.find({ product_id: 10 }).sort({ transaction_date: -1 })
```

**Output:**


**6. Count how many products are in each category**

```
db.products.aggregate([
  { $group: {
      _id: "$category_id",
      count: { $sum: 1 }
  }},
  { $lookup: {
      from: "categories",
      localField: "_id",
      foreignField: "category_id",
      as: "category"
  }},
  { $unwind: "$category" },
  { $project: {
      _id: 0,
      category: "$category.category_name",
      product_count: "$count"
  }}
])
```

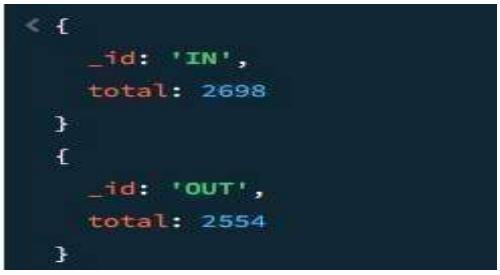
**Output:**




**7. Count total IN and OUT transactions**

```
db.transactions.aggregate([
  { $group: {
      _id: "$transaction_type",
      total: { $sum: "$quantity" }
  }}
])
```

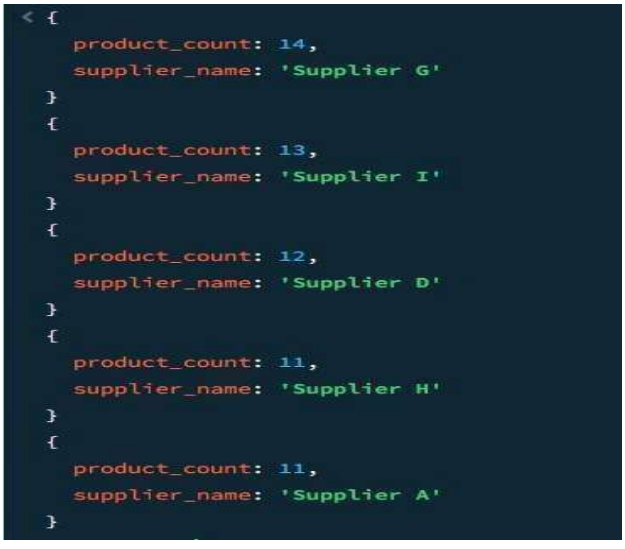
**Output:**




**8. Top 5 suppliers by number of products they supply**

```
db.products.aggregate([
  { $group: {
      _id: "$supplier_id",
      product_count: { $sum: 1 }
  }},
  { $sort: { product_count: -1 } },
  { $limit: 5 },
  { $lookup: {
      from: "suppliers",
      localField: "_id",
      foreignField: "supplier_id",
      as: "supplier"
  }},
  { $unwind: "$supplier" },
  { $project: {
      _id: 0,
      supplier_name: "$supplier.name",
      product_count: 1
  }}
])
```

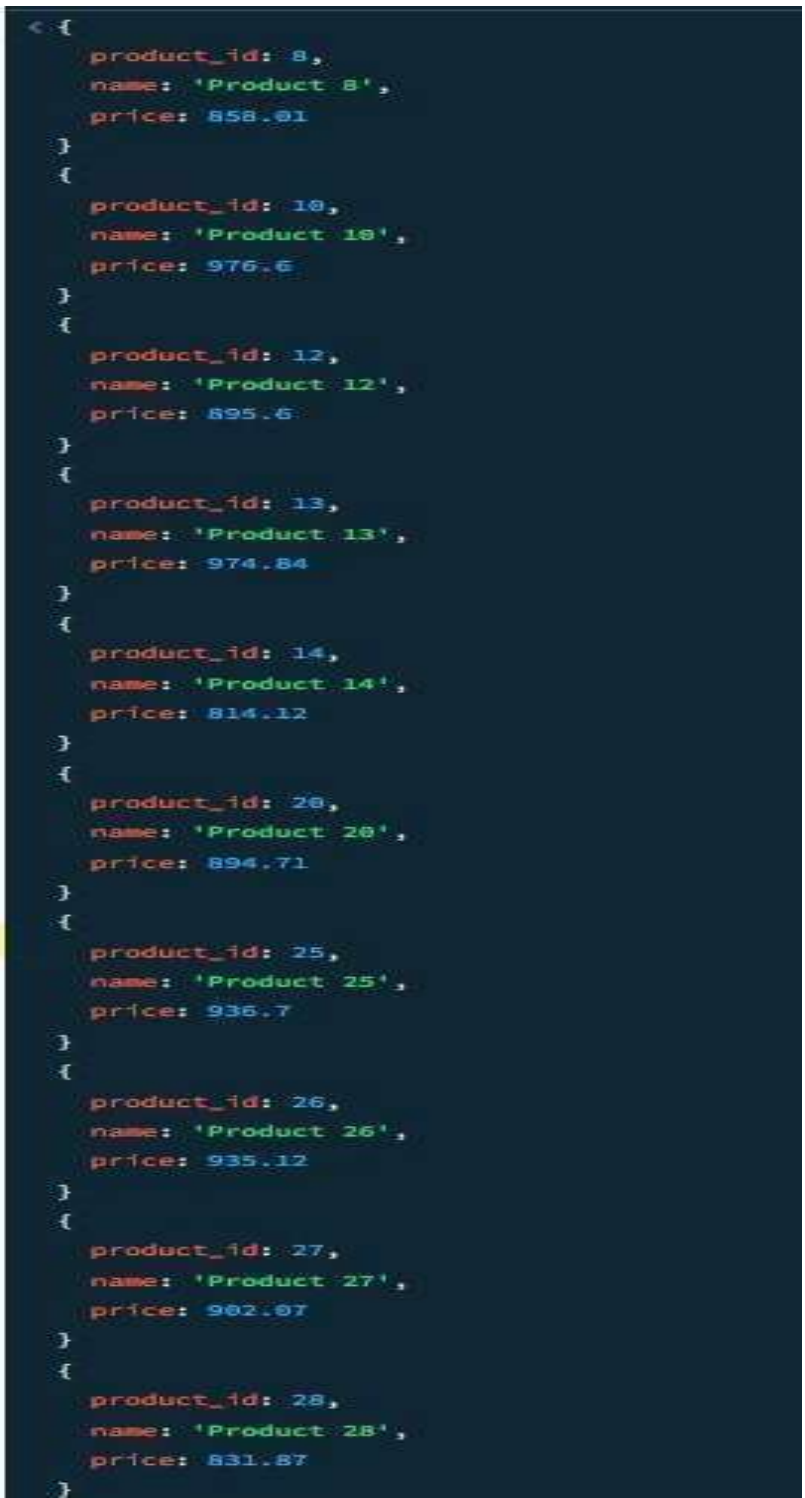
**Output:**



**9. List products priced above ₹800**

```
db.products.find(
  { price: { $gt: 800 } },
  { _id: 0, product_id: 1, name: 1, price: 1 }
).limit(10)
```

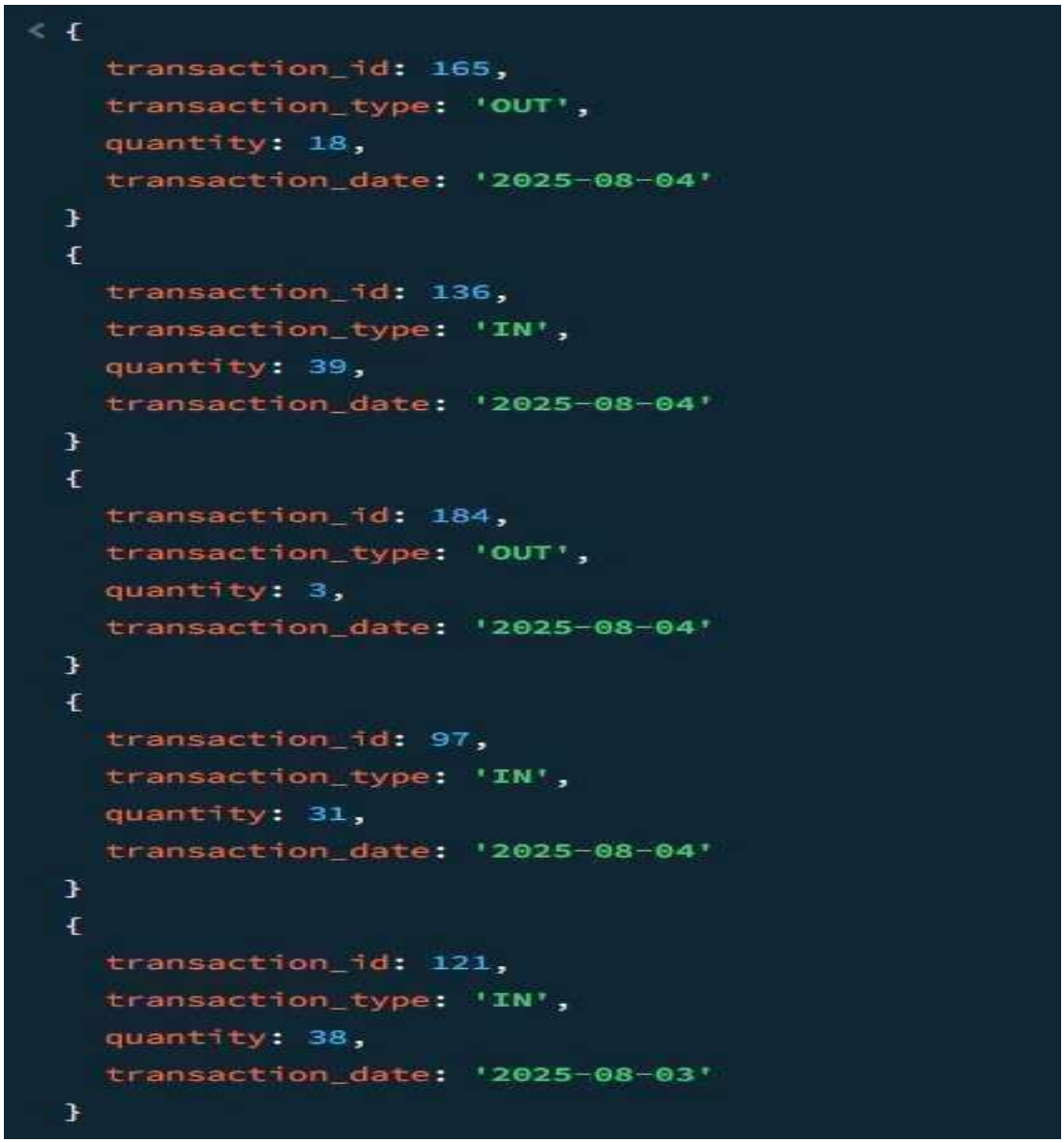
**Output:**



**10. Show last 5 transactions**

```
 db.transactions.find({}, {
_id: 0,
transaction_id: 1,
transaction_type: 1,
quantity: 1,
transaction_date: 1
}).sort({ transaction_date: -1 }).limit(5)
```

**Output:**



## 4.Conclusion:

In this experiment, we successfully installed and configured MongoDB, a popular NoSQL database. We learned how to start the MongoDB server and connect to it using the Mongo shell. Various queries, including inserting, updating, deleting, and retrieving documents, were executed to understand CRUD operations. Unlike traditional relational databases, MongoDB stores data in a document-oriented format (BSON/JSON), which offers great flexibility. Collections and documents effectively replace tables and rows in RDBMS, providing a more dynamic data structure. The experiment helped us handle unstructured and semi-structured data efficiently. We also explored indexing and query filtering to optimize data retrieval. Through practical exercises, we strengthened our understanding of database operations in NoSQL environments. The hands-on experience highlighted the differences between MongoDB and relational databases. Overall, this experiment enhanced our knowledge and confidence in working with modern database systems.

## 5.Reference:


- MongoDB, Inc. (2023). *MongoDB Manual: CRUD Operations*. Available at: https://www.mongodb.com/docs/manual/crud/
- Chodorow, K. (2013). *MongoDB: The Definitive Guide*. O'Reilly Media.
- Banker, K. (2011). *MongoDB in Action*. Manning Publications.
- MongoDB, Inc. (2023). *Introduction to MongoDB*. Available at: https://www.mongodb.com/what-is-mongodb
- Hows, D. (2017). *Learning MongoDB 3.x*. Packt Publishing.


## Sign and Remark:

| R1 (4 Marks) | R2 (4 Marks) | R3 (4 Marks) | R4 (3 Marks) | Total Marks (15 Marks) | Signature |
|---|---|---|---|---|---|
| | | | | | |