DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING(AI&ML)

2440597 NO SQL DATA BASES – (MONGO DB) LAB

B. Tech. II Year-II Sem L / T / P / C

0 / 0 / 2 / 1

LIST OF EXPERIMENTS:

Module I:

Introduction of MongoDB, No SQL Database, Advantage over RDBMS, MongoDB Data Types, Install MongoDB, MongoDB Data Modeling

1. Create and explore a NoSQL Document Structure

- Insert sample JSON documents to show flexible schemas.

2. Compare RDBMS vs. MongoDB with a Practical Schema

- Model the same data (e.g., user accounts) in SQL and MongoDB.

3. Explore MongoDB Data Types

- Insert and query documents using types like String, Number Int, Boolean, Array, Date, etc.

4. Basic MongoDB Data Modeling Example

- Design an embedded vs. referenced model for blog posts and comments.

Module II :

Operators: Query & Projection Operator, MongoDB Update Operator, Aggregation Pipeline Stages, MongoDB limit(), MongoDB sort(), Query Modifiers Commands: Aggregation Commands, Geospatial Command, Query and Write Operation Commands, Query Plan Cache Commands, Authentication Commands, User Management Commands, Role Management Commands, Replication Command, Shading Commands, Session Commands

1. Use Query and Projection Operators

- Demonstrate $eq, $gt, $lt, $in, $and, $or, $exists, and projection {field: 1}.

2. Update Operators and Aggregation Stages

- Use $set, $inc, $push in update, and pipeline stages like $match, $group, $sort.

3. Sorting, Limiting, and Modifying Queries

- Apply, limit(), sort(), and modifiers like, explain(), hint().

4. Geospatial Commands and User Management

- Insert geoJSON data and run $geoWithin queries; create users and assign roles.

Module III:

Create Database, Drop Database, Create Collection, Drop Collection

1. Create and Drop a Database

- Use use dbName, db.dropDatabase().

2. Create and Drop Collections

- db.createCollection("students"), db.students.drop().

3. Explore Collection Indexes and Options

- Create indexes and check with db.collection.getIndexes().

4. Set Up Schema Validation Rules

- Use JSON schema validation to restrict document structure.

Module IV:

Inset Documents, Update Documents, Delete Documents, Query Documents, SQL to MongoDB Mapping, MongoDB text search, Partial Updates & Document Limits, Removing Documents, Multi Update, Upsert, Wire Protocol, Bulk(), Operations and Methods, Common Commands, db.runCommand(), db.isMaster(), db.serverStatus(), db.currentOp() & db.killOp(), collection.stats() & collection.drop()

1. CRUD: Insert, Query, Update, Delete Documents

- Full example of inserting, querying with filters, updating fields, and deleting.

2. Use of db.runCommand() and Server Info

- Run db.runCommand({serverStatus: 1}) and db.isMaster().

3. Bulk Operations and Upsert Example

- Demonstrate bulkWrite() with mixed inserts and updates.

4. Check Collection Stats and Perform Partial Updates

- db.collection.stats(), $set with field targeting.

Module V:

MongoDB Shell, Shell Collection Methods, Cursor Method MongoDB Database Commands, Query Plan Cache Methods, User Management Method, Role Management Method, MongoDB Replication Methods Connectivity: Java MongoDB, PHP MongoDB, Python MongoDB

1. Using MongoDB Shell: Collection and Cursor Methods

- Demonstrate.find(), countDocuments(), forEach(), toArray().

2. Query Plan Cache and Role Management Commands

- db.collection.getPlanCache(), clear() and role creation with db.createRole().

3. Python MongoDB CRUD App using PyMongo

- Connect to MongoDB Atlas/local, and perform CRUD using Python.

4. Java MongoDB Connection Example

- Use MongoDB Java Driver to connect and run basic operations

INDEX

Department of Computer Science & Engineering (AI&ML)

CERTIFICATE

This is to certify that this manual is a Bonafide record of practical work carried out in the **MongoDB Laboratory** for the **B.Tech Computer Science Engineering (AI&ML) III Semester** Programme during the academic year **2025–2026.**

This manual has been prepared by **Mrs. B Soundarya (Assistant Professor),** Department of Computer Science Engineering (AI&ML), with my own efforts and to the best of our knowledge.

Signature of Lab Faculty **Signature of HOD**

PREFACE

This MongoDB Lab Manual provides a comprehensive introduction to the world of NoSQL databases and their growing importance in modern data-driven applications. MongoDB, as a document-oriented database, offers flexibility, scalability, and high performance for handling unstructured or semi-structured data. In this manual, we explore essential concepts such as data modeling, CRUD operations, aggregation, indexing, and schema design. The lab exercises are designed to help learners gain practical experience by applying core principles in real-time environments. With a focus on hands-on learning, this manual bridges theoretical knowledge with practical skills required in today's industry. Aimed at students, developers, and data enthusiasts, the content ensures a strong foundation in working with MongoDB. Our goal is to equip learners with the skills to build efficient, scalable database solutions that meet the needs of modern applications.

By,

Mrs. B Soundarya

ACKNOWLEDGEMENT

It was really a good experience, working at MongoDB Lab. First, I would like to thank Mrs. B Soundarya, Assistant Professor, Department of Computer Science & Engineering (AI&ML), Marri Laxman Reddy Institute of technology & Management for giving the technical support in preparing the document.

I express my sincere thanks to Dr. B Ravi Prasad, Head of the Department of Computer Science & Engineering (AI&ML), Marri Laxman Reddy Institute of technology & Management, for his concern towards me and gave me opportunity to prepare MongoDB laboratory manual.

I am deeply indebted and gratefully acknowledge the constant support and valuable patronage of Dr. B Ravi Prasad, Dean Academics, Marri Laxman Reddy Institute of technology & Management. I am unboundedly grateful to him for timely corrections and scholarly guidance.

I express my heartful thanks to Dr. P. Sridhar, Director, and Dr. R. Murali Prasad, Principal, Marri Laxman Reddy Institute of technology & Management, for giving me this wonderful opportunity for preparing the MangoDB laboratory manual.

At last, but not the least I would like to thank the entire Computer Science & Engineering Department faculties those who had inspired and helped me to achieve my goal.

By,

Mrs. B Soundarya

Department of Computer Science & Engineering (AI&ML)

GENERAL INSTRUCTIONS

1. Students are instructed to come to MongoDB laboratory on time. Late comers are not entertained in the lab.
2. Students should be punctual to the lab. If not, the conducted experiments will not be repeated.
3. Students are expected to come prepared at home with the experiments which are going to be performed.
4. Students are instructed to display their identity cards before entering into the lab.
5. Students are instructed not to bring mobile phones to the lab.
6. Any damage/loss of system parts like keyboard, mouse during the lab session, it is student's responsibility and penalty or fine will be collected from the student.
7. Students should update the records and lab observation books session wise. Before leaving the lab the student should get his/her lab observation book signed by the faculty.
8. Students should submit the lab records by the next lab to the concerned faculty members in the staff room for their correction and return.
9. Students should not move around the lab during the lab session.
10. If any emergency arises, the student should take the permission from faculty member concerned in written format.
11. The faculty members may suspend any student from the lab session on disciplinary grounds.
12. Never copy the output from other students. Write down your own outputs.

SAFETY MEASURES

To ensure the safe and efficient use of the Computer Science and Engineering(AI&ML) laboratory, all students must strictly adhere to the following safety guidelines:

1. General Conduct

- Maintain silence and discipline during lab sessions.
- Do not bring food, drinks, or chewing gum into the lab.
- Use lab resources responsibly and follow all instructions provided by the instructor or lab assistant.

2. Electrical Safety

- Do not touch electrical switches, sockets, or plugs with wet hands.
- Avoid overloading power sockets with unauthorized devices.
- Immediately report any loose connections, sparks, or unusual noises from equipment.

3. Computer and Equipment Handling

- Handle all computer systems, keyboards, mice, and peripherals with care.
- Do not attempt to open or tamper with any hardware components.
- Use only the assigned computer system; do not switch systems without permission.

4. Software and Data Safety

- Use only authorized software installed by the lab administrator.
- Do not attempt to install, uninstall, or modify any software without approval.
- Save your work frequently and ensure backups of important files.

5. Cybersecurity and Network Usage

- Keep your login credentials confidential.
- Do not attempt to access restricted websites or servers.

- Avoid activities such as hacking, gaming, or the use of pirated content.

6. Emergency Preparedness

- Be familiar with the location of emergency exits, fire extinguishers, and first aid kits.
- In the event of a fire, electrical hazard, or any emergency, remain calm and inform the lab instructor immediately.
- Follow the evacuation procedure as instructed.

7. Post-Lab Procedures

- Log out of your session and shut down the system properly after use.
- Leave your workstation clean and organized.
- Return any borrowed materials or equipment to their proper place.

8. Hygiene and Cleanliness

- Wash or sanitize your hands before and after using shared devices.
- Do not write or place unnecessary items on the workstation.
- Report any spills or cleanliness issues to the lab staff.

Department of Computer Science & Engineering (AI&ML)

VISION & MISSION OF THE INSTITUTE

Vision of the Institute:

To be a globally recognized institution that fosters innovation, excellence, and leadership in education, research, and technology development, empowering students to create sustainable solutions for the advancement of society.

Mission of the Institute:

- To foster a transformative learning environment that empowers students to excel in engineering, innovation, and leadership.
- To produce skilled, ethical, and socially responsible engineers who contribute to sustainable technological advancements and address global challenges.
- To shape future leaders through cutting-edge research, industry collaboration, and community engagement.

VISION & MISSION OF THE DEPARTMENT

Department Vision:

To nurture globally competent professionals in Artificial Intelligence and Machine Learning through excellence in education, research, and innovation, committed to developing sustainable and impactful solutions for the betterment of society.

Department Mission:

- To provide a transformative learning environment that equips students with in-depth knowledge and practical skills in Artificial Intelligence and Machine Learning, fostering innovation, leadership, and lifelong learning.
- To advance AI and ML through cutting-edge research, strong industry collaboration, and community engagement, preparing students to address real-world challenges on a global scale.
- To produce competent and ethical AI professionals who contribute to technological progress while addressing societal and environmental challenges with sustainable solutions.
- To foster a research-driven culture by partnering with industry and academia, encouraging entrepreneurship, and engaging in community-centered technology development.

Program Educational Objectives (PEOs)

PEO:

Professional Competence:

Graduates will possess strong theoretical and practical knowledge in Artificial Intelligence and Machine Learning, enabling them to solve complex real-world problems, pursue higher education, or excel in professional careers.

Innovation and Research Orientation:

Graduates will engage in innovative practices, cutting-edge research, and contribute to the advancement of AI and ML technologies through collaboration with industry and academia.

Leadership and Lifelong Learning:

Graduates will exhibit leadership qualities, effective communication, and teamwork skills, and will continuously upgrade their knowledge to adapt to evolving technological landscapes.

Entrepreneurial and Community Engagement:

Graduates will leverage entrepreneurial skills and a sense of civic responsibility to create AI-driven solutions that benefit local and global communities.

Course Outcomes (COs)

CO No	Course Outcome
CO1	Understand the fundamentals of NoSQL databases and explore the advantages of MongoDB over RDBMS.
CO2	Apply various MongoDB operators and commands to query, update, and manage data.
CO3	Perform CRUD operations and implement effective data modeling using collections and documents.
CO4	Analyze the usage of indexing, aggregation, geospatial queries, and advanced MongoDB features.
CO5	Develop and demonstrate connectivity with MongoDB using Java, Python, and PHP programming.

Program Outcomes (POs)

PO:

PO1.Engineering Knowledge:

Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 2. Problem Analysis:

Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO 3.Design/Development of Solutions:

Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO 4. Conduct Investigations of Complex Problems:

Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO 5. Modern Tool Usage:

Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO 6. The Engineer and Society:

Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO 7. Environment and Sustainability:

Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO 8.Ethics:

Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9. Individual and Team Work:

Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 10. Communication:

Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO 11. Project Management and Finance:

Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO 12. Life-long Learning:

Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

SYLLABUS

LIST OF EXPERIMENTS:

Module I:

Introduction of MongoDB, No SQL Database, Advantage over RDBMS, MongoDB Data Types, Install MongoDB, MongoDB Data Modeling

1. Create and explore a NoSQL Document Structure

o Insert sample JSON documents to show flexible schemas.

2. Compare RDBMS vs. MongoDB with a Practical Schema

o Model the same data (e.g., user accounts) in SQL and MongoDB.

3. Explore MongoDB Data Types

o Insert and query documents using types like String, Number Int, Boolean, Array, Date, etc.

4. Basic MongoDB Data Modeling Example

o Design an embedded vs. referenced model for blog posts and comments.

Module II :

Operators: Query & Projection Operator, MongoDB Update Operator, Aggregation Pipeline Stages, MongoDB limit(), MongoDB sort(), Query Modifiers Commands: Aggregation Commands, Geospatial Command, Query and Write Operation Commands, Query Plan Cache Commands, Authentication Commands, User Management Commands, Role Management Commands, Replication Command, Shading Commands, Session Commands

1. Use Query and Projection Operators

o Demonstrate $eq, $gt, $lt, $in, $and, $or, $exists, and projection { field: 1 }.

2. Update Operators and Aggregation Stages

o Use $set, $inc, $push in update, and pipeline stages like $match, $group, $sort.

3. Sorting, Limiting, and Modifying Queries

o Apply, limit(), sort(), and modifiers like, explain(), hint().

4. Geospatial Commands and User Management

o Insert geoJSON data and run $geoWithin queries; create users and assign roles.

Module III:

Create Database, Drop Database, Create Collection, Drop Collection

1. Create and Drop a Database

o Use use dbName, db.dropDatabase().

2. Create and Drop Collections

o db.createCollection("students"), db.students.drop().

3. Explore Collection Indexes and Options

o Create indexes and check with db.collection.getIndexes().

4. Set Up Schema Validation Rules

o Use JSON schema validation to restrict document structure.

Module IV:

Inset Documents, Update Documents, Delete Documents, Query Documents, SQL to MongoDB Mapping ,MongoDB text search, Partial Updates & Document Limits, Removing Documents , Multi Update , Upsert, Wire Protocol ,Bulk() Operations and Methods ,Common Commands, db.runCommand(), db.isMaster(), db.serverStatus(), db.currentOp() & db.killOp(), collection.stats() & collection.drop()

1. CRUD: Insert, Query, Update, Delete Documents

o Full example of inserting, querying with filters, updating fields, and deleting.

2. Use of db.runCommand() and Server Info

o Run db.runCommand({ serverStatus: 1 }) and db.isMaster().

3. Bulk Operations and Upsert Example

o Demonstrate bulkWrite() with mixed inserts and updates.

4. Check Collection Stats and Perform Partial Updates

o db.collection.stats(), $set with field targeting.

Module V:

MongoDB Shell, Shell Collection Methods, Cursor Method, MongoDB Database Commands, Query Plan Cache Methods, User Management Method, Role Management Method MongoDB Replication Methods Connectivity: Java MongoDB, PHP MongoDB, Python MongoDB

1. Using MongoDB Shell: Collection and Cursor Methods

o Demonstrate.find(), countDocuments(), forEach(), toArray().

2. Query Plan Cache and Role Management Commands

o db.collection.getPlanCache(), clear() and role creation with db.createRole().

3. Python MongoDB CRUD App using PyMongo

o Connect to MongoDB Atlas/local, and perform CRUD using Python.

4. Java MongoDB Connection Example

Use MongoDB Java Driver to connect and run basic operations

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

MONGODB LABORATORY

Virtual lab details

Name of the Virtual Lab:

Virtual Lab Host Institute:

URL/Link to Lab

Academic Year

Semester

List of Experiments Available in Virtual Lab

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

MONGODB LABORATORY

LAB PLANNER
CSE-B BATCH-1

S.No	Experiment	CO	Virtual Lab Availabilty	Date planned	Date Conducted
1	Create and Explore a NoSQL Document Structure Compare RDBMS vs. MongoDB with a Practical Schema	CO1	NA		
2	Explore MongoDB Data Types Basic MongoDB Data Modeling Example	CO1	NA		
3	Use Query and Projection Operators Update Operators and Aggregation Stages	CO2	NA		
4	Sorting, Limiting, and Modifying Queries Geospatial Commands and User Management	CO2	NA		
5	Create and Drop a Database Create and Drop Collections	CO3	NA		
6	Explore Collection Indexes and Options Set Up Schema Validation Rules	CO3	NA		
MID-I					
7	CRUD: Insert, Query, Update, Delete Documents Use of db.runCommand() and Server Info	CO4	NA		
8	Bulk Operations and Upsert Example Check Collection Stats and Perform Partial Updates	CO4	NA		
9	Using MongoDB Shell: Collection and Cursor Methods Query Plan Cache and Role Management Commands	CO5	NA		
10	Python MongoDB CRUD App using PyMongo	CO5	NA		
11	Java MongoDB Connection Example	CO5	NA		
MID-II					

CSE-B BATCH-2

S.No	Experiment	CO	Virtual Lab Availabilty	Date planned	Date Conducted
1	Create and Explore a NoSQL Document Structure Compare RDBMS vs. MongoDB with a Practical Schema	CO1	NA		
2	Explore MongoDB Data Types Basic MongoDB Data Modeling Example	CO1	NA		
3	Use Query and Projection Operators Update Operators and Aggregation Stages	CO2	NA		
4	Sorting, Limiting, and Modifying Queries Geospatial Commands and User Management	CO2	NA		
5	Create and Drop a Database Create and Drop Collections	CO3	NA		
MID-I					
6	Explore Collection Indexes and Options Set Up Schema Validation Rules	CO3	NA		
7	CRUD: Insert, Query, Update, Delete Documents Use of db.runCommand() and Server Info	CO4	NA		
8	Bulk Operations and Upsert Example Check Collection Stats and Perform Partial Updates	CO4	NA		
9	Using MongoDB Shell: Collection and Cursor Methods Query Plan Cache and Role Management Commands	CO5	NA		
10	Python MongoDB CRUD App using PyMongo	CO5	NA		
11	Java MongoDB Connection Example	CO5	NA		
MID-II					

CSE-C BATCH-1

S.No	Experiment	CO	Virtual Lab Availabilty	Date planned	Date conducted
1	Create and Explore a NoSQL Document Structure Compare RDBMS vs. MongoDB with a Practical Schema	CO1	NA		
2	Explore MongoDB Data Types Basic MongoDB Data Modeling Example	CO1	NA		
3	Use Query and Projection Operators Update Operators and Aggregation Stages	CO2	NA		
4	Sorting, Limiting, and Modifying Queries Geospatial Commands and User Management	CO2	NA		
5	Create and Drop a Database Create and Drop Collections	CO3	NA		
6	Explore Collection Indexes and Options Set Up Schema Validation Rules	CO3	NA		
MID-I					
7	CRUD: Insert, Query, Update, Delete Documents Use of db.runCommand() and Server Info	CO4	NA		
8	Bulk Operations and Upsert Example Check Collection Stats and Perform Partial Updates	CO4	NA		
9	Using MongoDB Shell: Collection and Cursor Methods Query Plan Cache and Role Management Commands	CO5	NA		
10	Python MongoDB CRUD App using PyMongo	CO5	NA		
11	Java MongoDB Connection Example	CO5	NA		
MID-II					

CSE-C BATCH-2

S.No	Experiment	CO	Virtual Lab Availabilty	Date planned	Date Conducted
1	Create and Explore a NoSQL Document Structure Compare RDBMS vs. MongoDB with a Practical Schema	CO1	NA		
2	Explore MongoDB Data Types Basic MongoDB Data Modeling Example	CO1	NA		
3	Use Query and Projection Operators Update Operators and Aggregation Stages	CO2	NA		
4	Sorting, Limiting, and Modifying Queries Geospatial Commands and User Management	CO2	NA		
5	Create and Drop a Database Create and Drop Collections	CO3	NA		
6	Explore Collection Indexes and Options Set Up Schema Validation Rules	CO3	NA		
MID-I					
7	CRUD: Insert, Query, Update, Delete Documents Use of db.runCommand() and Server Info	CO4	NA		
8	Bulk Operations and Upsert Example Check Collection Stats and Perform Partial Updates	CO4	NA		
9	Using MongoDB Shell: Collection and Cursor Methods Query Plan Cache and Role Management Commands	CO5	NA		
10	Python MongoDB CRUD App using PyMongo	CO5	NA		
11	Java MongoDB Connection Example	CO5	NA		
MID-II					

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

MONGODB LABORATORY

LAB PLANNER

Date planed																															
Date conducted																															
Roll Number	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*		Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	Exp No	CO	VL*	

Note:VL*-Virtual Lab Availabilty

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

MONGODB LABORATORY

RUBRICS USED TO ASSESS LEARNINGS IN LABORATORIES

1. RUBRICS FOR DAY TO DAY EVALUATION

Parameter	Max Marks	Level-1 (Very Poor)	Level-2 (Poor)	Level-3 (Average)	Level-4 (Good)	Level-5 (Excellent)
Observation Book	05	No observations or irrelevant data. (0–1)	Incomplete or incorrect data. (2)	Basic values with some errors. (3)	Mostly correct with good format. (4)	Fully correct, clear, and well-formatted. (5)
Record Writing	05	Not submitted. (0-1)	Submitted but mostly incomplete. (2)	Submitted with some missing/wrong parts. (3)	Submitted with minor issues. (4)	Fully complete, correct algorithm & flowchart. (5)
Result	05	No result or major errors. (0–1)	Result partially obtained. (2)	Acceptable result with limited error. (3)	Near-correct result and reasonable error. (4)	Accurate result. (5)
Viva-Voce	05	Did not answer any questions. (1)	Answered very few questions. (2)	Answered some questions with help. (3)	Answered most questions correctly. (4)	Answered all questions accurately. (5)

2. RUBRICS FOR INTERNAL EVALUATION

Criterion	Max Marks	Level-1 *(Very Poor)*	Level-2 *(Poor)*	Level-3 *(Average)*	Level-4 *(Good)*	Level-5 *(Excellent)*
Design/Tool/Apparatus Selection	2 Marks	Incorrect tool/design and no reasoning. **(0)**	Tool/design selection attempted with unclear logic. **(0.5)**	Satisfactory selection with partial justification. **(1)**	Correct selection and proper analysis with few errors. **(1.5)**	Smart selection with accurate, relevant analysis. **(2)**
Execution (Code/Debug/Run) /Analysis/Method Used	4 Marks	Did not attempt or completely failed to execute. **(0)**	Attempted but unable to proceed or with major errors. **(1)**	Partial execution with some logic/syntax errors. **(2)**	Mostly correct execution with minimal help. **(3)**	Fully correct and independently executed program. **(4)**
Results& Documentation	2 Marks	Incomplete or poorly presented. **(0)**	Basic structure but lacks clarity or formatting. **(0.5)**	Complete but generic or with formatting issues. **(1)**	Well-structured and mostly clear. **(1.5)**	Well-organized, professional, and engaging documentation. **(2)**
Viva-Voce (Understanding of Concepts)	2 Marks	No understanding; could not answer questions. **(0)**	Answered a few with difficulty. **(0.5)**	Answered half the questions with basic clarity. **(1)**	Good understanding with confident answers. **(1.5)**	Answered all questions with clarity and depth. **(2)**

3. RUBRICS FOR SEMESTER END EXAMINATIONS

Criterion	Max Marks	Level-1 (Very Poor *(0–2 marks)*	Level-2 (Poor)*(3–4 marks)*	Level-3 (Average)*(5–6 marks)*	Level-4 (Good)*(7–9 marks)*	Level-5 (Excellent)*(10–12 marks)*
Preparedness for the Experiment	12 marks	No clarity on objective or procedure. Unable to explain basics.	Limited idea of the objective/procedure. Needed prompting.	Has basic understanding; minor gaps in concept or preparation.	Well-prepared, with clear understanding of steps and background.	Fully prepared with strong conceptual clarity and confident explanation.
Performance in the Laboratory	12 marks	Unable to perform experiment. Relied entirely on examiner's help.	Performed with multiple errors and constant support.	Performed with some errors; required occasional help.	Performed mostly independently with minimal support.	Performed independently, efficiently, and with precision.
Calculations & Graphs	12 marks	No or incorrect calculations. Graphs missing or irrelevant.	Multiple calculation errors. Graphs/plots inaccurate or poorly labeled.	Calculations partially correct. Graphs present but with some flaws.	Correct calculations and graphs with minor errors.	Accurate calculations and well-labeled graphs with proper interpretation.
Results & Error Analysis	12 marks	No result or invalid result. No error analysis attempted.	Incorrect result with vague or no error discussion.	Acceptable result. Error analysis attempted but limited.	Correct result with sound error discussion.	Accurate result with detailed and relevant error analysis.
Viva-Voce (Subject Knowledge)	12 marks	Unable to answer any questions. No conceptual understanding.	Answered few questions with poor logic.	Answered half of the questions with average understanding.	Answered most questions with clarity and confidence.	Answered all questions with depth, clarity, and reasoning.

MODULE - I

Introduction to MongoDB

MongoDB is a NoSQL (Not Only SQL) database that is designed to store large amounts of unstructured data in a flexible, scalable, and easy-to-use manner. Unlike traditional relational databases (RDBMS) that store data in tables with rows and columns, MongoDB uses a document-oriented model where data is stored in BSON (Binary JSON) format. It is one of the most popular NoSQL databases, mainly because it is highly scalable, efficient for large-scale applications, and allows for high availability and quick data retrieval.

What is NoSQL Database?

NoSQL stands for "Not Only SQL," which is a broad category of databases that do not use traditional relational database structures. Unlike SQL databases that store data in rows and tables, NoSQL databases store data in various formats like key-value pairs, documents, wide-columns, or graphs.

- NoSQL databases are often used for:
- Big Data applications
- Real-time web applications
- Distributed systems
- Scalable and high-availability systems
- Advantages of MongoDB over RDBMS

Schema Flexibility:

- MongoDB does not require a predefined schema. This allows you to store data in a more flexible format. If you want to add or change fields in your data, you can do so without disrupting the entire system.
- In RDBMS, you have to define the schema in advance (tables, columns, types), making changes more cumbersome.

Scalability:

- MongoDB supports horizontal scaling, meaning you can distribute data across multiple servers and handle large amounts of traffic and data. It uses sharding to partition data across a distributed cluster.
- RDBMS generally scale vertically (adding more resources to a single server), which can be

more expensive and less effective for high-traffic applications.

- High Availability:
- MongoDB provides built-in replication, which means your data is automatically copied to multiple servers, ensuring high availability.
- In RDBMS, you often have to set up and manage replication manually or use third-party tools.

Performance:

MongoDB is designed for faster reads and writes, especially with large datasets and unstructured data. It supports indexing and aggregation for efficient querying.

While RDBMS can also handle large amounts of data, its performance may degrade when the database schema becomes complex or data grows rapidly.

Big Data Support:

MongoDB is built for handling large-scale data and unstructured data types such as logs, documents, videos, and images.

RDBMS typically struggles with handling such big data and requires a more complex infrastructure for managing large, unstructured datasets.

Data Model:

MongoDB stores data as JSON-like documents (BSON), allowing it to model complex data with nested structures. It is a natural fit for applications that require hierarchical data representation. RDBMS uses a tabular model, which can make representing hierarchical or complex data less intuitive.

MongoDB Data Types

MongoDB uses BSON (Binary JSON) format to store data, which supports more data types than standard JSON. Some of the common data types in MongoDB include:

String:

Used for storing text.

Example: "name": "John Doe"

- Integer:

Used for storing numeric values (32-bit or 64-bit).

Example: "age": 30

- Double:

Used for storing floating-point numbers.

Example: "price": 199.99

- Boolean:

Used for storing true/false values.

Example: "isActive": true

- Array:

Used for storing lists of values.

Example: "tags": ["mongodb", "database", "nosql"]

- Object:

Used for storing embedded documents or nested data structures.

Example: "address": {"city": "New York", "state": "NY"}

- Null:

Used for storing null values.

Example: "middleName": null

- Date:

Used for storing date and time.

Example: "createdAt": ISODate("2025-04-12T00:00:00Z")

- ObjectId:

A unique identifier used by MongoDB to uniquely identify documents within a collection.

Example: "id": ObjectId("5f8e9b6ed5b8e2d1f1f1f1f1")

- Binary Data:

Used for storing binary data (e.g., images, files).

Example: "imageData": BinData(0, "somebinarydata")

- Regular Expression:

Used for storing regex patterns.

Example: "email": /@example\.com$/

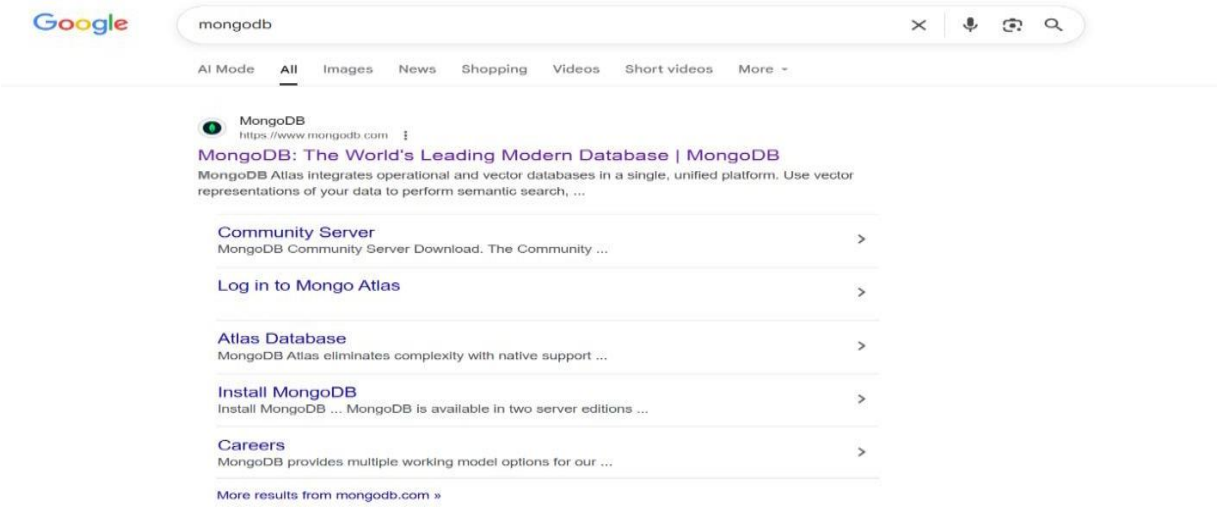
How to Install MongoDB

MongoDB Installation

Installation Procedure step by step

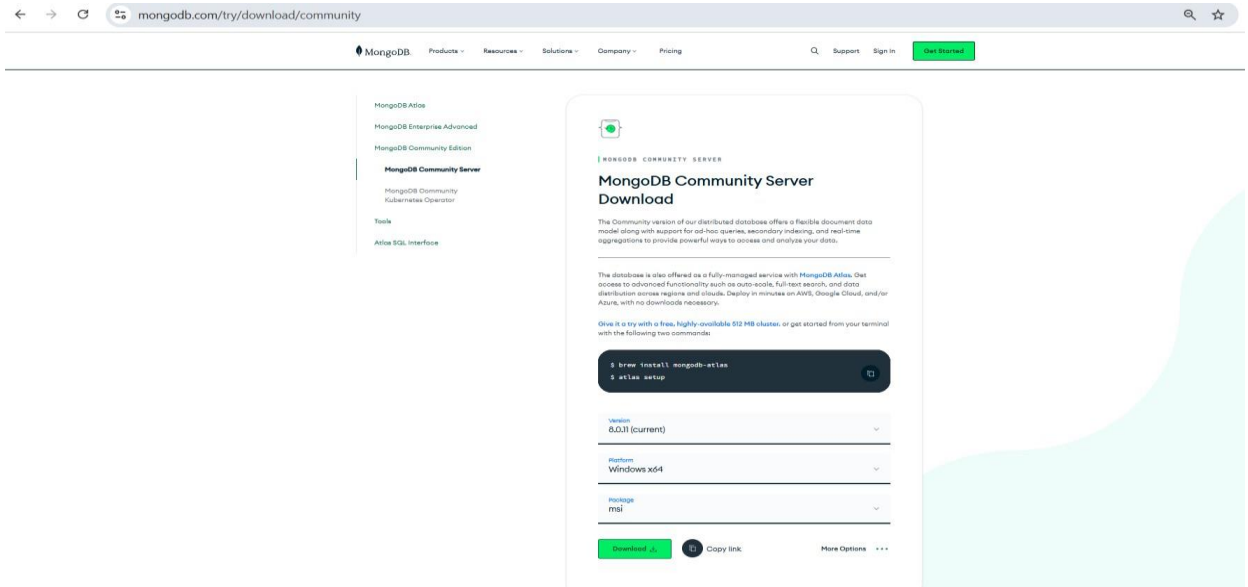
Step-1:

Search for MongoDB official website in browser



Step-2:

Search for MongoDB Community Server Edition and then select required version, Platform and Package and then click on Download button

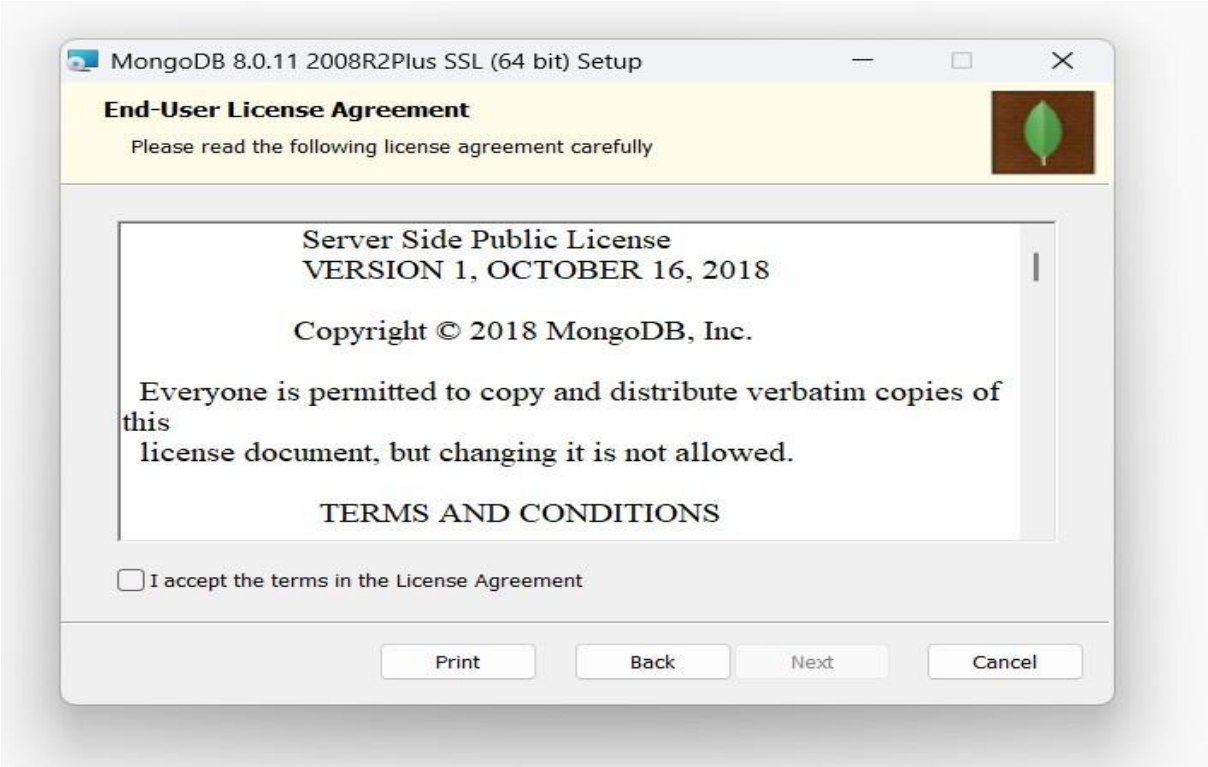


Step-3:

Double click on the MongoDB Installer to begin the Installation and click on Next button.

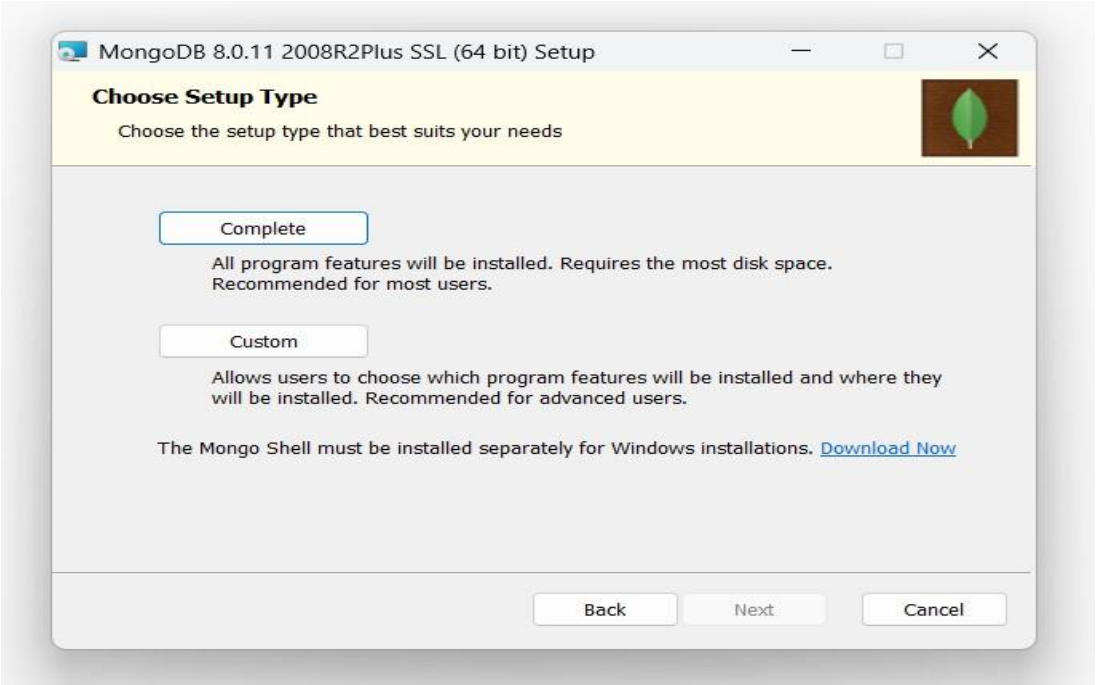
Step-4:

Accept the terms in the License Agreement by check marking the check box.



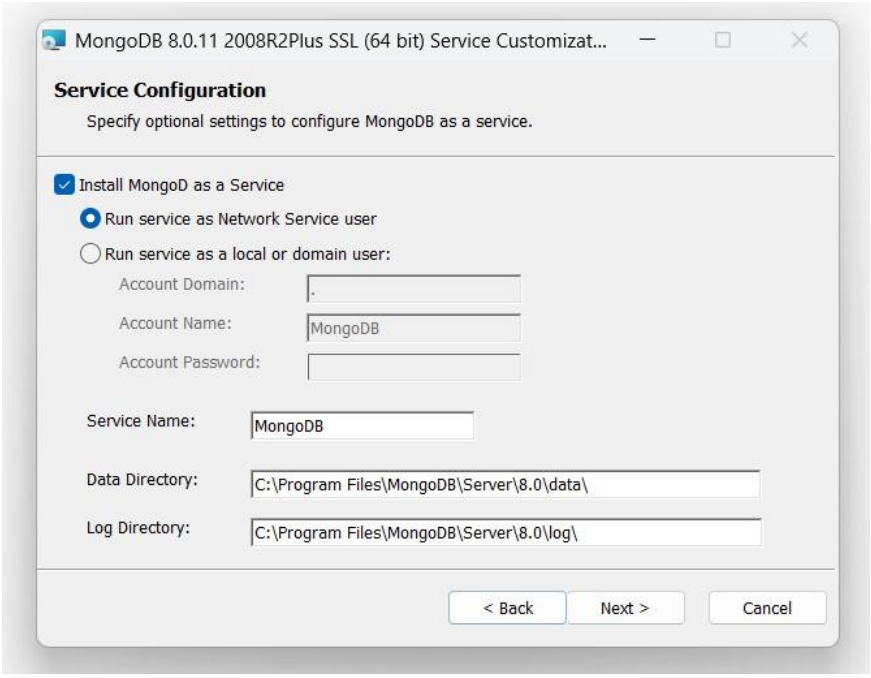
Step-5:

Select the Complete button to install all Program features.



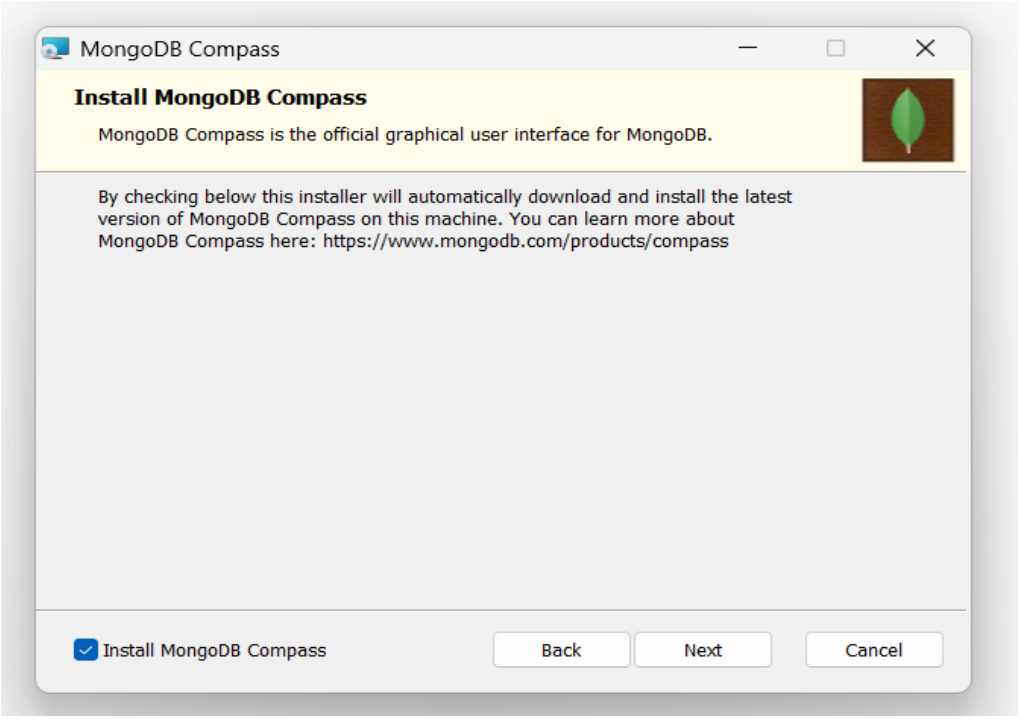
Step-6:

Select the below settings as shown in screenshot



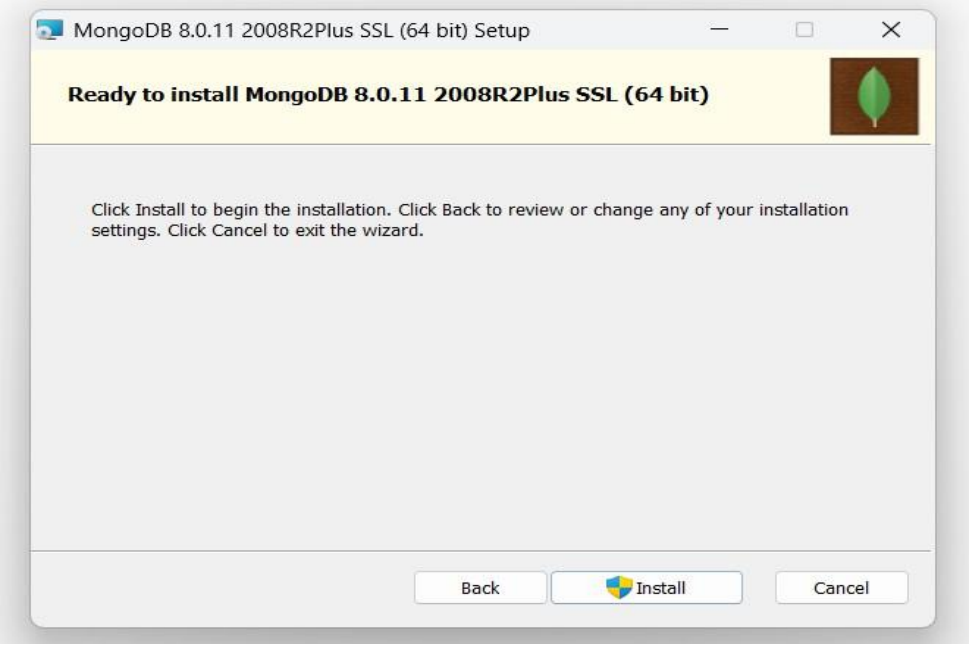
Step-7:

By check marking the check box of Install MongoDB Compass and then click on Next button



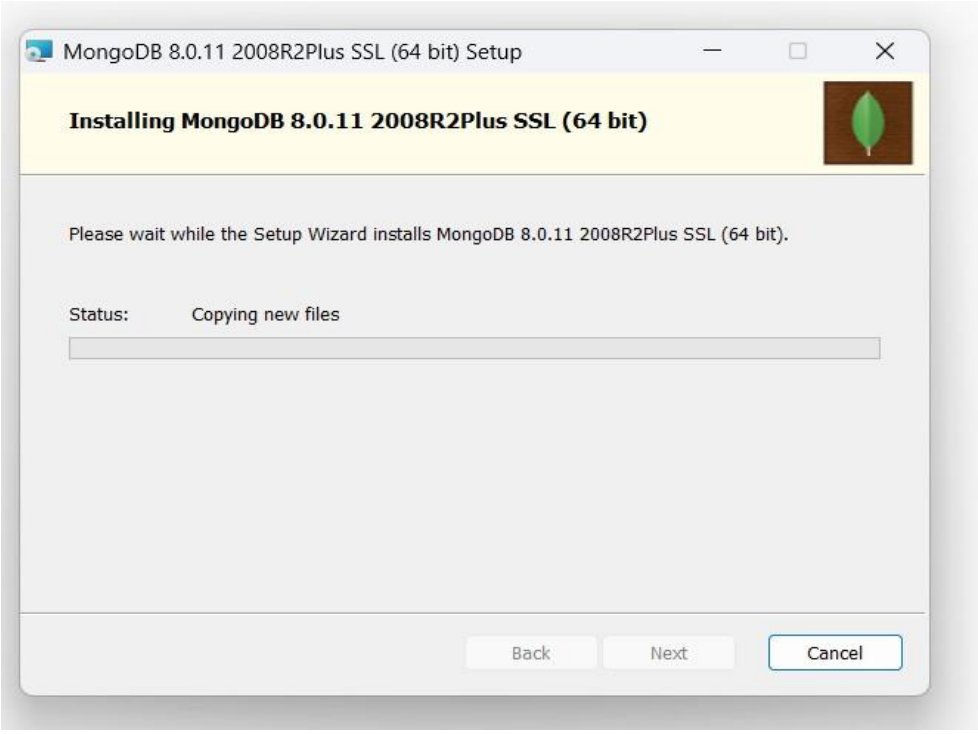
Step-8:

Click on Install button to begin the Installation.



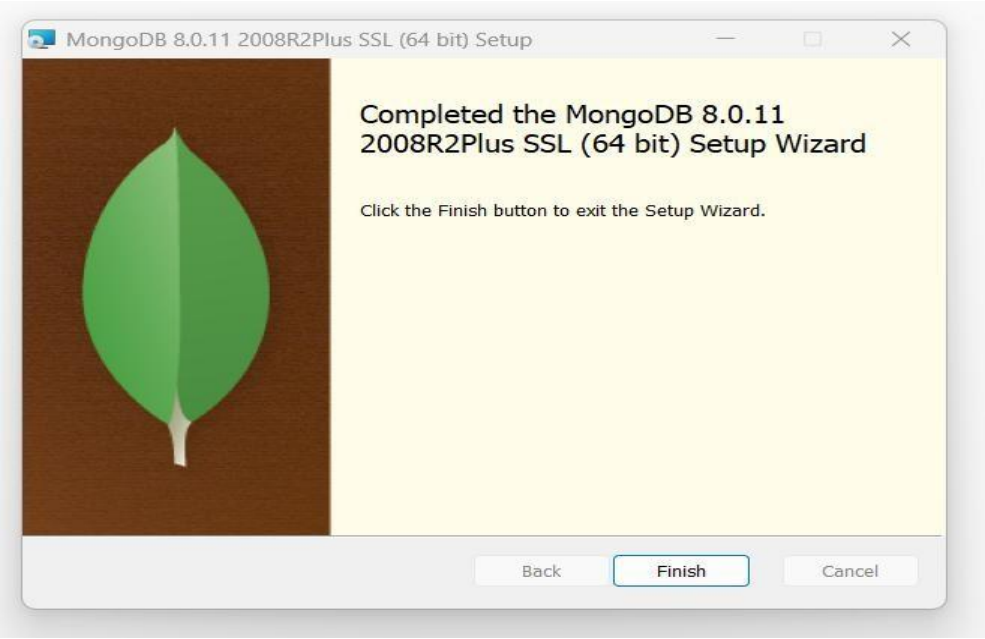
Step-9:

Installation Setup wizard will start installing.



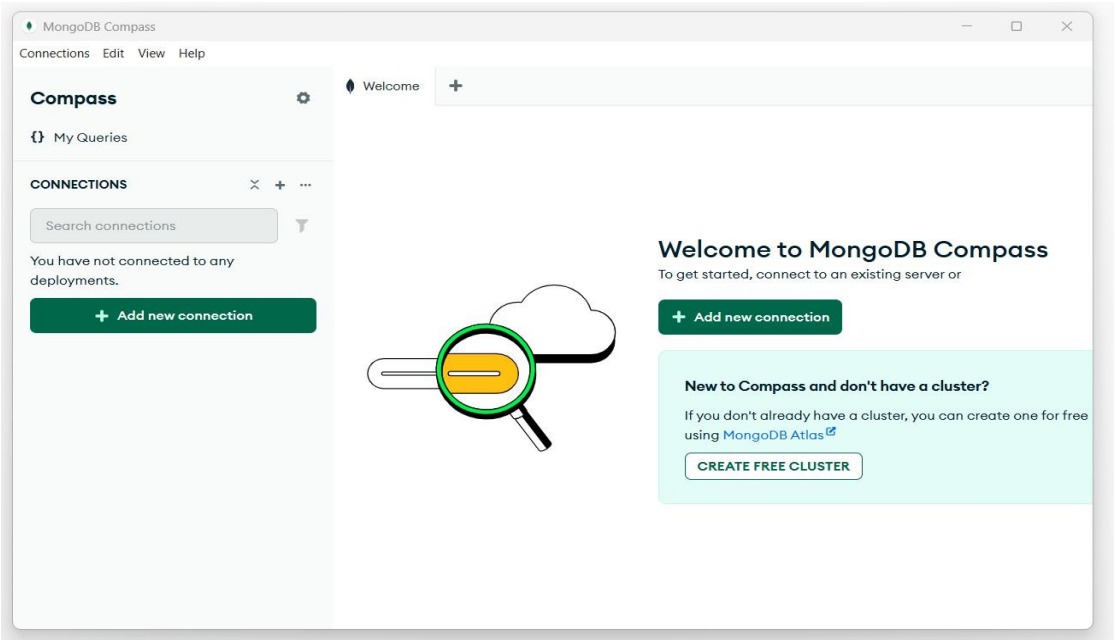
Step-10:

Click on Finish button to exit the Setup Wizard after completion of Installation.



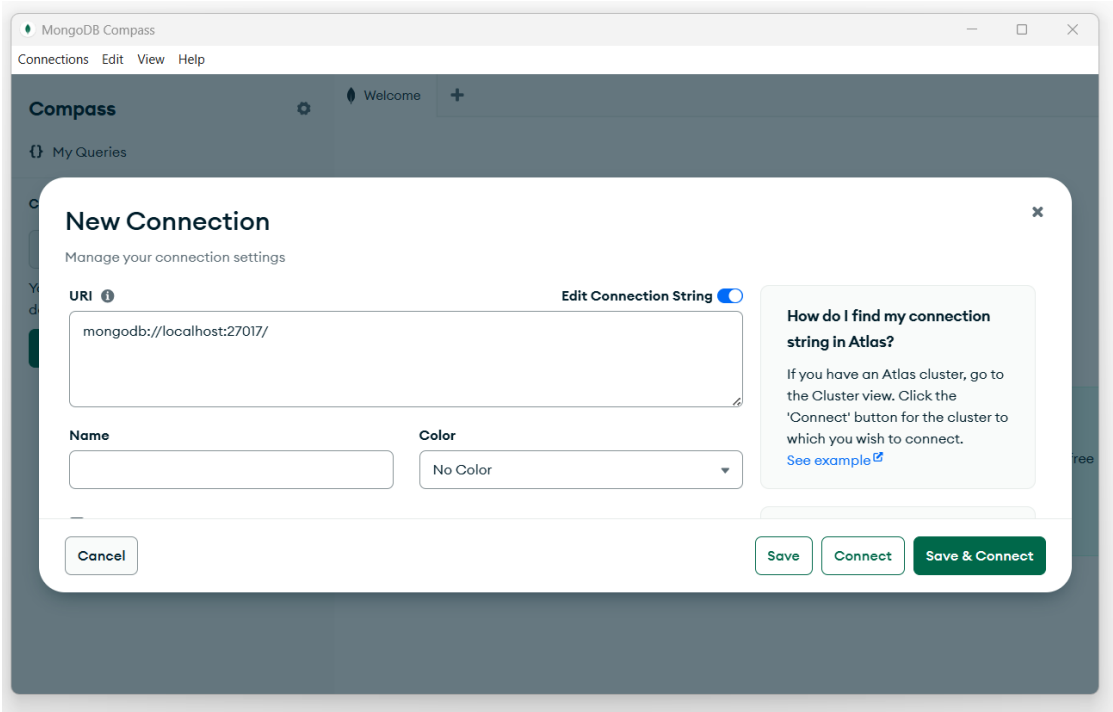
Step-11:

Open MongoDB compass application which provides GUI and then click on Add new connection button to establish new connection.



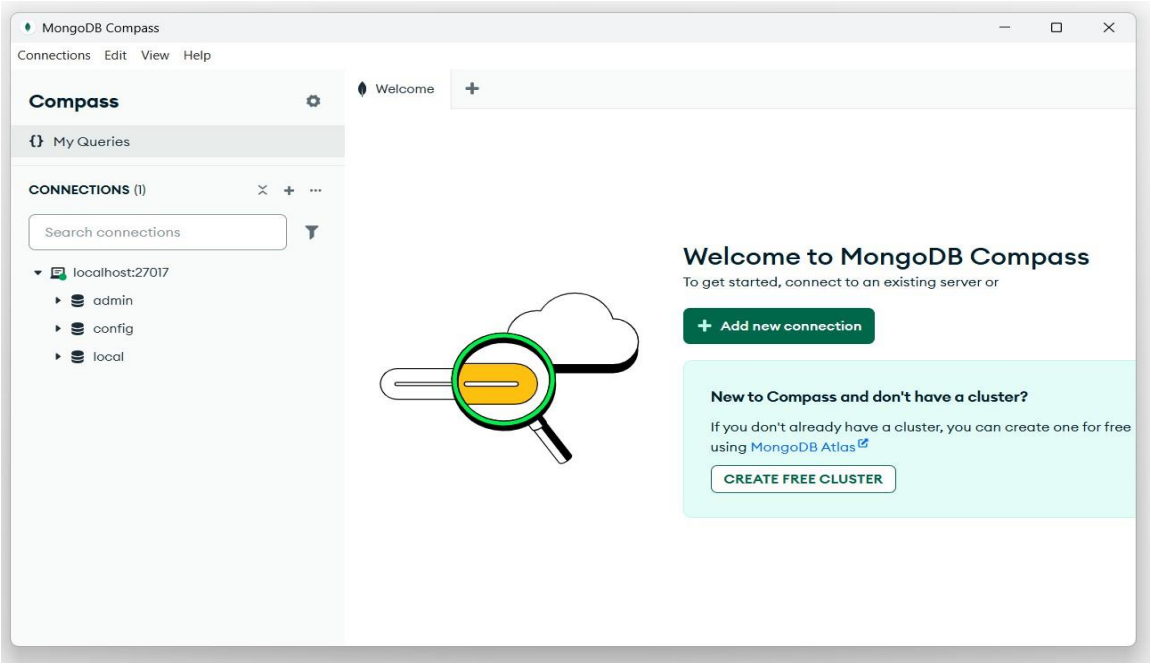
Step-12:

Click on Save & Connect button to save the connection settings as below.



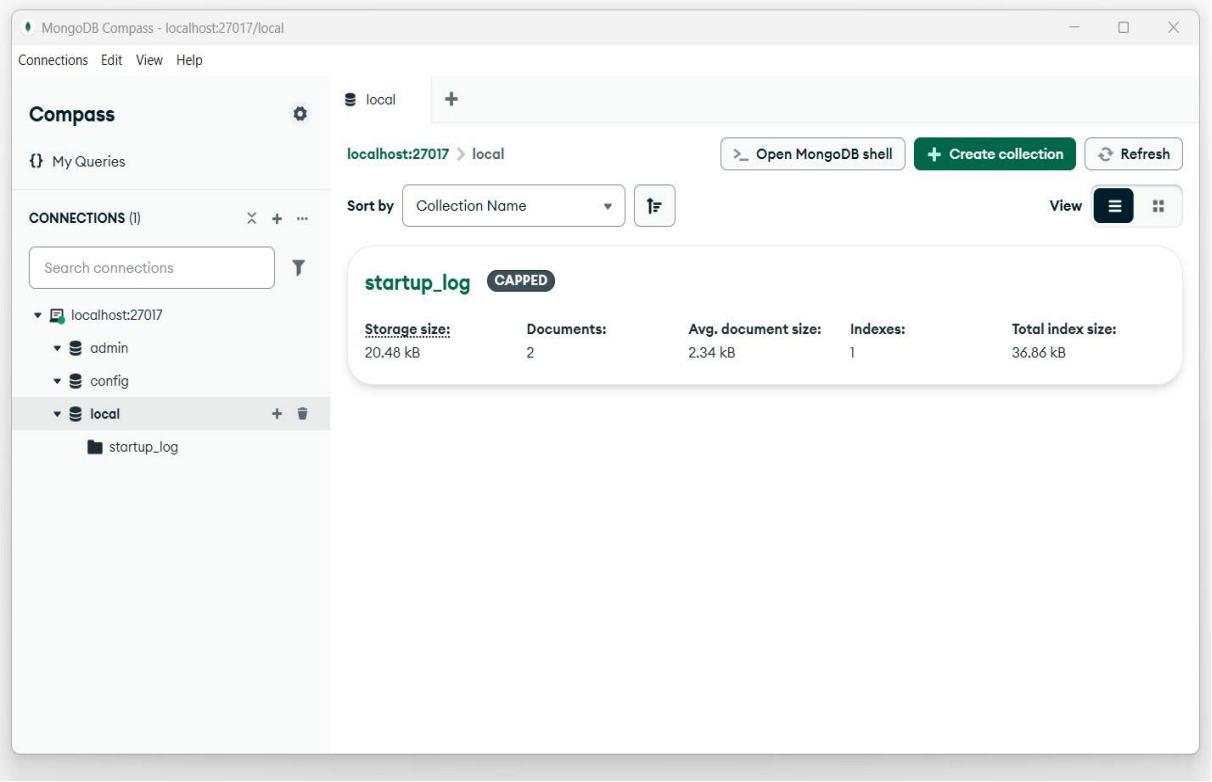
Step-13:

Now we can see the localhost connected to port 27017 under Connections.



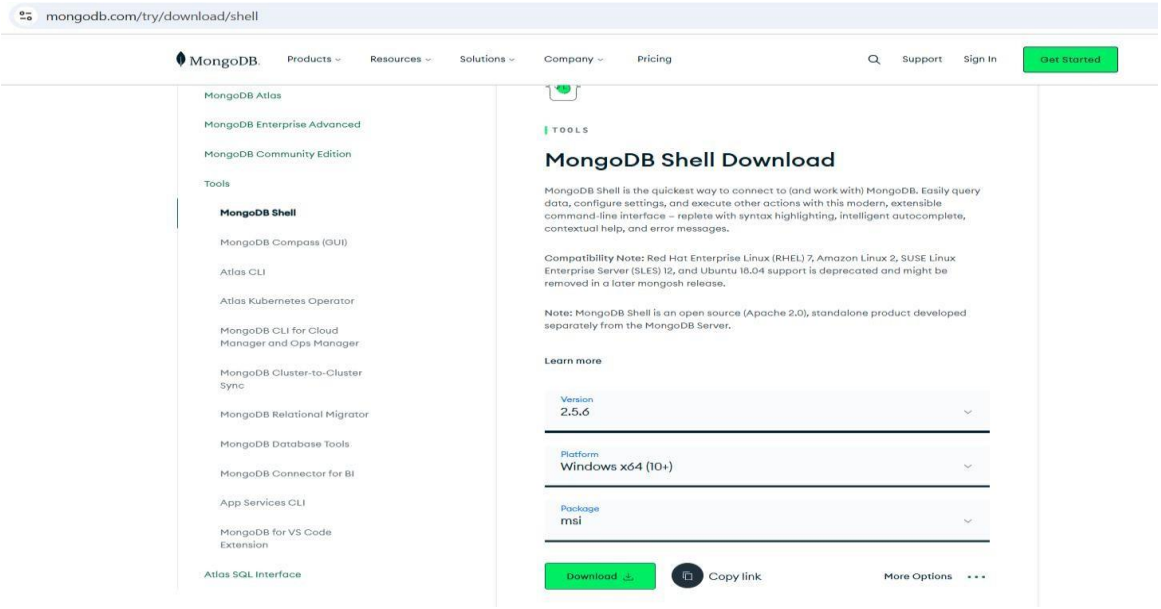
Step-14:

We can see details of the new localhost:27017 connection



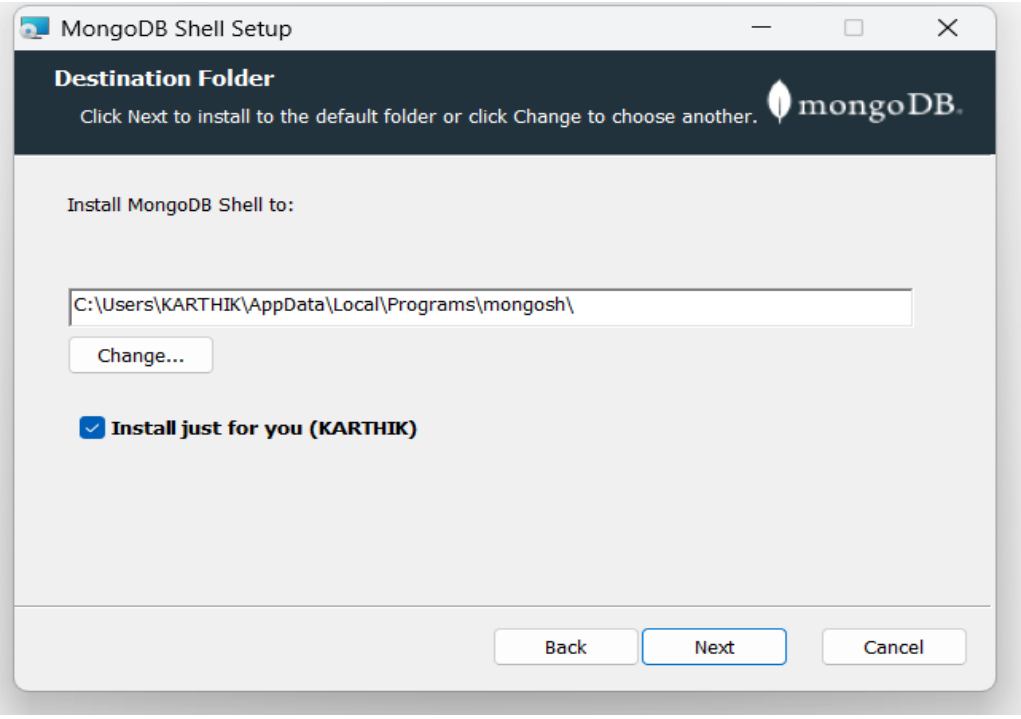
Step-15:

Search for MongoDB Shell under Tools in MongoDB website and then select required version, Platform and Package and then click on Download button.



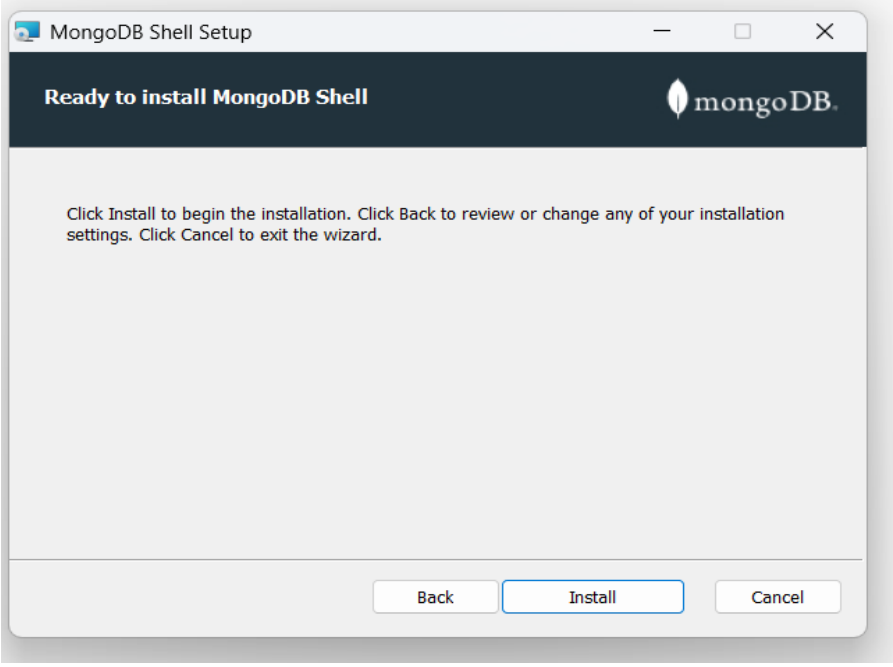
Step-16:

Select the Installation path and click on Next button



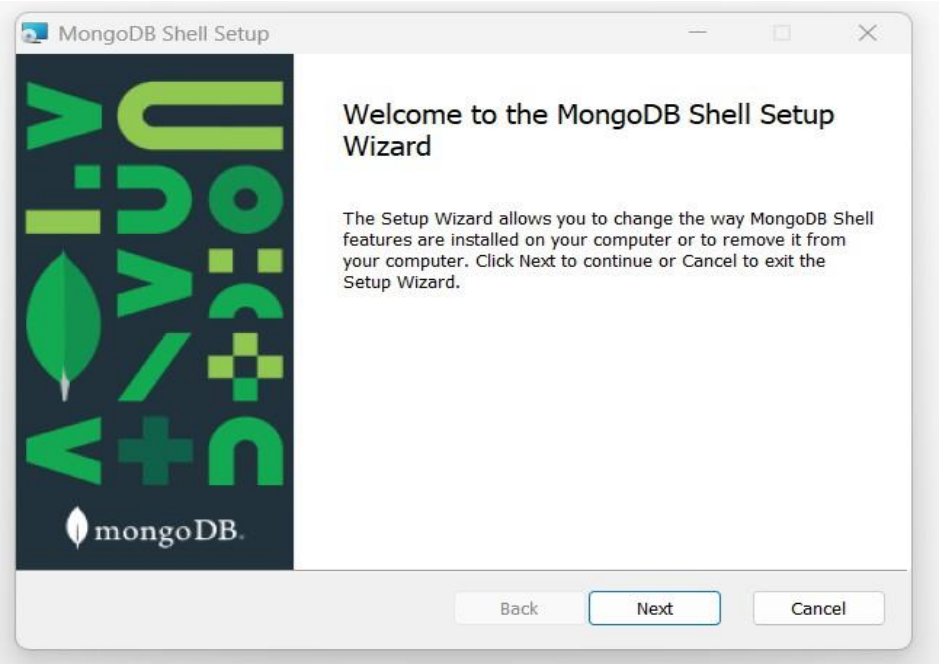
Step-17:

Click on Install button to begin the Installation.



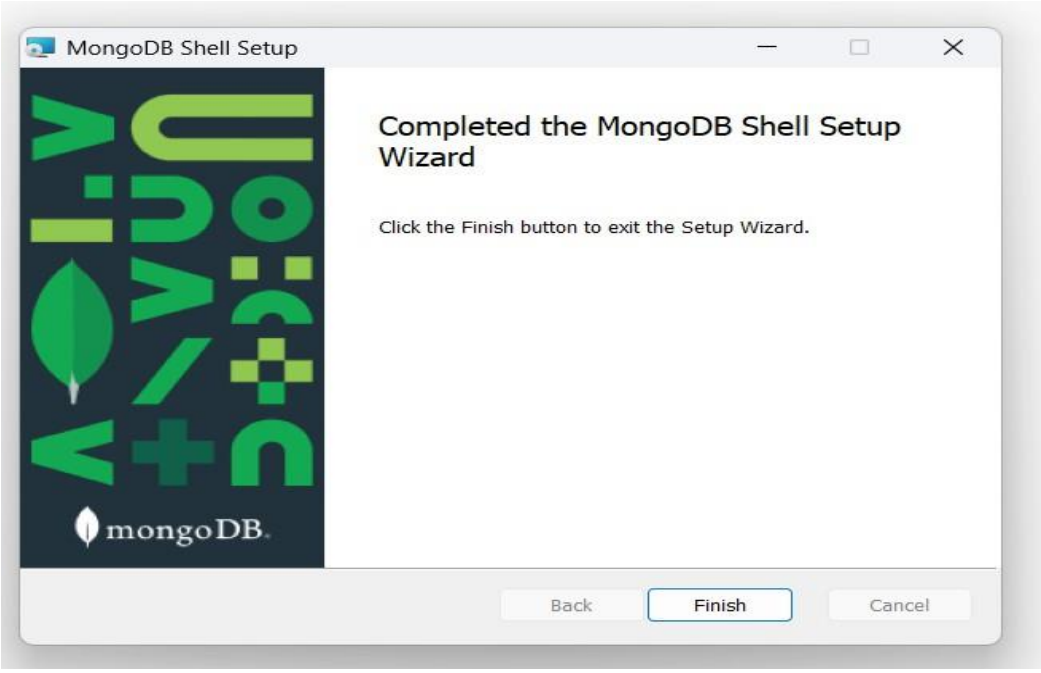
Step-18:

Click on Next button to continue the Installation.



Step-19:

Click on Finish button to exit the Setup Wizard after completion of Installation.



Step-20:-

After installation.

Now, the mongodb is installed on the path C:/Program Files/MongoDB/Server/3.2/bin. Instead of version 3.2, there could be some other version for your case. The path name would be changed accordingly.

bin directory contain several binary file along with mongod and mongo. To run it from other folder, you

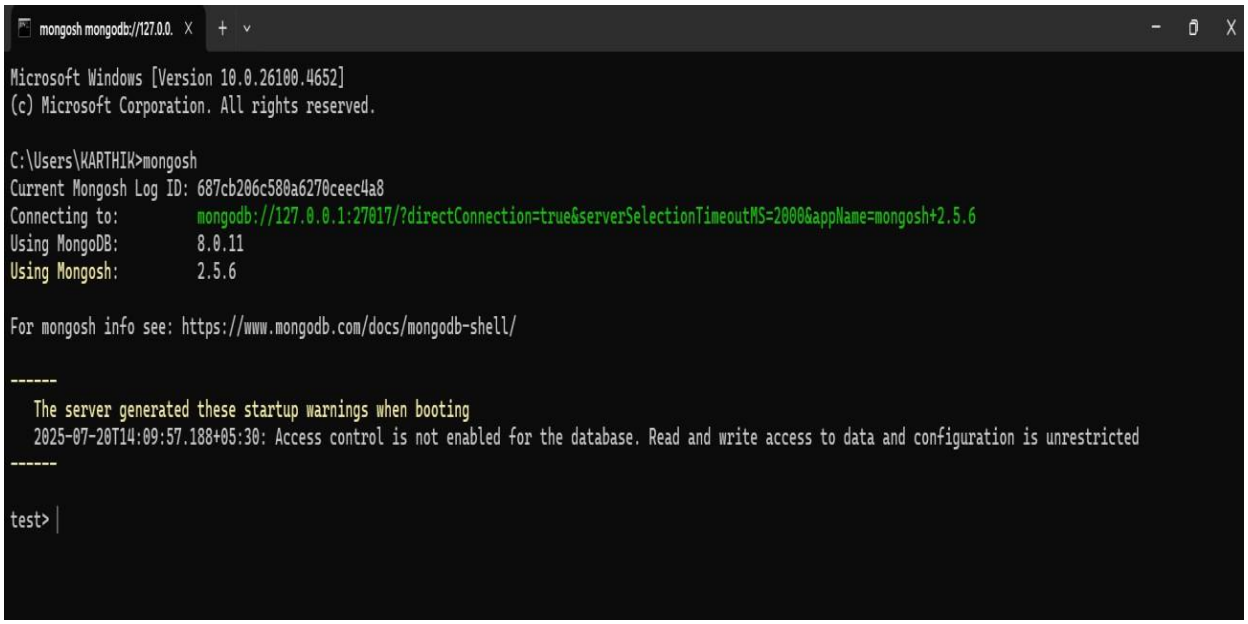
could add the path in system path. To do it:

- Right click on My Computer and select Properties.
- Click on Advanced system setting on the left pane.
- Click on Environment Variables... under the Advanced tab.
- Select Path from System variables section and click on Edit....
- Before Windows 10, append a semi-colon and paste the path given above. From Windows 10,
- there is a New button to add new path.
- Click OKs to save changes.

Start command prompt from their. Either changing the path in cmd or clicking on Open command

Step-21:

Open the command Prompt and type the command mongosh and hit enter button to check MongoDB connection establishment.



Experiment-1:

Aim: Create and Explore a NoSQL Document Structure

- Insert sample JSON documents to show flexible schemas

Procerdure:

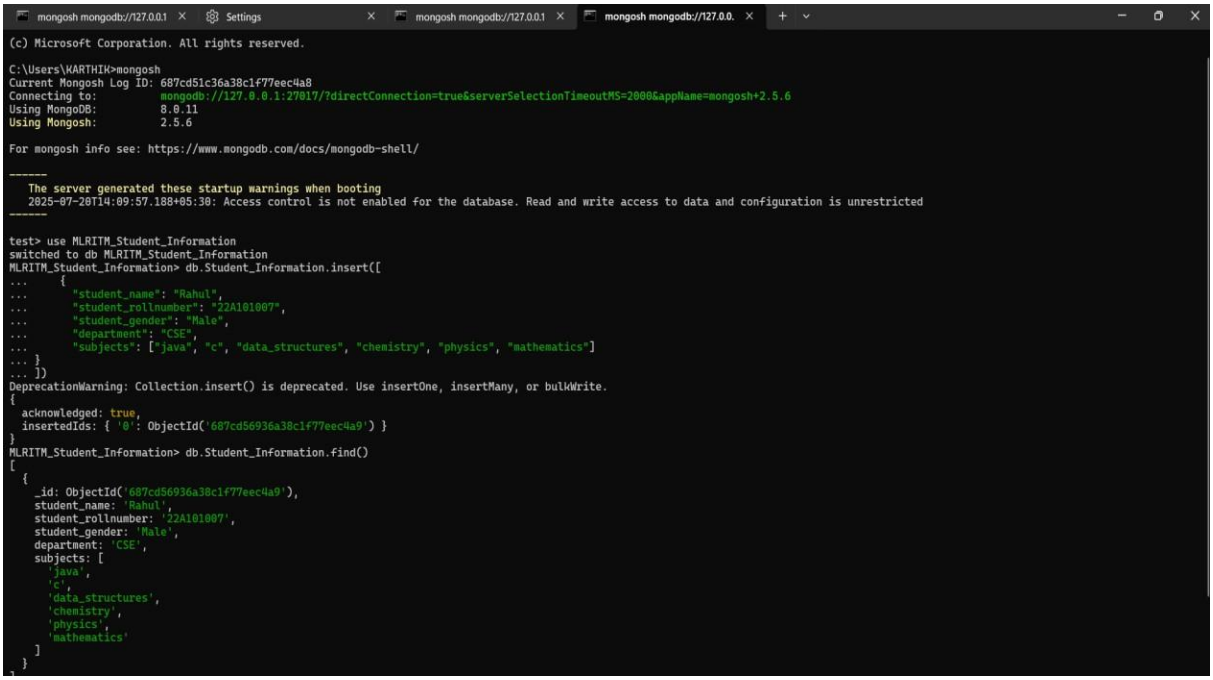
To create a database give command as following use MLRITM_Student_Information and to insert data into collection name Student_Information the following JSON code should be written.

Code:

```
db.Student_Information.insert([
   {
    "student_name": "Rahul",
    "student_rollnumber": "22A101007",
    "student_gender": "Male",
    "department": "CSE",
    "subjects": ["java", "c", "data_structures", "chemistry", "physics", "mathematics"]
   }
])
```

To find the inserted data we need to give following command db.Student_Information.find()

Output:



```
mongosh mongodb://127.0.0.1 | Settings | mongosh mongodb://127.0.0.1 | mongosh mongodb://127.0.0.

(c) Microsoft Corporation. All rights reserved.

C:\Users\KARTHIK>mongosh
Current Mongosh Log ID: 687cd51c36a38c1f77eec4a8
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.6
Using MongoDB:          8.0.11
Using Mongosh:          2.5.6

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-07-20T14:09:57.188+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

test> use MLRITM_Student_Information
switched to db MLRITM_Student_Information
MLRITM_Student_Information> db.Student_Information.insert([
...     {
...       "student_name": "Rahul",
...       "student_rollnumber": "22A101007",
...       "student_gender": "Male",
...       "department": "CSE",
...       "subjects": ["java", "c", "data_structures", "chemistry", "physics", "mathematics"]
... }
... ])
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId('687cd56936a38c1f77eec4a9') }
}
MLRITM_Student_Information> db.Student_Information.find()
[
  {
    _id: ObjectId('687cd56936a38c1f77eec4a9'),
    student_name: 'Rahul',
    student_rollnumber: '22A101007',
    student_gender: 'Male',
    department: 'CSE',
    subjects: [
      'java',
      'c',
      'data_structures',
      'chemistry',
      'physics',
      'mathematics'
    ]
  }
```

Experiment-2:

Aim: Compare RDBMS vs. MongoDB with a Practical Schema

- Model the same data (e.g., user accounts) in SQL and MongoDB.

Procerdure:

To create a database give command as following use Employee and to insert data into collection name Employee_Information for MongoDB
For RDBMS we use the query for writing in SQL for creating table Employees as

Code:

```
CREATE TABLE Employees (
   employee_number VARCHAR(10) PRIMARY KEY,
   employee_name VARCHAR(50) NOT NULL,
   employee_gender VARCHAR(10),
   department VARCHAR(50),
   salary DECIMAL(10, 2),
   address VARCHAR(255)
);
```

To insert data into the table we write below query

```
INSERT INTO Employees (employee_number, employee_name, employee_gender, department, salary, address) VALUES
('1001', 'Rahul', 'Male', 'Accounts', 50000.00, 'HYDERABAD'),
('1002', 'Rohit', 'Male', 'Cashier', 40000.00, 'HYDERABAD'),
('1003', 'Swetha', 'Female', 'Manager', 70000.00, 'HYDERABAD'),
('1004', 'Swapna', 'Female', 'Accounts', 50000.00, 'HYDERABAD');
```

- To retrieve all employee information:

```
SELECT * FROM Employees;
```

- To retrieve employees from a specific department:

```
SELECT * FROM Employees WHERE department = 'Accounts';
```

Similarly in MongoDB we use the following Json script

Code:

```
db.Employee_Information.insertMany([
  {
    "Employee_name": "Rahul",
    "Employee_number": "1001",
    "Employee_gender": "Male",
    "Department": "Accounts",
    "Salary": 50000,
    "Address": "HYDERABAD"
  },
  {
    "Employee_name": "Rohit",
    "Employee_number": "1002",
    "Employee_gender": "Male",
    "Department": "Cashier",
    "Salary": 40000,
    "Address": "HYDERABAD"
  },
  {
    "Employee_name": "Swetha",
    "Employee_number": "1003",
    "Employee_gender": "Female",
    "Department": "Manager",
    "Salary": 70000,
    "Address": "HYDERABAD"
  },
  {
```

```
    "Employee_name": "Swapna",
    "Employee_number": "1004",
    "Employee_gender": "Female",
    "Department": "Accounts",
    "Salary": 50000,
    "Address": "HYDERABAD"
  },
]);
```

- To retrieve all employee information:

```
db.Employee_Information.find();
```

- To retrieve employees from a specific department:

```
db.Employee_Information.find({ Department: "Accounts" });
```

Output:

```
mongosh mongodb://127.0.0.1 X   Settings X   mongosh mongodb://127.0.0.1 X   mongosh mongodb://127.0.0.1 X   mongosh mongodb://127.0.0. X

   The server generated these startup warnings when booting
   2025-07-20T14:09:57.188+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

test> use Employee
switched to db Employee
Employee> db.Employee_Information.insertMany([
...   {
...     "Employee_name": "Rahul",
...     "Employee_number": "1001",
...     "Employee_gender": "Male",
...     "Department": "Accounts",
...     "Salary": 50000,
...     "Address": "HYDERABAD"
...   },
...   {
...     "Employee_name": "Rohit",
...     "Employee_number": "1002",
...     "Employee_gender": "Male",
...     "Department": "Cashier",
...     "Salary": 40000,
...     "Address": "HYDERABAD"
...   },
...   {
...     "Employee_name": "Swetha",
...     "Employee_number": "1003",
...     "Employee_gender": "Female",
...     "Department": "Manager",
...     "Salary": 70000,
...     "Address": "HYDERABAD"
...   },
...   {
...     "Employee_name": "Swapna",
...     "Employee_number": "1004",
...     "Employee_gender": "Female",
...     "Department": "Accounts",
...     "Salary": 50000,
...     "Address": "HYDERABAD"
...   },
... ]);
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('687cdcdd88ffee49e3eec4a9'),
    '1': ObjectId('687cdcdd88ffee49e3eec4aa'),
    '2': ObjectId('687cdcdd88ffee49e3eec4ab'),
    '3': ObjectId('687cdcdd88ffee49e3eec4ac')
```

```
Employee> db.Employee_Information.find();
[
  {
    _id: ObjectId('687cdcdd88ffee49e3eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    Address: 'HYDERABAD'
  },
  {
    _id: ObjectId('687cdcdd88ffee49e3eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    Address: 'HYDERABAD'
  },
  {
    _id: ObjectId('687cdcdd88ffee49e3eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    Address: 'HYDERABAD'
  },
  {
    _id: ObjectId('687cdcdd88ffee49e3eec4ac'),
    Employee_name: 'Swapna',
    Employee_number: '1004',
    Employee_gender: 'Female',
    Department: 'Accounts',
    Salary: 50000,
    Address: 'HYDERABAD'
  }
]
Employee> db.Employee_Information.find({ Department: "Accounts" });
[
  {
    _id: ObjectId('687cdcdd88ffee49e3eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    Address: 'HYDERABAD'
  },
  {
    _id: ObjectId('687cdcdd88ffee49e3eec4ac'),
    Employee_name: 'Swapna',
    Employee_number: '1004',
    Employee_gender: 'Female',
    Department: 'Accounts',
    Salary: 50000,
    Address: 'HYDERABAD'
  }
```

Experiment-3:

Aim:Explore MongoDB Data Types

- Insert and query documents using types like String, NumberInt, Boolean, Array, Date, etc.

Procedure:

To create a database give command as following use Employee_1 and to insert data into collection name Employee_details for MongoDB.

MongoDB provides various query operators like $all, $gt, $lt, $lte and $gte to effectively filter and retrieve documents based on specific criteria within these data types.

- $all – filter is applied to all fields
- $gt – greater than
- $lt – less than
- $lte – less than equal
- $gte – greater than equal
- ISODate()- gives present date and time

Code:

```
db.Employee_details.insertMany([
  {
    "Employee_name": "Rahul",
    "Employee_number": "1001",
    "Employee_gender": "Male",
    "Department": "Accounts",
    "Salary":50000,
    "is_active": true,
    "Date": ISODate(),
    "Address": "HYDERABAD",
    "skills": ["SQL", "Excel", "Auditing"],
    "performance_rating": 4.5
}, {
```

"Employee_name": "Rohit",
"Employee_number": "1002",
"Employee_gender": "Male",
"Department": "Cashier",
"Salary":40000,
"is_active": true,
"Date": ISODate(),
"Address": "HYDERABAD",
"skills": ["Cash Handling", "Customer Service"],
"performance_rating": 4.0
},
{
"Employee_name": "Swetha",
"Employee_number": "1003",
"Employee_gender": "Female",
"Department": "Manager",
"Salary":70000,
"is_active": false,
"Date": ISODate(),
"Address": "HYDERABAD",
"skills": ["Leadership", "Project Management", "Reporting"],
"performance_rating": 4.8
},
{
"Employee_name": "Indu",
"Employee_number": "1005",

```
    "Employee_gender": "Female",
    "Department": "Cashier",
    "Salary":50000,
    "is_active": true,
    "Date": ISODate(),
    "Address": "HYDERABAD",
    "skills": ["Accounting Software", "Taxation","bills"],
    "performance_rating": 4.7
},
 {
    "Employee_name": "Sohail",
    "Employee_number": "1007",
    "Employee_gender": "Male",
    "Department": "Assistant_Manager",
    "Salary":62000,
    "is_active": false,
    "Date": ISODate(),
    "Address": "HYDERABAD",
    "skills": ["Leadership", "Project Management", "Reporting"],
    "performance_rating": 4.7
},
]);
```

Output:

```
test> use Employee3
switched to db Employee3
Employee3> db.Employee_details.insertMany([
...     {
...       "Employee_name": "Rahul",
...       "Employee_number": "1001",
...       "Employee_gender": "Male",
...       "Department": "Accounts",
...       "Salary":50000,
...       "is_active": true,
...       "Date": ISODate(),
...       "Address": "HYDERABAD",
...       "skills": ["SQL", "Excel", "Auditing"],
...       "performance_rating": 4.5
... }, {
...       "Employee_name": "Rohit",
...       "Employee_number": "1002",
...       "Employee_gender": "Male",
...       "Department": "Cashier",
...       "Salary":40000,
...       "is_active": true,
...       "Date": ISODate(),
...       "Address": "HYDERABAD",
...       "skills": ["Cash Handling", "Customer Service"],
...       "performance_rating": 4.0
... },
...  {
...       "Employee_name": "Swetha",
...       "Employee_number": "1003",
...       "Employee_gender": "Female",
...       "Department": "Manager",
...       "Salary":70000,
...       "is_active": false,
...       "Date": ISODate(),
...       "Address": "HYDERABAD",
...       "skills": ["Leadership", "Project Management", "Reporting"],
...       "performance_rating": 4.8
... },
...  {
...       "Employee_name": "Indu",
...       "Employee_number": "1005",
...       "Employee_gender": "Female",
...       "Department": "Cashier",
...       "Salary":50000,
...       "is_active": true,
...       "Date": ISODate(),
...       "Address": "HYDERABAD",
...       "skills": ["Accounting Software", "Taxation","bills"],
...       "performance_rating": 4.7
... },
...  {
...       "Employee_name": "Sohail",
...       "Employee_number": "1007",
...       "Employee_gender": "Male",
...       "Department": "Assistant_Manager",
...       "Salary":62000,
...       "is_active": false,
...       "Date": ISODate(),
...       "Address": "HYDERABAD",
...       "skills": ["Leadership", "Project Management", "Reporting"],
...       "performance_rating": 4.7
... },
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('687d091df6b37b6651eec4a9'),
    '1': ObjectId('687d091df6b37b6651eec4aa'),
```

Aim: To find an employee by their name:

Code: db.Employee_details.find({ Employee_name: "Rahul" });

Output:

```
Employee_1> db.Employee_details.find({ Employee_name: "Rahul" });
[
  {
    _id: ObjectId('687ce80f2a64f44de6eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  }
]
```

Aim: To find employees who are currently active:

Code: db.Employee_details.find({ is_active: true });

Output:

```
Employee_1> db.Employee_details.find({ is_active: true });
[
  {
    _id: ObjectId('687ce80f2a64f44de6eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687ce80f2a64f44de6eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687ce80f2a64f44de6eec4ac'),
    Employee_name: 'Swapna',
    Employee_number: '1004',
    Employee_gender: 'Female',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation' ],
    performance_rating: 4.2
  }
]
```

Aim: To find employees with a specific skill (e.g., "SQL"):

Code: db.Employee_details.find({ skills: "SQL" });

Output:

```
Employee_1> db.Employee_details.find({ skills: "SQL" });
[
  {
    _id: ObjectId('687ce80f2a64f44de6eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  }
]
Employee_1> db.Employee_details.find({ skills: { $all: ["Leadership", "Reporting"] } });
[
  {
    _id: ObjectId('687ce80f2a64f44de6eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  }
]
```

Aim: To find employees with multiple specific skills (e.g., "Leadership" and "Reporting"):

Code: db.Employee_details.find({ skills: { $all: ["Leadership", "Reporting"] } });

Output:

```
Employee_1> db.Employee_details.find({ skills: { $all: ["Leadership", "Reporting"] } });
[
  {
    _id: ObjectId('687ce80f2a64f44de6eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  }
]
```

Aim: To find employees who joined on a specific date:

Code: db.Employee_details.find({ Date: ISODate("2025-07-20T12:58:55.406Z") });

Output:

```
Employee_1> db.Employee_details.find({ Date: ISODate("2025-07-20T12:58:55.406Z") });
[
  {
    _id: ObjectId('687ce80f2a64f44de6eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687ce80f2a64f44de6eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687ce80f2a64f44de6eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687ce80f2a64f44de6eec4ac'),
    Employee_name: 'Swapna',
    Employee_number: '1004',
    Employee_gender: 'Female',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation' ],
    performance_rating: 4.2
  }
]
```

Aim: To find employees who joined within a specific date range:

Code: db.Employee_details.find({

```
  Date: {
    $gte: ISODate("2025-07-20T12:58:55.406Z"),
    $lte: ISODate("2025-07-21T12:58:55.406Z")
  }
});
```

Output:

```
Employee_1> db.Employee_details.find({
...   Date: {
...     $gte: ISODate("2025-07-20T12:58:55.406Z"),
...     $lte: ISODate("2025-07-21T12:58:55.406Z")
...   }
... });
[
  {
    _id: ObjectId('687ce80f2a64f44de6eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687ce80f2a64f44de6eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687ce80f2a64f44de6eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687ce80f2a64f44de6eec4ac'),
    Employee_name: 'Swapna',
    Employee_number: '1004',
    Employee_gender: 'Female',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation' ],
    performance_rating: 4.2
  }
]
Employee_1> db.Employee_details.find({ Salary: 50000 });
```

Aim: To find employees with a specific salary:

Code: db.Employee_details.find({ Salary: 50000 });

Output:

```
Employee_1> db.Employee_details.find({ Salary: 50000 });
[
  {
    _id: ObjectId('687ce80f2a64f44de6eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687ce80f2a64f44de6eec4ac'),
    Employee_name: 'Swapna',
    Employee_number: '1004',
    Employee_gender: 'Female',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation' ],
    performance_rating: 4.2
  }
]
```

Aim: To find employees with a salary greater than a specific amount (using the $gt operator):

Code: db.Employee_details.find({ Salary: { $gt: 60000 } });

Output:

```
Employee_1> db.Employee_details.find({ Salary: { $gt: 60000 } });
[
  {
    _id: ObjectId('687ce80f2a64f44de6eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  }
]
```

Aim: To find employees with a performance rating greater than or equal to 4.5 (using $gte):

Code: db.Employee_details.find({ performance_rating: { $gte: 4.5 } });

Output:

```
Employee_1> db.Employee_details.find({ performance_rating: { $gte: 4.5 } });
[
  {
    _id: ObjectId('687ce80f2a64f44de6eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687ce80f2a64f44de6eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T12:58:55.406Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  }
]
```

Experiment 4:

Aim: Basic MongoDB Data Modeling Example

- Design an embedded vs. referenced model for blog posts and comments.

Procedure:

To create a database give command as following use blogpost1 and to insert data into collection name posts_embedded for MongoDB.

Code:

A. Embedded Model:

```
blogpost> use blogpost1

switched to db blogpost1

blogpost1> db.posts_embedded.insertOne({

  title: "How to Learn MongoDB",

  author: "Jane Doe",

  content: "MongoDB is a NoSQL database...",

  created_at: ISODate("2025-07-18T10:00:00Z"),

  comments: [

    {

      commenter: "John Smith",

      comment: "Great article!",

      date: ISODate()

    },

    {

      commenter: "Alice Lee",

      comment: "Very helpful, thanks!",

      date: ISODate()

    }

  ] });
```

To Fetch Post with Comments:

Code: db.posts_embedded.findOne({ title: "How to Learn MongoDB" })

Output:

```
switched to db blogpost1
blogpost1> db.posts_embedded.insertOne({
...   title: "How to Learn MongoDB",
...   author: "Jane Doe",
...   content: "MongoDB is a NoSQL database...",
...   created_at: ISODate("2025-07-18T10:00:00Z"),
...   comments: [
...     {
...       commenter: "John Smith",
...       comment: "Great article!",
...       date: ISODate()
...
...     },
...     {
...       commenter: "Alice Lee",
...       comment: "Very helpful, thanks!",
...       date: ISODate()
...
...     }
...   ]
... });
{
  acknowledged: true,
  insertedId: ObjectId('687cfc31bfe3e5bfffeec4aa')
}
blogpost1> db.posts_embedded.findOne({ title: "How to Learn MongoDB" })
{
  _id: ObjectId('687cfc31bfe3e5bfffeec4aa'),
  title: 'How to Learn MongoDB',
  author: 'Jane Doe',
  content: 'MongoDB is a NoSQL database...',
  created_at: ISODate('2025-07-18T10:00:00.000Z'),
  comments: [
    {
      commenter: 'John Smith',
      comment: 'Great article!',
      date: ISODate('2025-07-20T14:24:49.835Z')
    },
    {
      commenter: 'Alice Lee',
      comment: 'Very helpful, thanks!',
      date: ISODate('2025-07-20T14:24:49.835Z')
    }
  ]
}
```

B. Referenced Model

```
const post = db.posts_referenced.insertOne({
  title: "How to Learn MongoDB",
  author: "Jane Doe",
  content: "MongoDB is a NoSQL database...",
  created_at: ISODate("2025-07-18T10:00:00Z")
});
```

- To Insert Comments (referencing post.insertedId)

Code:

```
db.comments.insertMany([
  {
    post_id: post.insertedId,
    commenter: "John Smith",
    comment: "Great article!",
    date: ISODate("2025-07-18T10:30:00Z")
  },
  {
    post_id: post.insertedId,
    commenter: "Alice Lee",
    comment: "Very helpful, thanks!",
    date: ISODate("2025-07-18T11:00:00Z")
  }
]);
```

Output:

```
blogpost1> const post = db.posts_referenced.insertOne({
...   title: "How to Learn MongoDB",
...   author: "Jane Doe",
...   content: "MongoDB is a NoSQL database...",
...   created_at: ISODate("2025-07-18T10:00:00Z")
... });

blogpost1> db.comments.insertMany([
...   {
...     post_id: post.insertedId,
...     commenter: "John Smith",
...     comment: "Great article!",
...     date: ISODate("2025-07-18T10:30:00Z")
...   },
...   {
...     post_id: post.insertedId,
...     commenter: "Alice Lee",
...     comment: "Very helpful, thanks!",
...     date: ISODate("2025-07-18T11:00:00Z")
...   }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('687cfc8bbfe3e5bfffeec4ac'),
    '1': ObjectId('687cfc8bbfe3e5bfffeec4ad')
  }
}
```

Aim:Use $lookup Aggregation (Join)

Code: db.posts_referenced.aggregate([

```
 { $match: { title: "How to Learn MongoDB" } },
 {
  $lookup: {
    from: "comments",
    localField: "_id",
    foreignField: "post_id",
    as: "comments"    }  }])
```

Output:

```
blogpost1> db.posts_referenced.aggregate([
...   { $match: { title: "How to Learn MongoDB" } },
...   {
...     $lookup: {
...       from: "comments",
...       localField: "_id",
...       foreignField: "post_id",
...       as: "comments"
...     }
...   }
... ])
[
  {
    _id: ObjectId('687cfc60bfe3e5bfffeec4ab'),
    title: 'How to Learn MongoDB',
    author: 'Jane Doe',
    content: 'MongoDB is a NoSQL database...',
    created_at: ISODate('2025-07-18T10:00:00.000Z'),
    comments: [
      {
        _id: ObjectId('687cfc8bbfe3e5bfffeec4ac'),
        post_id: ObjectId('687cfc60bfe3e5bfffeec4ab'),
        commenter: 'John Smith',
        comment: 'Great article!',
        date: ISODate('2025-07-18T10:30:00.000Z')
      },
      {
        _id: ObjectId('687cfc8bbfe3e5bfffeec4ad'),
        post_id: ObjectId('687cfc60bfe3e5bfffeec4ab'),
        commenter: 'Alice Lee',
        comment: 'Very helpful, thanks!',
        date: ISODate('2025-07-18T11:00:00.000Z')
      }
    ]
  }
]
```

VIVA Questions

S.No	Question	CO	Blooms Taxonomy
1	What is MongoDB?	CO1	Knowledge
2	What is NoSQL?	CO1	Knowledge
3	Name some types of NoSQL databases.	CO1	Knowledge
4	How does MongoDB differ from RDBMS?	CO1	Comprehension
5	What are some advantages of MongoDB over RDBMS?	CO1	Comprehension
6	What are the steps to install MongoDB on Windows?	CO1	Application
7	How do you start MongoDB from the terminal?	CO1	Application
8	What is the role of mongo shell?	CO1	Comprehension
9	What is a document in MongoDB?	CO1	Knowledge
10	How do you insert a document?	CO1	Application
11	What is a flexible schema?	CO1	Comprehension
12	List common MongoDB data types.	CO1	Knowledge
13	What is ObjectId?	CO1	Knowledge
14	How do you query by data type?	CO1	Application
15	How do you create a document?	CO1	Application
16	How do you read documents?	CO1	Application
17	How do you update a field?	CO1	Application
18	How do you delete a document?	CO1	Application
19	What is data modeling in MongoDB?	CO1	Knowledge
20	What is an embedded document?	CO1	Knowledge
21	What is a referenced document?	CO1	Knowledge
22	When do you use embedded vs. referenced?	CO1	Analysis
23	How is a user account represented in SQL?	CO1	Comprehension
24	How is the same user account represented in MongoDB?	CO1	Comprehension
25	Why is MongoDB preferred for fast prototyping?	CO1	Evaluation
26	What is a collection?	CO1	Knowledge
27	Can you change a document's schema later?	CO1	Evaluation
28	What are indexes in MongoDB?	CO1	Comprehension
29	What is db in the Mongo shell?	CO1	Knowledge
30	How do you list all collections in a database?	CO1	Application

MODULE - II

Operators and Commands

Experiment 5: Use Query and Projection Operators

- Demonstrate $eq, $gt, $lt, $in, $and, $or, $exists, and projection { field: 1 }.

Procedure:

Use Employee_details Collection to perform the below operations

$eq (equal to)

Aim: Find employees with a salary of 50000

Code: db.Employee_details.find({ "Salary": { $eq: 50000 } });

```
Employee3> db.Employee_details.find({ "Salary": { $eq: 50000 } });
[
  {
    _id: ObjectId('687d091df6b37b6651eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4ac'),
    Employee_name: 'Indu',
    Employee_number: '1005',
    Employee_gender: 'Female',
    Department: 'Cashier',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ],
    performance_rating: 4.7
  }
]
Employee3>
```

$gt (greater than)

Aim: Find employees with a salary greater than 50000

Code: db.Employee_details.find({ "Salary": { $gt: 50000 } });

```
Employee3> db.Employee_details.find({ "Salary": { $gt: 50000 } });
[
  {
    _id: ObjectId('687d091df6b37b6651eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.7
  }
]
Employee3>
```

$lt (less than)

Aim: Find employees with a salary less than 50000:

Code: db.Employee_details.find({ "Salary": { $lt: 50000 } });

```
Employee3> db.Employee_details.find({ "Salary": { $lt: 50000 } });
[
  {
    _id: ObjectId('687d091df6b37b6651eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  }
]
Employee3> |
```

$in (in array)

Aim: Find employees in the "Accounts" or "Cashier" departments:

Code: db.Employee_details.find({ "Department": { $in: ["Accounts", "Cashier"] } });

```
Employee3> db.Employee_details.find({ "Department": { $in: ["Accounts", "Cashier"] } });
[
  {
    _id: ObjectId('687d091df6b37b6651eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4ac'),
    Employee_name: 'Indu',
    Employee_number: '1005',
    Employee_gender: 'Female',
    Department: 'Cashier',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ],
    performance_rating: 4.7
  }
]
Employee3>
```

$and (logical AND)

Aim: Find active employees with a salary greater than 50000:

Code: db.Employee_details.find({

```
  $and: [
    { "Employee_gender": "Male" },
    { "Salary": { $gt: 50000 } }
  ]
});
```

```
Employee3> db.Employee_details.find({
...     $and: [
...         { "Employee_gender": "Male" },
...         { "Salary": { $gt: 50000 } }
...     ]
... });
[
  {
    _id: ObjectId('687d091df6b37b6651eec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.7
  }
]
Employee3>
```

$or (logical OR)

Aim: Find male employees or employees in the "Manager" department:

Code: db.Employee_details.find({

```
  $or: [
      { "Employee_gender": "Male" },
      { "Department": "Manager" }
  ]
});
```

```
Employee3> db.Employee_details.find({
...     $or: [
...         { "Employee_gender": "Male" },
...         { "Department": "Manager" }
...     ]
... });
[
  {
    _id: ObjectId('687d091df6b37b6651eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.7
  }
]
Employee3>
```

$exists

Aim: Find employees with the "skills" field:

Code: db.Employee_details.find({ "skills": { $exists: true } });

```
Employee3> db.Employee_details.find({ "skills": { $exists: true } });
[
  {
    _id: ObjectId('687d091df6b37b6651eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4ac'),
    Employee_name: 'Indu',
    Employee_number: '1005',
    Employee_gender: 'Female',
    Department: 'Cashier',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687d091df6b37b6651eec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-20T15:19:57.012Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.7
  }
```

projection { field: 1 }

Aim: Display the Employee_name and Department for employees in the "Accounts" or "Cashier" departments:

Code: db.Employee_details.find(

{ "Department": { $in: ["Accounts", "Cashier"] } },

{ "Employee_name": 1, "Department": 1, "_id": 0 }

);

```
Employee3> db.Employee_details.find(
...     { "Department": { $in: ["Accounts", "Cashier"] } },
...     { "Employee_name": 1, "Department": 1, "_id": 0 }
... );
[
  { Employee_name: 'Rahul', Department: 'Accounts' },
  { Employee_name: 'Rohit', Department: 'Cashier' },
  { Employee_name: 'Indu', Department: 'Cashier' }
]
Employee3>
```

Aim: Find documents where the "skills" field exists and project only Employee_name, Employee_number, and skills:

Code: db.Employee_details.find(

{ "skills": { $exists: true } },

{ "Employee_name": 1, "Employee_number": 1, "skills": 1, "_id": 0 });

```
Employee3> db.Employee_details.find(
...     { "skills": { $exists: true } },
...     { "Employee_name": 1, "Employee_number": 1, "skills": 1, "_id": 0 }
... );
[
  {
    Employee_name: 'Rahul',
    Employee_number: '1001',
    skills: [ 'SQL', 'Excel', 'Auditing' ]
  },
  {
    Employee_name: 'Rohit',
    Employee_number: '1002',
    skills: [ 'Cash Handling', 'Customer Service' ]
  },
  {
    Employee_name: 'Swetha',
    Employee_number: '1003',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ]
  },
  {
    Employee_name: 'Indu',
    Employee_number: '1005',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ]
  },
  {
    Employee_name: 'Sohail',
    Employee_number: '1007',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ]
  }
]
Employee3>
```


```
Employee3> db.Employee_details.find(
...     { "skills": { $exists: true } },
...     { "Employee_name": 1, "Employee_number": 1, "skills": 1, "_id": 0 }
... );
[
  {
    Employee_name: 'Rahul',
    Employee_number: '1001',
    skills: [ 'SQL', 'Excel', 'Auditing' ]
  },
  {
    Employee_name: 'Rohit',
    Employee_number: '1002',
    skills: [ 'Cash Handling', 'Customer Service' ]
  },
  {
    Employee_name: 'Swetha',
    Employee_number: '1003',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ]
  },
  {
    Employee_name: 'Indu',
    Employee_number: '1005',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ]
  },
  {
    Employee_name: 'Sohail',
    Employee_number: '1007',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ]
  }
]
Employee3>
```

Experiment 6:

Aim: Update Operators and Aggregation Stages

- Use $set, $inc, $push in update, and pipeline stages like $match, $group, $sort.

Procedure:

Use Employee_details Collection to perform the following operations

Update operations:

$set:

Aim: To update an existing field or add a new one. For example, let's update Rahul's Department to "Finance":

Code: db.Employee_details.updateOne(

{"Employee_name": "Rahul"},

{$set: {"Department": "Finance"}}

)

```
Employee4> db.Employee_details.updateOne(
...     {"Employee_name": "Rahul"},
...     {$set: {"Department": "Finance"}}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Employee4>
```

$inc:

Aim: To increment a numeric field. For example, let's give Indu a $5000 salary raise:

Code: db.Employee_details.updateOne(

{"Employee_name": "Indu"},

{$inc: {"Salary": 5000}}

)

```
Employee4> db.Employee_details.updateOne(
...     {"Employee_name": "Indu"},
...     {$inc: {"Salary": 5000}}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Employee4> |
```

$push:

Aim: To add an element to an array field. Let's add "Communication" as a skill for Sohail:

Code: db.Employee_details.updateOne(

{"Employee_name": "Sohail"},

{$push: {"skills": "Communication"}}

)

```
Employee4> db.Employee_details.updateOne(
...     {"Employee_name": "Sohail"},
...     {$push: {"skills": "Communication"}}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Employee4>
```

Aggregation pipeline stages:

$match:

Aim: To filter documents based on specified criteria. For example, let's find all employees who are Male and have a performance rating greater than or equal to 4.5:

Code: db.Employee_details.aggregate([

{$match: {"Employee_gender": "Male", "performance_rating": {$gte: 4.5}}}])

```
Employee4> db.Employee_details.aggregate([
...     {$match: {"Employee_gender": "Male", "performance_rating": {$gte: 4.5}}}
... ])
[
  {
    _id: ObjectId('687d126b17f8149631eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Finance',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T15:59:39.812Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687d126b17f8149631eec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-20T15:59:39.812Z'),
    Address: 'HYDERABAD',
    skills: [
      'Leadership',
      'Project Management',
      'Reporting',
      'Communication'
    ],
    performance_rating: 4.7
  }
]
Employee4> |
```

$group:

Aim: To group the documents by a specified field. For example, let's group employees by Department and Average Salary

Code: db.Employee_details.aggregate([

{$group: {"_id": "$Department", "averageSalary": {$avg: "$Salary"}}}

])

```
Employee4> db.Employee_details.aggregate([
...     {$group: {"_id": "$Department", "averageSalary": {$avg: "$Salary"}}}
... ])
[
  { _id: 'Finance', averageSalary: 50000 },
  { _id: 'Manager', averageSalary: 70000 },
  { _id: 'Cashier', averageSalary: 47500 },
  { _id: 'Assistant_Manager', averageSalary: 62000 }
]
Employee4> |
```

$sort:

Aim: To sort the documents by a specified field. For example, let's sort employees by performance_rating in descending order:

Code: db.Employee_details.aggregate([

{$sort: {"performance_rating": -1}}])

```
]
Employee4> db.Employee_details.aggregate([
...     {$sort: {"performance_rating": -1}}
... ])
[
  {
    _id: ObjectId('687d126b17f8149631eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T15:59:39.812Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ]
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687d126b17f8149631eec4ac'),
    Employee_name: 'Indu',
    Employee_number: '1005',
    Employee_gender: 'Female',
    Department: 'Cashier',
    Salary: 55000,
    is_active: true,
    Date: ISODate('2025-07-20T15:59:39.812Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687d126b17f8149631eec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-20T15:59:39.812Z'),
    Address: 'HYDERABAD',
    skills: [
      'Leadership',
      'Project Management',
      'Reporting',
      'Communication'
    ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687d126b17f8149631eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Finance',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T15:59:39.812Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687d126b17f8149631eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T15:59:39.812Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  }
]
Employee4>
```

Match and Group: The aggregation pipeline to use Employee_gender in the $match stage.let's find the group of average salary of employees in each Department, and then sort the result by average salary:

Code: db.Employee_details.aggregate([

{$match: {"Employee_gender": "Female"}}, // Filter for employees where gender is Male

{$group: {"_id": "$Department", "averageSalary": {$avg: "$Salary"}}},

{$sort: {"averageSalary": -1}}

])

```
Employee4> db.Employee_details.aggregate([
...     {$match: {"Employee_gender": "Female"}}, // Filter for employees where gender is Male
...     {$group: {"_id": "$Department", "averageSalary": {$avg: "$Salary"}}},
...     {$sort: {"averageSalary": -1}}
... ])
[
  { _id: 'Manager', averageSalary: 70000 },
  { _id: 'Cashier', averageSalary: 55000 }
]
Employee4>
```

Experiment 7:

Aim: Sorting, Limiting, and Modifying Queries

- Apply .limit(), .sort(), and modifiers like .explain(), .hint().

Procedure:

Use Employee_details Collection to Perfom the operations

Sorting (.sort())

The .sort() method sorts the results of a query in ascending (1) or descending (-1) order based on specified fields.

Aim: Sort by Salary (ascending):

Code: db.Employee_details.find().sort({"Salary": 1})

```
Employee5> db.Employee_details.find().sort({"Salary": 1})
[
  {
    _id: ObjectId('687d27302e8e0507cdeec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687d27302e8e0507cdeec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687d27302e8e0507cdeec4ac'),
    Employee_name: 'Indu',
    Employee_number: '1005',
    Employee_gender: 'Female',
    Department: 'Cashier',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687d27302e8e0507cdeec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687d27302e8e0507cdeec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  }
]
```

Sort by performance_rating (descending) and then Employee_name (ascending):

Code: db.Employee_details.find().sort({"performance_rating": -1, "Employee_name": 1})

```
Employee5> db.Employee_details.find().sort({"performance_rating": -1, "Employee_name": 1})
[
  {
    _id: ObjectId('687d27302e8e0507cdeec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687d27302e8e0507cdeec4ac'),
    Employee_name: 'Indu',
    Employee_number: '1005',
    Employee_gender: 'Female',
    Department: 'Cashier',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687d27302e8e0507cdeec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687d27302e8e0507cdeec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687d27302e8e0507cdeec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  }
]
```

Limiting Results (.limit())

The .limit() method restricts the number of documents returned by a query.

Retrieve the top 3 highest-paid employees:

Code: db.Employee_details.find().sort({"Salary": -1}).limit(3)

```
Employee5> db.Employee_details.find().sort({"Salary": -1}).limit(3)
[
  {
    _id: ObjectId('687d27302e8e0507cdeec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687d27302e8e0507cdeec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687d27302e8e0507cdeec4ac'),
    Employee_name: 'Indu',
    Employee_number: '1005',
    Employee_gender: 'Female',
    Department: 'Cashier',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T17:28:16.864Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ],
    performance_rating: 4.7
  }
]
Employee5>
```

Modifiers

.explain() - Analyzing query execution plans:The .explain() method provides detailed information about the query execution plan, helping to understand how MongoDB processes the query and whether indexes are used efficiently.

Code: db.Employee_details.find({"Department": "Cashier"}).explain("executionStats")

```
Employee5> db.Employee_details.find({"Department": "Cashier"}).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'Employee5.Employee_details',
    parsedQuery: { Department: { '$eq': 'Cashier' } },
    indexFilterSet: false,
    queryHash: '9E4738E0',
    planCacheShapeHash: '9E4738E0',
    planCacheKey: 'AAA567C0',
    optimizationTimeMillis: 5,
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    prunedSimilarIndexes: false,
    winningPlan: {
      isCached: false,
      stage: 'COLLSCAN',
      filter: { Department: { '$eq': 'Cashier' } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 2,
    executionTimeMillis: 8,
    totalKeysExamined: 0,
    totalDocsExamined: 5,
    executionStages: {
      isCached: false,
      stage: 'COLLSCAN',
      filter: { Department: { '$eq': 'Cashier' } },
      nReturned: 2,
      executionTimeMillisEstimate: 0,
      works: 6,
      advanced: 2,
      needTime: 3,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      direction: 'forward',
      docsExamined: 5
    }
  },
  queryShapeHash: 'C5C8F443E8AC3A2ACE89CA8424ACCFD9BD858814EFEFD26F43D45C82736A5038',
  command: {
    find: 'Employee_details',
    filter: { Department: 'Cashier' },
    '$db': 'Employee5'
  },
  serverInfo: {
    host: 'DESKTOP-9QG6EM7',
    port: 27017,
    version: '8.0.11',
    gitVersion: 'bed99f699da6cb2b74262aa6d473446c41476643'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'trySbeRestricted',
```

.hint() - Forcing index usage (for advanced optimization and testing):While MongoDB's query optimizer usually chooses the most efficient index, .hint() allows you to manually specify which index to use for a particular query. This can be beneficial for performance testing and in scenarios where the optimizer may not select the optimal index by default.First, you might create an index, for instance on the Department field:

Code: db.Employee_details.createIndex({"Department": 1})

Code: db.Employee_details.find({"Department":

"Accounts"}).hint({"Department": 1}).explain("executionStats")

```
Employee5> db.Employee_details.createIndex({"Department": 1})
Department_1
Employee5> db.Employee_details.find({"Department": "Accounts"}).hint({"Department": 1}).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'Employee5.Employee_details',
    parsedQuery: { Department: { '$eq': 'Accounts' } },
    indexFilterSet: false,
    queryHash: '9E4738E0',
    planCacheShapeHash: '9E4738E0',
    planCacheKey: '933F39E8',
    optimizationTimeMillis: 10,
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    prunedSimilarIndexes: false,
    winningPlan: {
      isCached: false,
      stage: 'FETCH',
      inputStage: {
        stage: 'IXSCAN',
        keyPattern: { Department: 1 },
        indexName: 'Department_1',
        isMultiKey: false,
        multiKeyPaths: { Department: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { Department: [ '["Accounts", "Accounts"]' ] }
      }
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 1,
    executionTimeMillis: 51,
    totalKeysExamined: 1,
    totalDocsExamined: 1,
    executionStages: {
      isCached: false,
      stage: 'FETCH',
      nReturned: 1,
      executionTimeMillisEstimate: 40,
      works: 2,
      advanced: 1,
      needTime: 0,
      needYield: 0,
      saveState: 2,
      restoreState: 2,
      isEOF: 1,
      docsExamined: 1,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
```

Output continued:

```
      docsExamined: 1,
      alreadyHasObj: 0,
      inputStage: {
        stage: 'IXSCAN',
        nReturned: 1,
        executionTimeMillisEstimate: 40,
        works: 2,
        advanced: 1,
        needTime: 0,
        needYield: 0,
        saveState: 2,
        restoreState: 2,
        isEOF: 1,
        keyPattern: { Department: 1 },
        indexName: 'Department_1',
        isMultiKey: false,
        multiKeyPaths: { Department: [] },
        isUnique: false,
        isSparse: false,
        isPartial: false,
        indexVersion: 2,
        direction: 'forward',
        indexBounds: { Department: [ '["Accounts", "Accounts"]' ] },
        keysExamined: 1,
        seeks: 1,
        dupsTested: 0,
        dupsDropped: 0
      }
    }
  },
  queryShapeHash: 'C5C8F443E8AC3A2ACE89CA8424ACCFD9BD858814EFEFD26F43D45C82736A5038',
  command: {
    find: 'Employee_details',
    filter: { Department: 'Accounts' },
    hint: { Department: 1 },
    '$db': 'Employee5'
  },
  serverInfo: {
    host: 'DESKTOP-9QG6EM7',
    port: 27017,
    version: '8.0.11',
    gitVersion: 'bed99f699da6cb2b74262aa6d473446c41476643'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'trySbeRestricted',
    internalQueryPlannerIgnoreIndexWithCollationForRegex: 1
  },
  ok: 1
}
Employee5> |
```

Experiment 8:

Aim: Geospatial Commands and User Management

- Insert geoJSON data and run $geoWithin queries; create users and assign roles.

Procedure:

to add a location field with GeoJSON point data (longitude, latitude):

db.Employee_details.updateMany(

{"Employee_name": "Rahul"},

{$set: {"location": {"type": "Point", "coordinates": [78.4867, 17.3850]}}}

)

db.Employee_details.updateMany(

{"Employee_name": "Rohit"},

{$set: {"location": {"type": "Point", "coordinates": [78.4864, 17.3852]}}}

)

```
Employee6> db.Employee_details.updateMany(
...     {"Employee_name": "Rahul"},
...     {$set: {"location": {"type": "Point", "coordinates": [78.4867, 17.3850]}}}
... )
...
... db.Employee_details.updateMany(
...     {"Employee_name": "Rohit"},
...     {$set: {"location": {"type": "Point", "coordinates": [78.4864, 17.3852]}}}
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Create a 2dsphere index on the location field to enable efficient geospatial queries

db.Employee_details.createIndex({"location": "2dsphere"})

```
}
Employee6> db.Employee_details.createIndex({"location": "2dsphere"})
location_2dsphere
```

Find employees within a defined rectangular area in Hyderabad:

```
db.Employee_details.find({
  "location": {
    $geoWithin: {
      $geometry: {
        type: "Polygon",
        coordinates: [[
          [78.4, 17.3], // Longitude, Latitude
          [78.5, 17.3],
          [78.5, 17.4],
          [78.4, 17.4],
          [78.4, 17.3]  // Close the polygon
        ]]
      }
    }
  }
})
```

```
Employee6> db.Employee_details.find({
...     "location": {
...         $geoWithin: {
...             $geometry: {
...                 type: "Polygon",
...                 coordinates: [[
...                     [78.4, 17.3], // Longitude, Latitude
...                     [78.5, 17.3],
...                     [78.5, 17.4],
...                     [78.4, 17.4],
...                     [78.4, 17.3]  // Close the polygon
...                 ]]
...             }
...         }
...     }
... })
...
[
  {
    _id: ObjectId('687d2e1c8fe02bddf4eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T17:57:48.195Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5,
    location: { type: 'Point', coordinates: [ 78.4867, 17.385 ] }
  },
  {
    _id: ObjectId('687d2e1c8fe02bddf4eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T17:57:48.195Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4,
    location: { type: 'Point', coordinates: [ 78.4864, 17.3852 ] }
  }
]
```

We can also search within a circular area using $centerSphere, for example:

Code: db.Employee_details.find({

"location": {

$geoWithin: {

$centerSphere: [[78.4867, 17.3850], 10 / 6378.1] // Center point [longitude, latitude], radius in radians

}

}

})

```
Employee6> db.Employee_details.find({
...     "location": {
...         $geoWithin: {
...             $centerSphere: [[78.4867, 17.3850], 10 / 6378.1] // Center point [longitude, latitude], radius in radians
...         }
...     }
... })
[
  {
    _id: ObjectId('687d2e1c8fe02bddf4eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-20T17:57:48.195Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5,
    location: { type: 'Point', coordinates: [ 78.4867, 17.385 ] }
  },
  {
    _id: ObjectId('687d2e1c8fe02bddf4eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-20T17:57:48.195Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4,
    location: { type: 'Point', coordinates: [ 78.4864, 17.3852 ] }
  }
]
Employee6> |
```

Aim: User management and roles

MongoDB uses role-based access control (RBAC) to manage user permissions.

Step 1: Create a user and assign roles:

Code: db.createUser({

```
    user: "appUser",
    pwd: "password123",
    roles: [
        { role: "readWrite", db: "Employee_details" },
        { role: "dbAdmin", db: "Employee_details" }
    ]
})
```

Output:

```
Employee6> db.createUser({
...     user: "appUser",
...     pwd: "password123",
...     roles: [
...         { role: "readWrite", db: "Employee_details" },
...         { role: "dbAdmin", db: "Employee_details" }
...     ]
... })
{ ok: 1 }
```

Step 2: Authenticate as the newly created user (for verification or further operations):

db.auth("appUser", "password123")

```
Employee6> db.auth("appUser", "password123")
{ ok: 1 }
Employee6>
```

VIVA Questions

S.No	Question	CO	Blooms Taxonomy
1	What does the $eq operator do in MongoDB queries?	CO2	Apply
2	How does the $gt operator differ from $lt in filtering documents?	CO2	Analyze
3	What is the purpose of the $in operator in MongoDB queries?	CO2	Apply
4	How is $or used to combine query conditions in MongoDB?	CO2	Analyze
5	What does the $exists operator check for in a document?	CO2	Understand
6	How do you project only selected fields in MongoDB query results?	CO2	Apply
7	What does $set do in MongoDB update operations?	CO2	Apply
8	How is $inc used to increment numeric fields in MongoDB documents?	CO2	Apply
9	What is the function of $push in array field updates?	CO2	Apply
10	What role does $match play in the MongoDB aggregation pipeline?	CO2	Apply
11	Explain the purpose of $group in MongoDB aggregation pipelines.	CO2	Analyze
12	How do you apply $sort stage in an aggregation pipeline?	CO2	Apply
13	How do you sort documents in a standard MongoDB query using .sort()?	CO2	Apply
14	How is .limit() used to restrict results in a MongoDB query?	CO2	Apply
15	What information does .explain() return for a MongoDB query?	CO2	Evaluate
16	How can .hint() help optimize MongoDB queries?	CO2	Evaluate
17	What is geoJSON and how is it used in MongoDB to store geospatial data?	CO2	Understand
18	How do you use a $geoWithin query to find documents within a specified region?	CO2	Apply
19	What is the syntax to create a new MongoDB user with specific roles?	CO2	Apply
20	How are roles assigned to a user during creation or modification?	CO2	Apply
21	What does the db.auth() function do in MongoDB authentication?	CO2	Apply

22	What are some built-in roles available in MongoDB, and what do they allow?	CO2	Remember
23	What is the process to create a custom role in MongoDB?	CO2	Create
24	What administrative privileges are granted by the dbAdmin role?	CO2	Understand
25	What is replication in MongoDB, and why is it used?	CO2	Understand
26	How do you initiate a replica set in MongoDB?	CO2	Apply
27	What is sharding in MongoDB, and how does it help scale data?	CO2	Understand
28	Describe the steps to enable sharding for a specific MongoDB collection.	CO2	Apply
29	What is a session in MongoDB and how is it used in transactions?	CO2	Understand
30	What is the Query Plan Cache in MongoDB, and how is it managed?	CO2	Analyze

MODULE – III

Experiment 9:

Aim: Create and Drop a Database

o Use use dbName, db.dropDatabase().

Procedure:

Verify the database and collection

After inserting the documents, the database and collection will be created and visible.

>show dbs

```
Employee7> show dbs
Employee                    40.00 KiB
Employee1                   40.00 KiB
Employee2                   40.00 KiB
Employee3                   40.00 KiB
Employee4                   56.00 KiB
Employee5                   60.00 KiB
Employee6                   92.00 KiB
Employee7                   40.00 KiB
Employee_1                  72.00 KiB
MLRITM                      72.00 KiB
MLRITM_Student_Information  40.00 KiB
admin                      132.00 KiB
blogpost                    40.00 KiB
blogpost1                  120.00 KiB
config                      96.00 KiB
local                       24.00 KiB
test                        40.00 KiB
Employee7>
```

>show collections

```
Employee7> show collections
Employee_details
Employee7>
```

>Drop the collection

```
Employee7> db.Employee_details.drop()
true
Employee7> show collections

Employee7> |
```

Drop the database

To drop the entire Employee7 database and all its collections, first ensure you're within the database using use Employee7, and then execute the db.dropDatabase() command:

>use Employee7

>db.dropDatabase()

```
Employee7> show dbs
Employee                    40.00 KiB
Employee1                   40.00 KiB
Employee2                   40.00 KiB
Employee3                   40.00 KiB
Employee4                   56.00 KiB
Employee5                   60.00 KiB
Employee6                   92.00 KiB
Employee7                   40.00 KiB
Employee_1                  72.00 KiB
MLRITM                      72.00 KiB
MLRITM_Student_Information  40.00 KiB
admin                      132.00 KiB
blogpost                    40.00 KiB
blogpost1                  120.00 KiB
config                      96.00 KiB
local                       24.00 KiB
test                        40.00 KiB
Employee7> use Employee7
already on db Employee7
Employee7> db.dropDatabase()
{ ok: 1, dropped: 'Employee7' }
Employee7> show dbs
Employee                    40.00 KiB
Employee1                   40.00 KiB
Employee2                   40.00 KiB
Employee3                   40.00 KiB
Employee4                   56.00 KiB
Employee5                   60.00 KiB
Employee6                   92.00 KiB
Employee_1                  72.00 KiB
MLRITM                      72.00 KiB
MLRITM_Student_Information  40.00 KiB
admin                      132.00 KiB
blogpost                    40.00 KiB
blogpost1                  120.00 KiB
config                      96.00 KiB
local                       24.00 KiB
test                        40.00 KiB
Employee7> |
```

Experiment 10:

Aim: Create and Drop Collections

Procedure:

To create a database give command as following use Students and to insert data into collection name Student_details to perform operations on dbName, db.dropDatabase().

Code:

db.createCollection("students")

```
Students> db.createCollection("students")
{ ok: 1 }
Students>
```

Verify the database and collection

After inserting the documents, the database and collection will be created and visible.

You can confirm the existence of the database with the following command:

>show dbs

You can also verify the existence of the collection within the selected database:

show collections

Drop the collection (optional)

To remove the Student_details collection from the Student

// Drop the "students" collection

db.students.drop()

```
Students>
true
```

Experiment 11:

Aim: Explore Collection Indexes and Options

- Create indexes and check with db.collection.getIndexes().

Procedure:

To create a compound index on the Department (ascending) and Salary (descending) fields, use the following command:

```
}
Employee9> db.Employee_details.createIndex({ "Employee_number": 1 });
Employee_number_1
Employee9> db.Employee_details.createIndex({ "Department": 1, "Salary": -1 });
Department_1_Salary_-1
Employee9>
```

Check indexes:

To view the indexes for the Employee_details collection, use the db.collection.getIndexes() method:

db.Employee_details.getIndexes();

```
Employee9> db.Employee_details.createIndex({ "Employee_number": 1 });
Employee_number_1
Employee9> db.Employee_details.createIndex({ "Department": 1, "Salary": -1 });
Department_1_Salary_-1
Employee9> db.Employee_details.getIndexes();
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { Employee_number: 1 }, name: 'Employee_number_1' },
  {
    v: 2,
    key: { Department: 1, Salary: -1 },
    name: 'Department_1_Salary_-1'
  }
]
Employee9> |
```

Experiment 12:

Aim: Set Up Schema Validation Rules

- Use JSON schema validation to restrict document structure.

Procedure:

To create a database give command as following use Employee12 and to insert data into collection Empdetails to Use JSON schema validation to restrict document structure.

Code:

```
db.createCollection("Empdetails", {
validator: {
  $jsonSchema: {
    bsonType: "object",
    required: ["name", "age"],
    properties: {
      name: {
        bsonType: "string",
        description: "must be a string and is required"
      },
      age: {
        bsonType: "int",
        minimum: 18,
        description: "must be an integer >= 18 and is required"
      },
      email: {
        bsonType: "string",
        pattern: "^.+@.+\\..+$",
        description: "must be a valid email address if provided"
      }
```

```
    },
    additionalProperties: false
  }
},
validationLevel: "strict",
validationAction: "error"
});
```

Output:

```
test> use Employee12
switched to db Employee12
Employee12> db.createCollection("Empdetails", {
...   validator: {
...     $jsonSchema: {
...       bsonType: "object",
...       required: ["name", "age"],
...       properties: {
...         name: {
...           bsonType: "string",
...           description: "must be a string and is required"
...         },
...         age: {
...           bsonType: "int",
...           minimum: 18,
...           description: "must be an integer >= 18 and is required"
...         },
...         email: {
...           bsonType: "string",
...           pattern: "^.+@.+\\..+$",
...           description: "must be a valid email address if provided"
...         }
...       },
...       additionalProperties: false
...     }
...   },
...   validationLevel: "strict",
...   validationAction: "error"
... });
...
{ ok: 1 }
Employee12>
```

Example of Valid Document:

Code: db.Empdetails.insertOne({

name: "John Doe",

age: 30,

email: "john.doe@example.com"

});

```
Employee12> db.Empdetails.insertOne({
...   name: "John Doe",
...   age: 30,
...   email: "john.doe@example.com"
... });
Uncaught:
MongoServerError: Document failed validation
Additional information: {
  failingDocumentId: ObjectId('687dd25c1450ce40e7eec4a9'),
  details: {
    operatorName: '$jsonSchema',
    schemaRulesNotSatisfied: [
      {
        operatorName: 'additionalProperties',
        specifiedAs: { additionalProperties: false },
        additionalProperties: [ '_id' ]
      }
    ]
  }
}
Employee12>
```

Example of Invalid Document (age < 18):

Code: db.Empdetails.insertOne({

name: "Alice",

age: 16});

```
Employee12> db.Empdetails.insertOne({
...   name: "Alice",
...   age: 16
... });
Uncaught:
MongoServerError: Document failed validation
Additional information: {
  failingDocumentId: ObjectId('687dd2bb1450ce40e7eec4aa'),
  details: {
    operatorName: '$jsonSchema',
    schemaRulesNotSatisfied: [
      {
        operatorName: 'properties',
        propertiesNotSatisfied: [
          {
            propertyName: 'age',
            description: 'must be an integer >= 18 and is required',
            details: [
              {
                operatorName: 'minimum',
                specifiedAs: { minimum: 18 },
                reason: 'comparison failed',
                consideredValue: 16
              }
            ]
          }
        ]
      },
      {
        operatorName: 'additionalProperties',
        specifiedAs: { additionalProperties: false },
        additionalProperties: [ '_id' ]
      }
    ]
  }
}
Employee12>
```

Example of Invalid Document (extra field):

Code: db.Empdetails.insertOne({

name: "Bob",

age: 25,

department: "HR"

});

```
Employee12> db.Empdetails.insertOne({
...   name: "Bob",
...   age: 25,
...   department: "HR"
... });
Uncaught:
MongoServerError: Document failed validation
Additional information: {
  failingDocumentId: ObjectId('687dd3341450ce40e7eec4ab'),
  details: {
    operatorName: '$jsonSchema',
    schemaRulesNotSatisfied: [
      {
        operatorName: 'additionalProperties',
        specifiedAs: { additionalProperties: false },
        additionalProperties: [ '_id', 'department' ]
      }
    ]
  }
}
Employee12>
```

VIVA Questions

S.No	Question	CO	Blooms Taxonomy
1	How do you create a new database in MongoDB?	CO3	Apply
2	How can you drop a database in MongoDB?	CO3	Apply
3	What command creates a collection explicitly?	CO3	Apply
4	How do you drop a collection?	CO3	Apply
5	What happens if you insert data into a non-existent collection?	CO3	Understand
6	How do you create an index on a field?	CO3	Apply
7	How do you list all indexes for a collection?	CO3	Apply
8	What is the purpose of indexes in MongoDB?	CO3	Understand
9	Can a collection have multiple indexes?	CO3	Remember
10	What is a compound index?	CO3	Understand
11	What is schema validation in MongoDB?	CO3	Understand
12	How do you apply schema validation to a collection?	CO3	Apply
13	Can schema validation rules be updated later?	CO3	Apply
14	How do you restrict a field to a string using validation?	CO3	Apply
15	What does "required" do in schema validation?	CO3	Understand
16	What does use myDB do?	CO3	Remember
17	How do you verify the current database in use?	CO3	Apply
18	What command shows all databases in MongoDB?	CO3	Remember
19	How do you list all collections in a database?	CO3	Apply
20	How do you check if a collection exists?	CO3	Apply
21	How do you create a unique index in MongoDB?	CO3	Apply
22	What happens when duplicate values are inserted into a unique indexed field?	CO3	Analyze
23	How do you enforce field type and required fields in the same validator?	CO3	Apply
24	Can you combine multiple validation rules in a schema validator?	CO3	Understand
25	What is the collMod command used for in MongoDB?	CO3	Understand
26	What happens when you drop a collection in MongoDB?	CO3	Understand
27	What happens when you drop a database in MongoDB?	CO3	Understand
28	How do you drop all indexes from a collection except _id?	CO3	Apply
29	Can you create a capped collection in MongoDB?	CO3	Apply
30	How do capped collections behave differently than normal ones?	CO3	Analyze

MODULE - IV

Experiment 13:

Aim: CRUD: Insert, Query, Update, Delete Documents

- Full example of inserting, querying with filters, updating fields, and deleting.

Procedure:

Read (Query Documents)

To retrieve documents, you use the find() method. You can specify filters to narrow down the results.

A. Get all employees

```
Employee13> db.Employee_details.find({});
[
  {
    _id: ObjectId('687dd75e6c720b4434eec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687dd75e6c720b4434eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687dd75e6c720b4434eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687dd75e6c720b4434eec4ac'),
    Employee_name: 'Indu',
    Employee_number: '1005',
    Employee_gender: 'Female',
    Department: 'Cashier',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687dd75e6c720b4434eec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.7
  }
]
```

B. Query for employees in a specific department

Code: db.Employee_details.find({ "Department": "Cashier" });

```
Employee13> db.Employee_details.find({ "Department": "Cashier" });
[
  {
    _id: ObjectId('687dd75e6c720b4434eec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687dd75e6c720b4434eec4ac'),
    Employee_name: 'Indu',
    Employee_number: '1005',
    Employee_gender: 'Female',
    Department: 'Cashier',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ],
    performance_rating: 4.7
  }
]
Employee13> |
```

C. Query for employees with a salary greater than a certain amount

Code: db.Employee_details.find({ "Salary": { $gt: 50000 } });

```
Employee13> db.Employee_details.find({ "Salary": { $gt: 50000 } });
[
  {
    _id: ObjectId('687dd75e6c720b4434eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687dd75e6c720b4434eec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.7
  }
]
Employee13> |
```

D. Query for employees with specific skills (using the $in operator)

Code: db.Employee_details.find({ "skills": { $in: ["Leadership", "Reporting"] } });

```
Employee13> db.Employee_details.find({ "skills": { $in: ["Leadership", "Reporting"] } });
[
  {
    _id: ObjectId('687dd75e6c720b4434eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
  {
    _id: ObjectId('687dd75e6c720b4434eec4ad'),
    Employee_name: 'Sohail',
    Employee_number: '1007',
    Employee_gender: 'Male',
    Department: 'Assistant_Manager',
    Salary: 62000,
    is_active: false,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.7
  }
]
Employee13>
```

E. Query for inactive female employees

Code: db.Employee_details.find({

"Employee_gender": "Female",

"is_active": false

});

```
]
Employee13> db.Employee_details.find({
...   "Employee_gender": "Female",
...   "is_active": false
... });
[
  {
    _id: ObjectId('687dd75e6c720b4434eec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-21T05:59:58.647Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  }
]
Employee13> |
```

F. Query and project specific fields

Code: db.Employee_details.find(

{ "Department": "Manager" },

{ "Employee_name": 1, "Department": 1, "_id": 0 } // Returns only name and department, excluding _id

);

```
]
Employee13> db.Employee_details.find(
...   { "Department": "Manager" },
...   { "Employee_name": 1, "Department": 1, "_id": 0 } // Returns only name and department, excluding _id
... );
[ { Employee_name: 'Swetha', Department: 'Manager' } ]
Employee13>
```

Update (Modify Documents)

You can update existing documents using methods like updateOne(), updateMany(), or replaceOne()

A. Update a single employee's department and salary

Code: db.Employee_details.updateOne(

{ "Employee_number": "1001" },

{ $set: { "Department": "Finance", "Salary": 55000 } }

);

```
Employee13> db.Employee_details.updateOne(
...   { "Employee_number": "1001" },
...   { $set: { "Department": "Finance", "Salary": 55000 } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Employee13>
```

B. Increment an employee's performance rating

Code: db.Employee_details.updateOne(

{ "Employee_number": "1002" },

{ $inc: { "performance_rating": 0.2 } }

);

```
}
Employee13> db.Employee_details.updateOne(
...   { "Employee_number": "1002" },
...   { $inc: { "performance_rating": 0.2 } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Employee13> |
```

C. Update the is_active status for all employees in a specific department

Code: db.Employee_details.updateMany(

{ "Department": "Cashier" },

{ $set: { "is_active": false } }

);

```
}
Employee13> db.Employee_details.updateMany(
...   { "Department": "Cashier" },
...   { $set: { "is_active": false } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
Employee13>
```

D. Rename a field

Code: db.Employee_details.updateMany(

{}, // Empty filter updates all documents

{ $rename: { "performance_rating": "rating" } });

```
Employee13> db.Employee_details.updateMany(
...   {}, // Empty filter updates all documents
...   { $rename: { "performance_rating": "rating" } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
Employee13> |
```

Delete (Remove Documents)

A. Delete a single employee by their employee number

Code: db.Employee_details.deleteOne({ "Employee_number": "1007" });

```
  upsertedCount: 0
}
Employee13> db.Employee_details.deleteOne({ "Employee_number": "1007" });
{ acknowledged: true, deletedCount: 1 }
Employee13>
```

B. Delete all employees in a specific department

Code: db.Employee_details.deleteMany({ "Department": "Cashier" });

```
Employee13> db.Employee_details.deleteMany({ "Department": "Cashier" });
{ acknowledged: true, deletedCount: 2 }
Employee13> |
```

Experiment 14:

Aim: Use of db.runCommand() and Server Info

- Run db.runCommand({ serverStatus: 1 }) and db.isMaster().

Procedure: By running the command db.runCommand({ serverStatus: 1 });

we get information as a comprehensive overview of the MongoDB instance's current state and health, etc. particularly useful for administrative and diagnostic tasks

Code: db.runCommand({ serverStatus: 1 });

Output:

```
mongosh mongodb://127.0.0.1   mongosh mongodb://127.0.0.1   mongosh mongodb://127.0.0.

   The server generated these startup warnings when booting
   2025-07-20T14:09:57.188+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

test> db.runCommand({ serverStatus: 1 });
{
  host: 'DESKTOP-9QG6EM7',
  version: '8.0.11',
  process: 'C:\\Program Files\\MongoDB\\Server\\8.0\\bin\\mongod.exe',
  service: [ 'shard' ],
  pid: Long('1164'),
  uptime: 78541,
  uptimeMillis: Long('78535445'),
  uptimeEstimate: Long('78535'),
  localTime: ISODate('2025-07-21T06:28:54.570Z'),
  asserts: {
    regular: 0,
    warning: 0,
    msg: 0,
    user: 298,
    tripwire: 0,
    rollovers: 0
  },
  batchedDeletes: {
    batches: 34,
    docs: 130,
    stagedSizeBytes: 31920,
    timeInBatchMillis: 21,
    refetchesDueToYield: 0
  },
  catalogStats: {
    collections: 20,
    capped: 0,
    clustered: 0,
    timeseries: 0,
    views: 0,
    internalCollections: 4,
    internalViews: 0,
    csfle: 0,
    queryableEncryption: 0
  },
  changeStreamPreImages: {
    purgingJob: {
      totalPass: Long('0'),
      docsDeleted: Long('0'),
      bytesDeleted: Long('0'),
      scannedCollections: Long('0'),
      scannedInternalCollections: Long('0'),
      maxStartWallTimeMillis: Long('0'),
      timeElapsedMillis: Long('0')
    }
  },
  collectionCatalog: { numScansDueToMissingMapping: Long('0') },
  connections: {
    current: 15,
    available: 999985,
    totalCreated: 224,
    rejected: 0,
    active: 4,
    threaded: 15,
    exhaustIsMaster: Long('0'),
    exhaustHello: Long('2'),
    awaitingTopologyChanges: Long('3'),
    loadBalanced: Long('0')
```

Output continued..

```
  loadBalanced: Long('0')
},
electionMetrics: {
  stepUpCmd: { called: Long('0'), successful: Long('0') },
  priorityTakeover: { called: Long('0'), successful: Long('0') },
  catchUpTakeover: { called: Long('0'), successful: Long('0') },
  electionTimeout: { called: Long('0'), successful: Long('0') },
  freezeTimeout: { called: Long('0'), successful: Long('0') },
  numStepDownsCausedByHigherTerm: Long('0'),
  numCatchUps: Long('0'),
  numCatchUpsSucceeded: Long('0'),
  numCatchUpsAlreadyCaughtUp: Long('0'),
  numCatchUpsSkipped: Long('0'),
  numCatchUpsTimedOut: Long('0'),
  numCatchUpsFailedWithError: Long('0'),
  numCatchUpsFailedWithNewTerm: Long('0'),
  numCatchUpsFailedWithReplSetAbortPrimaryCatchUpCmd: Long('0'),
  averageCatchUpOps: 0
},
extra_info: {
  note: 'fields vary by platform',
  page_faults: 245844,
  usagePageFileMB: 413,
  totalPageFileMB: 20214,
  availPageFileMB: 2817,
  ramMB: 7926
},
featureCompatibilityVersion: { major: 8, minor: 0, transitioning: 0 },
flowControl: {
  enabled: true,
  targetRateLimit: 1000000000,
  timeAcquiringMicros: Long('0'),
  locksPerKiloOp: 0,
  sustainerRate: 0,
  isLagged: false,
  isLaggedCount: 0,
  isLaggedTimeMicros: Long('0')
},
globalLock: {
  totalTime: Long('78540917000'),
  currentQueue: { total: 0, readers: 0, writers: 0 },
  activeClients: { total: 0, readers: 0, writers: 0 }
},
indexBuilds: {
  total: 3,
  killedDueToInsufficientDiskSpace: 0,
  failedDueToDataCorruption: 0,
  phases: {
    scanCollection: 3,
    drainSideWritesTable: 3,
    drainSideWritesTablePreCommit: 3,
    waitForCommitQuorum: 3,
    drainSideWritesTableOnCommit: 3,
    processConstraintsViolatonTableOnCommit: 3,
    commit: 3
  }
},
indexBulkBuilder: {
  count: Long('3'),
  resumed: Long('0'),
  filesOpenedForExternalSort: Long('0'),
  filesClosedForExternalSort: Long('0'),
  spilledRanges: Long('0'),
  bytesSpilledUncompressed: Long('0'),
```

db.runCommand({ hello: 1 }) or db.isMaster()

The hello command (which replaces itsMaster in MongoDB 5.0 and later) returns a document that describes the role of the mongodb instance in a replica set or sharded cluster.

```
test> db.runCommand({ hello: 1 });
{
  isWritablePrimary: true,
  topologyVersion: {
    processId: ObjectId('687cab59e2b8768abfac25d3'),
    counter: Long('0')
  },
  maxBsonObjectSize: 16777216,
  maxMessageSizeBytes: 48000000,
  maxWriteBatchSize: 100000,
  localTime: ISODate('2025-07-21T06:48:47.791Z'),
  logicalSessionTimeoutMinutes: 30,
  connectionId: 223,
  minWireVersion: 0,
  maxWireVersion: 25,
  readOnly: false,
  ok: 1
}
test> |
```

Experiment 15:

Aim: Bulk Operations and Upsert Example

- Demonstrate bulkWrite() with mixed inserts and updates

Procedure:

To create a database give command as following use Employee14 and use the bulkWrite() method in MongoDB to efficiently perform multiple write operations, including inserts and updates (with upsert), within a single command.

Upsert is a meaning used for update and insert operation together.

It refers to a database operation that checks if a record with specific criteria already exists in a table or collection. If it exists, the record is updated with new information. If it doesn't exist, a new record is inserted

Code:

```
db.Employee_details.bulkWrite([
  // Insert new documents
  {
    insertOne: {
      document: {
        "Employee_name": "Ravi",
        "Employee_number": "1008",
        "Employee_gender": "Male",
        "Department": "Sales",
        "Salary": 55000,
        "is_active": true,
        "Date": ISODate(),
        "Address": "BANGALORE",
        "skills": ["Negotiation", "Customer Relationship Management"],
        "performance_rating": 4.6
```

```
    }
  }
},
{
  insertOne: {
    document: {
      "Employee_name": "Priya",
      "Employee_number": "1009",
      "Employee_gender": "Female",
      "Department": "Marketing",
      "Salary": 60000,
      "is_active": true,
      "Date": ISODate(),
      "Address": "CHENNAI",
      "skills": ["Social Media Marketing", "Content Creation"],
      "performance_rating": 4.7
    }
  }
},
// Update an existing document (based on Employee_number)
{
  updateOne: {
    filter: {
      "Employee_number": "1001"
    },
    update: {
```

```
      $set: {
        "Salary": 52000,
        "skills": ["SQL", "Excel", "Auditing", "Financial Reporting"]
      }
    }
  }
},

// Upsert: Update if Employee_number exists, otherwise insert
{
  updateOne: {
    filter: {
      "Employee_number": "1004"
    }, // Searching for Employee_number 1004
    update: {
      $set: {
        "Employee_name": "John",
        "Employee_gender": "Male",
        "Department": "IT",
        "Salary": 65000,
        "is_active": true,
        "Date": ISODate(),
        "Address": "MUMBAI",
        "skills": ["Python", "Java", "Cloud Computing"],
        "performance_rating": 4.9
      }
```

```
    },
    upsert: true // If not found, insert this document
  } }
]);
```

Output:

```
test> use Employee14
switched to db Employee14
Employee14> db.Employee_details.bulkWrite([
...   // Insert new documents
...   {
...     insertOne: {
...       document: {
...         "Employee_name": "Ravi",
...         "Employee_number": "1008",
...         "Employee_gender": "Male",
...         "Department": "Sales",
...         "Salary": 55000,
...         "is_active": true,
...         "Date": ISODate(),
...         "Address": "BANGALORE",
...         "skills": ["Negotiation", "Customer Relationship Management"],
...         "performance_rating": 4.6
...       }
...     }
...   },
...   {
...     insertOne: {
...       document: {
...         "Employee_name": "Priya",
...         "Employee_number": "1009",
...         "Employee_gender": "Female",
...         "Department": "Marketing",
...         "Salary": 60000,
...         "is_active": true,
...         "Date": ISODate(),
...         "Address": "CHENNAI",
...         "skills": ["Social Media Marketing", "Content Creation"],
...         "performance_rating": 4.7
...       }
...     }
...   },
...
...   // Update an existing document (based on Employee_number)
...   {
...     updateOne: {
...       filter: {
...         "Employee_number": "1001"
...       },
...       update: {
...         $set: {
...           "Salary": 52000,
...           "skills": ["SQL", "Excel", "Auditing", "Financial Reporting"]
...         }
...       }
...     }
...   },
...
...   // Upsert: Update if Employee_number exists, otherwise insert
...   {
...     updateOne: {
...       filter: {
...         "Employee_number": "1004"
...       }, // Searching for Employee_number 1004
...       update: {
...         $set: {
...           "Employee_name": "John",
...           "Employee_gender": "Male",
...           "Department": "IT",
...           "Salary": 65000,
...           "is_active": true,
...           "Date": ISODate(),
...           "Address": "MUMBAI",
...           "skills": ["Python", "Java", "Cloud Computing"],
...           "performance_rating": 4.9
...         }
...       },
...       upsert: true // If not found, insert this document
...     }
...   }
... ]);
{
  acknowledged: true,
  insertedCount: 2,
  insertedIds: {
```

Code: db.Employee_details.find();

```
Employee14> db.Employee_details.find()
[
  {
    _id: ObjectId('687de3cd6d71ad8a5feec4a9'),
    Employee_name: 'Ravi',
    Employee_number: '1008',
    Employee_gender: 'Male',
    Department: 'Sales',
    Salary: 55000,
    is_active: true,
    Date: ISODate('2025-07-21T06:53:01.078Z'),
    Address: 'BANGALORE',
    skills: [ 'Negotiation', 'Customer Relationship Management' ]
    performance_rating: 4.6
  },
  {
    _id: ObjectId('687de3cd6d71ad8a5feec4aa'),
    Employee_name: 'Priya',
    Employee_number: '1009',
    Employee_gender: 'Female',
    Department: 'Marketing',
    Salary: 60000,
    is_active: true,
    Date: ISODate('2025-07-21T06:53:01.078Z'),
    Address: 'CHENNAI',
    skills: [ 'Social Media Marketing', 'Content Creation' ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687de3cde2b8768abfac25db'),
    Employee_number: '1004',
    Address: 'MUMBAI',
    Date: ISODate('2025-07-21T06:53:01.078Z'),
    Department: 'IT',
    Employee_gender: 'Male',
    Employee_name: 'John',
    Salary: 65000,
    is_active: true,
    performance_rating: 4.9,
    skills: [ 'Python', 'Java', 'Cloud Computing' ]
  }
]
```

Experiment 16:

Aim: Check Collection Stats and Perform Partial Updates

- db.collection.stats(), $set with field targeting

Procedure:

The db.collection.stats() method provides a comprehensive overview of a collection's characteristics, including its size, number of documents, and index details.

Code: db.Employee_details.stats();

Output:

```
mongosh mongodb://127.0.0.1 X   mongosh mongodb://127.0.0.1 X   mongosh mongodb://127.0.0. X   +

    'column-store variable-size RLE encoded values': 0,
    'column-store variable-size deleted values': 0,
    'column-store variable-size leaf pages': 0,
    'fixed-record size': 0,
    'maximum internal page size': 4096,
    'maximum leaf page key size': 2867,
    'maximum leaf page size': 32768,
    'maximum leaf page value size': 67108864,
    'maximum tree depth': 3,
    'number of key/value pairs': 0,
    'overflow pages': 0,
    'row-store empty values': 0,
    'row-store internal pages': 0,
    'row-store leaf pages': 0
  },
  cache: {
    'bytes currently in the cache': 2293,
    'bytes dirty in the cache cumulative': 1514,
    'bytes read into cache': 0,
    'bytes written from cache': 934,
    'checkpoint blocked page eviction': 0,
    'checkpoint of history store file blocked non-history store page eviction': 0,
    'data source pages selected for eviction unable to be evicted': 0,
    'eviction gave up due to detecting a disk value without a timestamp behind the last update on the chain': 0,
    'eviction gave up due to detecting a tombstone without a timestamp ahead of the selected on disk update': 0,
    'eviction gave up due to detecting a tombstone without a timestamp ahead of the selected on disk update after validating the update chain': 0,
    'eviction gave up due to detecting update chain entries without timestamps after the selected on disk update': 0,
    'eviction gave up due to needing to remove a record from the history store but checkpoint is running': 0,
    'eviction gave up due to no progress being made': 0,
    'eviction walk passes of a file': 0,
    'eviction walk target pages histogram - 0-9': 0,
    'eviction walk target pages histogram - 10-31': 0,
    'eviction walk target pages histogram - 128 and higher': 0,
    'eviction walk target pages histogram - 32-63': 0,
    'eviction walk target pages histogram - 64-128': 0,
    'eviction walk target pages reduced due to history store cache pressure': 0,
    'eviction walks abandoned': 0,
    'eviction walks gave up because they restarted their walk twice': 0,
    'eviction walks gave up because they saw too many pages and found no candidates': 0,
    'eviction walks gave up because they saw too many pages and found too few candidates': 0,
    'eviction walks random search fails to locate a page, results in a null position': 0,
    'eviction walks reached end of tree': 0,
    'eviction walks restarted': 0,
    'eviction walks started from root of tree': 0,
    'eviction walks started from saved location in tree': 0,
    'hazard pointer blocked page eviction': 0,
    'history store table insert calls': 0,
    'history store table insert calls that returned restart': 0,
    'history store table reads': 0,
    'history store table reads missed': 0,
    'history store table reads requiring squashed modifies': 0,
    'history store table resolved updates without timestamps that lose their durable timestamp': 0,
    'history store table truncation by rollback to stable to remove an unstable update': 0,
    'history store table truncation by rollback to stable to remove an update': 0,
    'history store table truncation to remove all the keys of a btree': 0,
    'history store table truncation to remove an update': 0,
    'history store table truncation to remove range of updates due to an update without a timestamp on data page': 0,
    'history store table truncation to remove range of updates due to key being removed from the data page during reconciliation': 0,
    'history store table truncations that would have happened in non-dryrun mode': 0,
    'history store table truncations to remove an unstable update that would have happened in non-dryrun mode': 0,
    'history store table truncations to remove an update that would have happened in non-dryrun mode': 0,
    'history store table updates without timestamps fixed up by reinserting with the fixed timestamp': 0,
    'history store table writes requiring squashed modifies': 0,
    'in-memory page passed criteria to be split': 0,
```

Output contined..

```
mongosh mongodb://127.0.0.1 X   mongosh mongodb://127.0.0.1 X   mongosh mongodb://127.0.0. X   +  v
Employee14> db.Employee_details.stats();
{
  ok: 1,
  capped: false,
  wiredTiger: {
    metadata: { formatVersion: 1 },
    creationString: 'access_pattern_hint=none,allocation_size=4KB,app_metadata=(formatVersion=1),assert=(commit_timestamp=none,durable_timestamp=none,read_timestamp=none,write_timestamp=off),block_allocation=best,block_compressor=snappy,cache_resident=false,checksum=on,colgroups=,collator=,columns=,dictionary=0,encryption=(keyid=,name=),exclusive=false,extractor=,format=btree,huffman_key=,huffman_value=,ignore_in_memory_cache_size=false,immutable=false,import=(compare_timestamp=oldest_timestamp,enabled=false,file_metadata=,metadata_file=,panic_corrupt=true,repair=false),internal_item_max=0,internal_key_max=0,internal_key_truncate=true,internal_page_max=4KB,key_format=q,key_gap=10,leaf_item_max=0,leaf_key_max=0,leaf_page_max=32KB,leaf_value_max=64MB,log=(enabled=true),lsm=(auto_throttle=true,bloom=true,bloom_bit_count=16,bloom_config=,bloom_hash_count=8,bloom_oldest=false,chunk_count_limit=0,chunk_max=5GB,chunk_size=10MB,merge_custom=(prefix=,start_generation=0,suffix=),merge_max=15,merge_min=0),memory_page_image_max=0,memory_page_max=10m,os_cache_dirty_max=0,os_cache_max=0,prefix_compression=false,prefix_compression_min=4,source=,split_deepen_min_child=0,split_deepen_per_child=0,split_pct=90,tiered_storage=(auth_token=,bucket=,bucket_prefix=,cache_directory=,local_retention=300,name=,object_target_size=0),type=file,value_format=u,verbose=[],write_timestamp_usage=none',
    type: 'file',
    uri: 'statistics:table:collection-50-7455869660606749032',
    LSM: {
      'bloom filter false positives': 0,
      'bloom filter hits': 0,
      'bloom filter misses': 0,
      'bloom filter pages evicted from cache': 0,
      'bloom filter pages read into cache': 0,
      'bloom filters in the LSM tree': 0,
      'chunks in the LSM tree': 0,
      'highest merge generation in the LSM tree': 0,
      'queries that could have benefited from a Bloom filter that did not exist': 0,
      'sleep for LSM checkpoint throttle': 0,
      'sleep for LSM merge throttle': 0,
      'total size of bloom filters': 0
    },
    autocommit: {
      'retries for readonly operations': 0,
      'retries for update operations': 0
    },
    backup: {
      'total modified incremental blocks with compressed data': 0,
      'total modified incremental blocks without compressed data': 0
    },
    'block-manager': {
      'allocations requiring file extension': 4,
      'blocks allocated': 4,
      'blocks freed': 0,
      'checkpoint size': 4096,
      'file allocation unit size': 4096,
      'file bytes available for reuse': 0,
      'file magic number': 120897,
      'file major version number': 1,
      'file size in bytes': 20480,
      'minor version number': 0
    },
    btree: {
      'btree checkpoint generation': 776,
      'btree clean tree checkpoint expiration time': Long('9223372036854775807'),
      'btree compact pages reviewed': 0,
      'btree compact pages rewritten': 0,
      'btree compact pages skipped': 0,
      'btree expected number of compact bytes rewritten': 0,
      'btree expected number of compact pages rewritten': 0,
      'btree number of pages reconciled during checkpoint': 2,
      'btree skipped by compaction as process would not reduce size': 0,
      'column-store fixed-size leaf pages': 0,
      'column-store fixed-size time windows': 0,
      'column-store internal pages': 0,
      'column-store variable-size RLE encoded values': 0,
      'column-store variable-size deleted values': 0,
      'column-store variable-size leaf pages': 0,
```

Updating a top-level field

Let's say you want to update the Salary of the employee with Employee_number "1001" to 55000:

Code: db.Employee_details.updateOne(

{ "Employee_number": "1001" }, // Filter: Match documents by Employee_number

{ $set: { "Salary": 55000 } } // Update: Set the Salary field to the new value

);

```
      'update conflicts': 0
    }
  },
  sharded: false,
  size: 827,
  count: 3,
  numOrphanDocs: 0,
  storageSize: 20480,
  totalIndexSize: 20480,
  totalSize: 40960,
  indexSizes: { _id_: 20480 },
  avgObjSize: 275,
  ns: 'Employee14.Employee_details',
  nindexes: 1,
  scaleFactor: 1
}
Employee14> db.Employee_details.updateOne(
...   { "Employee_number": "1001" }, // Filter: Match documents by Employee_number
...   { $set: { "Salary": 55000 } } // Update: Set the Salary field to the new value
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 0
}
Employee14>
```

For Adding a new field

Code: ndb.Employee_details.updateOne(

{ "Employee_number": "1002" },

{ $set: { "Email": "rohit.example@email.com" } }

);

```
Employee14> db.Employee_details.updateOne(
...   { "Employee_number": "1008" },
...   { $set: { "Email": "ravi.example@email.com" } }
... );
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Employee14> db.Employee_details.find()
[
  {
    _id: ObjectId('687de3cd6d71ad8a5feec4a9'),
    Employee_name: 'Ravi',
    Employee_number: '1008',
    Employee_gender: 'Male',
    Department: 'Sales',
    Salary: 55000,
    is_active: true,
    Date: ISODate('2025-07-21T06:53:01.078Z'),
    Address: 'BANGALORE',
    skills: [ 'Negotiation', 'Customer Relationship Management' ],
    performance_rating: 4.6,
    Email: 'ravi.example@email.com'
  },
  {
    _id: ObjectId('687de3cd6d71ad8a5feec4aa'),
    Employee_name: 'Priya',
    Employee_number: '1009',
    Employee_gender: 'Female',
    Department: 'Marketing',
    Salary: 60000,
    is_active: true,
    Date: ISODate('2025-07-21T06:53:01.078Z'),
    Address: 'CHENNAI',
    skills: [ 'Social Media Marketing', 'Content Creation' ],
    performance_rating: 4.7
  },
  {
    _id: ObjectId('687de3cde2b8768abfac25db'),
    Employee_number: '1004',
    Address: 'MUMBAI',
    Date: ISODate('2025-07-21T06:53:01.078Z'),
    Department: 'IT',
    Employee_gender: 'Male',
    Employee_name: 'John',
    Salary: 65000,
    is_active: true,
    performance_rating: 4.9,
    skills: [ 'Python', 'Java', 'Cloud Computing' ]
  }
]
Employee14> |
```

VIVA Questions

S.No	Question	CO	Blooms Taxonomy
1	How do you insert a document into a MongoDB collection?	CO2	Apply
2	How do you query documents with a filter in MongoDB?	CO2	Apply
3	What is the difference between updateOne() and updateMany() in MongoDB?	CO2	Analyze
4	How do you delete a document in MongoDB?	CO2	Apply
5	What is an upsert operation in MongoDB?	CO2	Understand
6	What does db.runCommand({ serverStatus: 1 }) do in MongoDB?	CO3	Understand
7	What is the purpose of db.isMaster() in MongoDB?	CO3	Understand
8	How do you check the status of the MongoDB server?	CO3	Apply
9	What does db.currentOp() do in MongoDB?	CO3	Apply
10	How do you terminate a running operation in MongoDB?	CO3	Apply
11	What is bulkWrite() in MongoDB and how is it used?	CO4	Apply
12	How do you perform a partial update on a document in MongoDB?	CO4	Apply
13	What is the difference between insertOne() and insertMany() in MongoDB?	CO2	Understand
14	How do you check collection statistics in MongoDB?	CO3	Apply
15	What happens when you drop a collection in MongoDB?	CO3	Understand
16	How do you check current operations running on MongoDB?	CO3	Apply
17	How do you kill a running operation in MongoDB?	CO3	Apply
18	What is the use of db.serverStatus() in MongoDB?	CO3	Understand
19	What does db.collection.drop() do in MongoDB?	CO3	Apply
20	How do you perform a multi-document update in MongoDB?	CO2	Apply
21	What is the Wire Protocol in MongoDB and what is it used for?	CO4	Understand
22	How does SQL SELECT query translate to a MongoDB find query?	CO4	Apply
23	How does SQL UPDATE query translate to a MongoDB update operation?	CO4	Apply
24	How do you remove documents conditionally in MongoDB?	CO2	Apply
25	What is meant by partial update in MongoDB?	CO4	Understand

26	What are the advantages of using bulkWrite() in MongoDB?	CO4	Analyze
27	Can bulkWrite() mix insert and update operations in a single command?	CO4	Understand
28	How do you limit the number of documents returned in a MongoDB query?	CO2	Apply
29	What is MongoDB text search and how do you enable it?	CO4	Apply
30	What does collection.stats() return in MongoDB?	CO3	Remember

MODULE - V

MongoDB Shell

The MongoDB shell (mongosh) is an interactive JavaScript interface to MongoDB. It lets users interact with databases, collections, and documents through commands and scripts. You can perform CRUD operations, indexing, aggregations, and more directly from the shell.

Shell Collection Methods

These methods operate on collections:

- db.collection.find() – Retrieves documents.
- db.collection.insertOne() / insertMany() – Inserts document(s).
- db.collection.updateOne() / updateMany() – Updates document(s).
- db.collection.deleteOne() / deleteMany() – Deletes document(s).
- db.collection.aggregate() – Performs aggregation operations.
- db.collection.createIndex() – Creates indexes.

Cursor Methods

When a query returns multiple documents, MongoDB uses a cursor. Common cursor methods:

- .forEach() – Iterates documents.
- .next() – Moves to the next document.
- .hasNext() – Checks if more documents are present.
- .toArray() – Converts cursor to array.
- .pretty() – Formats the output in a readable format.

MongoDB Database Commands

MongoDB supports various administrative commands:

- show dbs – Lists databases.
- use <db> – Switches to a specific database.
- db.createCollection() – Creates a new collection.
- db.dropDatabase() – Deletes the current database.
- db.stats() – Displays statistics about the current database.

Query Plan Cache Methods

These commands help analyze and clear cached query plans:

- db.collection.getPlanCache().clear() – Clears all cached plans.
- db.collection.getPlanCache().clearPlansByQuery(query) – Clears cached plans for a specific query.
- db.collection.getPlanCache().listQueryShapes() – Lists all query shapes in the plan cache.

User Management Methods

Used to create and manage database users:

- db.createUser() – Adds a new user with roles.
- db.updateUser() – Updates user information.
- db.dropUser() – Deletes a user.
- db.getUsers() – Lists all users in the current database.

Role Management Methods

Used for creating and assigning custom roles:

- db.createRole() – Defines a new custom role.
- db.updateRole() – Modifies existing roles.
- db.dropRole() – Removes a role.
- db.getRoles() – Lists roles in the database.

MongoDB Replication Methods

Replication ensures data redundancy and availability:

- rs.initiate() – Initializes a replica set.
- rs.add() – Adds a new member to the replica set.
- rs.status() – Shows the status of the replica set.
- rs.conf() – Displays the configuration.
- rs.stepDown() – Forces primary to step down.

Connectivity

Java -> MongoDB

Use the MongoDB Java Driver to connect and operate:

```
MongoClient mongoClient = new MongoClient("localhost", 27017);
MongoDatabase db = mongoClient.getDatabase("mydb");
```

PHP -> MongoDB

Use the MongoDB PHP Extension:

```
$manager = new MongoDB\Driver\Manager("mongodb://localhost:27017");
```

Python -> MongoDB

Use the PyMongo library:

```
from pymongo import MongoClient
client = MongoClient("mongodb://localhost:27017/")
db = client["mydb"]
```

Experiment 17:

Aim: Using MongoDB Shell: Collection and Cursor Methods

#Find all employee details

Code: db.Employee_details.find()

Output:

```
Employee> db.Employee_details.find()
[
  {
    _id: ObjectId('687f1a20a14c59eaaaeec4a9'),
    Employee_name: 'Rahul',
    Employee_number: '1001',
    Employee_gender: 'Male',
    Department: 'Accounts',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-22T04:57:04.894Z'),
    Address: 'HYDERABAD',
    skills: [ 'SQL', 'Excel', 'Auditing' ],
    performance_rating: 4.5
  },
  {
    _id: ObjectId('687f1a20a14c59eaaaeec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
Employee>
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-22T04:57:04.894Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687f1a20a14c59eaaaeec4ab'),
    Employee_name: 'Swetha',
    Employee_number: '1003',
    Employee_gender: 'Female',
    Department: 'Manager',
    Salary: 70000,
    is_active: false,
    Date: ISODate('2025-07-22T04:57:04.894Z'),
    Address: 'HYDERABAD',
    skills: [ 'Leadership', 'Project Management', 'Reporting' ],
    performance_rating: 4.8
  },
```

#Find employees in the "Cashier" department

db.Employee_details.find({Department:"Cashier"})

```
Employee> db.Employee_details.find({Department:"Cashier"})
[
  {
    _id: ObjectId('687f1a20a14c59eaaaeec4aa'),
    Employee_name: 'Rohit',
    Employee_number: '1002',
    Employee_gender: 'Male',
    Department: 'Cashier',
    Salary: 40000,
    is_active: true,
    Date: ISODate('2025-07-22T04:57:04.894Z'),
    Address: 'HYDERABAD',
    skills: [ 'Cash Handling', 'Customer Service' ],
    performance_rating: 4
  },
  {
    _id: ObjectId('687f1a20a14c59eaaaeec4ac'),
    Employee_name: 'Indu',
    Employee_number: '1005',
    Employee_gender: 'Female',
    Department: 'Cashier',
    Salary: 50000,
    is_active: true,
    Date: ISODate('2025-07-22T04:57:04.894Z'),
    Address: 'HYDERABAD',
    skills: [ 'Accounting Software', 'Taxation', 'bills' ],
    performance_rating: 4.7
  }
]
```

#Count all employees

db.Employee_details.countDocuments()

#Count active employees only

db.Employee_details.countDocuments({is_active:true})

```
Employee> db.Employee_details.countDocuments()
5
Employee> db.Employee_details.countDocuments({is_active:true})
3
Employee>
```

 # Print the name of every employee

```
db.Employee_details.find().forEach(function(doc) {
print("Name:", doc.Employee_name)
})
```

```
Employee> db.Employee_details.find().forEach(function(doc) {
...   print("Name:", doc.Employee_name)
... })
Name: Rahul
Name: Rohit
Name: Swetha
Name: Indu
Name: Sohail
```

 # Print all department names

```
db.Employee_details.find().forEach(function(doc) {
print("Department:", doc.Department)
})
```

```
Employee> db.Employee_details.find().forEach(function(doc) {
...   print("Department:", doc.Department)
... })
Department: Accounts
Department: Cashier
Department: Manager
Department: Cashier
Department: Assistant_Manager
```

 # Here we Store all employee documents in an array

```
var employees = db.Employee_details.find().toArray()
```

 # Print number of employees with salary above 50000

```
print(employees.filter(emp => emp.Salary > 50000).length)
```

```
Employee> var employees = db.Employee_details.find().toArray()

Employee> print(employees.filter(emp => emp.Salary > 50000).length)
2
```

Experiment 18:

Query Plan Cache and Role Management Commands

 # Access the plan cache of a collection
db.Employee_details.getPlanCache()

 # Clear the cached query plans for Employee_details
db.Employee_details.getPlanCache().clear()
Note: After execution of the above line the code will not be visible

```
Employee> db.Employee_details.getPlanCache()
PlanCache for collection Employee_details.
Employee>
{ ok: 1 }
Employee>
```

Role Management: db.createRole()

MongoDB allows creation of custom roles with specific privileges to collections or actions.

Example: Create a role called "employeeViewer" that only allows reading from the Employee_details collection:

```
db.createRole({
  role: "employeeViewer",
  privileges: [
    {
      resource: { db: "Employee", collection: "Employee_details" },
      actions: ["find"]
    }
  ],
  roles: []
})
```

```
Employee>
{ ok: 1 }
...   role: "employeeViewer",
...   privileges: [
...     {
...       resource: { db: "your_database_name", collection: "Employee_details" },
...       actions: ["find"]
...     }
...   ],
...   roles: []
... })
MongoServerError[InvalidRoleModification]: Roles on the 'Employee' database cannot be granted privileges that target other databases or the cluster
...   role: "employeeViewer",
...   privileges: [
...     {
...       resource: { db: "Employee", collection: "Employee_details" },
...       actions: ["find"]
...     }
...   ],
...   roles: []
... })
{ ok: 1 }
Employee>
```

Experiment 19:

Aim:Python MongoDB CRUD App using Pymongo

Using MongoDB with Python is a powerful combination for managing and interacting with NoSQL databases. Python's pymongo library is the most commonly used library for connecting and working with MongoDB. Here's a quick guide to get you started:

1. Install Required Libraries

Make sure you have pymongo installed. You can install it using pip:

Go to Command Prompt and write

>pip install pymongo

```
C:\Users\DELL>pip install pymongo
Collecting pymongo
  Downloading pymongo-4.13.2-cp311-cp311-win_amd64.whl.metadata (22 kB)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
Downloading pymongo-4.13.2-cp311-cp311-win_amd64.whl (851 kB)
   ---------------------------------------- 851.5/851.5 kB 1.2 MB/s eta 0:00:00
Downloading dnspython-2.7.0-py3-none-any.whl (313 kB)
   ---------------------------------------- 313.6/313.6 kB 1.1 MB/s eta 0:00:00
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.7.0 pymongo-4.13.2

C:\Users\DELL>
```

After Installation of Pymongo we have Import and Create a MongoDB Client

Now in notepad write the following code first the below code should be Executed

Code: from pymongo import MongoClient

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

db = client["company"]

collection = db["Employee_details"]

After running the above code add the below lines in the file to Create

 # Insert one document

Code: collection.insert_one({

```
    "Employee_name": "Kiran",
    "Department": "Sales",
    "Salary": 45000
})
```

 # Insert multiple documents

Code: collection.insert_many([

```
    {"Employee_name": "Tina", "Department": "HR", "Salary": 52000},
    {"Employee_name": "Amar", "Department": "IT", "Salary": 60000}
])
```

To read the collection from the file add the below code to the file

 # Find one document

Code: employee = collection.find_one({"Employee_name": "Kiran"})

```
print(employee)
```

 # Find all documents

Code: for emp in collection.find():

```
    print(emp)
```

 # Filter documents

Code: high_salary = collection.find({"Salary": {"$gt": 50000}})

```
for emp in high_salary:
    print(emp)
```

```
C:\Users\DELL\Desktop\mongo>python cncn.py
{'_id': ObjectId('687f33a7d37092e08b3e5f1e'), 'Employee_name': 'Kiran', 'Department': 'Sales', 'Salary': 45000}
{'_id': ObjectId('687f33a7d37092e08b3e5f1e'), 'Employee_name': 'Kiran', 'Department': 'Sales', 'Salary': 45000}
{'_id': ObjectId('687f33a7d37092e08b3e5f1f'), 'Employee_name': 'Tina', 'Department': 'HR', 'Salary': 52000}
{'_id': ObjectId('687f33a7d37092e08b3e5f20'), 'Employee_name': 'Amar', 'Department': 'IT', 'Salary': 60000}
{'_id': ObjectId('687f33a7d37092e08b3e5f1f'), 'Employee_name': 'Tina', 'Department': 'HR', 'Salary': 52000}
{'_id': ObjectId('687f33a7d37092e08b3e5f20'), 'Employee_name': 'Amar', 'Department': 'IT', 'Salary': 60000}

C:\Users\DELL\Desktop\mongo>
```

Update

 # Update one document

Code: collection.update_one(

{"Employee_name": "Tina"},

{"$set": {"Salary": 55000}}

)

 # Update many documents

Code: collection.update_many(

{"Department": "Sales"},

{"$inc": {"Salary": 5000}}

)

In Order to display the content write the Find and Print:

Code: for emp in collection.find():

print(emp)

```
C:\Users\DELL\Desktop\mongo>python cncn.py
{'_id': ObjectId('687f33a7d37092e08b3e5f1e'), 'Employee_name': 'Kiran', 'Department': 'Sales', 'Salary': 50000}
{'_id': ObjectId('687f33a7d37092e08b3e5f1f'), 'Employee_name': 'Tina', 'Department': 'HR', 'Salary': 55000}
{'_id': ObjectId('687f33a7d37092e08b3e5f20'), 'Employee_name': 'Amar', 'Department': 'IT', 'Salary': 60000}

C:\Users\DELL\Desktop\mongo>
```

Delete

 # Delete one document

Code: collection.delete_one({"Employee_name": "Amar"})

 # Delete multiple documents

Code: collection.delete_many({"Salary": {"$lt": 50000}})

```
C:\Users\DELL\Desktop\mongo>python cncn.py

C:\Users\DELL\Desktop\mongo>python cncn.py
{'_id': ObjectId('687f33a7d37092e08b3e5f1e'), 'Employee_name': 'Kiran', 'Department': 'Sales', 'Salary': 50000}
{'_id': ObjectId('687f33a7d37092e08b3e5f1f'), 'Employee_name': 'Tina', 'Department': 'HR', 'Salary': 55000}

C:\Users\DELL\Desktop\mongo>
```

Experiment 20:
Aim: Java MongoDB Connection Example

Use MongoDB Java Driver to connect and run basic operations

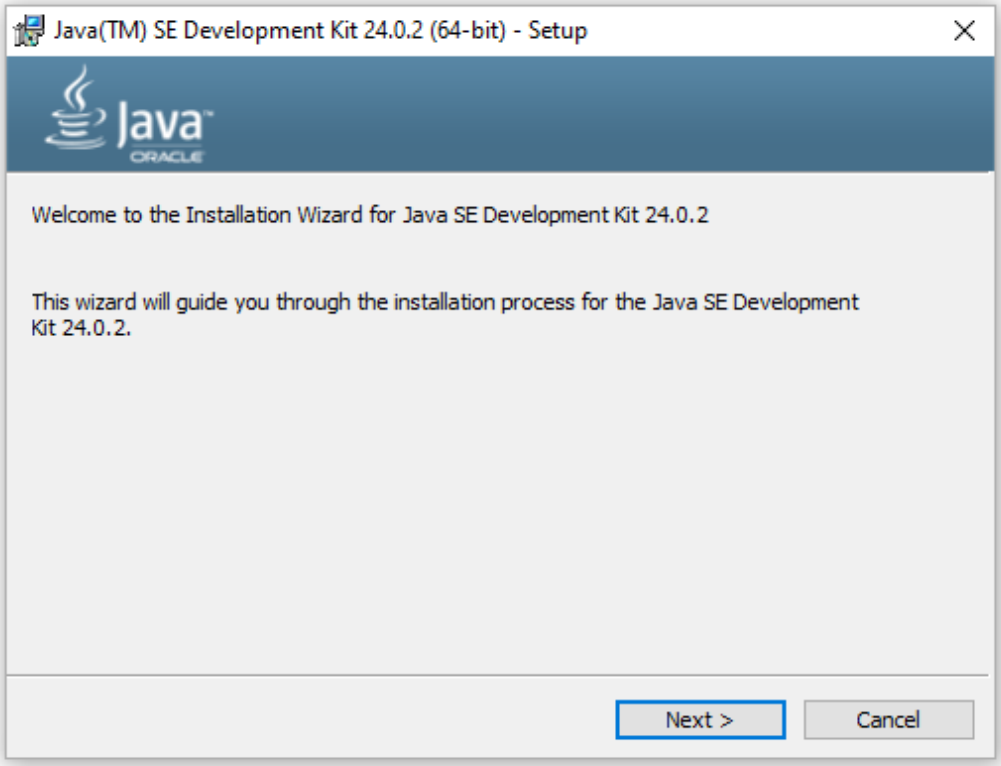
To Connect MongoDB with Java we have to Install JDK, Ecllipse and Maven

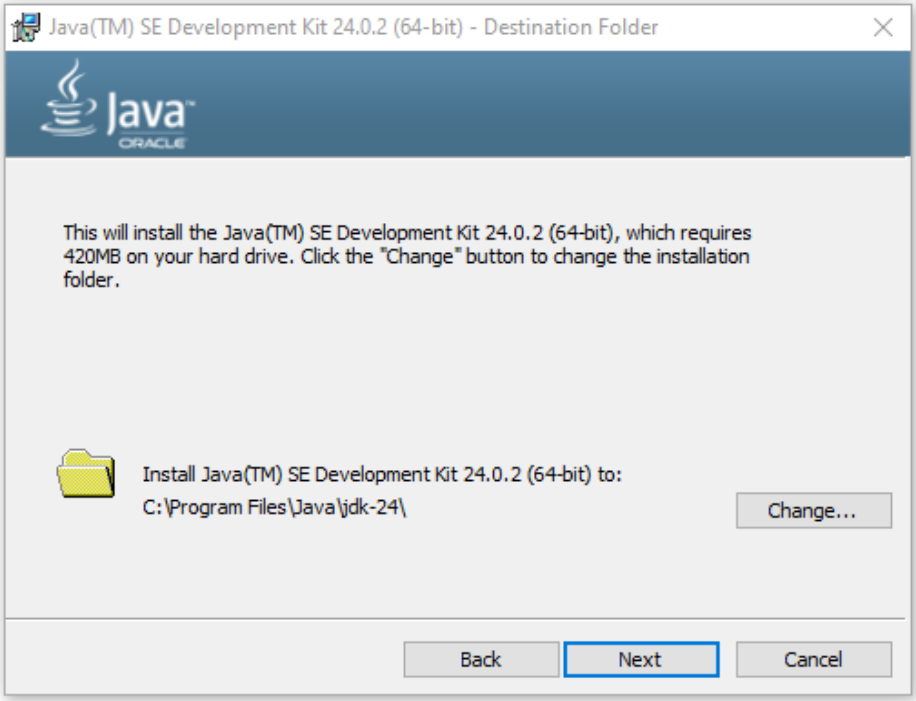
JDK

- Download the JDK Installer
- Visit the Oracle JDK Downloads page or OpenJDK and choose the latest version (e.g., JDK 24).
- Select the Windows x64 installer (.exe or .msi).

Run the Installer

- Double-click the downloaded file.
- Follow the wizard prompts and accept the license agreement.
- Choose the default installation path or customize it.

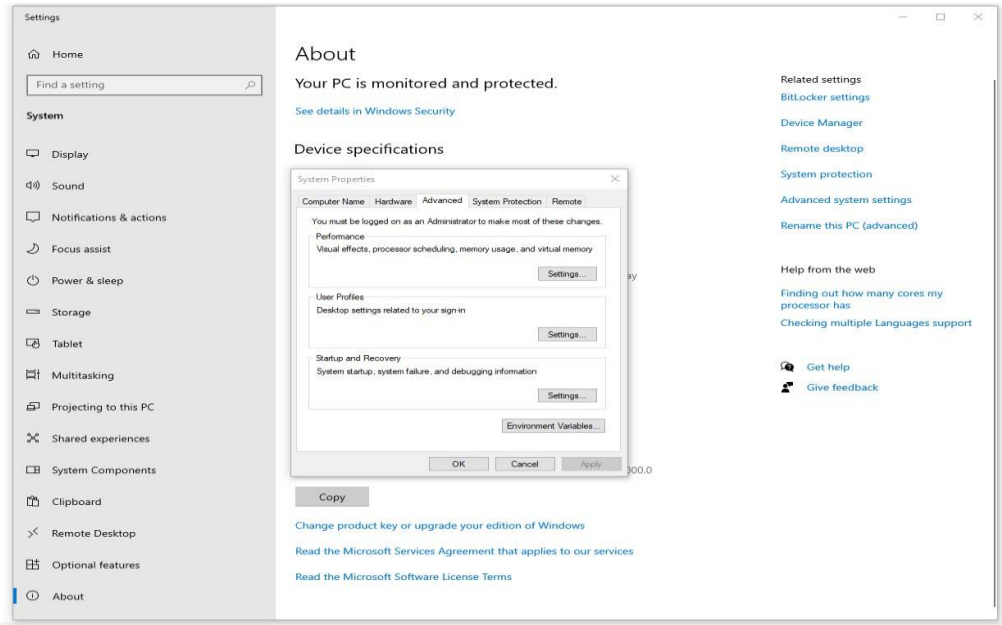



Java(TM) SE Development Kit 24.0.2 (64-bit) - Destination Folder
Java
ORACLE
This will install the Java(TM) SE Development Kit 24.0.2 (64-bit), which requires 420MB on your hard drive. Click the "Change" button to change the installation folder.
Install Java(TM) SE Development Kit 24.0.2 (64-bit) to:
C:\Program Files\Java\jdk-24\
Change...
Back
Next
Cancel


Java(TM) SE Development Kit 24.0.2 (64-bit) - Complete
Java
ORACLE
Java(TM) SE Development Kit 24.0.2 (64-bit) Successfully Installed
Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.
Next Steps
Close

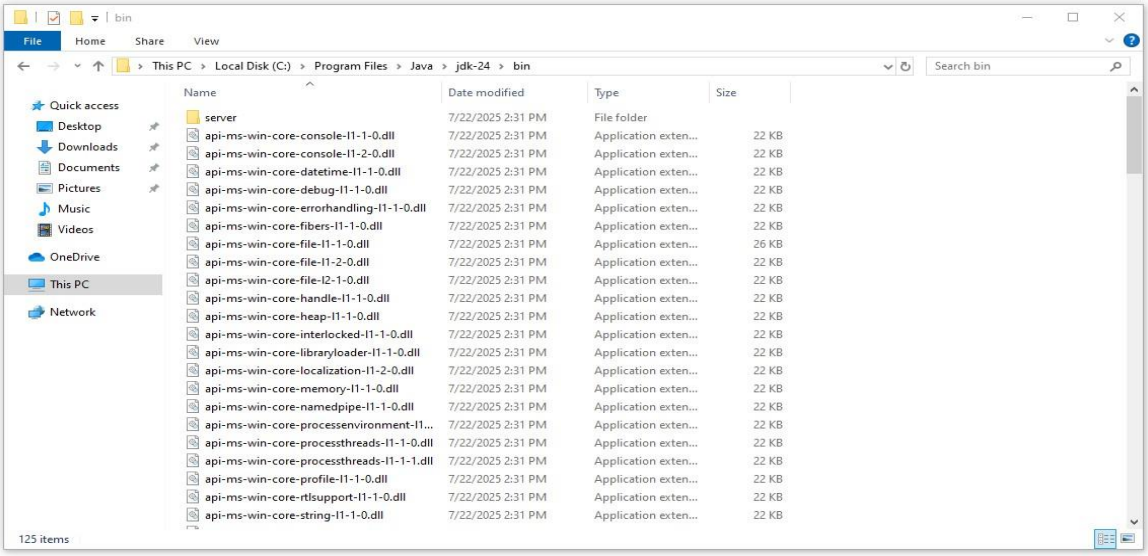
Set Environment Variables

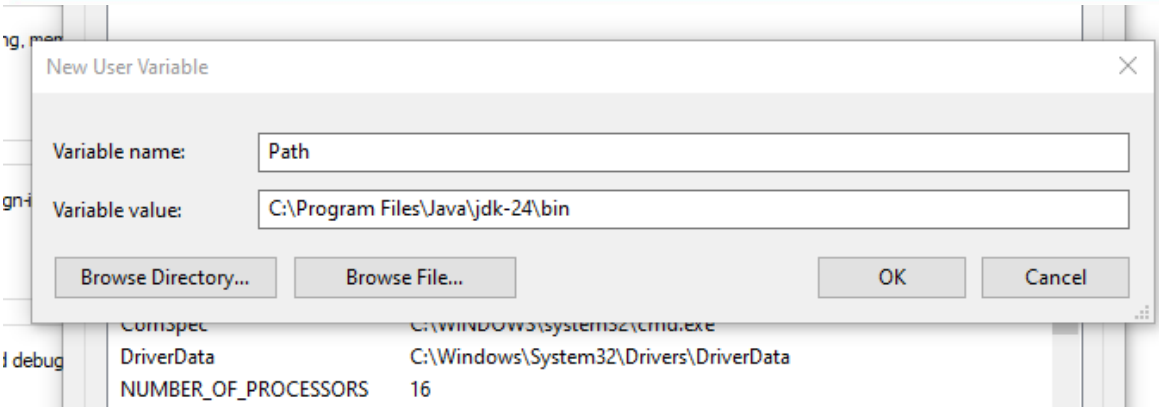
Open System Properties → Advanced → Environment Variables.



Under System Variables, add:

JAVA_HOME → C:\Program Files\Java\jdk-24
Edit Path → Add C:\Program Files\Java\jdk-24\bin

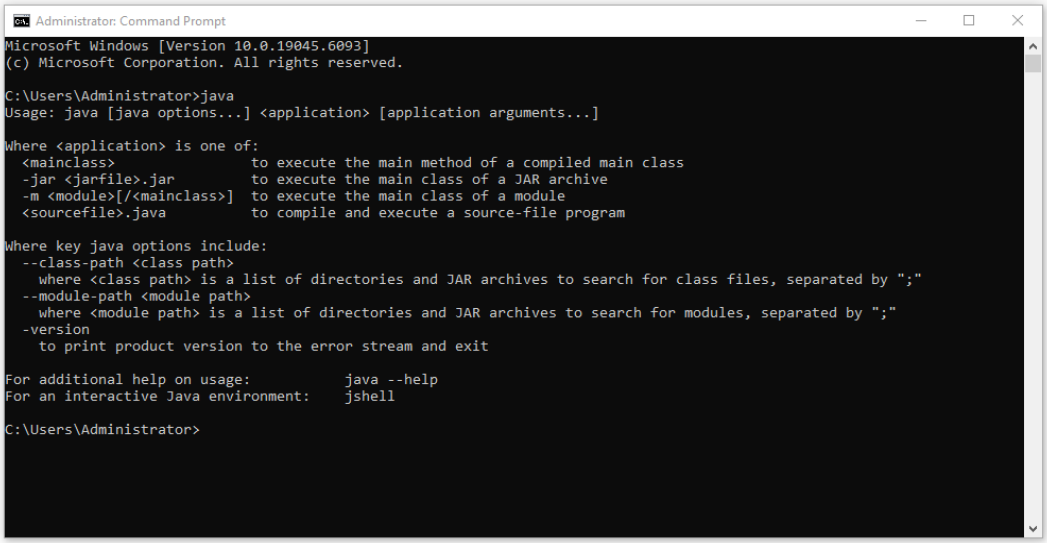




Verify Installation

Open Command Prompt and run:

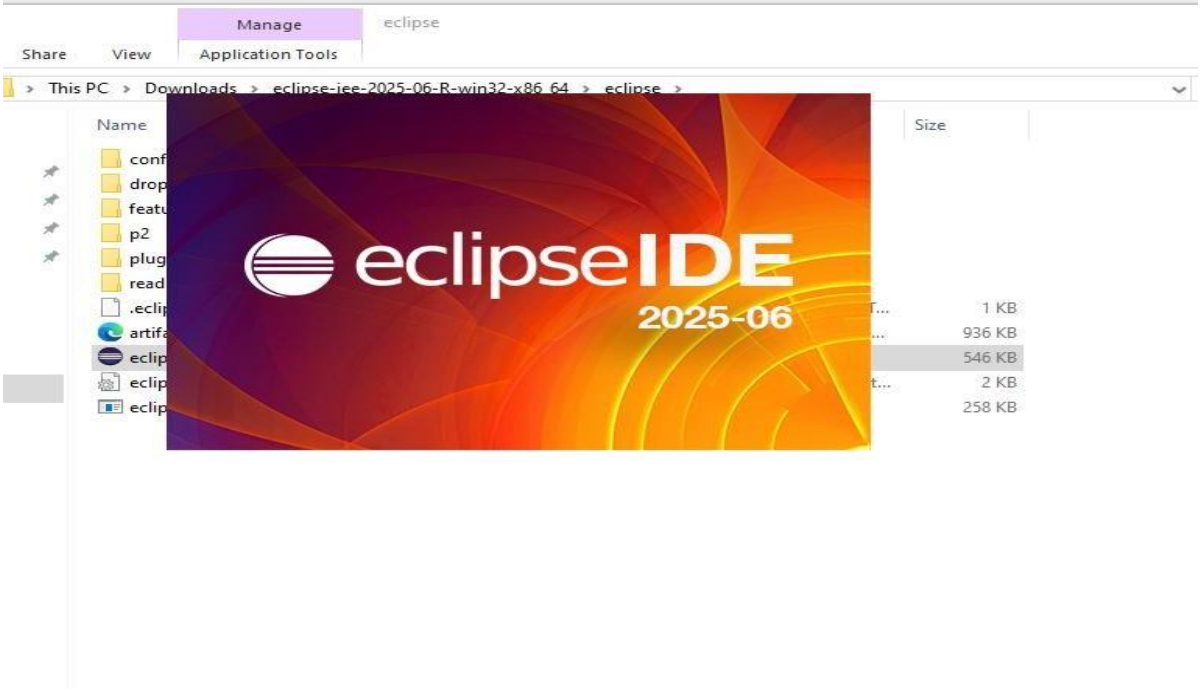
java -version
javac -version



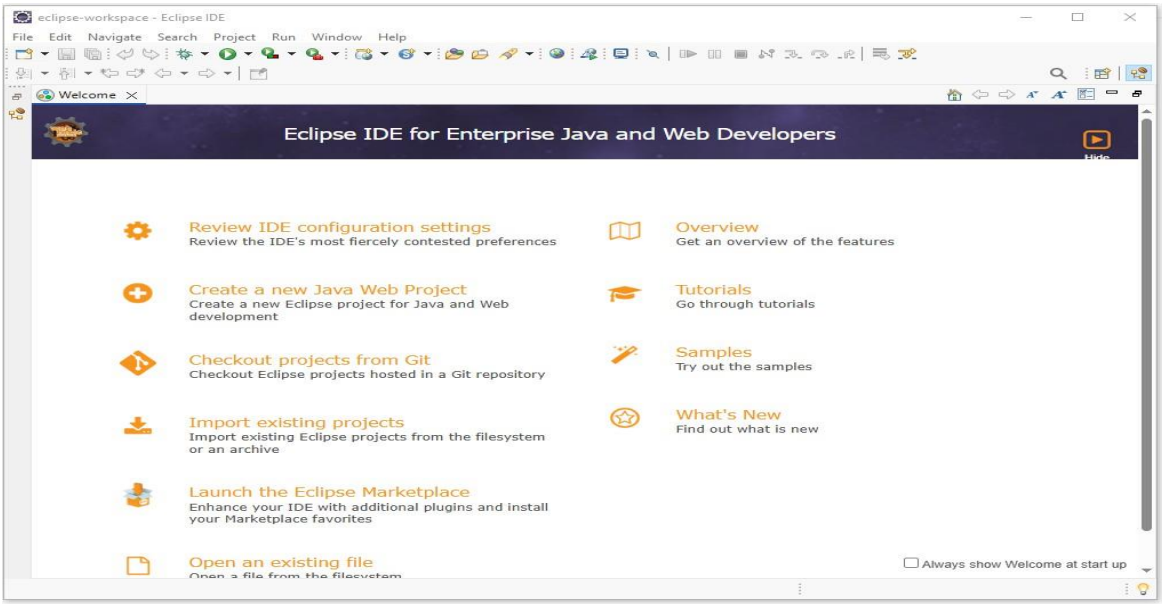
Ecllipse

- Download the Eclipse Installer
- Visit the official Eclipse downloads page and click Download x86_64 for Windows.
- Alternatively, you can go directly to the Eclipse Installer 2025-06 R which includes a built-in JRE.

- Run the Installer
- Locate the downloaded file (eclipse-inst-jre-win64.exe) and double-click to launch.
- Choose Installation Folder
- Select your preferred installation directory.
- Click Install and wait for the setup to complete.
- Launch Eclipse



After installation, click Launch.



- Choose a workspace folder (where your projects will be saved).
- Verify Java Integration
- Make sure you have JDK installed and configured.
- Eclipse should automatically detect it, but you can set the path manually in preferences if needed.

- Create Your First Java Project
- Go to File → New → Java Project.
- Name your project and start coding!
- You should see the installed version details.
- Maven
- To Download Maven
- Go to the Apache Maven Downloads page
- Download the latest binary zip (e.g., apache-maven-3.9.6-bin.zip)

Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the installation instructions. Use a source archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the release bundles against the public KEYS used by the Apache Maven developers.

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.9.11-bin.tar.gz	apache-maven-3.9.11-bin.tar.gz.sha512	apache-maven-3.9.11-bin.tar.gz.asc
Binary zip archive	apache-maven-3.9.11-bin.zip	apache-maven-3.9.11-bin.zip.sha512	apache-maven-3.9.11-bin.zip.asc
Source tar.gz archive	apache-maven-3.9.11-src.tar.gz	apache-maven-3.9.11-src.tar.gz.sha512	apache-maven-3.9.11-src.tar.gz.asc
Source zip archive	apache-maven-3.9.11-src.zip	apache-maven-3.9.11-src.zip.sha512	apache-maven-3.9.11-src.zip.asc

- 3.9.11 Release Notes and Release Reference Documentation
- latest source code from source repository

- Extract the Archive
- Unzip the downloaded file to a directory like: C:\Program Files\Maven\apache-maven-3.9.6
- Set Environment Variables
- Open System Properties → Advanced → Environment Variables
- Add a new System Variable:
- MAVEN_HOME → C:\Program Files\Maven\apache-maven-3.9.6
- Edit the Path variable:
- Add: %MAVEN_HOME%\bin
- Verify Installation

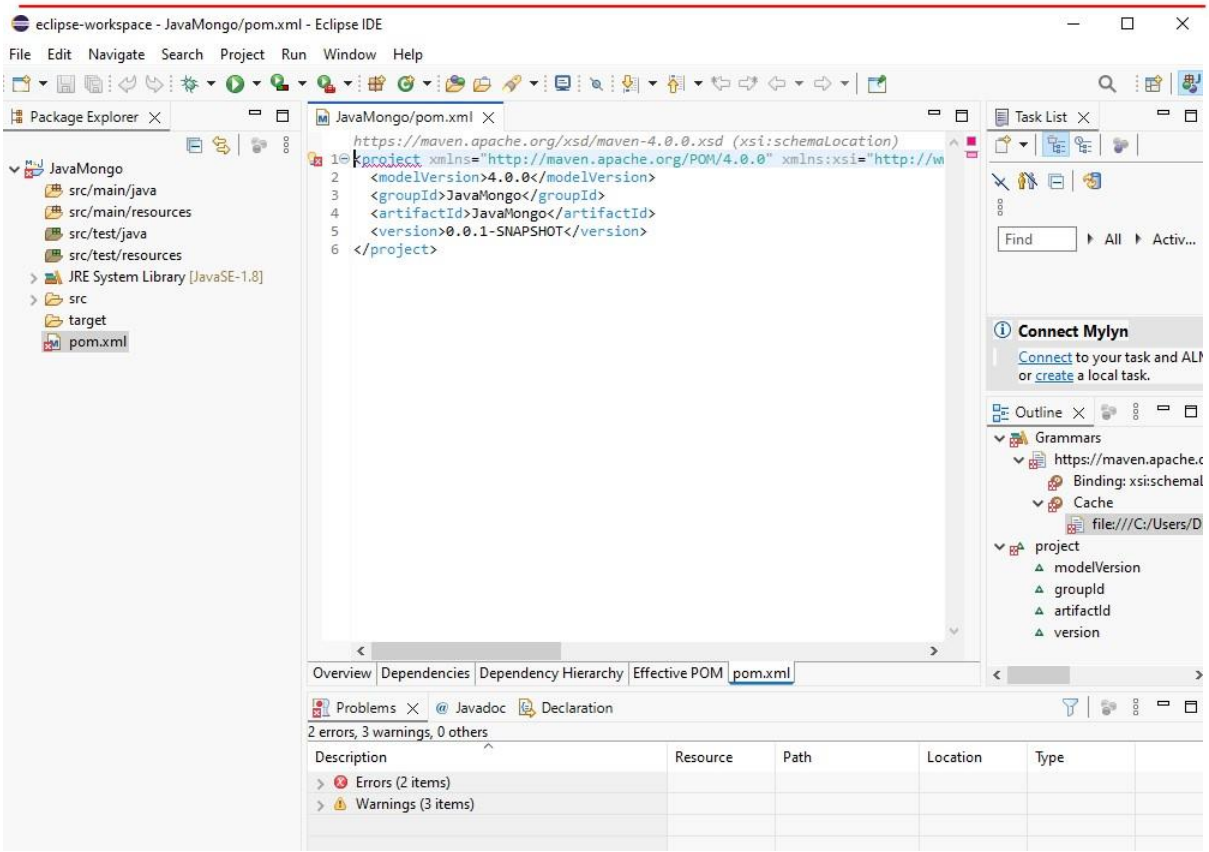
Open Command Prompt and run:

mvn -version

```
C:\Users\DELL>mvn -v
Apache Maven 3.9.11 (3e54c93a704957b63ee3494413a2b544fd3d825b)
Maven home: C:\apache-maven-3.9.11
Java version: 24.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-24
Default locale: en_US, platform encoding: UTF-8
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\DELL>
```

- Go to Ecllipse
- select New Java Project/ Maven Project
- which is used to add Dependencies to a Java Driver in order to connect mongodb and java together go to file
- Click on new
- then other and select Maven
- in maven select Maven Project
- click on next and select the check box of Create a Simple Project and next
- Give the GroupId and Artifact ID : JavaMongo
- And select Finish
- In the left bar of Package Explorer In JavaMongo select pom.xml which is used to add dependencies
- Maven will give access to dependencies to our mongodb and java driver

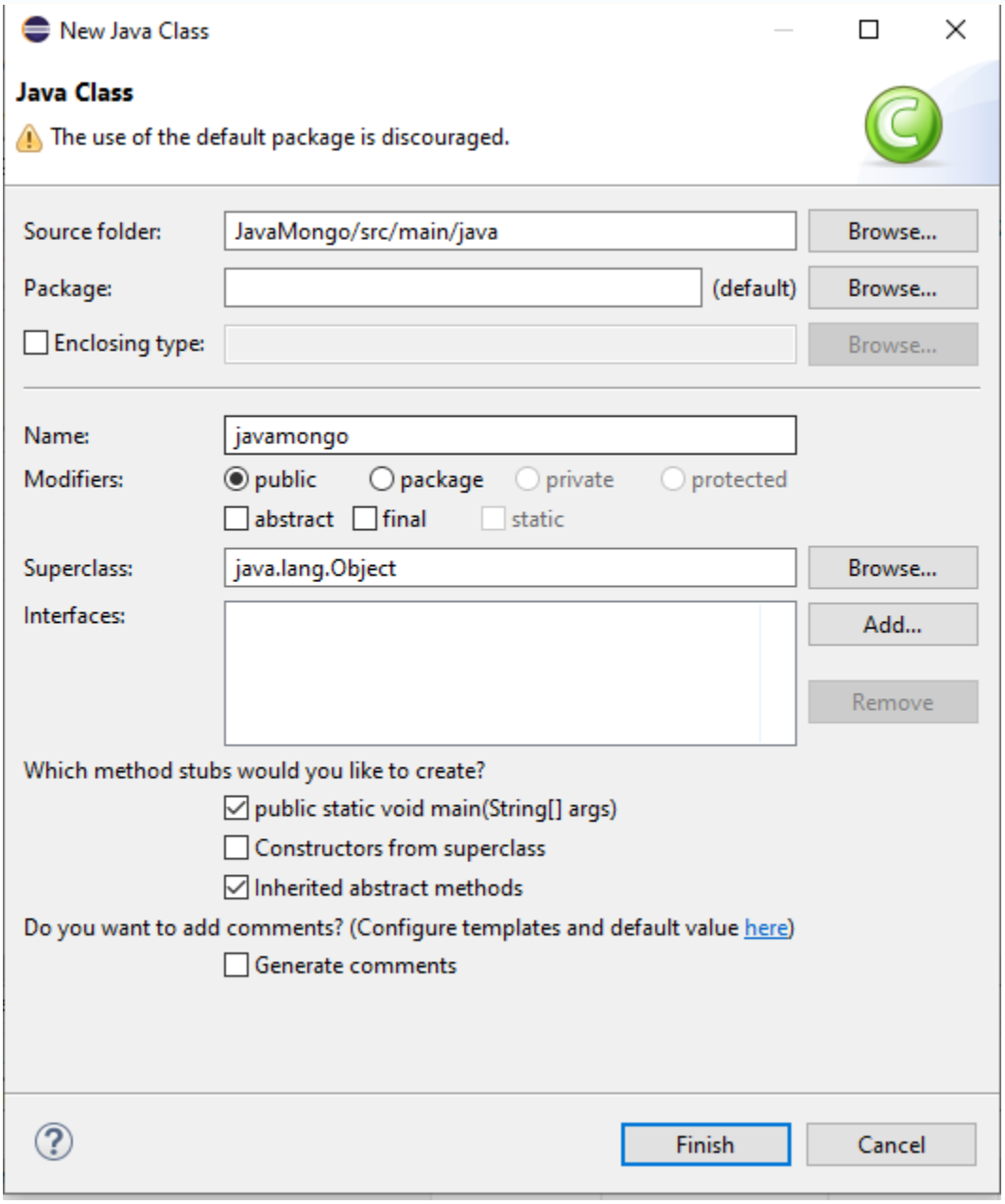


- Now we have to add our dependency after Version

Code: <dependencies>

```
<dependency>
<groupId>org.mongodb</groupId>
<artifactId>mongodb-driver-sync</artifactId>
<version>4.10.2</version> <!-- or latest stable -->
</dependency>
</dependencies>
```

- Now save the file by Control+s
- Now right click on src/main/java and select class give the name as javamongo and select Finish


New Java Class
Java Class
The use of the default package is discouraged.
Source folder:
JavaMongo/src/main/java
Browse...
Package:
(default)
Browse...
Enclosing type:
Browse...
Name:
javamongo
Modifiers:
public
package
private
protected
abstract
final
static
Superclass:
java.lang.Object
Browse...
Interfaces:
Add...
Remove
Which method stubs would you like to create?
public static void main(String[] args)
Constructors from superclass
Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
Generate comments
Finish
Cancel

Now add the Package and Import Drivers which are mentioned Below

Code:

```
package mongoJava;

import org.bson.Document;

import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

public class javamongo {

public static void main(String[] args) {
// TODO Auto-generated method stub
MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017");
System.out.println("Successfully Created MongoDB connection");
}

}
```

- Now go to C Drive go to MongoDB folder go to server and then to bin
- Now in the Address bar write cmd and press enter now we are directed to Command Prompt now write mongod and press enter
- Now we have Successfully made a connection with mongodb client
- Now select the Libraries and press Control+Shift+o
- Now right click and select run as>Java Application

JavaMongo/pom.xml | javamongo.java

```java
package mongoJava;

import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;

public class javamongo {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        MongoClient mongoClient = MongoClients.create("mongodb://localhost:27017");
        System.out.println("Successfully Created MongoDB connection");
    }

}
```

Problems | Javadoc | Declaration | Console

```
<terminated> javamongo (1) [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.
Jul 22, 2025 3:22:39 PM com.mongodb.internal.diagnostics.logging.Loggers shouldUseSLF4J
WARNING: SLF4J not found on the classpath.  Logging is disabled for the 'org.mongodb.driver' component
Successfully Created MongoDB connection
```

Code:

```
package mongoJava;
import javax.swing.text.Document;

import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;

public class javamongo {

public static void main(String[] args) {
// Creating a Mongo client
MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
System.out.println("Created Mongo Connection successfully");
MongoDatabase db = mongoClient.getDatabase("mongodbjava");
System.out.println("Get database is successful");
System.out.println("Below are list of databases present in MongoDB");
// To get all database names
 MongoCursor<String> dbsCursor = mongoClient.listDatabaseNames().iterator();
   while(dbsCursor.hasNext()) {
         System.out.println(dbsCursor.next());
   }
   //Inserting sample record by creating collection and document.
 MongoCollection<org.bson.Document>  collection= db.getCollection("javaprogram");
 org.bson.Document doc =(org.bson.Document) new org.bson.Document("name","hello world");
 collection.insertOne(doc);
System.out.println("########### Insertion is completed  ################");
MongoCursor<String> dbsCursord = mongoClient.listDatabaseNames().iterator();
while(dbsCursord.hasNext()) {
         System.out.println(dbsCursord.next());
      }
}

}
```

```
<terminated> javamongo (1) [Java Application] C:\Users\DELL\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.(
Jul 22, 2025 3:31:38 PM com.mongodb.internal.diagnostics.logging.Loggers shouldUseSLF4J
WARNING: SLF4J not found on the classpath.  Logging is disabled for the 'org.mongodb.driver' component
Successfully Created MongoDB connection
Below are list of databases present in MongoDB
Employee
Students
admin
company
config
local
```

```
Jul 22, 2025 3:47:38 PM com.mongodb.internal.diagnostics.logging.Loggers shouldUse
WARNING: SLF4J not found on the classpath.  Logging is disabled for the 'org.mongo
Successfully Created MongoDB connection
Get database is successful
Below are list of databases present in MongoDB
Employee
Students
admin
company
config
local
########### Insertion is completed  ###############
Employee
Students
admin
company
config
local
mongodbjava
```

Viva Questions

S.No	Question	CO	Blooms Taxonomy
1	What does the .find() method do in MongoDB?	CO5	Remember
2	How is .countDocuments() used to count documents?	CO5	Understand
3	Explain the usage of .forEach() in cursor operations.	CO5	Apply
4	What is the purpose of .toArray() method?	CO5	Understand
5	How do you clear the query plan cache in MongoDB?	CO5	Apply
6	How do you create a custom role in MongoDB?	CO5	Apply
7	How is a role assigned to a MongoDB user?	CO5	Apply
8	How do you establish a connection to MongoDB using Python?	CO5	Apply
9	How is a document inserted into a MongoDB collection using PyMongo?	CO5	Apply
10	How do you read documents using PyMongo?	CO5	Apply
11	How do you update a document in MongoDB using Python?	CO5	Apply
12	How do you delete a document in MongoDB using PyMongo?	CO5	Apply
13	How do you connect to MongoDB using Java?	CO5	Apply
14	How is a document inserted in MongoDB using Java?	CO5	Apply
15	How do you read documents in MongoDB using Java?	CO5	Apply
16	How do you connect to MongoDB in PHP?	CO5	Apply
17	How do you insert a document using MongoDB in PHP?	CO5	Apply
18	How do you retrieve documents in PHP from MongoDB?	CO5	Apply
19	What is replication in MongoDB?	CO5	Understand
20	What is a replica set in MongoDB?	CO5	Understand
21	How do you initiate a replica set in MongoDB shell?	CO5	Apply
22	How do you check the status of a replica set?	CO5	Apply
23	How do you add a member to a replica set?	CO5	Apply
24	What is the default read preference in a replica set?	CO5	Remember
25	How can you perform a read operation from a secondary replica?	CO5	Apply
26	What is the default port used by MongoDB?	CO5	Remember
27	How do you list all collections in a MongoDB database?	CO5	Apply
28	How do you drop a database using the MongoDB shell?	CO5	Apply
29	How do you drop a collection from a MongoDB database?	CO5	Apply
30	How do you list all users in a MongoDB database?	CO5	Apply