RQF LEVEL 5

SWDND501

SOFTWARE DEVELOPMENT

NoSQL Database Development

TRAINEE'S MANUAL

October, 2024

NOSQL DATABASE DEVELOPMENT

KOICA Korea International Cooperation Agency

TQUM TVET Quality Management Project

2024

AUTHOR'S NOTE PAGE (COPYRIGHT)

Original published version: October 2024

ACKNOWLEDGEMENTS

The publisher would like to thank the following for their assistance in the elaboration of this textbook:

Rwanda TVET Board (RTB) extends its appreciation to all parties who contributed to the development of the trainer's and trainee's manuals for the TVET Certificate V in Software Development, specifically for the module **"SWDND501: NOSQL Database Development"**

We extend our gratitude to KOICA Rwanda for its contribution to the development of these training manuals and for its ongoing support of the TVET system in Rwanda.

We extend our gratitude to the TQUM Project for its financial and technical support in the development of these training manuals.

We would also like to acknowledge the valuable contributions of all TVET trainers and industry practitioners in the development of this training manual.

The management of Rwanda TVET Board extends its appreciation to both its staff and the staff of the TQUM Project for their efforts in coordinating these activities.

This training manual was developed:

Under Rwanda TVET Board (RTB) guiding policies and directives

Under Financial and Technical support of

COORDINATION TEAM

RWAMASIRABO Aimable

MARIA Bernadette M. Ramos

MUTIJIMA Asher Emmanuel

Production Team

Authoring and Review

MUGISHA Pacifique

MUKAMUHOZA Liberee

NSENGIYUMVA Emmanuel

Validation

.NIYONSABA Godelive

HABANABAKIZE Jerome

Conception, Adaptation and Editorial works

HATEGEKIMANA Olivier

GANZA Jean Francois Regis

HARELIMANA Wilson

NZABIRINDA Aimable

DUKUZIMANA Therese

NIYONKURU Sylvestre

UMEREWENEZA Naasson

Formatting, Graphics, Illustrations, and infographics

YEONWOO Choe

SUA Lim

SAEM Lee

SOYEON Kim

WONYEONG Jeong

HABIMANA Emmanuel

Financial and Technical support

KOICA through TQUM Project

TABLE OF CONTENT

AUTHOR'S NOTE PAGE (COPYRIGHT) ---------- iii
ACKNOWLEDGEMENTS ---------- iv
TABLE OF CONTENT ---------- vii
ACRONYMS ---------- ix
INTRODUCTION ---------- 1
MODULE CODE AND TITLE: SWDND501 NOSQL DATABASE DEVELOPMENT ---------- 2
Learning Outcome 1: Prepare Database Environment ---------- 3
Key Competencies for Learning Outcome 1: Prepare Database Environment ---------- 4
Indicative content 1.1: Identifying database requirements ---------- 6
Indicative content 1.2: Analysing NoSQL database ---------- 20
Indicative content 1.3: Preparing database environment ---------- 27
Learning outcome 1 end assessment ---------- 38
References : ---------- 41
Learning Outcome 2: Design NoSQL Database ---------- 42
Key Competencies for Learning Outcome 2: Design No SQL Database ---------- 43
Indicative content 2.1: Selecting tools of drawing databases. ---------- 45
Indicative content 2.2: Creating conceptual data model. ---------- 52
Indicative content 2.3: Designing MongoDB database schema ---------- 66
Learning outcome 2 end assessment ---------- 74
References: ---------- 77
Learning Outcome 3: Implement Database Design ---------- 78
Key Competencies for Learning Outcome3: Implement Database Design ---------- 79
Indicative content 3.1: Perform MongoDB data definition ---------- 81
Indicative content 3.2: MongoDB data Manipulating ---------- 87
Indicative content 3.3: Applying query optimizations ---------- 101
Learning outcome 3 end assessment ---------- 107
References: ---------- 111
Learning Outcome 4: Manage MongoDB Database ---------- 112
Key Competencies for Learning Outcome 4: Manage Mongo DB Database ---------- 113

Indicative content 4.1: Management of database users----------115
Indicative content 4.2: Securing database----------131
Indicative content 4.3: Deployment of database ----------137
Learning outcome 4 end assessment ----------155
References: ----------158

ACRONYMS

CBA: Competency-Based Assessment

DFD: Data Flow Diagram

IAP: Industrial Attachment Program

ICT: Information and Communications Technology

IT: Information Technology

KOICA: Korea International Cooperation Agency

LO: Learning Outcome

LU: Learning Unit

NO SQL: Not only structured query language

OPLOG: Operations Log

RPL: Recognition of Prior Learning

RQF: Rwanda Qualification Framework

RTB: Rwanda TVET Board

SWD: Software Development

TQUM: TVET Quality Management

TVET: Technical and Vocational Education and Training

UML: Unified Modelling Language

INTRODUCTION

This trainee's manual includes all the knowledge and skills required in Software Development, specifically for the module of **"NOSQL Database Development"** Trainees enrolled in this module will engage in practical activities designed to develop and enhance their competencies.

The development of this training manual followed the Competency-Based Training and Assessment (CBT/A) approach, offering ample practical opportunities that mirror real-life situations.

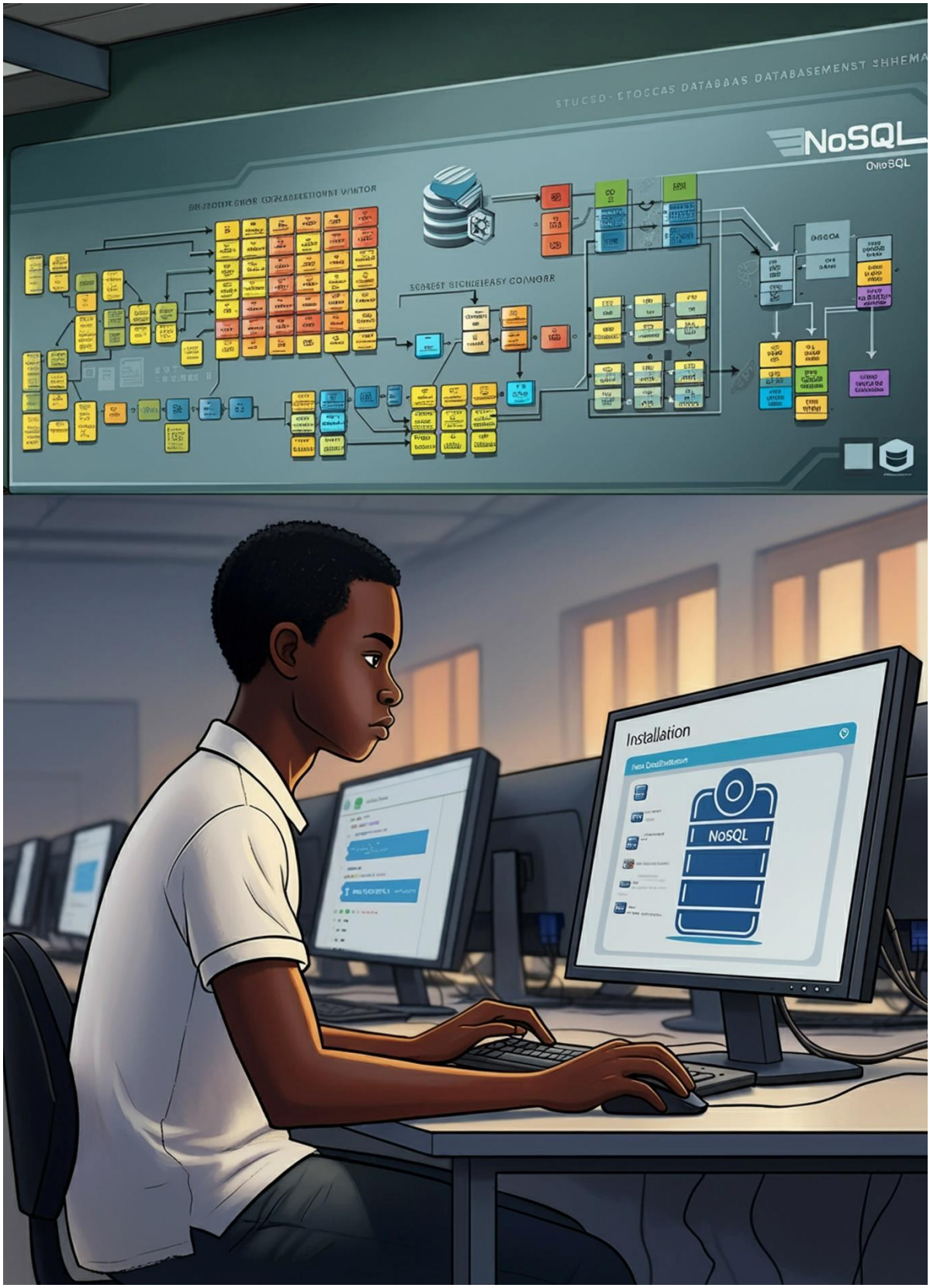
The trainee's manual is organized into Learning Outcomes, which is broken down into indicative content that includes both theoretical and practical activities. It provides detailed information on the key competencies required for each learning outcome, along with the objectives to be achieved.

As a trainee, you will start by addressing questions related to the activities, which are designed to foster critical thinking and guide you towards practical applications in the labour market. The manual also provides essential information, including learning hours, required materials, and key tasks to complete throughout the learning process.

All activities included in this training manual are designed to facilitate both individual and group work. After completing the activities, you will conduct a formative assessment, referred to as the end learning outcome assessment. Ensure that you thoroughly review the key readings and the 'Points to Remember' section.

MODULE CODE AND TITLE: SWDND501 NOSQL DATABASE DEVELOPMENT

Learning outcome 1: Prepare database environment
Learning outcome 2: Design NoSQL database
Learning outcome 3: Implement Database Design
Learning outcome 4: Manage Mongodb database


NoSQL
Installation
NoSQL

Indicative contents

1.1 Identifying Database requirements

1.2 Analysing NoSQL Database

1.3 Preparing Database environment

Key Competencies for Learning Outcome 1: Prepare Database Environment

Knowledge	Skills	Attitudes
• Identification of Database Requirements • Identification of user requirements • Identification of the Scalability of mongo DB • Description of requirements analysis process	• Creating use cases. • Performing Data Analysis • Implementing data validation • Setting up MongoDB environment	• Having. paying attention to detail while creating use cases • Having problem-Solving capability for creating use case and performing data analysis • Having Adaptability based on requirement analysis process. • Being collaborative while handling requirement analysis process.

Duration: 10 hrs

Learning outcome 1 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Describe correctly database requirement based on user requirement
2. Describe correctly the requirement analysis process based on database requirement
3. Prepare properly setup database environment based on established standard
4. Define correctly use case based on database requirements.
5. Identify correctly components of use cases based on database requirements.
6. Create properly use cases based on database requirements.

Resources

Equipment	Tools	Materials
● Computer	● Mongosh ● MongoDB compass ● Browser ● MongoDB ● Apache Cassandra	● Internet ● Electricity

Indicative content 1.1: Identifying database requirements

Duration: 3 hrs

Theoretical Activity 1.1.1: Introduction to NoSQL database

Tasks:

1: Answer the following questions:

i. Define the following terms:

a. NoSQL
b. MongoDB
c. Availability
d. Documents
e. Collection
f. Indexing
g. Optimistic Locking
h. Relationships
i. Data model
j. Schema
k. Mongosh

ii. Differentiate SQL from No SQL Database
iii. Explain the features of No SQL database
iv. What are the types of No SQL database?

2: Write your findings on papers or flipcharts.

3: Present your findings to the trainer or classmates.

4: Pay attention to the trainer's clarification and ask questions where necessary.

5: Read the key readings 1.1.1

Key readings 1.1.1.: Identification of Database Requirement

- **Definition of key terms**
  - ✓ **NoSQL**

NoSQL (Not Only SQL) databases are a class of databases that go beyond the relational database model. They are designed to handle large amounts of data, unstructured data, and distributed systems more efficiently. They often use key-value pairs, document-oriented, graph, or wide-column storage models.

NoSQL stands for Not only SQL. It is a type of database that uses non-relational data structures, such as documents, graph databases, and key-value stores to store and retrieve data. NoSQL systems are designed to be more flexible than traditional relational databases and can scale up or down easily to accommodate changes in usage or load. This makes them ideal for use in applications.

- **Applications of NoSQL Databases**

NoSQL databases, designed to handle large sets of unstructured or semi-structured data.

1. Real-time Analytics:

Processing large volumes of data in real-time such as IoT data, social media feeds, and financial market data and Analyzing data collected over time, like sensor readings, stock prices, or website traffic.

2. Content Management Systems:

Storing and managing large amounts of unstructured content, such as text, images, videos, and documents and Handling increasing volumes of content and user interactions.

3. Social Networking:

Representing complex relationships between users, posts, and groups, Handling rapidly changing data, such as likes, comments, and shares.

4. Gaming:

Maintaining and updating leader boards in real-time, Storing and managing user preferences and game progress.

5. Internet of Things (IoT): Storing and analysing data from a large number of connected devices, Processing sensor data in real-time for immediate actions or insights.

6. Big Data Analytics:

Handling massive datasets that are difficult to manage with traditional relational databases, Processing data of different formats and structures.

7. Content-Based Recommendation Systems:

Storing and analysing user preferences to recommend relevant content, Updating recommendations based on new user interactions or content.

8. Graph Databases: Analysing relationships between people or entities, Representing and querying complex relationships between concepts.

9. Geolocation-Based Services: Storing and querying spatial data, such as location coordinates and maps, Processing location data in real-time for location-based services.
10. Mobile Applications: Storing and accessing data offline for mobile applications and Synchronizing data between devices and the cloud.

- **Benefits of No SQL database**

Scalability

NoSQL databases are designed to scale out horizontally using distributed hardware clusters, rather than scaling up by adding servers.

High performance

NoSQL databases are optimized for specific data models and access patterns.

Ease of development

NoSQL databases enable faster and more iterative development because of their flexible schemas.

Cost-effectiveness

NoSQL databases can be cost-effective and can accommodate changing storage and processing requirements.

Data handling

NoSQL databases can handle large volumes of data and traffic growth. They can also analyze inventory and catalogs in real time.

Data storage

NoSQL databases can store data in a more free-form fashion without rigid schemas. They can easily handle different data formats, such as structured, semi-structured, and non-structured data.

- **Differences between SQL and NoSQL**

SQL	NO SQL
stands for structured query language	Stands for not only structured query language
Relational database management system(RDBMS)	Non-relational database management system
Suitable for structured data with predefined schema	Suitable for unstructured and semi-structured data
Data is stored in tables with columns and rows	Data is stored in collection of documents
Follows ACID properties (Atomicity, Consistency, Isolation, Durability) for transaction management	Does not necessary follow ACID properties
Supports join and complex queries	Does not Support join and complex queries
Use normalized data structures	Uses denormalized data structures

Requires vertical scaling to handle large volumes of data	Requires horizontal scaling to handle large volumes of data
Examples: Mysql, Postgresql, oracle, sql server, Microsoft sql server	Examples: mongoBD,Cassandra,Couchbase,Amazon,DynamoBD,Redis

- ✓ **MongoDB**

MongoDB is a popular NoSQL database that uses a document-oriented model. It stores data in flexible JSON-like documents, making it well-suited for handling complex and semi-structured data. MongoDB offers features like automatic sharding, replication, and indexing for scalability and performance.

- ✓ **Availability**

Availability refers to the ability of a system to be accessible and operational when needed. In the context of databases, it means that users can access and use the data without interruption. High availability systems often employ redundancy and failover mechanisms to minimize downtime.

- ✓ **Documents**

Documents are the basic unit of data storage in MongoDB. They are JSON-like structures that can contain nested objects and arrays, allowing for flexible data modeling. Documents can represent individual entities or groups of related data.

- ✓ **Collection**

Collection is a group of related documents in MongoDB. It's analogous to a table in a relational database. Collections can contain documents with different structures, as long as they share a common theme or purpose.

- ✓ **Indexing**

Indexing is a technique used to improve query performance in MongoDB. It creates data structures that allow the database to quickly locate specific documents based on their values. Indexes can be created on individual fields or combinations of fields.

- ✓ **Optimistic Locking**

Optimistic locking is a concurrency control technique that assumes conflicts are rare and checks for them only when a transaction is about to commit. If a conflict is detected, the transaction is aborted and the user is typically asked to retry the operation.

- ✓ **Relationships**

Relationships in MongoDB are often modeled using references. Documents can contain references to other documents, creating relationships between them. These relationships can be one-to-one, one-to-many, or many-to-many.

- ✓ **Data model**

Data model defines how data is organized and structured in a database. In MongoDB, the data model is based on documents and collections, providing flexibility and scalability.

- ✓ **Schema**

Schema is a formal description of the structure of data. In MongoDB, the schema is flexible and can evolve over time. While there is no strict schema enforcement, it's often good practice to define a schema to ensure data consistency and maintainability.

✓ **Mongosh**

Mongosh is the official MongoDB shell, a command-line interface for interacting with MongoDB databases. It provides a powerful and intuitive way to execute queries, manage databases, and perform administrative tasks.

- **Identifying user requirements**

Is a crucial step in creating successful software, applications, or products. It ensures that the final solution meets the needs and expectations of the target audience.

✓ **Key methods and considerations:**

- User Research:
  - Interviews: Conduct one-on-one interviews with potential users to gather their needs, pain points, and expectations.
  - Surveys: Distribute surveys to a wider audience to collect quantitative data and identify common patterns.
  - Observations: Observe users in their natural environment to understand their behaviors and workflows.
  - Focus Groups: Facilitate group discussions to uncover shared opinions and insights.
- Personas:
  - Create detailed personas: Develop fictional characters representing your target users.
  - Define goals and motivations: Outline their objectives and the challenges they face.
  - Empathize with their experiences: Understand their perspectives and emotions.
- User Stories:
  - Write user stories: Capture user requirements in a simple, narrative format.
  - Follow the INVEST criteria: Ensure stories are Independent, Negotiable, Valuable, Estimable, Small, and Testable.
  - Prioritize based on value: Determine which stories deliver the most significant benefits.
- Requirements Gathering Techniques:
  - Brainstorming: Facilitate creative sessions to generate ideas and potential solutions.
  - Prototyping: Create low-fidelity prototypes to visualize concepts and gather feedback.
  - Wireframing: Develop basic layouts and structures for interfaces.
  - Mockups: Design more detailed representations of the final product.
- User Testing:

- Conduct usability testing: Observe users interacting with prototypes or early versions of the product.
- Gather feedback: Identify areas for improvement and iterate on the design.
- Iterate and refine: Incorporate user insights to enhance the overall experience.

Additional Considerations:

- Involve stakeholders: Ensure that key stakeholders are involved in the requirements gathering process.
- Consider constraints: Be mindful of technical limitations, budget constraints, and timelines.
- Prioritize requirements: Rank requirements based on their importance and feasibility.
- Document requirements: Create a clear and concise requirements document.
- Remember: Effective user requirements gathering involves a combination of research, empathy, and collaboration.

- **Description of characteristics, features, and datatypes of NoSQL Databases**

✓ **Characteristics of collections.**

Collections in NoSQL databases, particularly document-oriented ones like MongoDB,

Key characteristics:

- Dynamic Schema: Unlike relational databases, collections don't require a fixed schema. Documents within a collection can have different structures, allowing for flexibility and adaptability to changing data requirements.
- Unstructured or Semi-Structured Data: Collections can store unstructured or semi-structured data, such as JSON, XML, or binary data. This makes them suitable for handling complex data formats that don't fit well into traditional relational tables.
- High Performance: Collections are often optimized for high-performance read and write operations. This is especially true for document-oriented databases that use indexing and sharding techniques to distribute data across multiple servers.
- Scalability: Collections can scale horizontally by adding more servers to a cluster. This allows for handling large datasets and increasing throughput without requiring significant changes to the application.
- Flexibility: Collections provide flexibility in terms of data modeling and query capabilities. You can query documents based on their structure and content, allowing for complex and ad-hoc analysis.

✓ **Features of NoSQL Databases**

NoSQL databases offer a range of features that make them well-suited for modern applications:

- Scalability: The ability to handle large datasets and high traffic loads by distributing data across multiple servers.
- Performance: Optimized for fast read and write operations, often using indexing and caching techniques.
- Flexibility: The ability to accommodate changing data structures and requirements without requiring significant schema changes.
- Fault Tolerance: Built-in mechanisms to ensure data consistency and availability even in the event of hardware failures or network outages.
- Distributed Architecture: The ability to run across multiple servers, providing redundancy and scalability.
- Schema-less or Flexible Schema and rich query language: No strict requirement for a predefined schema, allowing for more dynamic data modelling.
- High Availability: The ability to maintain continuous access to data, even in the event of failures or maintenance.

✓ **Types of NoSQL Databases**

A database is a collection of structured data or information which is stored in a computer system and can be accessed easily. A database is usually managed by a Database Management System (DBMS). NoSQL is a non-relational database that is used to store the data in the nontabular form. NoSQL stands for Not only SQL. The main types are documents, key-value, wide-column, and graphs. There are different Types of NoSQL Databases:

Document-based databases

- Key-value stores
- Column-oriented databases
- Graph-based databases

Document-based databases :

The document-based database is a nonrelational database. Instead of storing the data in rows and columns (tables), it uses the documents to store the data in the database. A document database stores data in JSON, BSON, or XML documents.

Documents can be stored and retrieved in a form that is much closer to the data objects used in applications which means less translation is required to use these data in the applications. In the Document database, the particular elements can be accessed by using the index value that is assigned for faster querying.

Collections are the group of documents that store documents that have similar contents. Not all the documents are in any collection as they require a similar schema because document databases have a flexible schema.

Features of Documents Database:

- Flexible schema: Documents in the database has a flexible schema. It means the documents in the database need not be the same schema.
- Faster creation and maintenance: the creation of documents is easy and minimal maintenance is required once we create the document.
- Suitable for unstructured data
- Easy to scale Horizontally
- No foreign keys: There is no dynamic relationship between two documents so documents can be independent of one another. So, there is no requirement for a foreign key in a document database.
- Open formats: To build a document we use XML, JSON, and others.

- **Key-Value Stores:**

A key-value store is a nonrelational database. The simplest form of a NoSQL database is a key-value store. Every data element in the database is stored in key-value pairs. The data can be retrieved by using a unique key allotted to each element in the database. The values can be simple data types like strings and numbers or complex objects. A key-value store is like a relational database with only two columns which is the key and the value.

Features of the Key-Value Store Database:

- Simplicity.
- Scalability.
- Speed

- **Column Oriented Databases**:

A column-oriented database is a non-relational database that stores the data in columns instead of rows. That means when we want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data. Columnar databases are designed to read data more efficiently and retrieve the data with greater speed. A columnar database is used to store a large amount of data. Wide-Column Databases: Store data in wide rows with columns that can vary in number and type. Examples include Cassandra and HBase.

Features of columnar oriented database:

- Scalability.
- Compression.
- Very responsive.

- **Graph Databases**:

Graph-based databases focus on the relationship between the elements. It stores the data in the form of nodes in the database. The connections between the nodes are

called links or relationships. Store data as a graph of interconnected nodes and relationships, making them suitable for modeling complex relationships between data. Examples include Neo4j, ArangoDB, and Amazon Neptune.

Features of graph database:

- It is easy to identify the relationship between the data by using the links.
- The Query's output is real-time results.
- The speed depends upon the number of relationships among the database elements.
- Updating data is also easy, as adding a new node or edge to a graph database is a straightforward task that does not require significant schema changes.

✓ **Data Types**

Common Data Types in NoSQL Databases:

- **Text/String**:
  - Used for storing plain text or strings.
  - Example: Names, addresses, or any text-based data.
  - **Supported By**: MongoDB, Cassandra, Couchbase, DynamoDB, etc.
- **Number**:
  - Handles integers, floating-point numbers, and sometimes complex numbers.
  - Example: Quantities, prices, sensor data.
  - **Supported By**: All NoSQL databases.
- **Boolean**:
  - Used for storing `true` or `false` values.
  - Example: Flags, status indicators, or binary states.
  - **Supported By**: Most NoSQL databases.
- **Array/Lists**:
  - Represents ordered collections of elements, which can be a mix of different types.
  - Example: A list of product IDs in an order.
  - **Supported By**: MongoDB, Couchbase, Cassandra (as sets, lists).
- **Object/Document**:
  - Stores complex data as key-value pairs where values can be other types or even nested documents/objects.
  - Example: JSON documents representing users or products.
  - **Supported By**: MongoDB, Couchbase, CouchDB, DynamoDB.
- **Binary/BLOB**:
  - Stores binary large objects such as files, images, or other media types.
  - Example: Images, videos, PDF files.
  - **Supported By**: MongoDB (Binary), Cassandra (BLOB), Couchbase.
- **Date/Time**:
  - Stores dates and timestamps.

- Example: Transaction times, event logs.
- **Supported By**: MongoDB, Cassandra, DynamoDB.

Geospatial Data:

- Handles geographic data like coordinates (latitude, longitude).
- Example: Location data for mapping applications.
- **Supported By**: MongoDB (Geospatial Indexing), Couchbase, Cassandra (via custom types).

UUID (Universally Unique Identifier):

- A unique identifier for objects or records.
- Example: User IDs, session tokens.
- **Supported By**: MongoDB, Cassandra, Couchbase, DynamoDB

Practical Activity 1.1.2: Defining use cases

Task:

1: Read the key readings 1.1.2

2: By referring to the previous activity 1.1.1, you are requested to define use cases based on this case study

A TVET (Technical and Vocational Education and Training) school is expanding its student base and curriculum. The institution offers a wide range of courses, from mechanical engineering and information technology to culinary arts and electrical installation. Managing student records, course materials, assessments, and operational data has become a challenge due to growing enrolment and the diverse nature of the programs. As database analyst, you are tasked to define or perform use cases database of the aforementioned school by showing the relationship between users.

3: Present your work to the trainer or classmates.

4: Ask for any clarification where necessary.

Key readings 1.1.2: Defining use cases

✓ **Defining use cases**

Use cases are a valuable tool for understanding and documenting the interactions between users and a system. They provide a clear and concise description of how a system will be used and what value it will deliver.

Key components of a use case:

- Actor: The person or entity that interacts with the system.
- Goal: The objective that the actor wants to achieve.
- Precondition: The conditions that must be met before the use case can begin.
- Post condition: The conditions that will be true after the use case is completed.
- Flow of events: A step-by-step description of how the actor interacts with the system.

Steps for defining use cases:

- Identify actors: Determine who will be using the system and what their roles are.
- Identify goals: Determine what the actors want to accomplish using the system.
- Create use cases: Write detailed descriptions of each use case, including the actor, goal, preconditions, post conditions, and flow of events.
- **Write use case description:** Provide a detailed narrative of each use case, including:
  i. Name: A clear and concise name.
  ii. Actors: The involved actors.
  iii. Precondition: Conditions that must be met before the use case begins.
  iv. Post condition: The expected outcomes after the use case is completed.
  v. Normal Flow: The typical sequence of steps.
  vi. Alternative Flows: Possible variations or exceptions.
  vii. Exception Flows: How the system handles errors or unexpected situations.
- Refine and prioritize: Review and refine the use cases to ensure they are clear, concise, and complete. Prioritize them based on their importance and complexity.

Examples of use case:

E-commerce Platforms (Document Store - MongoDB, Couchbase)

- **Use Case**: Managing dynamic product catalogs, customer profiles, shopping carts, and order histories.
- **Reason**: In e-commerce, product data structures can vary greatly (different attributes for electronics, clothing, etc.), and a document-based NoSQL database allows flexibility in schema, handling nested and variable fields efficiently.
- **Example**: Amazon uses DynamoDB (a document-oriented database) to manage shopping carts and ensure high availability and scalability during high-traffic shopping periods.

Social Media Networks (Key-Value Store - Redis, DynamoDB)

- **Use Case**: Storing user session data, likes, comments, and follower relationships in real-time.
- **Reason**: Social media apps require fast retrieval and storage of user-generated content and interactions. Key-value stores allow for rapid read/write operations, making them ideal for caching session data, posts, and timelines.
- **Example**: Twitter uses Redis as a fast, in-memory key-value store to store user timelines and deliver real-time updates.

IoT Data Management (Column-Family Store - Cassandra, HBase)

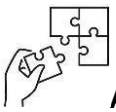
- **Use Case**: Collecting and analyzing massive volumes of sensor data in real-time.
- **Reason**: IoT devices generate vast amounts of time-series data, and column-family stores like Cassandra can efficiently store this type of data, allowing fast writes and optimized querying for analytics.
- **Example**: Companies that manage smart home devices use Cassandra to store sensor data from devices such as thermostats, cameras, and motion detectors.

Benefits of using use cases:

- Improved communication: Use cases provide a common language for stakeholders to discuss system requirements.
- Enhanced requirements gathering: Use cases help to identify and capture all necessary features and functionality.
- Better system design: Use cases can guide the design of the system's user interface and functionality.
- Improved testing: Use cases can be used to develop test cases to ensure that the system meets the specified requirements.

Points to Remember

- **NoSQL:** referred to as “not only SQL” or “non-SQL”, is an approach to database design that enables the storage and querying of data outside the traditional structures found in relational databases.
- **Collections:** Data is organized into documents, which are similar to rows in a traditional relational database. However, instead of being organized into tables, documents are grouped into collections**.**
- **There are differences between SQL and No SQL databases:**
  - ✓ SQL is Suitable for structured data with predefined schema whereas No SQL is Suitable for unstructured and semi-structured data.
  - ✓ SQL Data is stored in tables with columns and rows whereas No SQL Data is stored in collection of documents.
  - ✓ SQL Follows ACID properties (Atomicity, Consistency, Isolation and Durability) for transaction management whereas No SQL Does not necessary follow ACID properties.
- **Features of No SQL database are** flexible, faster creation and maintenance. They are suitable for unstructured data, easy to horizontally scale.
- **Types of No SQL database include** document-based databases, Key-value stores, Column-oriented databases, Graph-based databases.
- **Steps for defining use cases:**
  1) Identify actors
  2) Identify goals
  3) Create use cases
  4) Write use case description
  5) Refine and prioritize



Application of learning 1.1.

FinCorp's IT team is migrating from a relational database to MongoDB to enhance scalability and performance. As a trainee, you are tasked to define use cases in database requirement identification.

Indicative content 1.2: Analysing NoSQL database

Duration: 3 hrs

Theoretical Activity 1.2.1: Description of requirement analysis process

Tasks:

1: Answer the following questions:

i. What is requirement?
ii. What do you understand by requirement analysis?
iii. Identify factors to consider while choosing requirement analysis?
iv. Differentiate requirement analysis from perform analysis?

2: Write your findings on flipchart blackboard or white board

3: Present your findings to the trainer or classmates.

4: Pay attention to the trainer's clarification and ask for clarifications.

5: Read the key readings 1.2.1

Key readings 1.2.1: Description of requirement analysis process

✓ **Definition of Requirement Analysis**

Requirements analysis is a critical phase in software development that involves identifying, documenting, and validating the specific needs of a system or application. The goal is to ensure that the final product meets the expectations of its users and stakeholders.

- **Identify Key Stakeholders and End-Users.**
  - Determine who is involved: Identify all parties with an interest in the project, including users, customers, managers, developers, and other stakeholders.
  - Understand their needs: Understand the goals, expectations, and constraints of each stakeholder.
- **Capture Requirements.**
  - Use various techniques: Employ a combination of methods to collect requirements, such as interviews, surveys, workshops, and document analysis.
  - Consider different perspectives: Gather input from various stakeholders to ensure a comprehensive understanding of the system's needs.

- Use tools: Utilize tools like mind maps, user stories, and use cases to visually represent and organize requirements.

- **Categorize Requirements.**
  - Group similar requirements: Organize requirements into categories based on their function, priority, or other relevant criteria.
  - Prioritize requirements: Rank requirements based on their importance and impact on the project's success.
  - Consider constraints: Take into account factors such as budget, timeline, and technical limitations.
- **Interpret and Record Requirements.**
  - Document requirements: Write a clear and concise requirements document that outlines the system's functionalities, features, and constraints.
  - Use a consistent format: Employ a standardized template or notation (e.g., use cases, user stories) to ensure consistency and clarity.
  - Define terms: Ensure that all terms and acronyms used in the document are clearly defined.
- **Validate Requirements**

- Review with stakeholders: Present the requirements document to stakeholders for review and feedback.
- Address concerns: Incorporate any necessary changes or clarifications based on stakeholder input.
- Conduct feasibility studies: Evaluate the technical and economic feasibility of the requirements.
- Use techniques like:
  Traceability matrix: Map requirements to design elements and test cases.
  Walkthroughs: Have stakeholders review the requirements document to identify any inconsistencies or omissions.

- **Additional Considerations:**

- Iterative process: Requirements analysis is often an iterative process, so be prepared to refine and update requirements as the project progresses.
- Involve stakeholders: Ensure that stakeholders are actively involved throughout the requirements analysis process.
- Consider user experience: Pay attention to the user experience and design requirements that enhance usability and satisfaction.

Practical Activity 1.2.1: Performing data analysis

Task:

1: Read the key readings 1.2.1

2: By referring to the previous activity 1.1.1, you are requested to perform data analysis

3: Respect instruction from your trainer

4: Perform data analysis

5: Present your work to the trainer or classmates

6: Ask for any clarification where necessary

Key readings 1.2.1: Performing data analysis

- ✓ **Perform Data analysis**

Data analysis takes the data you already have and lets you easily see how decisions impact your business. It gives you a rod to steer the ship in the right direction by using facts you have gathered and presenting them in a meaningful way.

- Data cleaning and preparation: Identifying and correcting errors, inconsistencies, or missing values.
- Exploratory data analysis (EDA): Summarizing and understanding the key characteristics of the data through visualizations and statistical measures.
- Statistical analysis: Applying statistical techniques to test hypotheses, identify patterns, and make inferences.
- Predictive modeling: Building models to predict future outcomes based on historical data.
- Data visualization: Creating informative and visually appealing charts and graphs to communicate insights.

- **Steps for performing data analysis**

step1: Define the Problem and Research Question

It can be easy to jump straight into collecting the data you need and creating dashboards to try and answer every question thinking this will be helpful for everyone. But if it's not driving business value, the work done is useless.

Step 2: Collecting Data

Once you understand the question you need to provide insights for, it's time to gather the necessary data. There are different types of data that you can collect.

- First-Party Data: First-party data is collected from your customers and audiences on channels like your website or app.
- Second-Party Data: Second-party data refers to information shared by one organization with another, often as part of a partnership.
- Third-Party Data: Third-party data is owned and sold by a separate organization or individual and is often compiled from numerous sources.

Step 3: Preparing Customer Data

Now you have the data you need, it's time to transform it into a useful state.

You may be fortunate and have the data going into the data warehouse being modelled.

Some of the processes involved in cleaning the data could be:

- Removing any errors or duplicate values
- Filling in any missing data

- Structuring the data so it can be manipulated more easier
- Removing any data that isn't relevant to the analysis you are performing

Step 4: Analysing the Data

Now that you have data to work with that is relevant to the task you have at hand and are confident that it's accurate, it's time to analyze the data.

There are a few different analysis techniques.

- **Descriptive Analysis:** Descriptive analytics involves looking at past events and patterns. It is often the first step in data analysis before delving deeper into a subject.
- **Diagnostic Analysis:** Diagnostic analytics is a type of analytics aiming to understand a problem's root cause.
- **Predictive Analysis:** Predictive analytics is a type of data analysis that uses historical data to forecast future trends and growth.
- **Prescriptive Analysis:** Prescriptive analytics is a type of data analysis that enables users to make recommendations for future actions.

Step 5: Interpret the Results

Up to this point, we know what answers we are looking for in the data by clarifying research questions.

- ✓ **Implement data validation**

Data validation is a crucial step in data analysis to ensure data quality and accuracy. It involves checking data for errors, inconsistencies, and completeness.

Step 1. Range Checking:

- Verify data within limits: Ensure that numerical values fall within predefined ranges (e.g., age cannot be negative).

Step 2. Format Checking:

- Validate data format: Check that data adheres to specific formats (e.g., email addresses, dates, phone numbers).

Step 3. Consistency Checking:

- Verify data relationships: Ensure that related data values are consistent (e.g., a person's birth date should be earlier than their current age).

Step 4. Completeness Checking:

- Check for missing values: Verify that all required fields are filled in.

Step 5. Uniqueness Checking:

- Ensure unique values: Verify that certain fields have unique values (e.g., customer IDs or social security numbers).

Step 6. Cross-Validation:

- Compare data sources: Compare data from multiple sources to identify inconsistencies or errors.

Step 7. Business Rule Validation:

- Enforce specific rules: Check that data adheres to predefined business

rules (e.g., product prices must be positive).

Step 8. Regular Expression Validation:

- Use patterns: Employ regular expressions to validate data based on specific patterns (e.g., validating email addresses).

Step 9. Data Type Validation:

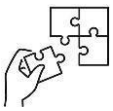
- Ensure correct types: Verify that data values are of the correct data type (e.g., a field for age should be numeric).

Step 10. Data Quality Checks:

- Assess data quality: Use data quality metrics to evaluate the accuracy, completeness, consistency, and timeliness of data.

Points to Remember

- Data validation factors are range checking, format checking, consistency checking, completeness checking and uniqueness checking.
- Consideration of requirement analysis are identify key stakeholders and end-user, capture requirements, categorize requirements, interpret and record requirements.
- Data analysis factors are data cleaning and preparation ,Exploratory data analysis, Statistical analysis, predictive modelling, Data visualization
- **Steps to perform data analysis are**:
  - ✓ Define the problem
  - ✓ Collect data
  - ✓ Prepare data
  - ✓ Analyse data
  - ✓ Interpret results
- **Steps to implement data validation are:**
  - ✓ Range checking
  - ✓ Format checking
  - ✓ Consistency checking
  - ✓ Completeness checking
  - ✓ Uniqueness checking
  - ✓ Cross-validation
  - ✓ Business rule validation
  - ✓ regular expression validation
  - ✓ Data type validation
  - ✓ Data quality checks.



Application of learning 1.2.

MXZY Ltd is a company that generates revenue from selling furniture products. The company uses file system (books) to record information about the sales and inventory. The company has a problem of non-efficient security, accessibility and management of information about the stock-in, stock-out and customers. Therefore, the company wants to switch to a digital system that can be accessed easily by authorized employees. As database analyst, you are requested to analyse the above case study and find out the elements that will be stored in database of the company.

Indicative content 1.3: Preparing database environment

Duration: 4 hrs

Theoretical Activity 1.3.1: Description of database environment

Tasks:

1: Answer the following questions:

i. What is database?
ii. What do you understand by environment?
iii. Outlines factors to consider while preparing database environment.
iv. Describe how to set up database environment within the workplace?
v. Differentiate atlas from compass environment?
vi. Differentiate shell from atlas environment?

2: Write your findings on papers, flipcharts

3: Present your findings to the trainer or classmates

4: Pay attention to the trainer's clarification and ask questions where necessary.

5: Read the key readings 1.3.1

Key readings 1.3.1.: Description of database environment

✓ **Identify scalability of mongoDB**

MongoDB's scalability is a key feature that makes it suitable for handling large datasets and high traffic loads. Here are the primary factors contributing to its scalability:

Horizontal Scalability:

- **Sharding:** MongoDB allows you to distribute data across multiple servers (shards) for horizontal scalability. This enables the database to handle increasing amounts of data without compromising performance.
- **Automatic Sharding:** MongoDB can automatically shard data based on predefined sharding keys, making it easier to scale the database as needed.
- **Balanced Sharding:** MongoDB strives to distribute data evenly across shards to ensure optimal performance and load balancing.

Vertical Scalability:

- **Memory Optimization:** MongoDB can efficiently use available memory to improve query performance and reduce I/O operations.
- **Indexing:** Creating appropriate indexes can significantly speed up query execution, especially for frequently accessed data.

- **Query Optimization:** MongoDB's query optimizer analyses queries and chooses the most efficient execution plans to optimize performance.

Other Factors:

- **Replication:** MongoDB supports replication to provide data redundancy and fault tolerance. This helps ensure that data is available even if a server fails.
- **High Availability**: MongoDB offers high availability features like replica sets and sharded clusters to minimize downtime and ensure data consistency.
- **Cloud Integration:** MongoDB is available on various cloud platforms, allowing you to leverage cloud infrastructure for scalability and management.
- **Sharding Key Selection**: Choose sharding keys carefully to ensure balanced data distribution and optimal query performance.
- **Indexing Strategy:** Create appropriate indexes to support common query patterns and improve query performance.
- **Data Modelling:** Design your data model to consider scalability requirements and avoid performance bottlenecks.
- **Monitoring and Tuning**: Continuously monitor your MongoDB cluster and tune performance settings as needed.

✓ **Setting up MongoDB environment**

✓ **Shell environment**

Download and Install MongoDB:

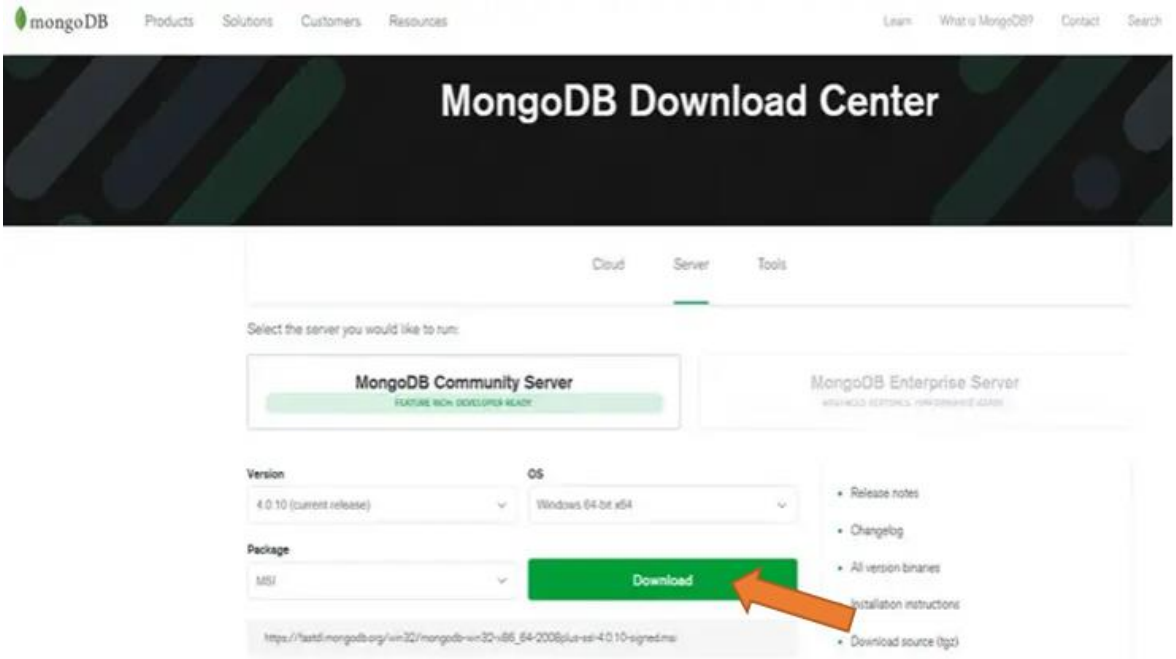
- Visit the official MongoDB website (https://www.mongodb.com/) and download the appropriate version for your operating system.
- Follow the installation instructions provided.

To get started with MongoDB, you have to install it in your system. You need to find and download the latest version of MongoDB, which will be compatible with your computer system. You can use this (http://www.mongodb.org/downloads) link and follow the instruction to install MongoDB in your PC. In this chapter, you will learn how to setup a complete environment to start working with MongoDB.

The process of setting up MongoDB in different operating systems is also different,
The website of MongoDB provides all the installation instructions, and MongoDB is supported by Windows, Linux as well as Mac OS.

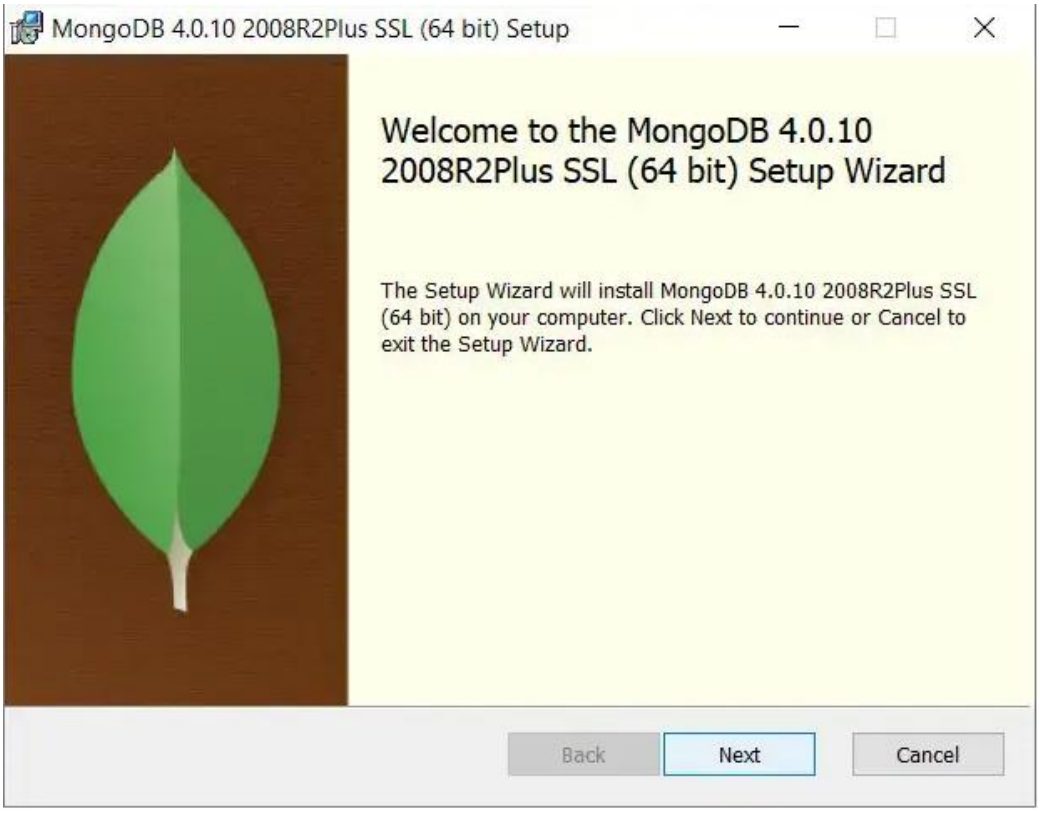
It is to be noted that, MongoDB will not run in Windows XP; so you need to install higher versions of windows to use this database.

Once you visit the link (http://www.mongodb.org/downloads), Install mongoDB in windows

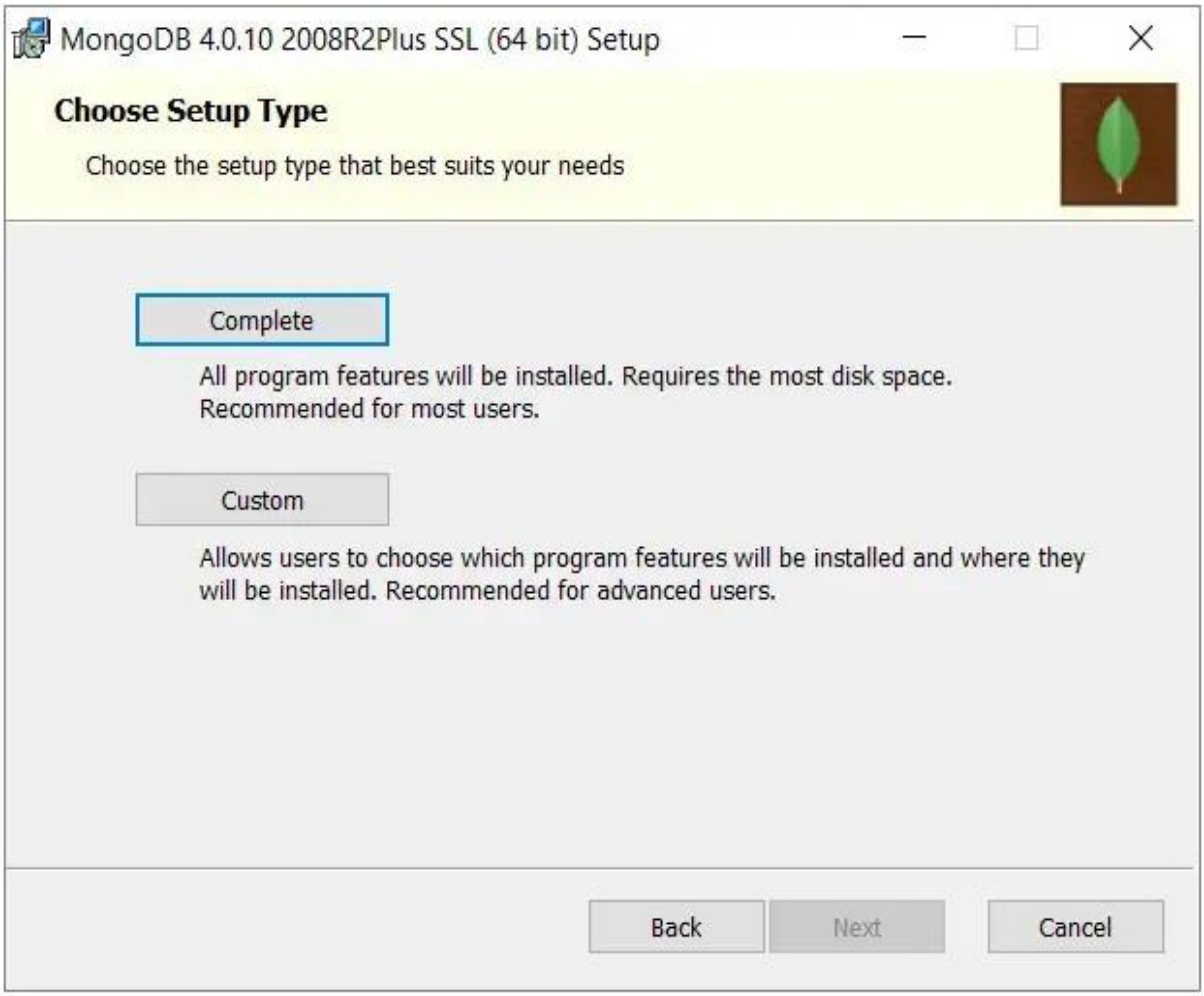


Once the download is complete, double click this setup file to install it. Follow the steps:

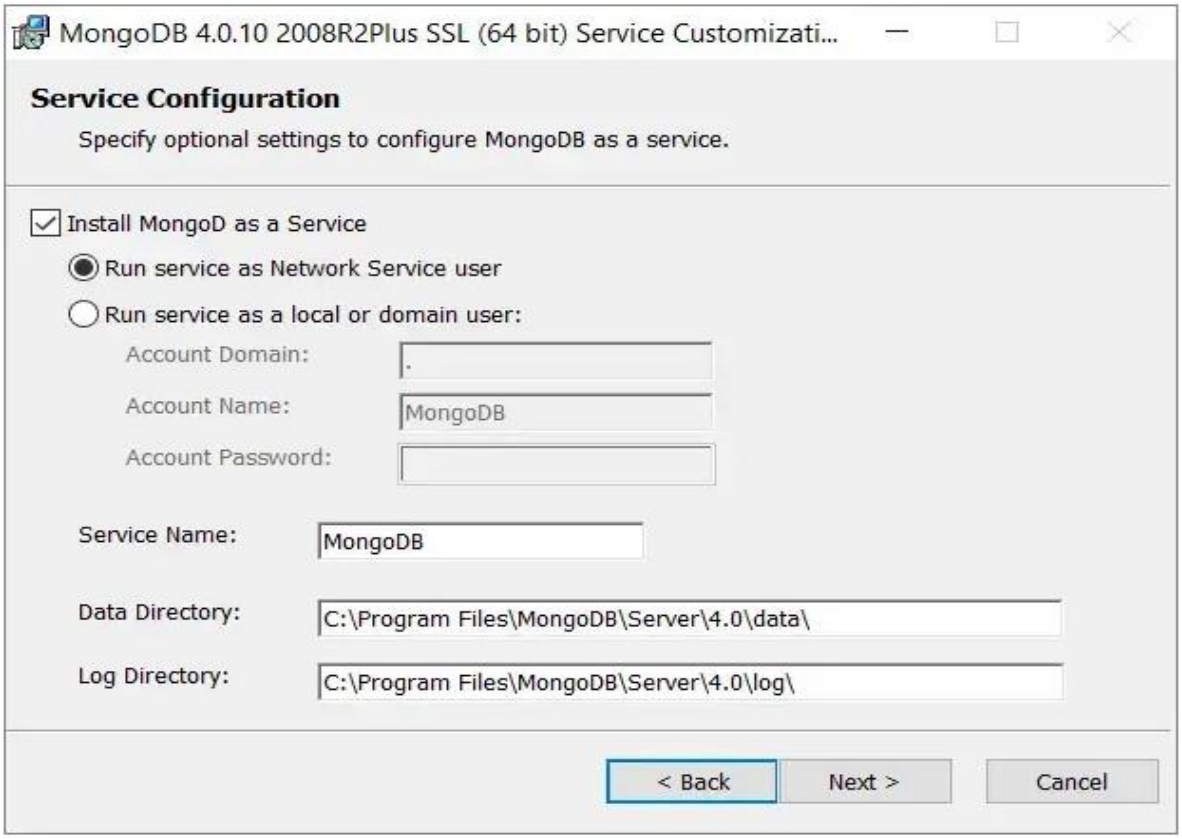
1.Click Next.



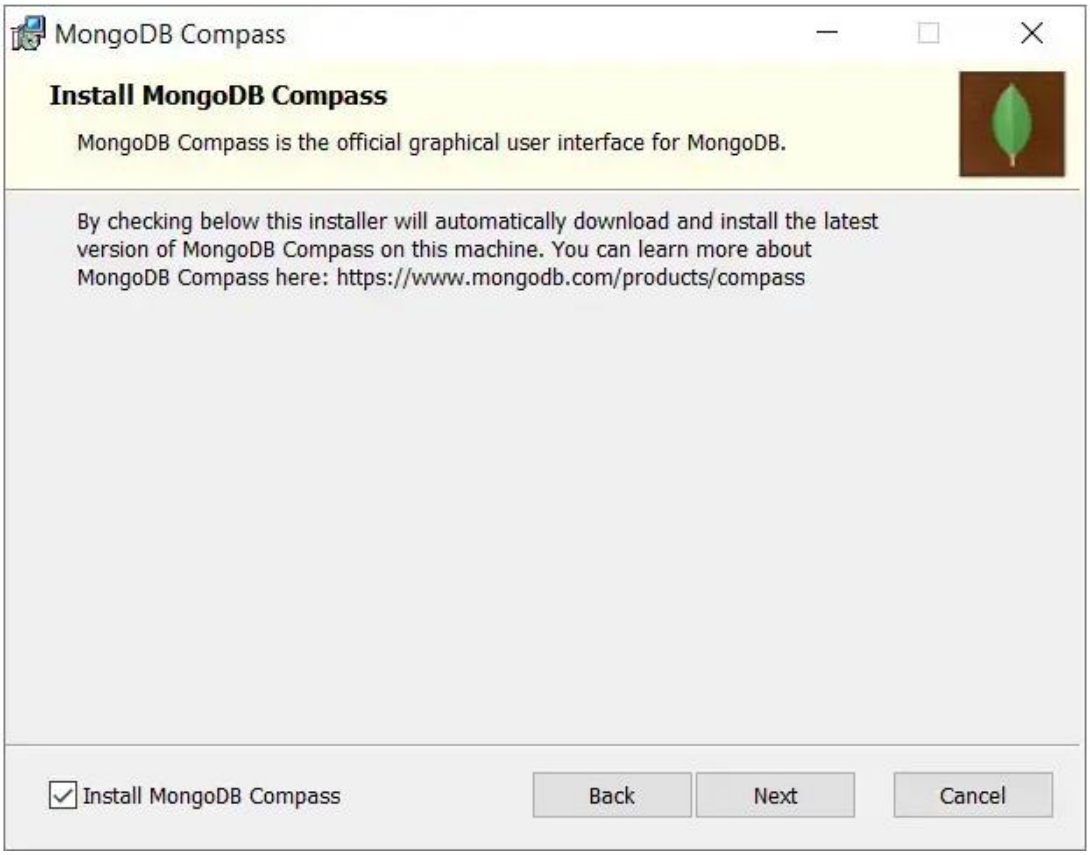
2. Now, choose Complete to install MongoDB completely.



3. Then, select the radio button "Run services as Network service user."



4.The setup system will also prompt you to install MongoDB Compass, which is MongoDB official graphical user interface (GUI). You can tick the checkbox to install that as well.

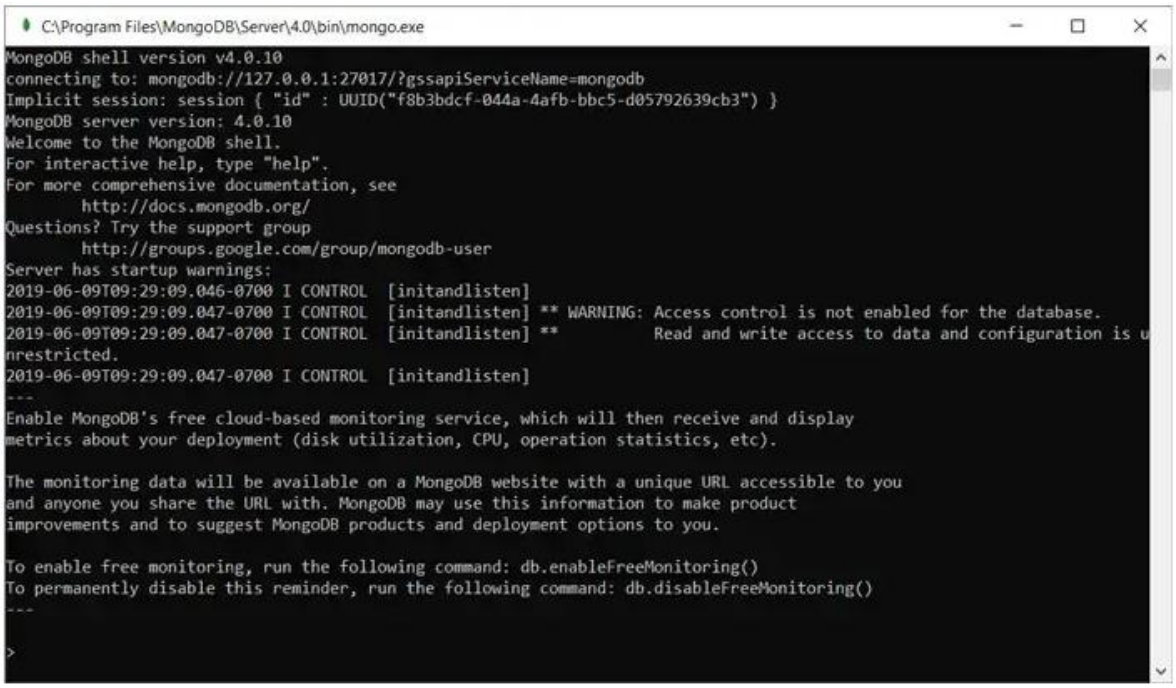


Once the installation is done completely, you need to start MongoDB and to do so follow the process:

1. Open Command Prompt.
2. Type: C:\Program Files\MongoDB\Server\4.0\bin
3. Now type the command simply: mongod to run the server.

In this way, you can start your MongoDB database. Now, for running MongoDB primary client system, you have to use the command:

C:\Program Files\MongoDB\Server\4.0\bin>mongo.exe



- ❖ **Set Environment Variables:**

Add the MongoDB bin directory to your system's PATH environment variable. This allows you to run MongoDB commands from any directory.
On Windows, you can modify the system environment variables. On macOS and Linux, you can edit your shell's configuration file (e.g., .bashrc, .zshrc).

Start the MongoDB Server:

- Open a terminal or command prompt and run the following command: mongod

Connect to the MongoDB Shell:

- Open a new terminal or command prompt and run the following command: mongo
  - ✓ Compass environment

Compass Environmental is a prominent environmental consulting firm that offers a wide range of services to help businesses and organizations address their environmental challenges. With a global presence and a team of experienced professionals, they provide expert guidance and solutions for a variety of environmental issues.

Key Services Offered by Compass Environmental:

- **Environmental Impact Assessments:** Evaluating the potential effects of projects on the environment.
- **Environmental Audits:** Assessing compliance with environmental regulations and identifying areas for improvement.

- **Waste Management:** Developing sustainable waste management strategies and solutions.

- **Water Quality Management:** Ensuring compliance with water quality standards and protecting water resources.
- **Air Quality Management:** Monitoring and managing air quality to minimize pollution.
- **Soil and Groundwater Remediation:** Cleaning up contaminated sites to protect human health and the environment.
- **Environmental Permitting:** Assisting clients in obtaining necessary permits and licenses for their projects.
- **Sustainability Consulting**: Developing and implementing sustainable business practices.

Why Choose Compass Environmental?

- **Expertise:** Their team of experienced professionals has a deep understanding of environmental regulations and best practices.
- **Global Reach:** They operate in multiple countries, providing tailored solutions to clients worldwide.
- **Customized Approach:** They work closely with clients to develop solutions that meet their specific needs and goals.
- **Commitment to Sustainability:** Compass Environmental is dedicated to promoting sustainable practices and protecting the environment.

✓ Atlas environment

Atlas is a renowned environmental consulting firm that offers a comprehensive range of services to help businesses and organizations address their environmental challenges. With a global footprint and a team of highly skilled professionals, they provide innovative solutions for a variety of environmental issues.

Key Services Offered by Atlas:

- **Environmental Impact Assessments:** Evaluating the potential effects of projects on the environment.
- **Environmental Audits:** Assessing compliance with environmental regulations and identifying areas for improvement.
- **Waste Management:** Developing sustainable waste management strategies and solutions.

- **Water Quality Management**: Ensuring compliance with water quality standards and protecting water resources.
- **Air Quality Management:** Monitoring and managing air quality to minimize pollution.
- Soil and Groundwater Remediation: Cleaning up contaminated sites to protect human health and the environment.
- **Environmental Permitting**: Assisting clients in obtaining necessary permits and licenses for their projects.
- **Sustainability Consulting:** Developing and implementing sustainable business practices.

Why Choose Atlas?

- **Expertise:** Their team of experienced professionals has a deep understanding of environmental regulations and best practices.
- **Global Reach:** They operate in multiple countries, providing tailored solutions to clients worldwide.
- **Innovative Solutions:** Atlas is committed to developing innovative and effective environmental solutions.
- **Commitment to Sustainability:** They are dedicated to promoting sustainable practices and protecting the environment.

Practical Activity 1.3.2: Setting database environment

Task:

1: Go to the computer lab and by referring to the previous theoretical activity 1.3.1, set database environment

2: Present the steps to set database environment

3: Referring to the steps provided in task 2, set the database environment

4: Present your work to the trainer or classmates.

5: Ask questions for clarification where necessary

6: Read the key readings 1.3.2

Key readings 1.3.2: Setting database environment

Setting up a NoSQL database environment involves several key steps. The exact process may vary depending on the specific NoSQL database you're using (e.g., MongoDB, Cassandra, Redis),

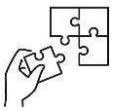
- ✓ **Choose a NoSQL Database**
- Evaluate your needs: Consider factors like scalability, performance, data consistency, and query patterns.
- Research popular options: Explore databases like MongoDB, Cassandra, Redis, Couchbase, and Amazon DynamoDB.
- ✓ **Install and Configure the Database**
- Download and install: Follow the installation instructions provided by the database vendor.
- Configure settings: Adjust parameters like storage location, memory usage, and network settings to optimize performance.
- ✓ **Create a Database and Collections**
- Database: A logical container for your data.
- Collections: Similar to tables in relational databases, they hold groups of related documents.
- ✓ **Connect to the Database**
- Use a driver: Choose a suitable driver for your programming language to interact with the database.
- Establish a connection: Connect to the database using the appropriate connection string.

Points to Remember

- Difference between atlas and compass environment is that

  Atlas is a renowned environmental while Compass Environmental is a prominent environment consulting firm that offers a comprehensive range of services to help businesses and organizations address their environmental challenges.

- Steps for setting database environment are as follows:
  - ✓ Choose a NoSQL database
  - ✓ Install and configure the database
  - ✓ Create a database and collections
  - ✓ Connect to the database



Application of learning 1.3.

Track Fini is a startup company which specialises in real-time financial tracking. The company is migrating from SQL to NoSQL for its scalability and efficient handling of large datasets.and You are tasked to help the company to create mongoDB database environment.

Learning outcome 1 end assessment

Theoretical assessment

Q1. Read the statements carefully, then circle the letter corresponding with the correct answer:

i. What is a key feature of NoSQL databases?
   a) Strictly structured schema
   b) Support for SQL queries
   c) Horizontal scaling
   d) Lack of flexibility for unstructured data

ii. In MongoDB, what is a collection?
   a) A group of documents
   b) A group of databases
   c) A database index
   d) A table

iii. Which term refers to the process of storing and retrieving data without fixed schemas?
   a) Relational Database
   b) Schema Validation
   c) NoSQL
   d) Data normalization

iv. What is MongoDB's equivalent of a table in relational databases?
   a) Row
   b) Column
   c) Document
   d) Collection

v. Which of the following is a benefit of using MongoDB's indexing feature?
   a) Slower query performance
   b) Faster data retrieval
   c) Improved data redundancy
   d) Larger storage space requirements

vi. Which type of NoSQL database does MongoDB fall under?
   a) Column-oriented
   b) Key-Value
   c) Graph
   d) Document

vii. What does optimistic locking help prevent in MongoDB?
   a) Document duplication
   b) Schema validation errors
   c) Simultaneous updates from overwriting data
   d) Indexing failures

viii. Which tool is used to interact with MongoDB in a shell environment?
   a) Compass
   b) Mongosh
   c) Atlas
   d) MySQL Workbench

ix. In requirement analysis for a NoSQL database, what is the first step?
   a) Perform Data Validation
   b) Identify Key Stakeholders and End-Users
   c) Interpret and Record Requirements
   d) Capture Data Models

x. Which of the following describes the scalability of MongoDB?
   a) Vertical scaling only
   b) Only suited for small datasets
   c) Supports horizontal scaling across distributed systems
   d) Limited to one server instance

Q2. Read the statement carefully then answer By TRUE for the correct statement Or FALSE for the wrong statement.

i. NoSQL databases are always schema-less and cannot enforce any structure on the data.
ii. MongoDB stores data in the form of tables similar to relational databases.
iii. In MongoDB, documents within the same collection can have different structures.
iv. Indexing in MongoDB can improve the performance of read operations by making data retrieval faster.
v. Optimistic locking prevents other users from accessing a document while it is being edited.
vi. In MongoDB, relationships between documents can be modeled by embedding documents or referencing them.
vii. Collections in MongoDB always require a predefined structure for documents.
viii. MongoDB Atlas is a cloud-based solution that automates the deployment and scaling of MongoDB databases.
ix. In the requirements analysis process for a NoSQL database, identifying key stakeholders and end-users is the first step.
x. MongoDB's Compass environment is a command-line tool used for database management.

Q3. Read carefully and then provide answers to the quetions below

i. What is NoSQL?
ii. What is MongoDB and how does it work?
iii. What is meant by Availability in a NoSQL context?
iv. What is a Document in MongoDB?
v. What is a Collection in MongoDB?
vi. How does Indexing work in MongoDB?
vii. How are Relationships managed in NoSQL databases like MongoDB?
viii. What is a Data Model in NoSQL?
ix. . What is a Schema in NoSQL databases?
x. How are user requirements identified for a database?
xi. What are the characteristics of collections in MongoDB?
xii. What are key features of NoSQL databases?
xiii. What are the types of NoSQL databases?
xiv. xv. What types of data type can NoSQL databases store?
xv. What are the steps in the requirements analysis process for a NoSQL database?
xvi. xvii. How is data validation implemented in NoSQL databases like MongoDB?
xvii. How does MongoDB ensure scalability?
xviii. xix. What are the different environments for working with MongoDB?

Practical assessment

Suppose that there is a newly opened Cybercafé located in your area and it needs a database Developer to set up their database infrastructure. You are asked to help the cybercafé to set up MongoDB using the mongosh shell and create a database called ecommerce db.

References :

(2023, April 5). MongoDB. TechTarget. https://www.techtarget.com/searchdatamanagement/definition/MongoDB

(Mongo DB in Action, 2016)

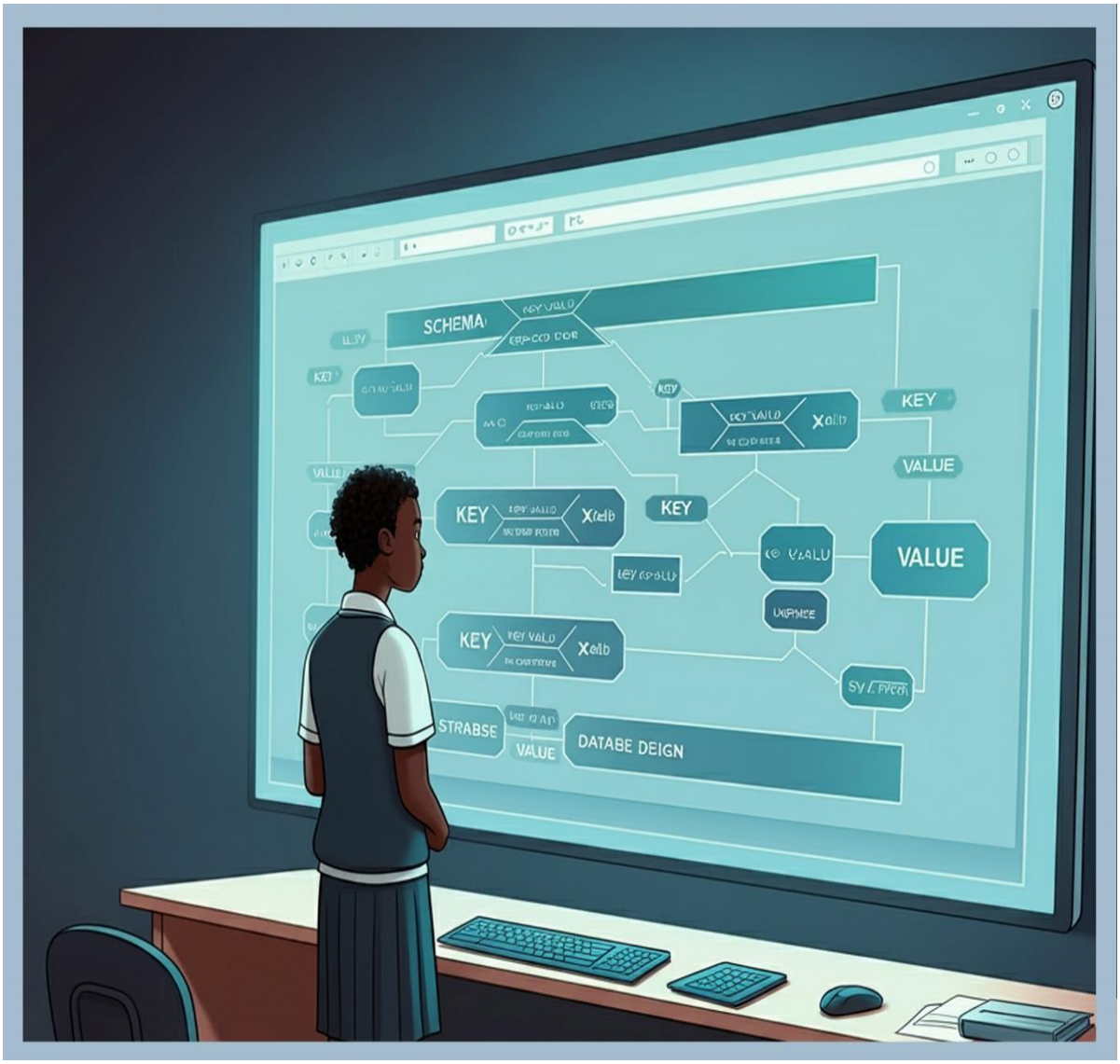
(Nosql for Mere Mortals, April 2015)

(Shakuntala Gupta Edward, 2015)

Amit V. Solutions. (2023, August 22). MongoDB 101: A practical guide to NoSQL databases. Medium. https://medium.com/@amitvsolutions/mongodb-101-a-practical-guide-to-nosql-databases-4bc47c36cb9e

GeeksforGeeks. (2023, October 14). Introduction to NoSQL. GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-nosql/SearchDataManagement.

Hightouch. (2023, June 15). Steps for data analysis. Hightouch. https://hightouch.com/blog/steps-for-data-analysis


SCHEMA
KEY
VALUE
KEY
KEY
VALUE
KEY
STRABSE
VALUE
DATABE DEIGN

Indicative contents

2.1 Selecting tools of drawing databases

2.2 Creating conceptual data model.

2.3 Designing MongoDB database schema

Key Competencies for Learning Outcome 2: Design No SQL Database

Knowledge	Skills	Attitudes
• Identification of No SQL drawing tools • Identification of Collections • Definition of Collection Structure	• Installing No SQL Drawing tools. • Designing MongoDB Database Schema • Validating and Normalizing Schema • Applying Design Patterns	• Having attention to Detail while creating use cases. • Having Problem-Solving capability for creating use case and performing data analysis. • Having adaptability based on requirement analysis process. • Being collaborative while handling requirement analysis process

Duration: 20 hrs

Learning outcome 2 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Identify correctly drawing tools based on database requirements.
2. Draw properly selected tools based on database requirements.
3. Install correctly drawing tools based on database requirement.
4. Create properly conceptual Data Modelling based on the structure of the data and its relationships.
5. Design clearly database schema according to Mongoose.
6. Validate clearly drawing tools based on the structure of the data and its relationships.
7. Apply properly design patterns according to mongoose.

Resources

Equipment	Tools	Materials
● Computer	● Edraw Max ● Mongosh ● MongoDB compass	● Internet ● Electricity

Indicative content 2.1: Selecting tools of drawing databases.

Duration: 3 hrs

Theoretical Activity 2.1.1: Identification of NoSQL drawing tools

Tasks:

1: Answer the following questions:

i. What do you understand by drawing tools?

ii. Identify NoSQL drawing tools.

2: Write your findings on papers or flipcharts

3: Present your findings to the trainer or classmates

4: Pay attention to the trainer's clarification and ask for clarifications where necessary.

5: Read the key readings 2.1.1

Key readings 2.1.1.: Identification of No SQL Drawing tools

✓ **Definition of NoSQL drawing tools**

NoSQL tools encompass a range of software designed to interact with NoSQL databases, each catering to different aspects of NoSQL database management, development, and operations. When working with NoSQL databases, visualizing data models, schemas, and interactions can be crucial for design, documentation, and communication

NoSQL databases come with various tools that cover different needs such as data management, monitoring, and optimization.

NoSQL tools and their descriptions:

❖ **MongoDB**

Description: MongoDB is a widely used document-oriented NoSQL database. It stores data in flexible, JSON-like documents, allowing for schema-less designs.

Key Features:

- Schema Flexibility: Documents can have different structures within the same collection.
- Rich Query Language: Supports a wide range of queries including filtering, sorting, and aggregations.

- Indexing: Offers various indexing options to optimize query performance.
- Replication and Sharding: Provides high availability through replica sets and horizontal scalability via sharding.

Tool:

- MongoDB Compass: A GUI tool for MongoDB that allows users to visualize and explore data, build queries, and manage indexes.

❖ **Redis**

Description: Redis is an in-memory key-value store known for its speed and efficiency. It is often used for caching, session storage, and real-time analytics.

Key Features:

- In-Memory Storage: Provides extremely fast data access.
- Data Structures: Supports various data types including strings, hashes, lists, sets, and sorted sets.
- Persistence Options: Offers mechanisms for data durability such as snapshots and append-only files.
- Pub/Sub: Supports publish/subscribe messaging patterns for real-time communication.

Tool:

- Redis Desktop Manager (RDM): A GUI tool for managing Redis databases, offering features like data visualization and management.

❖ **Apache Cassandra**

Description: Apache Cassandra is a highly scalable column-family store designed for handling large amounts of data across many servers with no single point of failure.

Key Features:

- Horizontal Scalability: Designed to scale out by adding more nodes to the cluster.
- High Availability: Provides continuous availability with no downtime.
- Tunable Consistency: Allows the configuration of consistency levels based on use case requirements.
- Distributed Architecture: Supports data replication across multiple nodes and data centers.

Tool:

- DataStax Studio: An interactive development environment for Apache Cassandra that allows users to visualize and interact with data.

❖ **Neo4j**

Description: Neo4j is a graph database designed to represent and query data with complex relationships. It is often used for applications involving networked data, such as social networks or recommendation engines.

Key Features:

- Graph Data Model: Stores data as nodes and relationships, making it ideal for highly interconnected data.

- Cypher Query Language: Provides a powerful query language specifically designed for graph traversal and pattern matching.
- ACID Compliance: Ensures data integrity through transactional support.

Tool:

- Neo4j Browser: An interactive web-based tool for querying and visualizing graph data in Neo4j.

❖ **Couchbase**

Description: Couchbase is a distributed document-oriented NoSQL database that combines key-value store capabilities with document database features.

Key Features:

- Multi-Model: Supports both key-value and document models.
- In-Memory Performance: Utilizes in-memory caching for fast data access.
- Global Distribution: Offers cross datacenter replication and automatic failover for high availability.
- N1QL Query Language: Provides SQL-like queries for JSON documents.

Tool:

- Couchbase Sync Gateway: A tool for synchronizing data between Couchbase Server and Couchbase Lite (mobile), allowing for offline-first applications.

❖ **Amazon DynamoDB**

Description: Amazon DynamoDB is a managed key-value and document database service provided by AWS. It is designed for high availability and seamless scaling.

Key Features:

- Managed Service: AWS handles maintenance tasks like backups, updates, and scaling.
- Automatic Scaling: Automatically adjusts throughput capacity based on workload demands.
- High Performance: Provides low-latency access to data and high throughput.
- Integration with AWS Ecosystem: Easily integrates with other AWS services such as Lambda, S3, and CloudWatch.

Tool:

- AWS DynamoDB Console: A web-based interface for managing DynamoDB tables, performing queries, and monitoring performance.

❖ **Apache HBase**

Description: Apache HBase is a column-family store built on top of the Hadoop Distributed File System (HDFS). It is used for real-time read/write access to large datasets.

Key Features:

- Scalability: Can handle large amounts of data across many servers.
- Strong Consistency: Ensures that read operations reflect the most recent writes.
- Integration with Hadoop: Integrates well with Hadoop for big data processing.

Tool:

- HBase Shell: A command-line tool for managing and querying HBase tables.

❖ **RavenDB**

Description: RavenDB is a document-oriented database with a focus on simplicity and developer productivity. It provides features for easy data management and query capabilities.

Key Features:

- Embedded or Standalone: Can be used as an embedded database or as a standalone server.
- Indexing and Querying: Supports full-text search and complex queries with automatic indexing.
- Transaction Support: Provides ACID transactions for reliable data operations.

Tool:

- RavenDB Studio: A web-based management tool that provides a user-friendly interface for database operations, querying, and monitoring.

Practical Activity 2.1.2: Installing of Edraw Max drawing

Task:

1: Read the key readings 2.1.2

2: You are tasked to install Edraw max drawing

3: Present the steps to install Edraw Max drawing

4: Referring to the steps provided in task 2, install Edraw max drawing

5: Present your work to the trainer or classmates

6: Ask questions for clarification where necessary

Key readings 2.1.2: installing Edraw Max drawing

✓ **Definition of Edraw max drawing**

Edraw Max is a versatile diagramming and drawing tool used for creating flowcharts, mind maps, organizational charts, floor plans, and more.

- step-by-step guide to installing Edraw Max:
- ❖ Download Edraw Max

- Visit the Official Website:

❖ Go to the Edraw Max official website.

- Select the Download Option:

❖ Navigate to the download section and choose the version suitable for your operating system (Windows, macOS, or Linux).

- Download the Installer:

❖ Click on the download link for the installer. This will download a setup file to your computer.

- Install Edraw Max on Windows
  - ❖ Run the Installer:
    - Locate the downloaded .exe file (e.g., EdrawMax_Setup.exe) and double-click it to start the installation process.
  - ❖ Start Installation:
    - The installation wizard will open. Click "Next" to proceed.
  - ❖ Read and Accept the License Agreement:
    - Review the End User License Agreement (EULA). If you agree, select "I Agree" to continue.
  - ❖ Choose Installation Folder:
    - Select the destination folder where you want to install Edraw Max or accept the default location. Click "Next".
  - ❖ Select Additional Tasks:
    - Choose any additional tasks or shortcuts you want to create. Click "Next".
  - ❖ Install:
    - Click "Install" to begin the installation. The process will take a few minutes.
  - ❖ Finish Installation:
    - Once the installation is complete, click "Finish" to exit the installer. You can now launch Edraw Max from your desktop or Start menu.
- **Install Edraw Max on macOS**
  - ❖ Run the Installer:
    - Locate the downloaded .dmg file (e.g., EdrawMax.dmg) and double-click it to open.
  - ❖ Drag and Drop to Applications Folder:
    - A window will appear with the Edraw Max application icon and a shortcut to the Applications folder. Drag the Edraw Max icon into the Applications folder.
  - ❖ Open Edraw Max:
    - Go to the Applications folder and double-click on Edraw Max to launch it. The first time you open it, you may need to confirm that you want to open an application downloaded from the internet.

Install Edraw Max on Linux

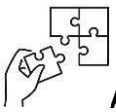
Edraw Max does not have a native Linux version, but you can run it using compatibility layers or virtual machines. Here's a common method using Wine:

- ❖ Install Wine:
  - First, install Wine on your Linux system. You can usually do this through your package manager. For example:
  - sudo apt update
  - sudo apt install wine
- ❖ Run the Installer with Wine:
  - Download the Windows installer for Edraw Max. Then, use Wine to run the installer**: wine EdrawMax_Setup.exe**
- ❖ Follow the Windows Installation Instructions:
- ❖ Post-Installation

- Activation: Launch Edraw Max. If you have a license key, enter it when prompted to activate the full version.
- Updates: Check for any updates or patches from within the application or on the official website.

Points to Remember

- **No SQL drawing tools are:**
  - ✓ MongoBD
  - ✓ Redis
  - ✓ Apache Cassandra
  - ✓ Neo4j
  - ✓ Couchbase
  - ✓ Amazon DynamoBD
  - ✓ Apache Hbase
  - ✓ RavenDB
- **Steps for installing Edraw max are as follows:**
  - ✓ Download Edraw max through official website
  - ✓ Install Edraw Max on different operating systems.



Application of learning 2.1.

Suppose that KKK Company needs to install software applications in its computers. Now, as software developer you are asked to help the company install Edraw Max (on Windows OS).

Indicative content 2.2: Creating conceptual data model.

Duration: 10 hrs

Theoretical Activity 2.2.1: Description of creation of conceptual data model

Tasks:

1:Answer the following questions:

1. Define the following terms:
    i. What is collection?
    ii. Entity relationships
    iii. Unified modelling language
    iv. Data flow diagram
2. What is the difference between sharding and replication?
3. What is the difference between levels of Data Flow Diagram?
4. Identify the types of relationships

2: Write your findings on papers or flipcharts

3: Present your findings to the trainer or classmates

4: Pay attention to the trainer's clarification and ask clarifications where necessary.

5: Read the key readings 2.2.1

Key readings 2.2.1.: Description of creation of conceptual data model

✓ **Identifying Collections**

Collections in NoSQL databases (like MongoDB) are equivalent to tables in relational databases. They hold documents or records that share some common characteristics, but unlike relational tables, collections don't enforce a fixed schema.

Example:

If you're designing a database for an e-commerce application, some key collections might include:

- users (for storing customer information)
- products (for listing items for sale)
- orders (for tracking customer purchases)

Each collection contains documents representing individual entities (like individual users, products, or orders)

✓ **Modeling Entity Relationships**

NoSQL databases focus on performance, scalability, and flexibility, requiring a different approach for modeling relationships based on the database type (document, key-value, column-family, graph). There are common techniques for modeling entity relationships in NoSQL databases:

- **Embedded Documents (One-to-One, One-to-Many)**
  - Use Case: Suitable for relationships where data is frequently accessed together and can be nested within the parent entity.
  - Example: In MongoDB, you can embed addresses inside the users collection.
  - Best For: One-to-one or one-to-many relationships where embedded data is small, tightly coupled, and unlikely to change independently.
- **References (One-to-Many, Many-to-One)**
  - Use Case: Links between entities where one entity references another through an identifier, keeping related data separate.
  - Example: In MongoDB, a post can reference an author by an ID.
  - Best For: Scenarios where related data is large, independently updated, or accessed separately, such as one-to-many or many-to-one relationships (e.g., posts and authors).
- **Denormalization (One-to-Many, Many-to-Many)**
  - Use Case: Repeatedly storing related data to improve query performance by avoiding joins at the cost of storage duplication.
  - Example: Storing product reviews inside each product document in a product catalog.
  - Best For: Read-heavy operations where relationships can be stored together and consistency between related entities can be managed by the application (e.g., product catalogs, social media feeds).
- **Linking Collections (Many-to-Many)**
  - Use Case: Handling many-to-many relationships by creating a separate linking collection (junction table in relational terms).
  - Example: In a music app, a users collection and a songs collection can be linked using a favorites collection.
  - Best For: Many-to-many relationships where maintaining the connection between entities is critical (e.g., users and their favorite songs, orders and products).
- **Graph Databases (Many-to-Many, Complex Relationships)**
  - Use Case: Graph databases like Neo4j are optimized for representing complex, many-to-many relationships using nodes and edges.
  - Example: Modeling a social network where users follow other users and interact with posts.
  - Best For: Applications where relationships are the primary focus, and query patterns involve traversing these relationships frequently (e.g., social networks, recommendation systems, fraud detection).

✓ **Sharding and Replication**

Sharding:

Definition: Sharding is a technique used to distribute data across multiple servers, or "shards," to horizontally scale a database. Each shard contains a subset of the data, which helps in handling large datasets or high-traffic applications.

Example: An online marketplace may shard its products collection by category. One shard could hold all electronic items, while another could hold clothing items.

Key Benefits:

- Improved performance
- Increased capacity
- Distributed workload.

Replication:

Definition: Replication involves creating multiple copies of the same data across different servers or data centers. This ensures data availability and redundancy in case of server failure.

Example: MongoDB uses replica sets to replicate data. A replica set contains a primary server and multiple secondary servers, where the secondary servers are read-only but can take over in case the primary fails.

Key Benefits:

- Fault tolerance
- High availability
- Load balancing.

✓ **Visualizing High-Level Data Model**

UML Class Diagram

❖ **Definition**: UML (Unified Modeling Language) class diagram is a graphical representation of the structure of a system by showing its classes, attributes, methods, and the relationships among object.

❖ **UML Class Diagrams** are used to represent the static structure of the system, including entities (classes) and the relationships between them.

❖ **Use Case Example: An E-commerce Platform with Users, Orders, Products, and Reviews.**

❖ **Entities:**

- **User**: Represents customers.
- **Order**: Represents purchase orders placed by users.
- **Product**: Represents products listed in the e-commerce store.

- **Review**: Represents user reviews for products.

❖ **UML Class Diagram Representation:**

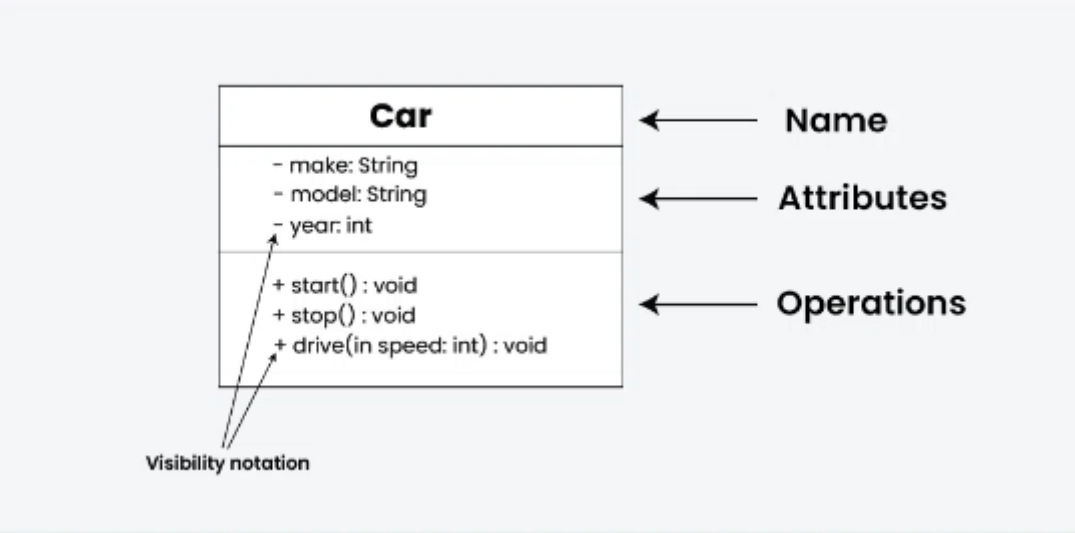
- **Classes**: Represent the entities (e.g., User, Product, Order, Review).
- **Attributes**: Represent the data each class will contain (e.g., userId, name, email, orderId, productId).
- **Associations**: Define the relationships between classes, such as has, owns, or contains (e.g., a User places multiple Orders, an Order contains multiple Products, etc.).

What is a class?

In object-oriented programming (OOP), a class is a blueprint or template for creating objects. Objects are instances of classes, and each class defines a set of attributes (data members) and methods (functions or procedures) that the objects created from that class will possess. The attributes represent the characteristics or properties of the object, while the methods define the behaviors or actions that the object can perform.

UML Class Notation

class notation is a graphical representation used to depict classes and their relationships in object-oriented modeling.



❖ Class Name: Is typically written in the top compartment of the class box and is cantered and bold.
❖ Attributes: Also known as properties or fields, represent the data members of the class. They are listed in the second compartment of the class box and often include the visibility (e.g., public, private) and the data type of each attribute.
❖ Methods: Also known as functions or operations, represent the behaviour or functionality of the class. They are listed in the third compartment of the class box

and include the visibility (e.g., public, private), return type, and parameters of each method.

- ❖ Visibility Notation: Indicate the access level of attributes and methods.
- ❖ Relationships between classes: In class diagrams, relationships between classes describe how classes are connected or interact with each other within a system.

Data Flow Diagrams (DFDs):

Definition: Data Flow Diagrams (DFDs) are an excellent way to visualize how data moves through a system, especially when dealing with NoSQL databases. They illustrate the flow of data between different components, showing how inputs are processed to produce outputs..

Characteristics of Data Flow Diagram (DFD)

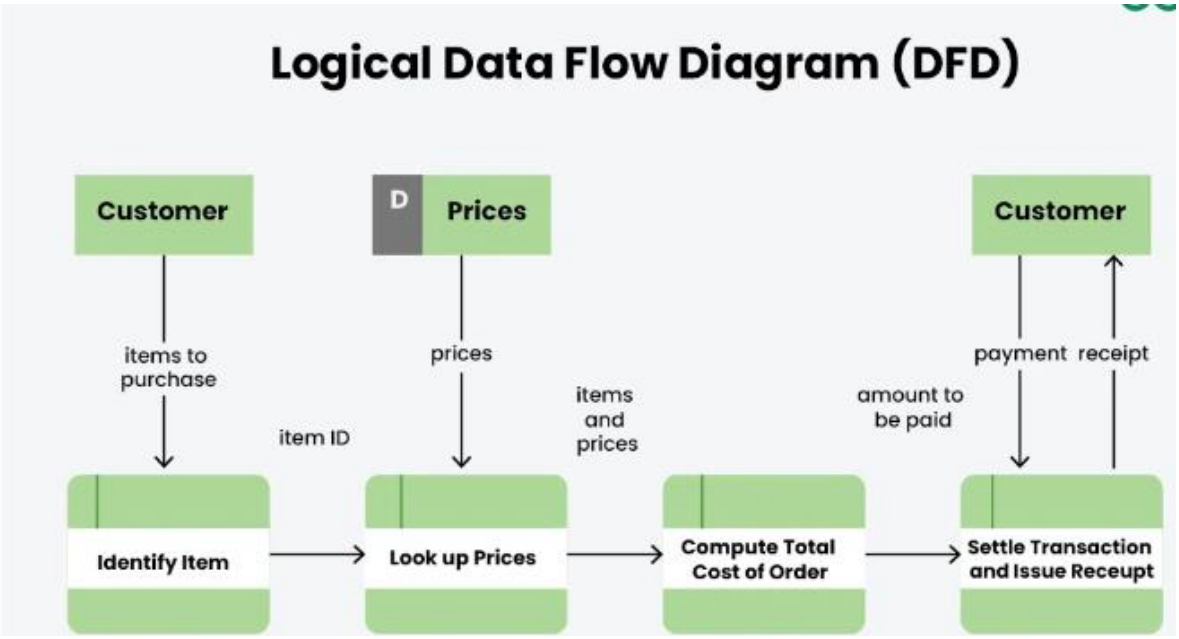
- Graphical Representation: Data Flow Diagram (DFD) use different symbols and notation to represent data flow within system. That simplify the complex model.
- Problem Analysis: Data Flow Diagram (DFDs) are very useful in understanding a system and can be effectively used during analysis. Data Flow Diagram (DFDs) are quite general and are not limited to problem analysis for software requirements specification.
- Abstraction: Data Flow Diagram (DFD) provides a abstraction to complex model i.e. DFD hides unnecessary implementation details and show only the flow of data and processes within information system.
- Hierarchy: Data Flow Diagram (DFD) provides a hierarchy of a system. High- level diagram i.e. 0-level diagram provides an overview of entire system while lower-level diagram like 1-level DFD and beyond provides a detailed data flow of individual process.
- Data Flow: The primary objective of Data Flow Diagram (DFD) is to visualize the data flow between external entity, processes and data store. Data Flow is represented by an arrow Symbol.
- Ease of Understanding: Data Flow Diagram (DFD) can be easily understand by both technical and non-technical stakeholders.
- Modularity: Modularity can be achieved using Data Flow Diagram (DFD) as it breaks the complex system into smaller module or processes. This provides easily analysis and design of a system.

Types of Data Flow Diagram (DFD)

- Logical Data Flow Diagram
- Physical Data Flow Diagram
- Logical Data Flow Diagram (DFD)

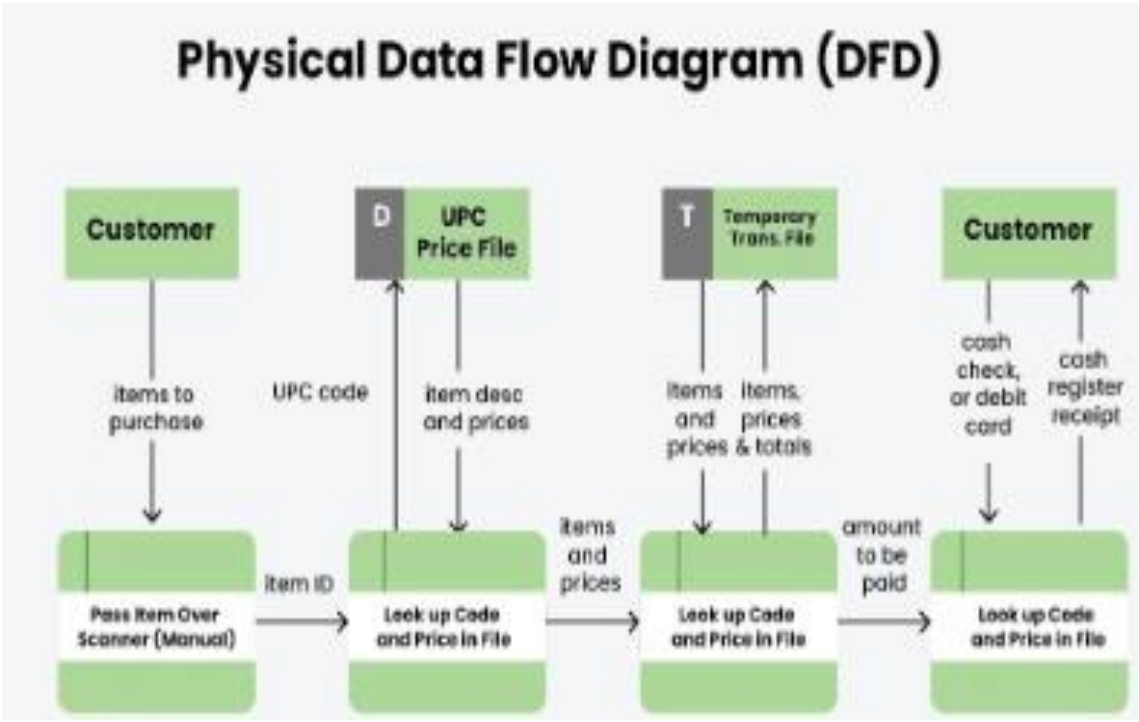
Logical data flow diagram mainly focuses on the system process. It illustrates how data flows in the system. Logical Data Flow Diagram (DFD) mainly focuses on high level processes and data flow without diving deep into technical implementation details. Logical

DFD is used in various organizations for the smooth running of system. Like in a Banking software system, it is used to describe how data is moved from one entity to another.



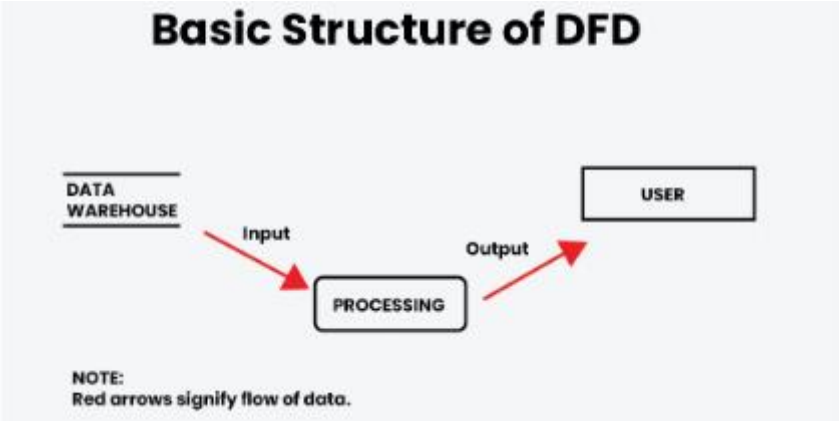
Physical Data Flow Diagram

Physical data flow diagram shows how the data flow is actually implemented in the system. In the Physical Data Flow Diagram (DFD), we include additional details such as data storage, data transmission, and specific technology or system components. Physical DFD is more specific and close to implementation.



❖ Components of Data Flow Diagrams (DFD)

- Process: Input to output transformation in a system takes place because of process function. The symbols of a process are rectangular with rounded corners, oval, rectangle or a circle.
- Data Flow: Describes the information transferring between different parts of the systems. The arrow symbol is the symbol of data flow. A relatable name should be given to the flow to determine the information which is being moved.
- Data Store : The data is stored in the warehouse for later use. Two horizontal lines represent the symbol of the store. The warehouse is simply not restricted to being a data file rather it can be anything like a folder with documents, an optical disc, a filing cabinet.
- Terminator (External Entity): Is an external entity that stands outside of the system and communicates with the system.

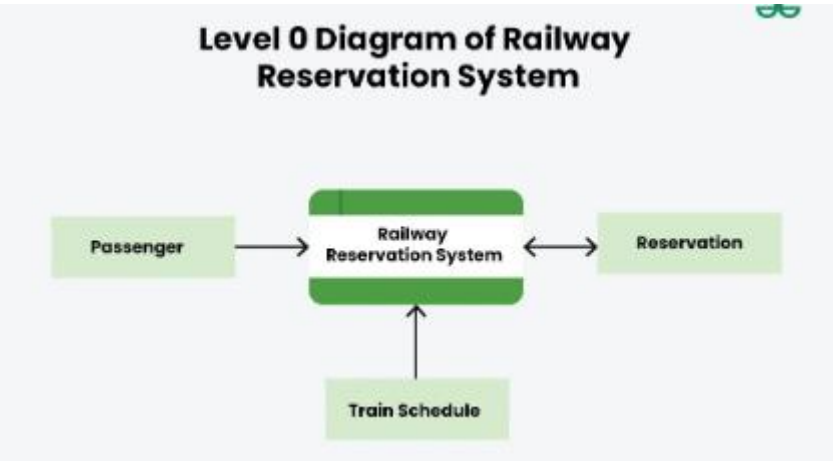


❖ DFD Levels:

Data Flow Diagram (DFD) uses hierarchy to maintain transparency thus multilevel Data Flow Diagram (DFD's) can be created. Levels of Data Flow Diagram (DFD) are as follows:

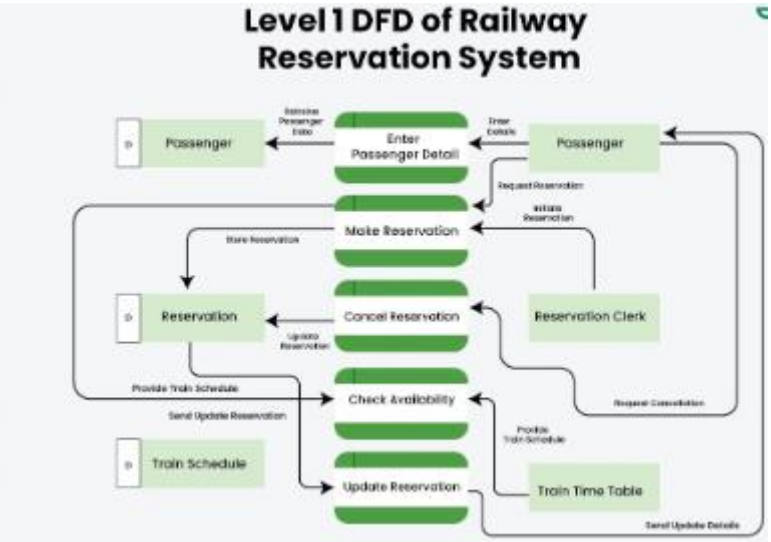
Level 0: Shows a high-level overview of the system.

It is also known as a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.



Level 1: Breaks down each major process into subprocesses.

This level provides a more detailed view of the system by breaking down the major processes identified in the level 0 DFD into sub-processes. Each sub-process is depicted as a separate process on the level 1 DFD. The data flows and data stores associated with each sub-process are also shown. In 1-level DFD, the context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main functions of the system and breakdown the high-level process of 0-level DFD into subprocesses.



Level 2 DFD

This level provides an even more detailed view of the system by breaking down the sub-processes identified in the level 1 DFD into further sub-processes. Each sub-process is depicted as a separate process on the level 2 DFD. The data flows and data stores associated with each sub-process are also shown

Rules for Data Flow Diagram (DFD)

- ❖ **Data can flow from:**
- Terminator or External Entity to Process
- Process to Terminator or External Entity
- Process to Data Store
- Data Store to Process
- Process to Process
- ❖ **Data Cannot Flow From:**
- Terminator or External Entity to Terminator or External Entity
- Terminator or External Entity to Data Store
- Data Store to Terminator or External Entity
- Data Store to Data Store

- ❖ **Advantages of Data Flow Diagram (DFD)**
- It helps us to understand the functioning and the limits of a system.
- It is a graphical representation which is very easy to understand as it helps visualize contents.
- Data Flow Diagram represent detailed and well explained diagram of system components.
- It is used as the part of system documentation file.
- Data Flow Diagrams can be understood by both technical or nontechnical person because they are very easy to understand.
- ❖ Disadvantages of Data Flow Diagram (DFD)
- At times Data Flow Diagram (DFD) can confuse the programmers regarding the system.
- Data Flow Diagram takes long time to be generated, and many times due to this reasons analysts are denied permission to work on it.
- ❖ steps to Draw Data Flow Diagram
- Understand the System
- Identify External Entities
- Identify Processes
- Identify Data Stores
- Use Standard Symbols
- Create Level 0 Diagram
- Identify Data Flows:
- Number Processes and Data Stores
- Review and Validate
- ✓ **Design a conceptual data model**

A conceptual data model is a high-level representation of an organization's data requirements, focusing on the entities and their relationships. It's a blueprint for the database design process.

- steps for designing conceptual data model:

Step 1.Identify Entities:

- Examples: Customers, Products, Orders, Employees, Departments

Step 2.Define Attributes:

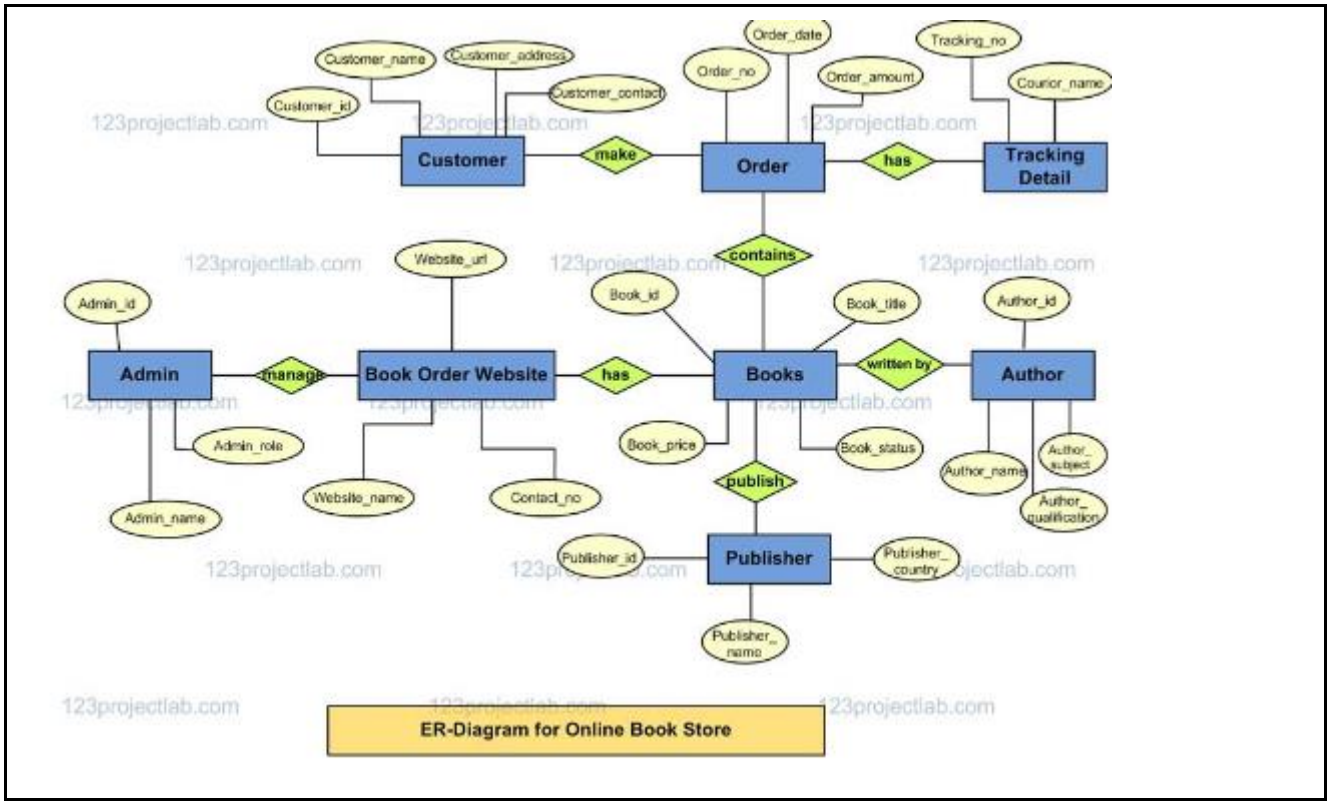
- Customer: CustomerID, CustomerName, Address, PhoneNumber
- Product: ProductID, ProductName, Description, Price

Step 3.Establish Relationships:

- One-to-many: One customer can have many orders.
- Many-to-many: One product can be sold in many orders, and one order can contain many products.
- One-to-one: One employee can have one desk.

Step 4.Create an Entity-Relationship Diagram (ERD):

- **Attributes:**
- Customer: CustomerID, CustomerName, Address, PhoneNumber
- Order: OrderID, OrderDate, TotalAmount
- Book: BookID, BookTitle, Author, Price
- **Relationships:**
- One-to-many: A customer can place many orders.
- Many-to-many: A book can be included in many orders, and an order can contain many books.



Practical Activity 2.2.2: Creating data flow diagram

Task:

1: Read the key readings 2.2.2

2: Go to the computer lab and by referring to the previous theoretical activity 2.2.1 , you create data flow diagram

2: Present the steps to create data flow diagram (DFD).

3: Referring to the steps provided in task 2, create the data flow diagram.

4: Present your work to the trainer or classmates.

5: Ask questions for clarification where necessary.

Key readings 2.2.2: Creating Data Flow Diagram

Steps Create a Data Flow Diagram

Now that you have some background knowledge on data flow diagrams and how they are categorized, you're ready to build your own DFD. The process can be broken down into 5 steps:

Step 1.Identify major inputs and outputs in your system

Nearly every process or system begins with input from an external entity and ends with the output of data to another entity or database. Identifying such inputs and outputs gives a macro view of your system, it shows the broadest tasks the system should achieve. The rest of your DFD will be built on these elements, so it is crucial to know them early on.

Step 2.Build a context diagram

Once you've identified the major inputs and outputs, building a context diagram is simple. Draw a single process node and connect it to related external entities. This node represents the most general process that information follows to go from input to output.

The data diagram flow example below shows how information flows between various entities via an online community. Data flows to and from the external entities, representing both input and output. The center node, "online community," is the general process.

Step 3.Expand the context diagram into a level 1 DFD

The single process node of your context diagram doesn't provide much information—you need to break it down into sub-processes. In your level 1 data flow diagram, you should include several process nodes, major databases, and all external entities. Walk through the flow of information: where does the information start and what needs to happen to it before each data store?

Step 4.Expand to a level 2+ DFD

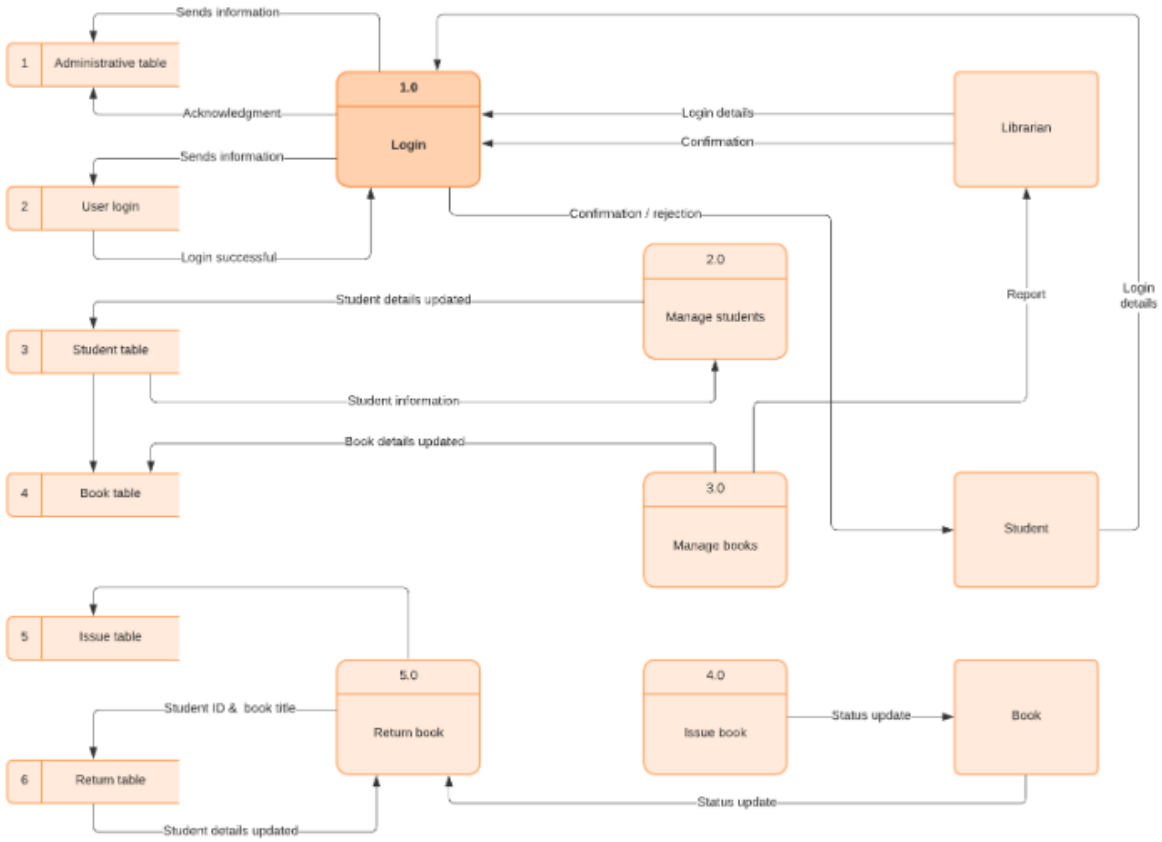
To enhance the detail of your data flow diagram, follow the same process as in step 3. The processes in your level 1 DFD can be broken down into more specific sub-processes. Once again, ensure you add any necessary data stores and flows—at this point, you should have a fairly detailed breakdown of your system. To progress beyond a level 2 data flow diagram, simply repeat this process. Stop once you've reached a satisfactory level of detail.

Step 5.Confirm the accuracy of your final diagram

When your diagram is completely drawn, walk through it. Pay close attention to the flow of information: does it make sense? Are all necessary data stores included? By looking at your final diagram, other parties should be able to understand the way your system functions. Before presenting your final diagram, check with co-workers to ensure your

diagram is comprehensible.

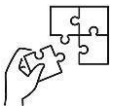
Example of data flow Diagram:



Points to Remember

- Sharding is a technique used to distribute data across multiple servers, or “shards,” to horizontally scale a database.
- Replication involves creating multiple copies of the same data across different servers or data centers.
- Unified Modeling Language is a visual modeling language that helps software developers visualize and construct new systems
- Aggregation is a specialized form of association that represents a “whole-part” relationship.

- An association represents a bi-directional relationship between two classes. It indicates that instances of one class are connected to instances of another class.
- Data Flow Diagrams (DFD) provide a graphical representation of the data flow of a system that can be understood by both technical and non-technical users.
- Components of Data flow Diagram are Process, Data flow, Data store and external entity.
- Levels of Data Flow Diagram are (**Level 0:** Shows a high-level overview of the system, (**Level 1:** Breaks down each major process into subprocesses), (**level 2 DFD** provides an even more detailed view of the system by breaking down the sub-processes identified in the level 1 DFD into further sub-processes).
- Steps for creating data flow diagram are:
    - ✓ Identify major inputs and outputs of the system
    - ✓ Build a context diagram(level 0 DFD)
    - ✓ Expand the content diagram into a level 1DFD
    - ✓ Expand to a level2+DFD
    - ✓ Confirm the accuracy of your final system



Application of learning 2.2.

ABC Company is migrating from SQL to NoSQL, then as a database designer, you are requested to help the company to create DFD on MongoDB conceptual data model for its e-commerce platform, focusing on users, products, orders, and reviews.

Indicative content 2.3: Designing MongoDB database schema

Duration: 7 hrs

Theoretical Activity 2.3.1: Identification of the application of workload

Tasks:

1: Answer the following questions:

i. What do you understand by workload?
ii. What is collection?
iii. What is relationship?
iv. Outlines different types of relationships
v. Give the difference between normalization and validation

2: Write your findings on papers or flipcharts

3: Present your findings to the trainer or classmates

4: Pay attention to the trainer's clarification and ask for clarification where necessary.

5: Read the key readings 2.3.1

Key readings 2.3.1.: Identification of the application of workload

✓ **Identify Application Workload**

Understanding the workload is crucial before designing a schema, as MongoDB schema design is highly dependent on read and write patterns.

- Identify key queries and operations: Determine the most frequent operations—whether they are read-heavy, write-heavy, or a balance of both.
- Understand data access patterns: Understand how data is accessed. Will most queries need to join or aggregate data? Will large datasets be paginated or fetched all at once?
- Estimate data volume: Estimate the size of the data and its growth over time.
- Example: For an e-commerce application:
  - Read-heavy operations: Browsing products, viewing product details.
  - Write-heavy operations: Adding items to the cart, placing orders.
  - Estimated growth: Millions of products and users, thousands of orders per day.

✓ **Define Collection Structure**

In MongoDB, collections are designed to store related documents. The structure depends on whether data is stored as embedded documents or separate collections with references.
Embedded vs. Referenced Documents:

- Embedded Documents: Store nested data within the same document. This is ideal for one-to-one or one-to-few relationships and for scenarios where the data is always accessed together.
- Referenced Documents: Store data in separate collections with references to related documents. This is useful for one-to-many or many-to-many relationships, or when data is accessed independently.

  Example:
  - Embedded: Storing product reviews within the product document for faster access.

```
{
  "_id": 101,
  "name": "Laptop",
  "price": 999.99,
  "reviews": [
    { "userId": 1, "rating": 5, "comment": "Great laptop!" },
    { "userId": 2, "rating": 4, "comment": "Good value for money." }
  ]
}
```

  - Referenced: Storing orders in a separate collection, with references to users and products.

```
{
  "_id": 5001,
  "userId": 1,
  "productIds": [101, 102],
  "total": 1999.98
}
```

✓ **Map Schema Relationships**

In MongoDB, schema relationships (e.g., one-to-one, one-to-many, many-to-many) are handled through either embedding or referencing.
Types of Relationships:

- One-to-One: Use embedded documents or references. E.g., storing user profile information in the same document as the user.

- One-to-Many: Typically use embedded documents for related data or references for data that grows large or is queried separately.
- Many-to-Many: Use references to relate data between collections.
- Example: For the e-commerce application:

User to Orders (One-to-Many): One user can have multiple orders, stored as references.

Order to Products (Many-to-Many): An order can contain multiple products, and a product can be part of multiple orders. This is best represented using references in both Orders and Products collections.

```
{
 "_id": 1,
 "name": "Alice",
 "orders": [5001, 5002] // References to Orders collection
}

{
 "_id": 5001,
 "userId": 1, // Reference to Users collection
 "products": [
  { "productId": 101, "quantity": 2 },
  { "productId": 102, "quantity": 1 }
 ]
}
```

✓ **Validate and Normalize Schema**

Even though MongoDB is schema-less, it's essential to maintain some level of validation and normalization for consistency and data integrity.

Validation:

- Schema Validation: MongoDB allows you to define validation rules at the collection level. You can enforce specific types and structures for your documents.
- Example: Ensure that all documents in the users collection contain name, email, and address fields.

```
{
 "validator": {
  "$jsonSchema": {
   "bsonType": "object",
   "required": [ "name", "email", "address" ],
   "properties": {
    "name": { "bsonType": "string" },
    "email": { "bsonType": "string", "pattern": "^.+@.+$" },
```

```
            "address": {
              "bsonType": "object",
              "properties": {
                "street": { "bsonType": "string" },
                "city": { "bsonType": "string" },
                "state": { "bsonType": "string" },
                "zip": { "bsonType": "string" }
              }
            }
          }
        }
      }
    }
```

- **Normalization:**

- Normalization in NoSQL often means minimizing data redundancy where necessary. However, you can denormalize if it improves performance (e.g., embedding data for faster reads).
- Example: Store a reference to a userID in orders to avoid duplicating user data in every order document.

✓ **Apply Design Patterns**

MongoDB offers various design patterns to solve common data modeling challenges.

Common MongoDB Design Patterns:

- **Bucket Pattern:**

- Description: Useful when data grows quickly. Instead of creating one document per event, you can store multiple related events in one document.
- Example: Instead of creating a new document for every individual user login, you can group logins for a user into a single document by day.

Schema:

```
{
  "userId": "user123",
  "logins": [
    { "date": "2023-09-10", "loginCount": 5 },
    { "date": "2023-09-11", "loginCount": 3 }
  ]
}
```

- **Extended Reference Pattern:**

- Description: This pattern is used when you need to minimize the number of joins (or $lookup queries) by embedding essential information from the referenced document.

- Example: Embedding a product's name and price within an order document instead of performing a join to get the product details during every order retrieval.

Schema:

```
{
 "_id": "order123",
 "userId": "user456",
 "items": [
  { "productId": "prod789", "productName": "Laptop", "price": 999.99, "quantity": 1 },
  { "productId": "prod654", "productName": "Mouse", "price": 19.99, "quantity": 2 }
 ],
 "totalAmount": 1039.97
}
```

Outlier Pattern:

- Description: For handling rare "outlier" documents that are much larger than others, move those documents to a separate collection to optimize general query performance.
- Example: If some products have extremely large descriptions or media files, move those fields to a separate productDetails collection.

Practical Activity 2.3.2: Drawing of entity relationship Diagram (ERD)

Task:

1: Read the case study below and then draw an entity relationship diagram (ERD)

Suppose that you are given the following requirements for a simple database for the National League (NL):

i. The NL has many teams,
ii. Each team has a name, a city, a coach, a captain, and a set of players,
iii. Each player belongs to only one team,
iv. Each player has a name, a position (such as left wing or goalie), a skill level, and a set of injury records,
v. The team captain is also a player, then construct a clean and concise ER diagram for the NL database

2: Respect instructions from your trainer

3: Draw an entity relationship diagram

3: Present your work to the trainer or classmates.

6: Read the key readings 2.3.2 and ask for clarification where necessary.

Key readings 2.3.1: Drawing an entity relationship diagram

- ✓ **Drawing an entity relationship diagram**
  - **Definition of entity relationship diagram**

An **Entity Relationship Diagram (ERD)** is a visual representation of the relationships between entities in a database. It helps in designing a database schema by identifying entities, their attributes, and the relationships among them. ERDs are widely used in both relational and NoSQL database design to provide a clear understanding of the structure and organization of data

Preparing to draw an ER diagram

Before starting to draw an ER diagram, it is essential to gather requirements and identify the entities involved. This preparation phase sets the groundwork for creating a comprehensive and effective ER diagram.

Consider the following steps:

- Start by understanding the goals and objectives of your project. Gather all necessary information about the system or database you are working with.
- Identify the main entities involved in the system. These entities can represent real-world objects, concepts, or people.
- Consider the relationships between the identified entities. Determine how they interact and depend on each other.

Step-by-step of drawing an ER diagram

Step 1: Defining entities

Start by identifying the main entities in your system. Entities are objects or concepts that have data to be stored.

Step 2: Establishing relationships

Establishing relationships between entities is a crucial aspect of an ER diagram. Common relationship types include one-to-one (each instance of an entity is associated with exactly one instance of another entity, and vice versa), one-to-many , and many-to-many

Step 3: Adding attributes

Attributes provide additional information about entities. Add relevant attributes to the entities identified in the previous steps.

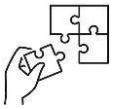
Step 4: Refining the Diagram

In this final step, we focus on refining the ER diagram to enhance clarity and readability. Organize the entities and relationships in a logical and intuitive manner. Group related entities together and arrange them in a way that reflects their connections.

Points to Remember

- Collections: is group of documents that are stored in MongoDB
- Embedded documents data are Store in nested data within the same document whereas Referenced documents data are Store in separate collections with references to related documents.
- Types of relationships includes one to one, one to many, many to one and many to many.

- Normalization is process of removing/minimizing data redundancy where necessary in database.
- Apply Design Patterns like Use MongoDB's design patterns like the bucket pattern, extended reference pattern, and outlier pattern to optimize the schema
- Steps for drawing entity relationship diagram
    - ✓ Defining entities
    - ✓ Establishing relationships
    - ✓ Add attributes to entities defined
    - ✓ Refining diagram



Application of learning 2.3.

FitTrack is a company that is seeking for a database developer to design a fitness tracking application. you are tasked to help the company identify the application's workload, define the collection structure, and map schema relationships, to validate and normalize the schema of FitTrack company

Learning outcome 2 end assessment

Theoretical assessment

Q1. Read the statement carefully, circle the letter corresponding to the correct answer based on the statement given

i. Which of the following is a tool used for drawing NoSQL databases?
   a) Microsoft Word
   b) Edraw Max
   c) Excel
   d) Photoshop

ii. What is the first step in installing Edraw Max?
   a) Run the software without downloading
   b) Download the installer from the official website
   c) Purchase a physical copy from a store
   d) Set up a MySQL database

iii. In MongoDB, what is a 'collection'?
   a) A set of tables
   b) A group of related documents
   c) An entity relationship model
   d) A data visualization tool

iv. Which diagram is commonly used to model entity relationships in databases?
   a) Bar Chart
   b) UML Class Diagram
   c) Flowchart
   d) Pie Chart

v. What does 'sharding' refer to in a NoSQL database like MongoDB?
   a) Data replication between servers
   b) Horizontal partitioning of data across multiple servers
   c) Normalizing data within a single server
   d) Archiving unused data

vi. Which of the following diagrams helps visualize how data flows through a system?
   a) Entity Relationship Diagram (ERD)
   b) UML Diagram
   c) Data Flow Diagram (DFD)
   d) Gantt Chart

vii. What should be considered when designing a conceptual data model?
   a) The file system structure
   b) The user interface design
   c) The high-level entities and their relationships
   d) Network configuration

viii. What is the primary goal of schema normalization in MongoDB?
   a) Improve query speed
   b) Ensure minimal data redundancy
   c) Increase indexing size
   d) Decrease replication time

ix. Which factor should be considered when defining a collection structure in MongoDB?
   a) The primary key in SQL
   b) Application workload and access patterns
   c) The number of users accessing the application
   d) The size of the database index

x. Which MongoDB design pattern is commonly applied for managing relationships between documents?
   a) One-to-One Mapping
   b) Embedding and Referencing
   c) Normalization
   d) Data Denormalization

Q2. Read carefully and then answer the following questions:

1. What are some popular NoSQL database drawing tools?

2. How do you install Edraw Max for database drawing?

3. What are collections in MongoDB, and how do you identify them in a conceptual data model?

4. How do you model entity relationships in a NoSQL database?

5. What is sharding, and how is it applied in MongoDB?

6. What is replication in MongoDB?

7. How do you visualize a high-level data model in a database design?

8. How would you design a conceptual data model for an e-commerce website using MongoDB?

9. How do you identify the application workload for MongoDB schema design?

10. What are common schema design patterns used in MongoDB?

Practical assessment

Imagine there is ABC Network Company that is located in your area. The company needs to draw an entity relationship diagram (ERD) based on following business rules:

a) A salesperson may manage many other salespeople.
b) A salesperson is managed by only one salespeople.
c) A salesperson can be an agent for many customers.
d) A customer is managed by one salespeople.
e) A customer can place many orders.
f) An order can be placed by one customer.
g) An order lists many inventory items.
h) An inventory item may be listed on many orders.
i) An inventory item is assembled from many parts.
j) A part may be assembled into many inventory items.
k) Many employees assemble an inventory item from many parts.
l) A supplier supplies many parts.
m) A part may be supplied by many suppliers.

Now, help the company carry out the above-mentioned tasks

References:

(Learning NoSQL — NoSQL Database Designing, 2019)

(Mongo DB in Action, 2016)
(NoSQL Data Modeling: Streamlined Database Design, 2024)

(NoSQL Database Design, 2023)

(NoSQL Database Design, 2024)https://www.geeksforgeeks.org/nosql-database-design/

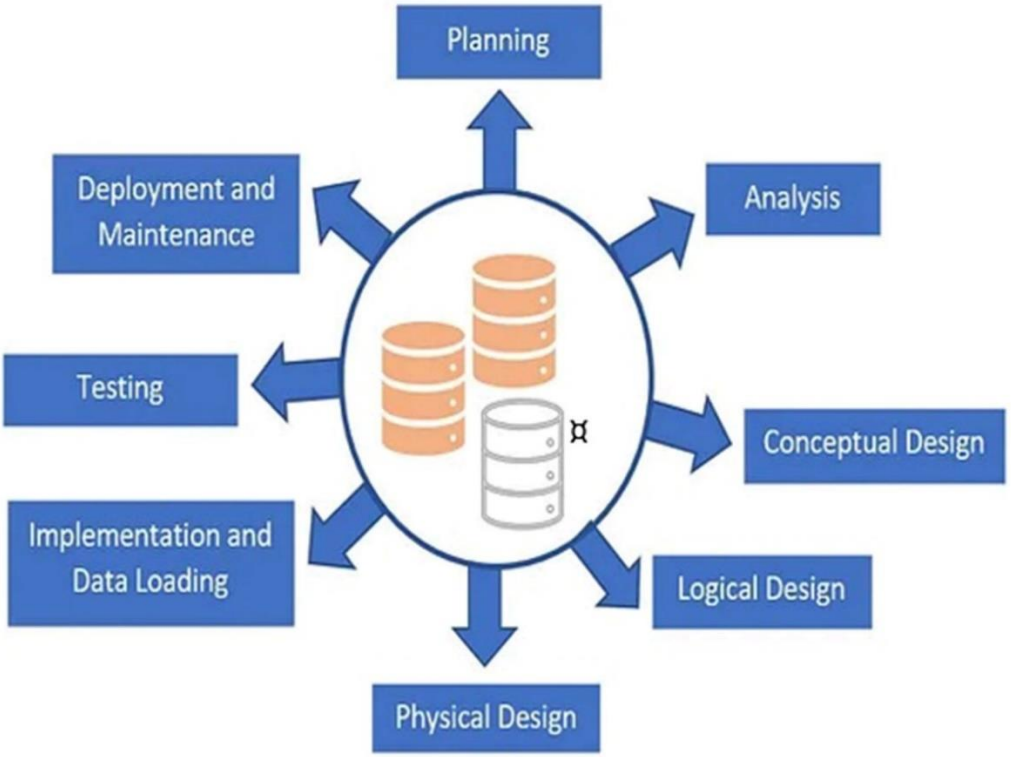
(Nosql for Mere Mortals, April 2015)
(Schema Design and Relationship in NoSQL Document-based databases, 2024)

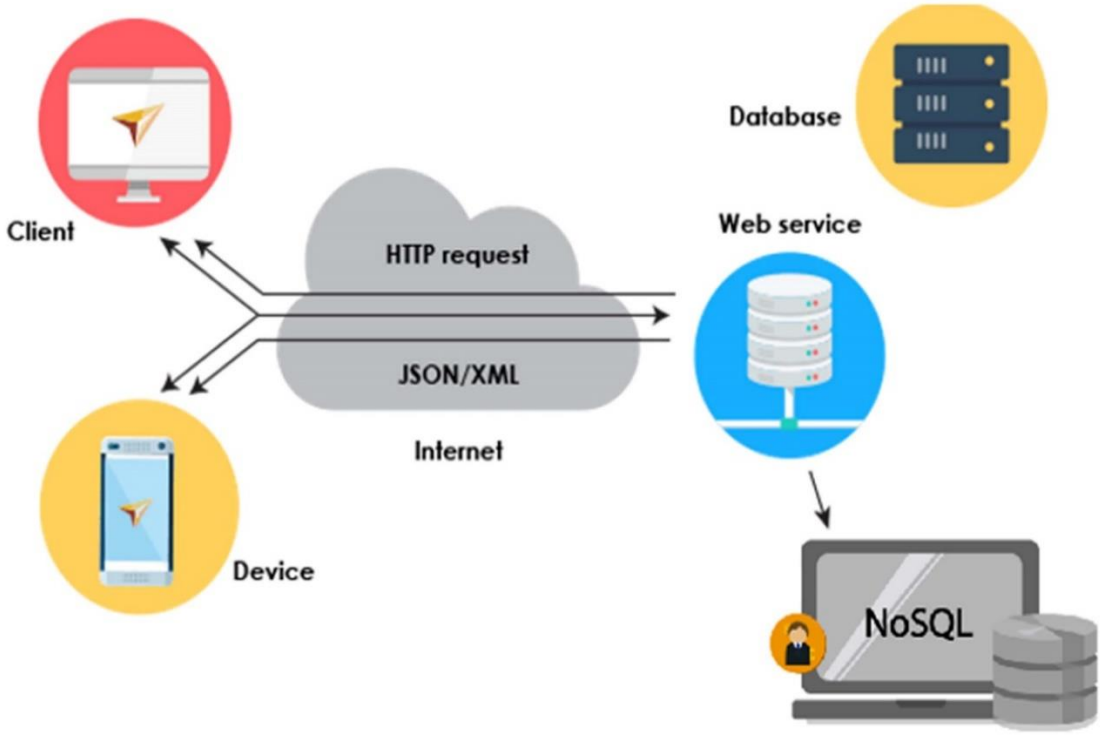
(Shakuntala Gupta Edward, 2015)

https://blog.usu.com/en-us/schema-design-and-relationship-in-nosql-document-based-databases

https://medium.com/@bubu.tripathy/nosql-database-design-1a42731c8265

https://solvaria.com/nosql-data-modeling-streamlined-database-design/


Planning
Analysis
Conceptual Design
Logical Design
Physical Design
Implementation and Data Loading
Testing
Deployment and Maintenance


Client
Database
Web service
HTTP request
JSON/XML
Internet
Device
NoSQL

Indicative contents
3.1 Perform MongoDB Data Definition
3.2 MongoDB data Manipulating
3.3 Apply Query optimizations

Key Competencies for Learning Outcome3: Implement Database Design

Knowledge	Skills	Attitudes
• Description of MongoDB • Description of MongoDB data manipulation • Identification of mongosh methods • Description of Optimization techniques • Identification of current operations performance.	• Performing MongoDB Data Definition • Executing of data manipulation • Applying Mongosh Methods • Evaluating Optimization techniques • Evaluating current operations performance.	• Having attention to detail while defining MongoDB data based on database requirements. • Having problem-solving capability for executing data manipulation based on database requirements • HavingContinuous learning based on query performance • Being collaborative based on query performance and database requirements.

Duration: 20 hrs

Learning outcome 3 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Define properly mongoDB data based on database requirements.
2. Describe clearly data manipulation language based on database requirements.
3. Identify correctly mongosh methods based on database requirements.
4. Describe appropriately optimization techniques based on query performance.
5. Identify effectively current operations performance based on query performance.
6. Evaluate correctly current operations performance based on query performance.
7. Apply properly mongosh methods based on query performance and database requirements.

 **Resources**

Equipment	Tools	Materials
● Computer	● MongoDB ● Mongosh ● MongoDB compass ● MongoDB Shell	● Electricity ● Internet

Indicative content 3.1: Perform MongoDB data definition

Duration: 5 hrs

Theoretical Activity 3.1.1: Description of mongoDB data definition

Tasks:

1: Answer the following questions:

i. What do you understand by mongoDB data?

ii. What is data manipulation language mean?

iii. List features of MongoDB

iv. Discuss advantage and disadvantage of mongoDB

v. Explain mongoDB

2: Write findings provided on papers or flipcharts.

3: Present your findings to the trainer or classmates.

4: Pay attention to the trainer's clarification and ask question where necessary.

5: Read the key readings 3.1.1

Key readings 3.1.1. Description of MongoDB data definition

✓ **MongoDB data definition**

MongoDB data definition refers to the structure and organization of data within a MongoDB database, which is designed to be flexible and scalable. In MongoDB, data is stored in collections, which are analogous to tables in relational databases. Each collection contains documents, which are the fundamental data units and are stored in a binary JSON-like format called BSON (Binary JSON).

❖ **Components of MongoDB data definition include:**

- **Collections**: Groups of related documents. Collections do not require a predefined schema, allowing for dynamic data storage.
- **Documents**: Individual records stored in collections. Each document can have a unique structure, containing fields and values that can vary from one document to another.
- **Fields**: Key-value pairs within a document, where keys are strings and values can be various data types, including strings, numbers, arrays, and even nested documents.

- **Indexes**: Structures that improve the speed of data retrieval operations on a collection. Indexes can be created on one or multiple fields within documents to optimize query performance.
- **Schema**: While MongoDB is schema-less, it can benefit from a defined schema to maintain consistency and structure. This is often achieved through schema validation rules, which can be applied to collections.
- **Data Types**: MongoDB supports various data types, including string, number, date, array, object, and others, which allow for versatile data representation.

✓ **Perform MongoDB Data Definition**

To perform various MongoDB Data Definition tasks, including creating, dropping, and renaming databases and collections.

Create a Database

To create a new database, you can use the use command, which switches to a specified database. If the database doesn't exist, it will be created when you first insert data into it.

use myDatabase

Create Collections

- Once you have a database, you can create collections within it.
  db.createCollection("myCollection")
- You can also create a collection implicitly by inserting a document:
  db.myCollection.insertOne({ name: "John Doe", age: 30 })

Drop a Database

To drop an entire database, you can use the following command. This will delete the database and all its collections:

db.dropDatabase()

Make sure to switch to the database you want to drop using use before executing this command.

Drop Collections

To drop a specific collection from a database, use:

db.myCollection.drop()

This will remove the specified collection and all the documents within it.

Rename a Database

MongoDB does not provide a direct command to rename a database. However, you can achieve this by copying the collections from the old database to a new one and then dropping the old database.

```
// Switch to the source database
use oldDatabase

// Create the new database
use newDatabase
```

```
// Copy each collection from oldDatabase to newDatabase
db.oldCollection1.find().forEach(function(doc) {
   db.newCollection1.insert(doc);
});

// Repeat for other collections as needed

// Drop the old database after transferring all collections
use oldDatabase
db.dropDatabase()
```

Rename Collections

To rename a collection, use the renameCollection command:

```
db.myOldCollection.renameCollection("myNewCollection")
```

Practical Activity 3.1.2: Performing data definition of MongoDB

Task:

1: Read the key readings 3.1.2

2: Go to the computer lab and by referring to the previous activity install mongo DB

2: Present the steps to perform data definition

3: Referring to the steps provided in task2, perform data defintion

4: Present your work to the trainer or classmates

5: Ask questions for clarification where necessary.

Key readings 3.1.2: Perform data definition MongoDB

✓ **Perform data definition**

Create Database & Collections

Step 1: Start MongoDB

- If you are running MongoDB locally, start the MongoDB server (mongod).
- If you are using a cloud service like MongoDB Atlas, log in to your cluster.

Step 2: Open the MongoDB Shell or MongoDB Compass

- For the MongoDB shell, open a terminal and type:
  mongo

- For MongoDB Compass, simply open the application and connect to your MongoDB instance.

Step 3: Create a Database

- To create a database, type the following command in the MongoDB shell. This switches to the new database, which will be created when you insert data:
  use myDatabase

  Note: MongoDB only creates the database once data is added.

Step 4: Create a Collection

You can create collections in two ways.

- **Implicit Creation by Inserting Data**: MongoDB will automatically create a collection when you insert a document into it.
  db.myCollection.insert({ name: "John", age: 30 })

- **Explicit Creation using createCollection()**: If you want to explicitly create a collection, use:
  db.createCollection("myCollection")

This creates a collection called myCollection.

Drop Database & Collections

Step 1: Drop a Collection

- If you want to delete a collection from your database, use the .drop() method:
  db.myCollection.drop()

- This command will remove the myCollection from the database.

Step 2: Drop a Database

- To delete an entire database, switch to the database using the use command and then use the dropDatabase() command:

i. use myDatabase
ii. db.dropDatabase()

- This will drop the myDatabase and all its collections.

Rename Collections

Step 1: Rename a Collection

To rename a collection, use the renameCollection() method. Make sure that you are connected to the database containing the collection you want to rename.

- First, insert some data into a collection if it doesn't exist:
  db.oldCollection.insert({ name: "Sample Document", value: 123 })

- Then, rename the collection:
  db.oldCollection.renameCollection("newCollectionName")

Points to Remember

- MongoDB Platforms are Create, Drop and Rename.
- **Steps for perform data definitions in MongoDB**

Create Database & Collections

Step 1: Start MongoDB

Step 2: Open the MongoDB Shell or MongoDB Compass

Step 3: Create a Database

Step 4: Create a Collection

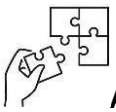
Drop Database & Collections

Step 1: Drop a Collection

Step 2: Drop a Database

Rename Collections

- First, insert some data into a collection if it doesn't exist.
- Then, rename the collection



Application of learning 3.1.

MXY Ltd is a company that generates revenue from selling Products. The Company uses file system (books) to store information about the sales and inventory of the products. The Company has a problem of non-efficient security and management of information about the products and Customers. As Database developer, you are requested to create Mongo Database with documents and collections used to store both product and customers information .

Indicative content 3.2: MongoDB data Manipulating

Duration: 10 hrs

Theoretical Activity 3.2.1: Description of mongoDB data manipulation

Tasks:

1: Answer the following questions:

i. What is data manipulation?

ii. Give the difference between delete and update with their respective syntax

iii. Give the Difference between replacing and querying documents.

iv. What is Bulk Write Operations?

v. What is aggregation operations?

2: Write your findings on papers or flipcharts.

3: Present your findings to the trainer or classmates.

4: Pay attention to the trainer's clarification and ask question where necessary

5: Read the key readings 3.2.1

Key readings 3.2.1.: Description of MongoDB data manipulation

- **MongoDB data Manipulating**

✓ **Execute data manipulation**

MongoDB provides powerful data manipulation capabilities that allow developers to insert, update, delete, query, and aggregate data efficiently.

Insert Documents

In MongoDB, inserting a document into a collection is straightforward using the insertOne() or insertMany() methods.

Example:

```
// Insert a single document into the Users collection
db.Users.insertOne({ name: "John Doe", email: "john@example.com", age: 30 });

// Insert multiple documents into the Workouts collection
```

```
db.Workouts.insertMany([
  { userId: 1, type: "Running", duration: 30 },
  { userId: 1, type: "Cycling", duration: 60 }
]);
```

Update Documents

You can update documents in a collection using methods like updateOne(), updateMany(), or replaceOne().

Example:

```
// Update a single document: Change the duration of the workout
db.Workouts.updateOne(
  { type: "Running" },
  { $set: { duration: 45 } }
);

// Update multiple documents: Add a new field to all documents
db.Workouts.updateMany(
  { userId: 1 },
  { $set: { completed: true } }
);
```

Delete Documents

You can delete documents from a collection using the deleteOne() or deleteMany() methods.

Example:

```
// Delete a single document
db.Workouts.deleteOne({ type: "Cycling" });

// Delete multiple documents
db.Workouts.deleteMany({ userId: 1 });
```

Replacing Documents

You can replace an entire document using the replaceOne() method, which substitutes an existing document with a new one.

Example:

```
// Replace the existing document with a new one
```

```
db.Users.replaceOne(
  { name: "John Doe" },
  { name: "John Smith", email: "john.smith@example.com", age: 35 }
);
```

Querying Documents

Querying in MongoDB allows you to retrieve data from collections using methods like find() and findOne().

Example:

```
// Find all users older than 30
db.Users.find({ age: { $gt: 30 } });

// Find one document
db.Users.findOne({ name: "John Smith" });
```

Indexes

Indexes improve the efficiency of queries by allowing faster lookups. You can create indexes on specific fields using the createIndex() method.

Example:

```
// Create an index on the email field
db.Users.createIndex({ email: 1 });
```

1. Bulk Write Operations

Bulk write operations in MongoDB allow you to perform multiple database operations (such as inserts, updates, and deletes) in a single command, which reduces the number of network roundtrips and can improve performance.

Example:

```
// Perform multiple operations in bulk on the Users collection
db.Users.bulkWrite([
  { insertOne: { document: { name: "Alice", email: "alice@example.com" } } },
  { updateOne: { filter: { name: "John Smith" }, update: { $set: { age: 36 } } } },
  { deleteOne: { filter: { name: "Alice" } } }
]);
```

This batch operation inserts a new document, updates an existing one, and deletes another—all in one command.

2. Aggregation Operations

Aggregation in MongoDB is a powerful way to process data and perform computations over large datasets. The aggregate () method is used to apply various transformations and filter criteria.

Example:

```
// Aggregate workout data to calculate the total duration for a specific user
db.Workouts.aggregate([
 { $match: { userId: 1 } },  // Filter by userId
 { $group: { _id: "$userId", totalDuration: { $sum: "$duration" } } }  // Group by userId and sum durations
]);
```

This query filters workout records by userId and calculates the total workout duration.

- ✓ Apply Mongosh Methods

Mongosh (MongoDB Shell) is an interactive command-line interface that allows you to interact with your MongoDB databases. It provides a rich set of methods for executing database operations, managing collections, and performing administrative tasks.

Mongosh Methods Overview

Collection Methods

Collection methods allow you to interact with and manipulate data within a collection.

- **Insert a Document**:

```
db.collection.insertOne({ name: "John", age: 30 })
```

- **Insert Multiple Documents**:

```
db.collection.insertMany([{ name: "Alice", age: 25 }, { name: "Bob", age: 22 }])
```

- **Update Documents**:

```
db.collection.updateOne({ name: "John" }, { $set: { age: 31 } })
```

- **Find Documents**:

```
db.collection.find({ age: { $gte: 25 } })
```

- **Delete a Document**:

```
db.collection.deleteOne({ name: "Alice" })
```

- **Create an Index**:

db.collection.createIndex({ name: 1 }) // Ascending index

Cursor Methods

Cursor methods are used to interact with the results returned from a query.

- **Iterate Over a Cursor**:

```
const cursor = db.collection.find()
while (cursor.hasNext()) {
printjson(cursor.next())
}
```

- **Limit Results**:

db.collection.find().limit(5)

- **Sort Results**:

db.collection.find().sort({ age: 1 }) // Sort by age in ascending order

- **Skip Results**:

db.collection.find().skip(10).limit(5) // Skip the first 10 documents, then limit to 5

Database Methods

Database methods provide functionalities related to the database itself.

- **Get the Current Database**:

db.getName()

- **List All Collections**:

db.getCollectionNames()

- **Database Stats**:

db.stats()

- **Drop the Current Database**:

db.dropDatabase()

Query Plan Cache Methods

Query plan cache methods allow you to manage and inspect cached query plans.

- **Get the Query Plan Cache**:

db.collection.getPlanCache().list()

- **Clear the Query Plan Cache**:

db.collection.getPlanCache().clear()

Bulk Operation Methods

Bulk operation methods are optimized for executing multiple write operations in a single command.

- **Initialize Bulk Operations**:

const bulk = db.collection.initializeOrderedBulkOp()

- **Add Insert Operation**:

bulk.insert({ name: "Charlie", age: 28 })

- **Add Update Operation**:

bulk.find({ name: "Alice" }).updateOne({ $set: { age: 26 } })

- **Add Delete Operation**:

bulk.find({ name: "Bob" }).deleteOne()

- **Execute Bulk Operations**:

bulk.execute()

User Management Methods

User management methods are used to create and manage users in MongoDB.

- **Create a User**:

```
db.createUser({
  user: "myUser",
  pwd: "myPassword",
  roles: [{ role: "readWrite", db: "myDatabase" }]
})
```

- **Drop a User**:

db.dropUser("myUser")

- **List Users**:

db.getUsers()

Role Management Methods

Role management methods allow you to manage user roles in the database.

- **Grant Role to User**:
db.grantRolesToUser("myUser", [{ role: "dbAdmin", db: "myDatabase" }])

- **Revoke Role from User**:
db.revokeRolesFromUser("myUser", [{ role: "dbAdmin", db: "myDatabase" }])

- **List Roles**:
db.getRoles()

Replication Methods

Replication methods allow you to manage and inspect replica sets.

- **Check Replica Set Status**:
rs.status()

- **Initiate a Replica Set**:
rs.initiate()

- **Add a Member to Replica Set**:
rs.add("newMember:27017")

Sharding Methods

Sharding methods enable you to manage sharded clusters.

- **Enable Sharding for a Database**:
sh.enableSharding("myDatabase")

- **Shard a Collection**:
sh.shardCollection("myDatabase.myCollection", { shardKeyField: 1 })

- **Check Sharding Status**:
sh.status()

Free Monitoring Methods

MongoDB provides features for free monitoring of your deployment.

- **Enable Free Monitoring**:
db.enableFreeMonitoring()

- **Disable Free Monitoring**:
db.disableFreeMonitoring()

Object Constructors and Methods

MongoDB provides constructors for common types.

- **Create a New ObjectId**:
  const id = ObjectId()

- **Create a Date Object**:
  const date = new Date()

Connection Methods

Connection methods are used to connect to MongoDB instances.

- **Connect to a MongoDB Instance**:
  mongo --host <hostname> --port <port> -u <username> -p <password> --authenticationDatabase <authDB>

- **Authenticate a User**:
  db.auth("username", "password")

Atlas Search Index Methods

MongoDB Atlas provides advanced querying capabilities through search indexes.

- **Create a Search Index**:
  db.collection.createIndex({ name: "text" })

- **List Search Indexes**:
  db.collection.getIndexes()

- **Delete a Search Index**:
  db.collection.dropIndex("name_text")

Practical Activity 3.2.2: Executing MongoDB data manipulation

Task:

1: Read key reading 3.2.2

2: Go to the computer lab to execute mongoDB data manipulation

2: Present the steps to perform data manipulation in MongoDB

3: Referring to the steps provided task2, perform data manipulation in MongoDB

4: Present your work to the trainer or classmates

5: Ask questions for clarification where necessary.

Key readings 3.2.2: Executing MongoDB data manipulation

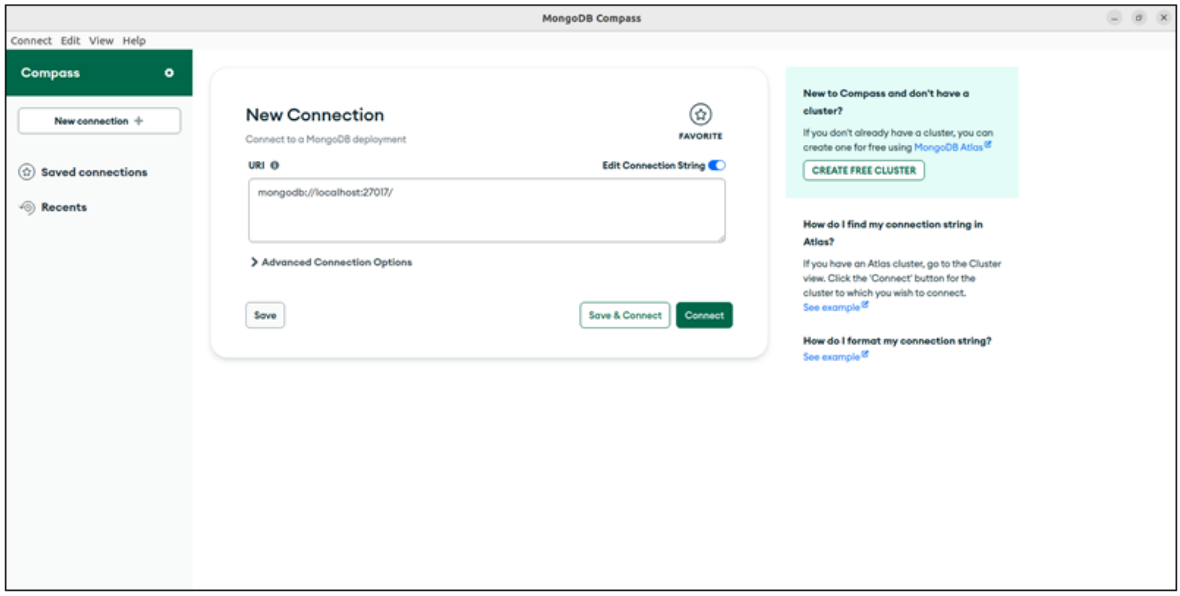
Data manipulation in MongoDB refers to the various operations that allow you to create, read, update, and delete (CRUD) documents stored in collections within a MongoDB database.

Steps to perform data manipulation in MongoDB

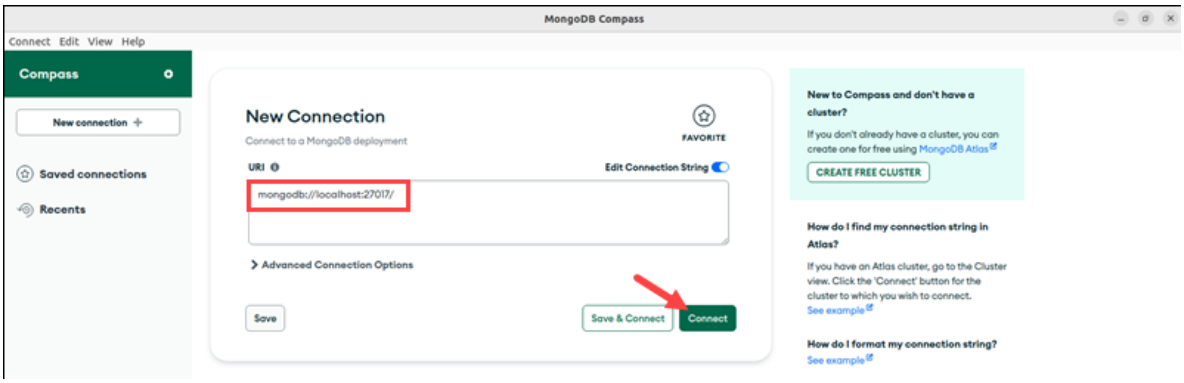
Step 1. Create a Database in MongoDB Using Compass

Step 2. Launch the MongoDB Compass program. If using Linux, run the following command in the terminal:

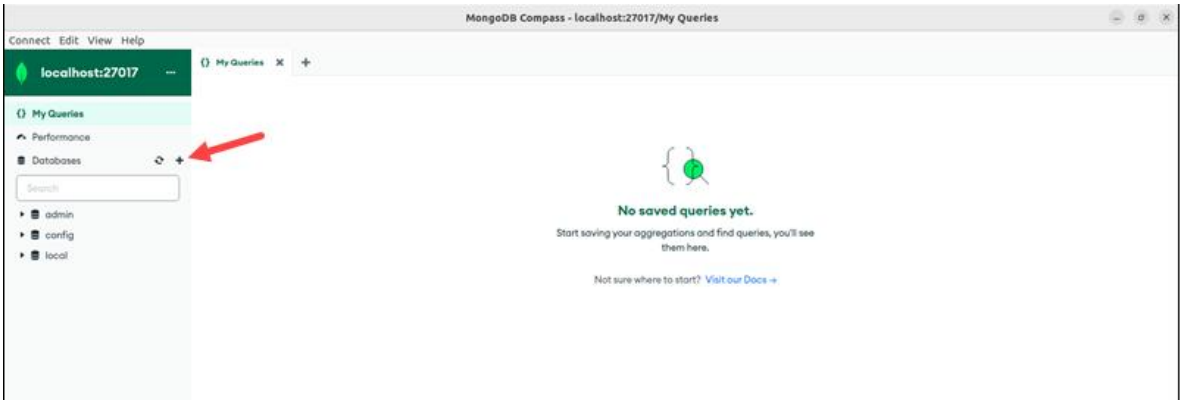
mongodb-compass



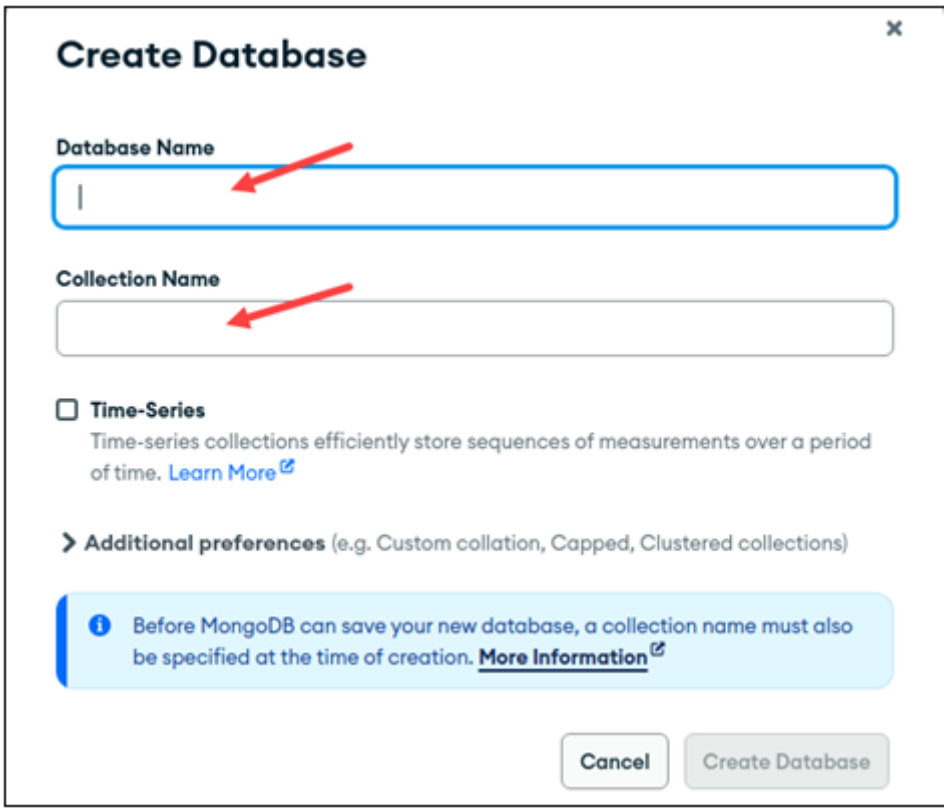
Step 3. Connect to the MongoDB instance. Adjust the URI if required and click Connect.



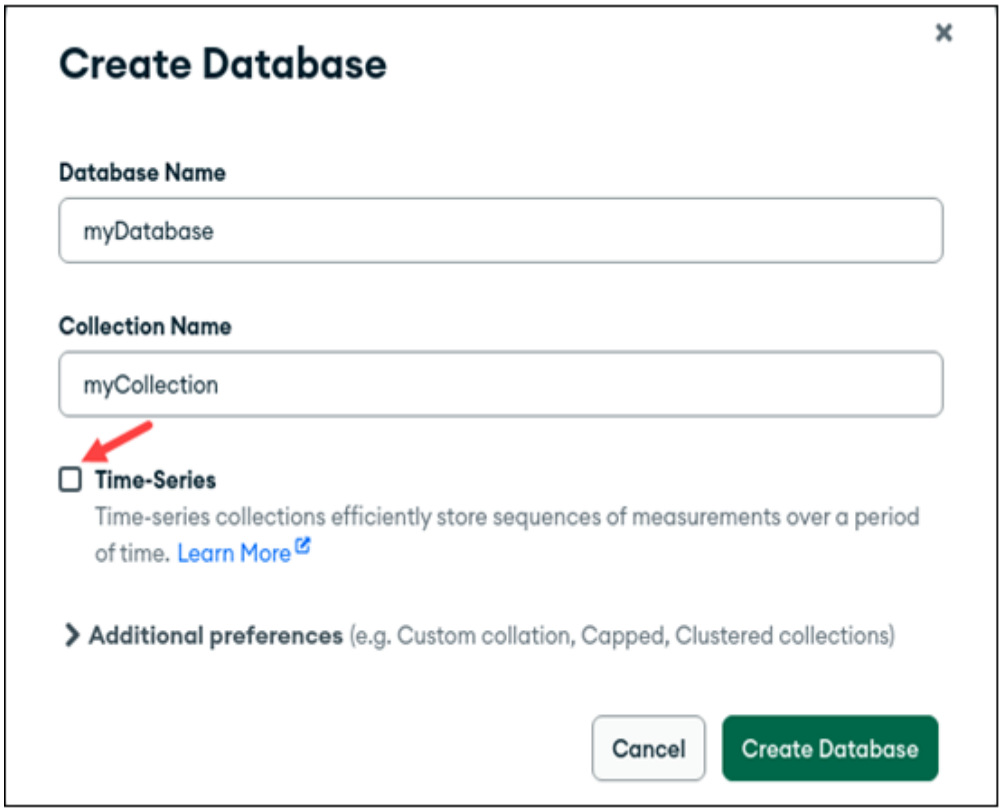
Step 4. Click the plus icon (+) next to Databases.



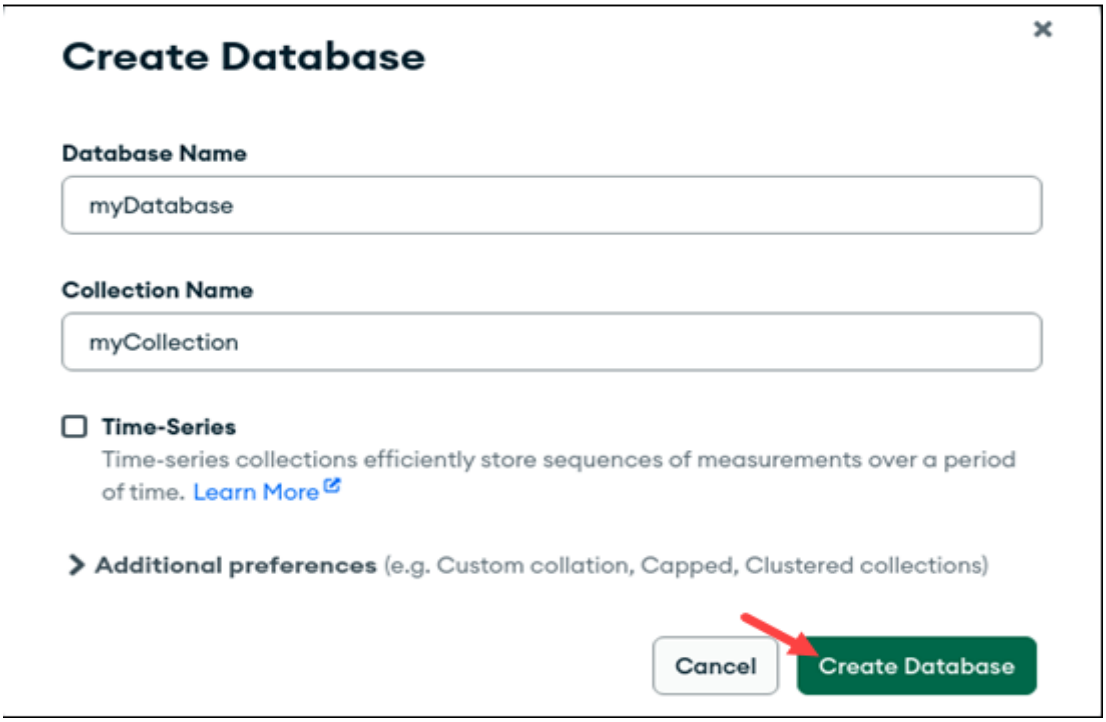
Step 5. Enter the database and collection name in the appropriate fields.



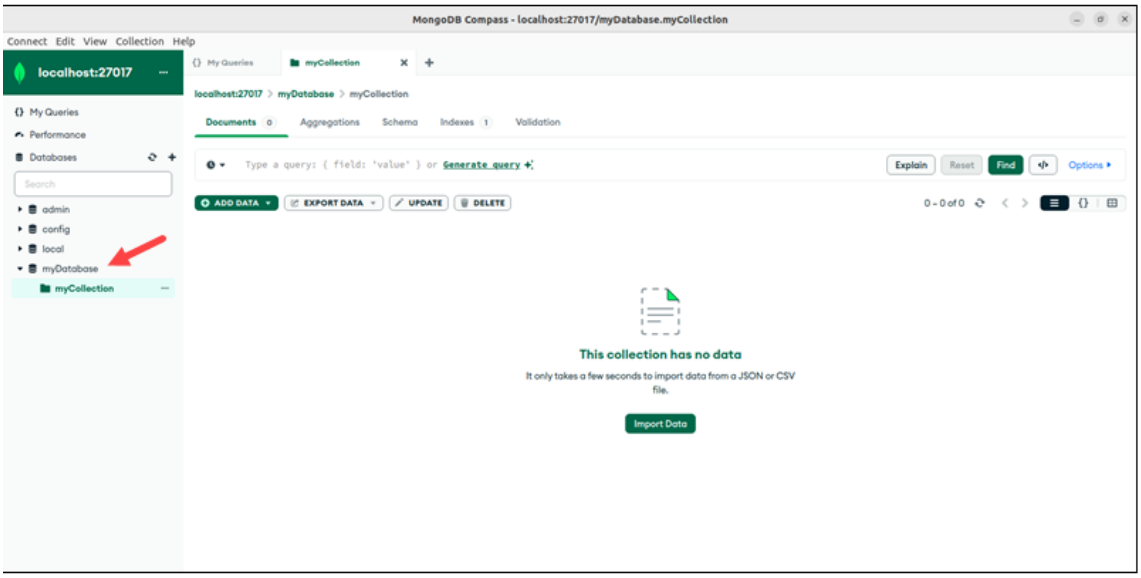
Step 6. (Optional) Check the Time-Series box if the database contains time-series data.



Step 7. Review the names and options. Once ready, click Create Database.



The database and collection appear in the database listing on the left.



Step 8.Perform Data Manipulation

Step 9. Apply Bulk Write Operations

Step 10. Apply Aggregation Operations

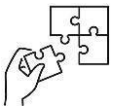
Step 11. Apply Mongosh Methods (Collection Methods, Cursor methods and Database Methods)

Points to Remember

- Insert document based on the insert () Method: To insert data into MongoDB collection, you need to use MongoDB's insert () or save () method.
- Update document based on update method: MongoDB's update () and save () methods are used to update document into a collection.
- Replacing Documents: You can replace a single document using the collection.replaceOne() method. replace One () accepts a query document and a replacement document.
- Querying Documents based on the find () Method: To query data from MongoDB collection, you need to use MongoDB's find () method and the pretty () Method To display the results in a formatted way, you can use pretty () method.
- Bulk Write Operations: MongoDB provides clients the ability to perform write operations in bulk. Bulk writes operations affect a single collection.
- Aggregations operations process data records and return computed results.
- Mongosh Methods are MongoDB Shell (Mongosh) provides methods for managing and interacting with various MongoDB entities like collection methods, cursor methods database methods, query plan cache methods, bulk operation methods, user management methods, replication methods, sharding methods and free monitoring methods.

- **Steps for creating MongoDB are as follows:**
  - ✓ Create mongoDB
  - ✓ Launch the MongoDB
  - ✓ Connect to the MongoDB instance. Adjust the URI if required and click Connect.
  - ✓ Click the plus icon (+) next to Databases.
  - ✓ Enter the database and collection name in the appropriate fields.
  - ✓ (Optional) Check the Time-Series box if the database contains time-series data.
  - ✓ Review the names and options. Once ready, click Create Database.

- ✓ Perform data manipulation
- ✓ Apply aggregation operations
- ✓ Apply bulk Write Operations
- ✓ Apply mongosh methods



Application of learning 3.2.

XY Ltd is a company that generates revenue from delivering products. The company has a problem of losing accessibility and management of information about the products and customers. You are tasked to help the company create Mongodb, collections and then execute data manipulation operations and then apply mongosh methods.

Indicative content 3.3: Applying query optimizations

Duration: 5 hrs

Theoretical Activity 3.3.1: Identification of query optimizations

Tasks:

1: Answer the following questions:

i. What do you understand by optimization?

ii. Differentiate types of index

2: Write your findings on papers or flipcharts.

3: Present your findings to the trainer or classmates

4: Pay attention to the trainer's clarification and ask questions where necessary

5: Read the key readings 3.3.1

Key readings 3.3.1.: Identification of query optimizations

✓ **Query Optimizations**

Applying query optimizations in MongoDB, including techniques, evaluation of current performance, and methods for optimizing query performance.

Describe Optimization Techniques

Optimization techniques in MongoDB help improve query performance and resource utilization. Key techniques include:

Indexing

- **Indexes** are special data structures that store a small portion of the data set, enabling faster queries. Create indexes on fields that are frequently queried, sorted, or used in join operations.
- **Types of Indexes**: Single-field, compound, text, geospatial, and wildcard indexes.

Query Profiling

- Use the **MongoDB profiler** to analyze query performance and identify slow queries. It can log operations that exceed a certain execution time.

Schema Design

- Design your schema based on the application's read and write patterns. Denormalization (embedding documents) can improve read performance, while normalization (referencing documents) can save space.

Projection

- Use projection to return only the necessary fields in your queries, reducing the amount of data transferred and processed.

  db.myCollection.find({}, { name: 1, age: 1 }) // Returns only name and age

Query Optimization

- Use the query optimizer to find the most efficient way to execute a query. This includes analyzing query shapes and filter criteria.

Caching

- Implement caching strategies (e.g., in-memory caching) to store frequently accessed data, reducing database load.

Evaluate Performance of Current Operations

To evaluate the performance of your current operations in MongoDB, consider the following steps:

Use the Explain Plan

- The explain() method provides insights into how MongoDB executes a query, including the stages and indexes used.

  db.myCollection.find({ age: { $gt: 30 } }).explain("executionStats")

- Analyze the output for metrics like execution time, number of documents examined, and index usage.

Profile Slow Queries

- Enable the profiler to log slow queries and examine the output.

  db.setProfilingLevel(1, { slowms: 100 }) // Log queries that take longer than 100ms

- Review the logs to identify queries that may need optimization.

Monitor Database Performance

- Use MongoDB's built-in tools or third-party monitoring solutions (like MongoDB Atlas) to track performance metrics such as query throughput, latency, and resource utilization.

Optimize query performance

Once you've evaluated the performance of current operations, you can implement specific optimizations:

Add Indexes

- Analyze the explain() output to determine if your queries could benefit from additional indexes. Create indexes on frequently queried fields.

```
db.myCollection.createIndex({ age: 1 })
```

✓ **Optimize query structure**

- Rewrite inefficient queries for better performance. Avoid using $where and regular expressions when possible, as they can be slow.

```
// Inefficient
db.myCollection.find({ $where: "this.age > 30" })

// More efficient
db.myCollection.find({ age: { $gt: 30 } })
```

✓ **Use Aggregation Pipelines Efficiently**

- Optimize aggregation pipelines by using stages that reduce data size early in the process (e.g., $match before $group).

✓ **Limit the Data Returned**

- Use limit() to restrict the number of documents returned, which can reduce network and processing overhead.

```
db.myCollection.find({ age: { $gte: 25 } }).limit(10)
```

✓ **Analyze Query Performance Regularly**

- Continually monitor and analyze query performance, updating indexes and optimizing queries as the application evolves and data grows.

Practical Activity 3.3.2: Creating index in MongoDB

Task:

1: Read the key readings 3.3.2

2: You are requested to go to the computer lab to create index in MongoDB of any company

2: Present the steps to create index in MongoDB

3: Referring to the steps provided in task2, create index in MongoDB

4: Present your work to the trainer or classmates

5: Ask questions for clarification where necessary.

Key readings 3.3.2: Creating index in MongoDB

✓ **Creates indexes on collections.**

Optimizing query performance in MongoDB is essential for ensuring that your application runs efficiently, especially as your data grows.

Steps to perform query optimization

Steps 1: Use Indexes Effectively

Indexes are the most critical tool for optimizing query performance.

- **Create Indexes**: Identify fields that are frequently queried or sorted and create indexes on them.

```
db.collection.createIndex({ fieldName: 1 })  // Ascending index
```

- **Compound Indexes**: Use compound indexes for queries that filter or sort by multiple fields.

```
db.collection.createIndex({ field1: 1, field2: -1 })  // Ascending on field1, descending on field2
```

- **Text Indexes**: Use text indexes for efficient searching of string content.

```
db.collection.createIndex({ content: "text" })
```

- **Wildcard Indexes**: Consider using wildcard indexes if your document structure varies widely.

```
db.collection.createIndex({ "$**": 1 })  // Indexes all fields
```

Steps 2: **Analyze Query Performance**

Use MongoDB tools to analyze and understand how your queries are performing.

- **Explain Plan**: Use the explain() method to get details about how a query is executed.
- **Profiler**: Enable the database profiler to track query performance and identify slow queries.

Steps 3: **Optimize Query Structure**

- **Projection**: Retrieve only the fields you need using projection.
- **Avoid Large Result Sets**: Use limit() and skip() to control the size of returned results, especially in pagination.
- **Use Query Operators Wisely**: Use specific query operators that can utilize indexes efficiently (e.g., $eq, $gt, $lt, etc.).

Steps 4: Optimize Data Schema

Design your schema to reduce the need for complex queries and joins.

- **Denormalization**: Consider denormalizing data to reduce the number of queries needed. Embed related data within documents if it makes sense for your application.
- **Avoid Unbounded Growth**: Be cautious of large arrays within documents, as they can impact performance. Consider breaking them into separate collections if necessary.

Steps 5: Maintain Your Database

Regular maintenance can also help improve query performance.

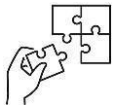
- **Compact Collections**: Use the compact command to reclaim disk space and improve performance.

```
db.runCommand({ compact: "collectionName" })
```

- **Remove Unused Indexes**: Regularly review and remove any unused indexes, as they can slow down write operations.

Points to Remember

- Types of indexes: single, compound, text and geospatial
- Techniques of optimization: indexing, query execution plan, query optimization, sharding and aggregation pipeline optimization.
- Retain that this is the syntax for creating index: db.collection.createIndex (<keys>, <options>, <commitQuorum>).



Application of learning 3.3.

HealthTrack's data engineering team needs to optimize MongoDB queries to improve system performance and reduce response times, identify bottlenecks, indexes to use and aggregation pipelines reduce data processing volume. you are tasked to optimize the performance of HealthTrack's database

Learning outcome 3 end assessment

Theoretical assessment

A) Read the statement carefully, then circle the letter corresponding to the correct answer on the given statement.

1. Which command is used to create a new database in MongoDB?
   a) CREATE DATABASE <db_name>
   b) USE <db_name>
   c) db.createDatabase(<db_name>)
   d) db.create(<db_name>)

2. What is the correct command to drop a collection named "users"?
   a) db.users.delete()
   b) db.dropCollection("users")
   c) db.users.remove()
   d) db.drop("users")

3. To rename a collection from "oldCollection" to "newCollection", which command should you use?
   a) db.oldCollection.rename("newCollection")
   b) db.renameCollection("oldCollection", "newCollection")
   c) db.oldCollection.renameCollection("newCollection")
   d) db.rename("oldCollection", "newCollection")

4. Which of the following methods is used to insert a document into a collection?
   a) db.collection.insertOne()
   b) db.collection.add()
   c) db.collection.create()
   d) db.collection.insertDocument()

5. What is the purpose of the **updateOne()** method in MongoDB?
   a) To remove a single document from a collection.
   b) To replace an entire document in a collection.
   c) To update a single document that matches the specified filter.
   d) To add a new document if it does not exist.

6. Which of the following operations will completely replace an existing document?
   a) updateOne()
   b) replaceOne()
   c) insertOne()
   d) upsert()

7. What method would you use to create an index on a field called "email" in a collection named "users"?
   a) db.users.createIndex({"email": 1})
   b) db.users.addIndex({"email": 1})
   c) db.createIndex({"users.email": 1})
   d) db.users.index({"email": 1})

8. Which of the following best describes a bulk write operation in MongoDB?
   a) A single write operation performed on multiple documents at once.
   b) Multiple write operations bundled together to be executed in one request.
   c) A write operation that affects all documents in a collection.
   d) A write operation that creates a new collection.

9. In MongoDB, which aggregation method is used to group documents by a specified field?
   a) $group
   b) $match
   c) $sort
   d) $project

10. What is the purpose of query optimization in MongoDB?
    a) To increase the number of documents returned by a query.
    b) To decrease the execution time of queries.
    c) To reduce the amount of data stored in the database.
    d) To create backups of the database.

B) Read the following statement carefully and then answer True for the correct statement or False for wrong statement

1. In MongoDB, a database can be created using the command db.createDatabase("myDB").

2. The command to drop a collection in MongoDB is db.collection.drop().

3. You can rename a collection in MongoDB using the command db.collection.rename("newName").

4. To insert a document into a MongoDB collection, you can use the db.collection.insertOne() method.

5. In MongoDB, the update() method can only add new fields to an existing document and cannot modify existing ones.

6. The command db.collection.deleteOne() is used to delete a single document from a collection.

7. Bulk write operations in MongoDB can only insert documents, not update or delete them.

8. Aggregation operations in MongoDB are used for performing complex data transformations and computations.

9. The cursor methods in MongoDB allow for iteration over the results of a query.

10. Optimization techniques in MongoDB can include creating indexes to improve query performance.

C) Match ColumnA with ColumnB respectively according to their MongoDB terms and Definitions ,then write the correct answer on column named with ANSWERS

ANSWERS	MongoDB TERMS	MongoDB Definitions
1……..	1.MongoDB	A. A type of database that does not require a fixed schema and can handle unstructured data
2…….	2.NoSQL	B. A method of organizing data in MongoDB that consists of key-value pairs.
3……..	3.Replica set	C.A group of MongoDB instances that maintain the same data set for high availability.
4……	4.Collections	D. A graphical interface that allows users to interact with the database using commands.
5…….	5.Schema – less	E. A storage structure in MongoDB that automatically overwrites its oldest entries when a size limit is reached.
6…….	6.Aggregation	F. Decrease the execution time of queries
7…….	7.Sharding	G. A format used to represent data in MongoDB that supports more data types than JSON.
8…….	8.Mongo shell	H. An open-source NoSQL database management program suitable for high-volume data storage.
9……..	9.Capped collection	I. Indexes are a critical optimization technique that helps speed up data retrieval operations.
10……	10.BSON	J. The process of combining multiple documents to generate aggregated results.
		K. A group of documents in MongoDB, similar to tables in relational databases.
		L. The partitioning of data across multiple servers to improve performance and manageability.

Practical assessment

HealthSync, a digital health record system, underwent a MongoDB infrastructure optimization, enhancing query performance, data manipulation, and query performance, resulting in faster data retrieval and improved system efficiency.as a design implement a database design of this healthSync.

References:

(Mongo DB in Action, 2016)
(MongoDB Schema Design, 2023)

(Nosql for Mere Mortals, April 2015)
(Shakuntala Gupta Edward, 2015)

(Soh, manipulating-data-with-mongodb, 2020)

EDUCBA. (n.d.). MongoDB list collections. EDUCBA. https://www.educba.com/mongodb-list-collections/

Eloutmadi, A. (2023, November 13). MongoDB schema design. Medium. https://medium.com/@eloutmadiabderrahim/mongodb-schema-d-23003eeb0199

MongoDB, Inc. (n.d.). Atlas UI: Databases. MongoDB. https://www.mongodb.com/docs/atlas/atlas-ui/databases/

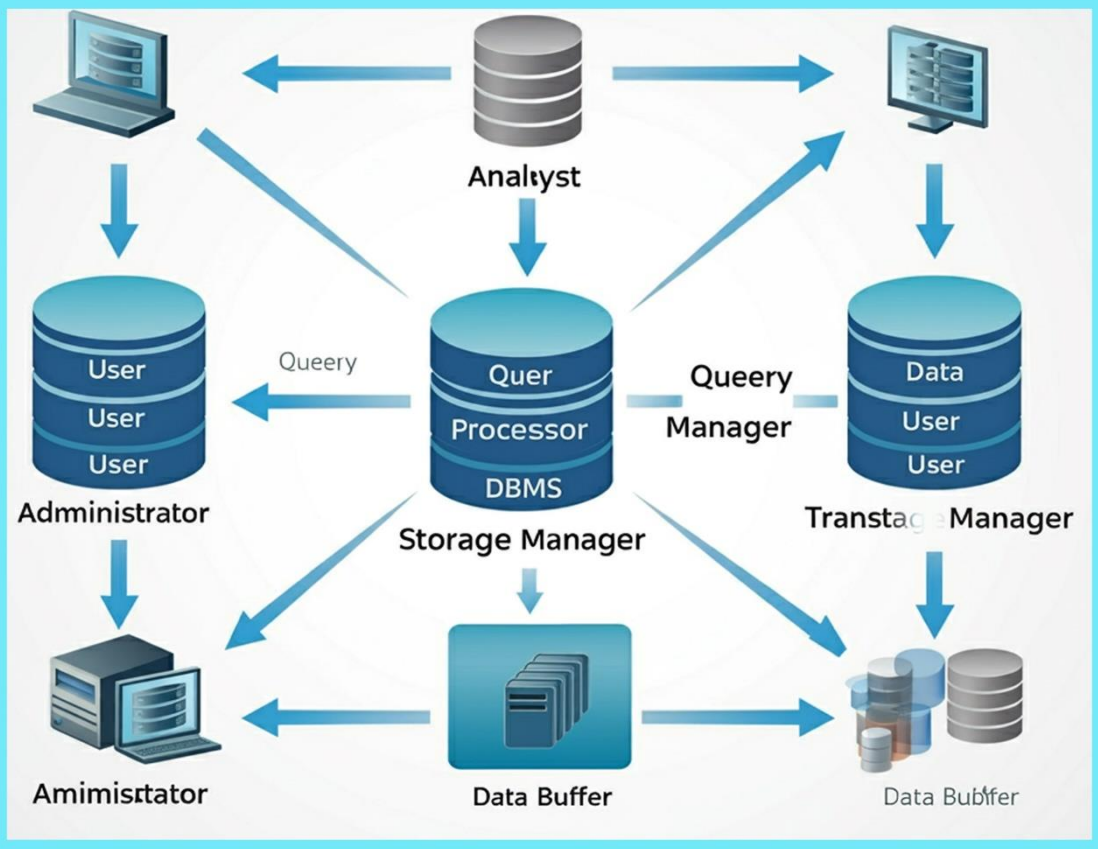
MongoDB, Inc. (n.d.). CRUD operations. MongoDB. https://www.mongodb.com/docs/manual/crud/

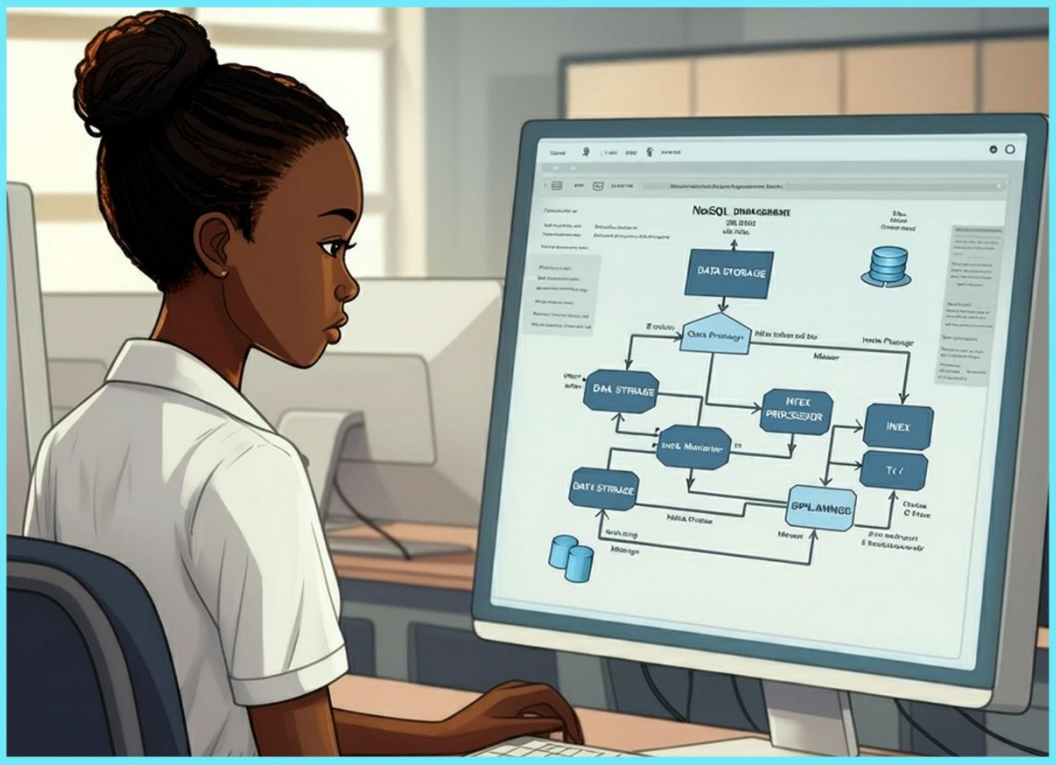
MongoDB, Inc. (n.d.). Data modeling. MongoDB. https://www.mongodb.com/docs/manual/data-modeling/

MongoDB, Inc. (n.d.). Databases and collections. MongoDB. https://www.mongodb.com/docs/manual/core/databases-and-collections/

NodeTeam. (2023, November 8). How to optimize MongoDB queries. Medium. https://nodeteam.medium.com/how-optimize-mongodb-queries-8b8a0f0fd049

Singh, A. (2023, October 23). Manipulating data with MongoDB. Towards Data Science. https://towardsdatascience.com/manipulating-data-with-mongodb-bd561f09d76a


Analyst
User
User
User
Administrator
Queery
Quer
Processor
DBMS
Storage Manager
Queery
Manager
Data
User
User
Transtag Manager
Amimistrator
Data Buffer
Data Buffer



Indicative contents

4.1 Management of Database users

4.2 Securing Database

4.3 Deployment of Database

Key Competencies for Learning Outcome 4: Manage Mongo DB Database

Knowledge	Skills	Attitudes
• Identification of the role of database users • Identification of MongoDB cluster architectures • Description of configuration of role-based access control • Description of the application of deployment options.	• Creating users • Managing roles and privilege • Enabling access control and enforce Authentication • Configuring of role-based access control • Performing backup and recovery • Applying deployment Options • Scaling MongoDB with sharding	• Having paying attention to details while creating use cases. • Having problem-solving capability for creating use case and performing data analysis • Having continuous learning based on requirement analysis process • Having aptability based on requirement analysis process. Being collaborative based on requirement analysis process.

Duration: 10 hrs

Learning outcome 4 objectives:

By the end of the learning outcome, the trainees will be able to:

1. Identify properly database user used in database managed with appropriate permissions.
2. Configure properly how to crate database user in database managed with appropriate permissions.
3. Configure properly roles-based access control in database managed with appropriate permissions.
4. Perform effectively how database is secured in line with best practices
5. Describe clearly MongoDB cluster architecture managed with appropriate permissions.
6. Apply effectively how to deploy database based on the target environment.

Resources

Equipment	Tools	Materials
● Computer	● Mongosh ● MongoDB compass ● MongoDB	● Electricity ● internet

Indicative content 4.1: Management of database users

Duration: 3 hrs

Theoretical Activity 4.1.1: Description of database user management

Tasks:

1: Answer the following questions:

i. Define the following terms:
   a) access control
   b) encryption

ii. Differentiate roles from privileges

2: Write your findings on Papers or flipcharts

3: Present your findings to the trainer or classmates

4: Pay attention to the trainer's clarification and ask question where necessary.

5: Read the key readings 4.1.1

Key readings 4.1.1.: Description of database user management

✓ **Database user management**

Definition: Database user management is the process of managing users who have access to a database. The system administrator, or admin, is responsible for managing database user access and permissions.

best practices for managing database users include:

- Access control: Use access controls to restrict unauthorized access to sensitive data.
- Authentication: Use authentication to validate a user's identity and ensure that only trusted users can access the database.
- Roles and groups: Use roles and groups to define user permissions.
- Least privilege: Implement the principle of least privilege.
- Encryption: Use encryption to provide high-level access control for users.
- Auditing: Monitor and record user database actions, including those performed by administrators.
- Separate servers: Separate the web server from the database server to prevent attackers from accessing the database.
- Policy: Define a clear policy for managing database users and permissions.
- Automation and monitoring: Automate and monitor the process of managing database users and permissions.
- Backup and testing: Test and back up the database.

Identify the Role of Database Users

Purpose of Database Users

- Database users interact with the MongoDB database to perform operations such as reading, writing, updating, and deleting data.
- Each user has specific roles that define their access and capabilities within the database.

Types of Roles

- **Built-in Roles**: MongoDB provides several predefined roles with specific permissions:

  - **read**: Allows users to read data from a specified database.
  - **readWrite**: Permits both reading and writing data.
  - **dbAdmin**: Grants administrative privileges, such as creating indexes and managing users.
  - **clusterAdmin**: Allows full administrative control over a cluster, including sharding and replication management.
  - **userAdmin**: Enables the management of users and roles.

- **Custom Roles**: Users can also be assigned custom roles tailored to specific application needs, which can include a combination of privileges.

Creating Users

Connect to the MongoDB Shell

- Launch the MongoDB shell by running:

```
mongo
```

Switch to the Admin Database

- Use the admin database or the relevant database where you want to create the user:

```
use admin
```

Create a New User

- Use the createUser() method to add a new user with specified roles and privileges.

```
db.createUser({
  user: "appUser",
  pwd: "securePassword123", // Choose a strong password
  roles: [
    { role: "readWrite", db: "myDatabase" }, // Grants read and write access to myDatabase
    { role: "dbAdmin", db: "myDatabase" }      // Grants administrative privileges on myDatabase
```

```
  ]
})
```

Parameters:

- user: Username for the new user.
- pwd: Password for the user.
- roles: An array of roles assigned to the user.

Manage Roles and Privileges

View Existing Users

- To list all users in the current database, execute:

```
db.getUsers()
```

Update User Roles

- To modify a user's roles, use the updateUser() method:

```
db.updateUser("appUser", {
  roles: [{ role: "read", db: "myDatabase" }] // Update the roles assigned to appUser
})
```

Remove a User

- To delete a user from the database, utilize the dropUser() method:

```
db.dropUser("appUser") // Remove appUser from the database
```

Create Custom Roles

- If needed, create a custom role with specific privileges:

```
db.createRole({
  role: "customRole",
  privileges: [
    { resource: { db: "myDatabase", collection: "" }, actions: ["find", "insert"] } // Custom
privileges
  ],
  roles: [] // No inherited roles
})
```

- Assign this custom role to a user:

```
db.grantRolesToUser("appUser", ["customRole"]) // Grant customRole to appUser
```

Revoke Roles from Users

- To revoke specific roles from a user, use the revokeRolesFromUser() method:

db.revokeRolesFromUser("appUser", ["dbAdmin"]) // Remove dbAdmin role from appUser

Practical Activity 4.1.2: Creating users

Task:

1: Read the key readings 4.1.2

2: Go to the computer lab and by referring to the theoretical activity, 4.1.1 , create users and assign them privileges depending on responsibilities of Mongo Database

3: Present the steps to create user in MongoDB

4: Referring to the steps provided in tasks2 , create user in Mongo DB

5: Present your work to the trainer or classmates

6: Ask questions for clarification where necessary.

Key readings 4.1.2: Creating of users

MongoDB GUI is a NoSQL database that is extremely popular for its convenience and features. There is no SQL here, which means it is a mechanism for processing data patterned in tabular format and storing it in a database. It is faster in speed, and easy to scale. One of the parts of Mongo's functioning is creating and adding new users to the system. It is easy enough if you have purchased our new product — NoSQL Manager. With it, you can easily make up specific databases, and the user will have access to this unique database. You can likewise specify the access level for this client in the database. MongoDB contains a considerable number of roles. By creating a user using our console, you can assign them one or more functions, thereby regulating access to your database.

Understanding the user's database

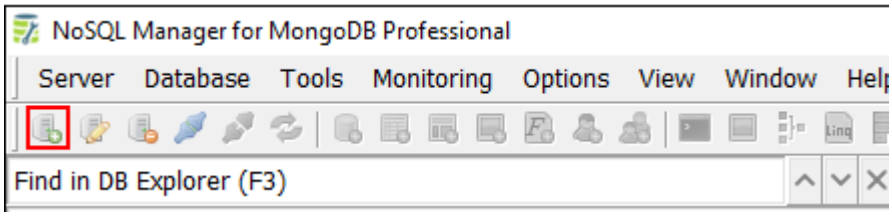
When you install a new instance of MongoDB, an admin database is automatically created. It is a particular database that provides functionality not available in regular databases. Some roles give users the authority to manage multiple databases, and these roles can only be created in the administrator's database. To create a user with the authority to manage all databases, you must add the user to the administrator's database.

When validating credentials, MongoDB will validate the account against the specified database and the admin one. It's easy to do this with the NoSQL Manager:

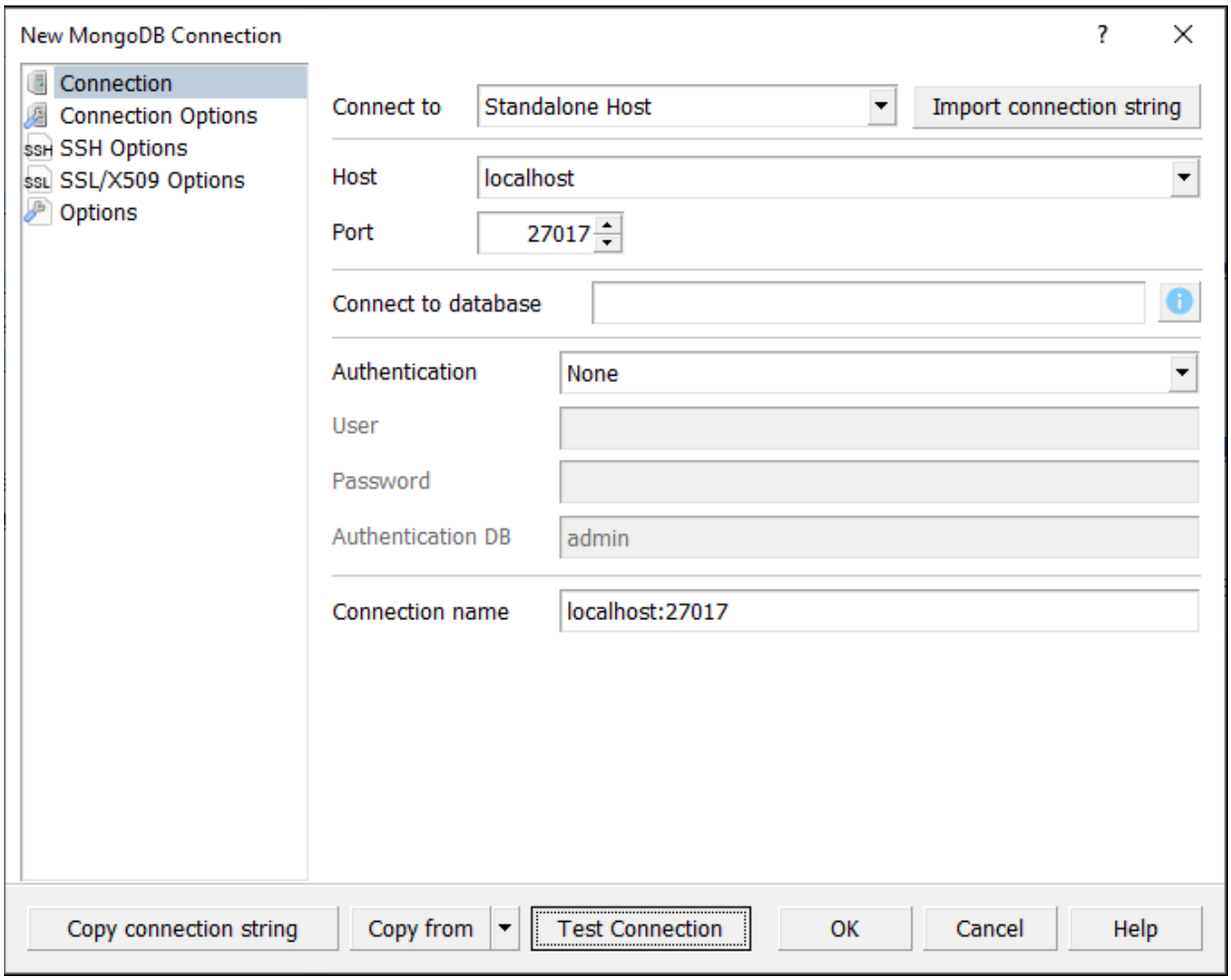
1.First, you need to create a database administrator in NoSQL Manager.

By default, access control (authorization) is disabled, so any person can connect to a just installed MongoDB server from allowed IP addresses. But do not worry, remote connections are not allowed by default also. In other words - if you have a fresh, just installed MongoDB instance, you can connect to it without authorization, but only from localhost. This means your client software (NoSQL Manager) should be started on the same computer where your MongoDB server is installed. Or you can use SSH tunneling feature. In this case your MongoDB instance recognizes your connection as established from localhost, but only in case if your SSH server and MongoDB server are located on the same host.

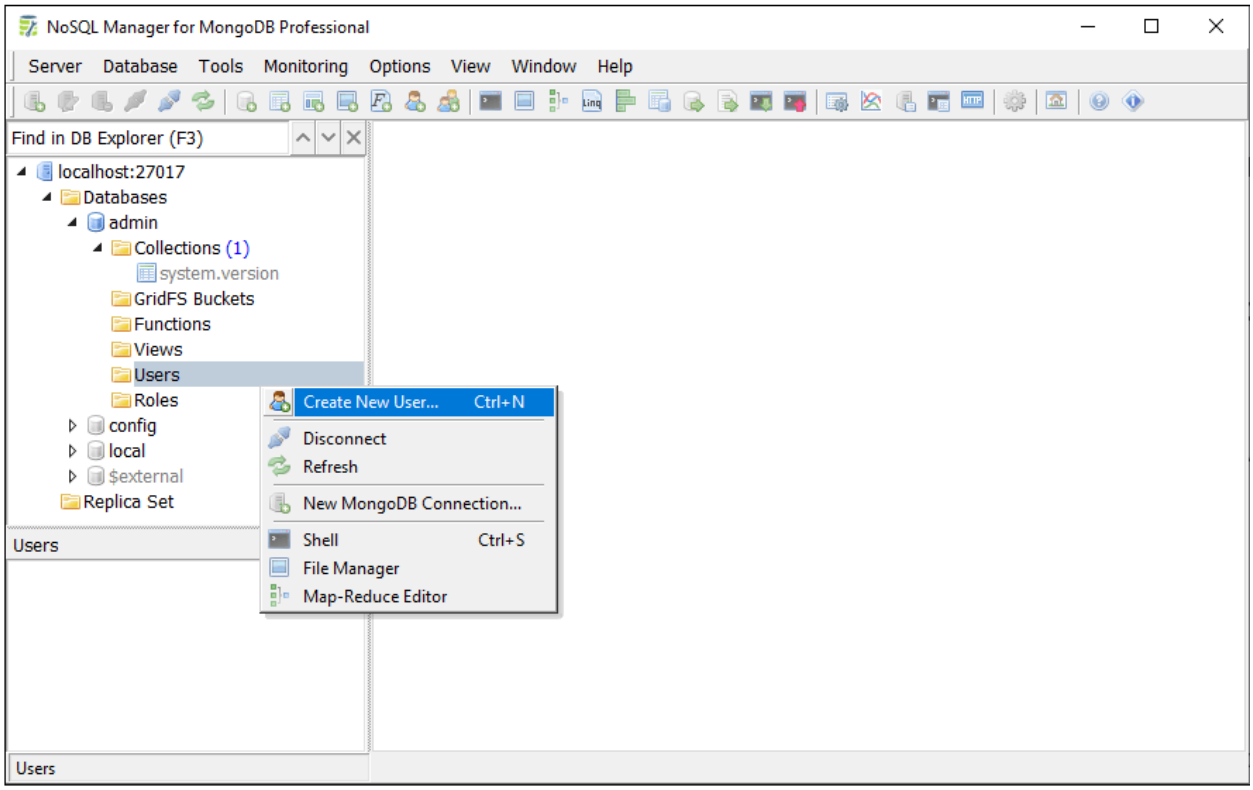
1.Open NoSQL Manager and click New MongoDB Connection button in the toolbar.



2. Next, specify your MongoDB host and port. Leave fields as-is if you are connecting to a local instance.



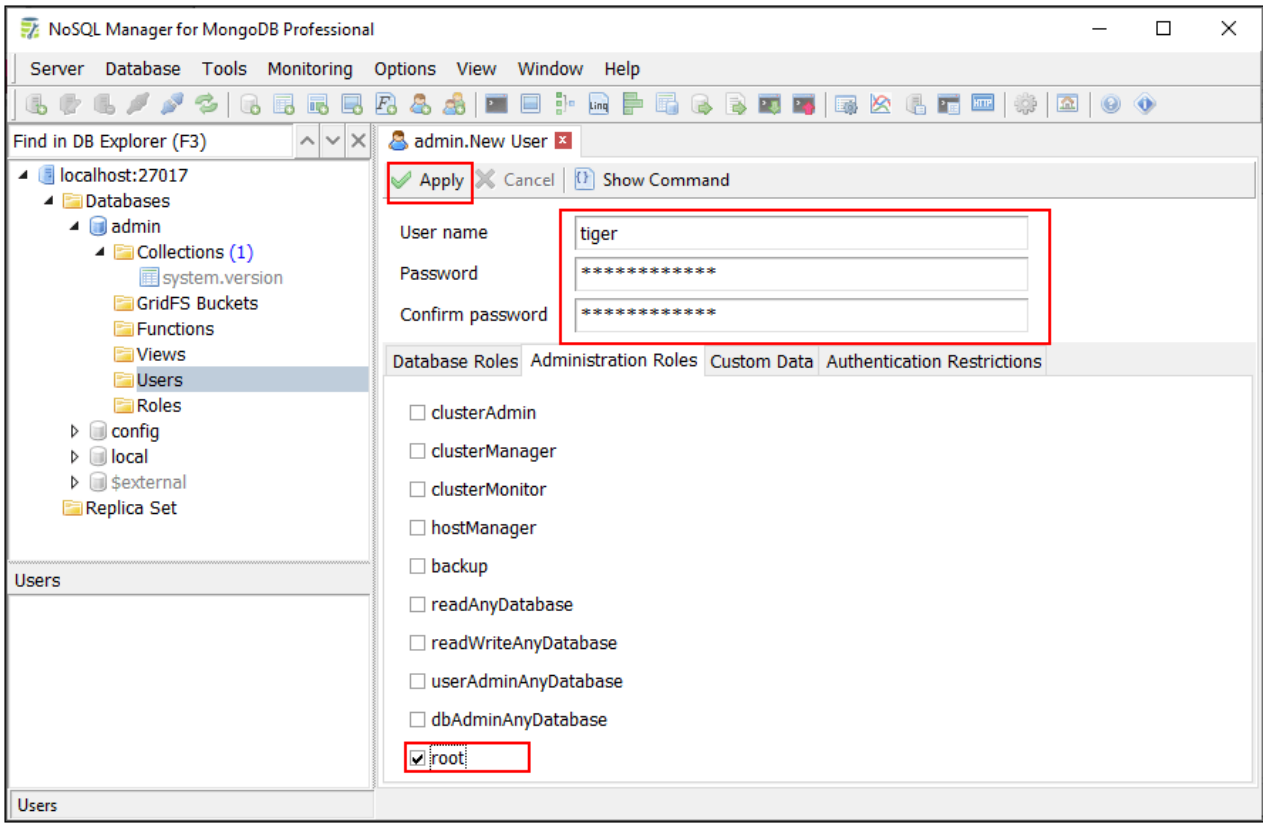
3. Test your connection with Test Connection button and click OK to save the connection.
4. Click double on your connection in DB Explorer, click double on the admin database, next click Main Menu|Database|Create New User... or click right on Users in DB Explorer and click Create New User in the context menu.



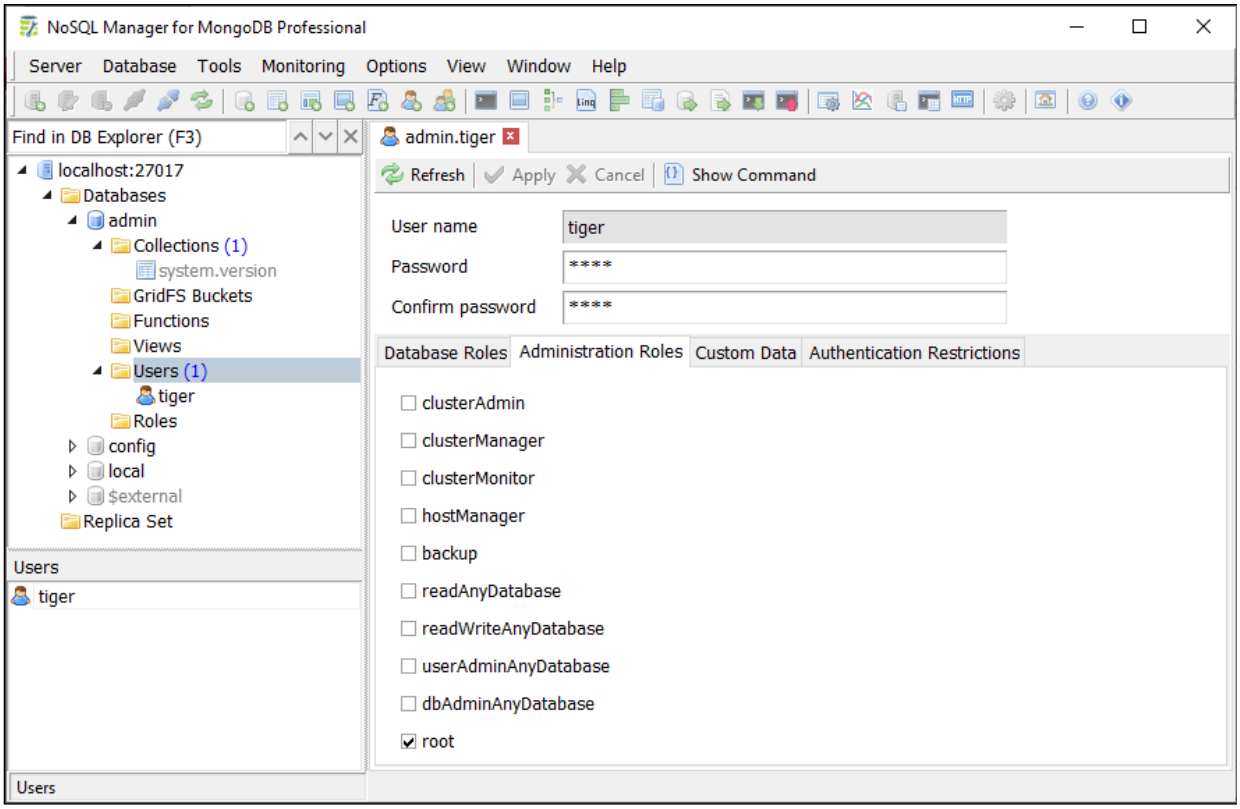
5. Specify the user name and password. For example we use the tiger name.

6. Go to Administration Roles tab and check on the root role.

7. Next click Apply button to create the user.



8. You have created the tiger user with root privileges. The root role is a combination of readWriteAnyDatabase, dbAdminAnyDatabase, userAdminAnyDatabase, clusterAdmin, restore and backup roles.



9. Disconnect the server before the next step.

2. Enable authentication for your MongoDB instance.

As you already know, access control (authorization) is disabled by default. This paragraph describes how to enable the authentication for an instance.

Stop your MongoDB instance and open the configuration file. The file is usually located at /etc/mongod.conf for Linux, <install directory>\bin\mongod.cfg for Windows. Detail information about the configuration files you can find at MongoDB web-site.

The minimal configuration file is:

```
storage:
  dbPath: /data/db
net:
  port: 27017
  bindIp: 127.0.0.1
```

You need to add the following options to this file.

```
security:
  authorization: enabled
```

setParameter:

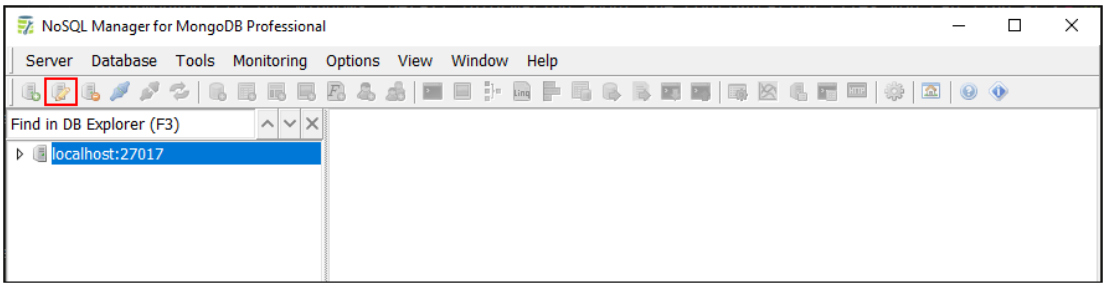
authenticationMechanisms: "SCRAM-SHA-256"

This means enable authentication using SCRAM-SHA-256 algorithm.

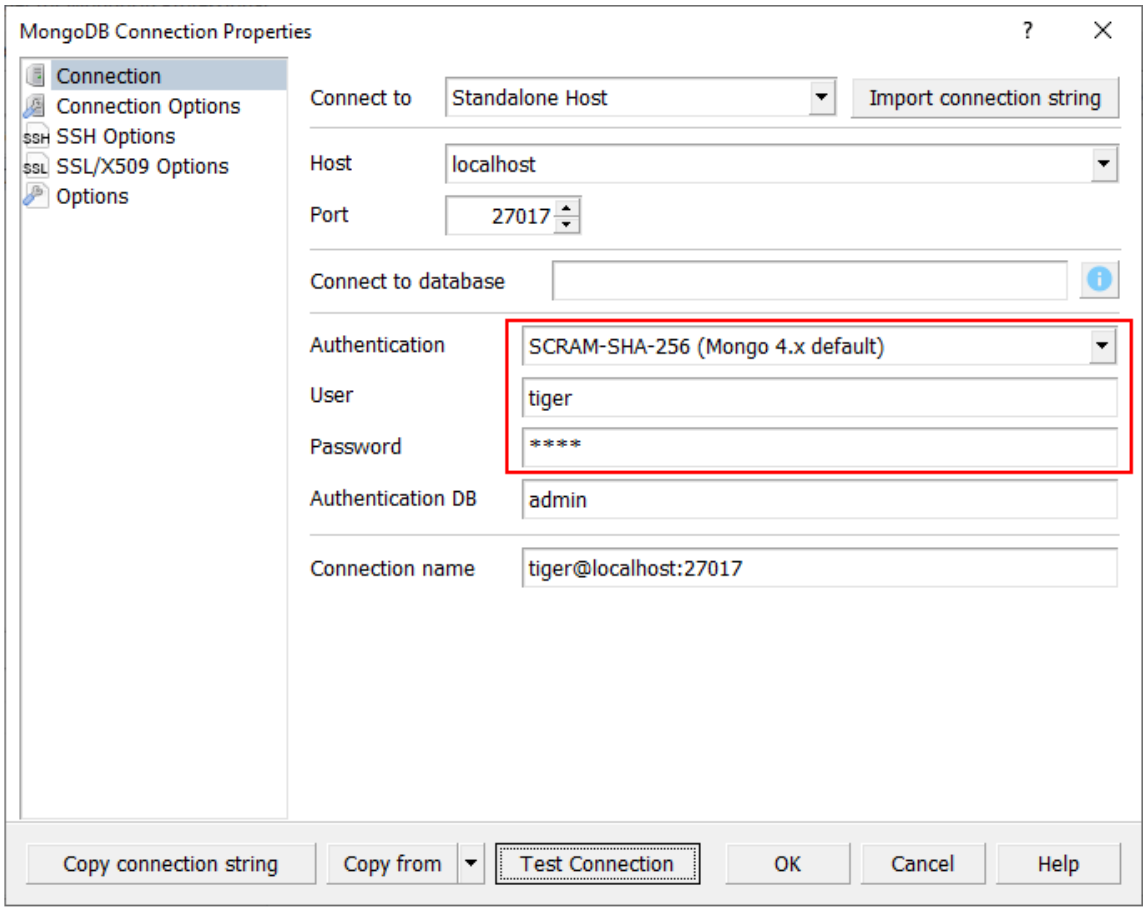
Restart your MongoDB instance.

3. Switch to the database administrator user in the NoSQL Manager.

Open NoSQL Manager, select your connection in DB Explorer and click Edit MongoDB Connection button in the toolbar.



Edit the Authentication, User and Password fields as described below and click OK to save the changes.

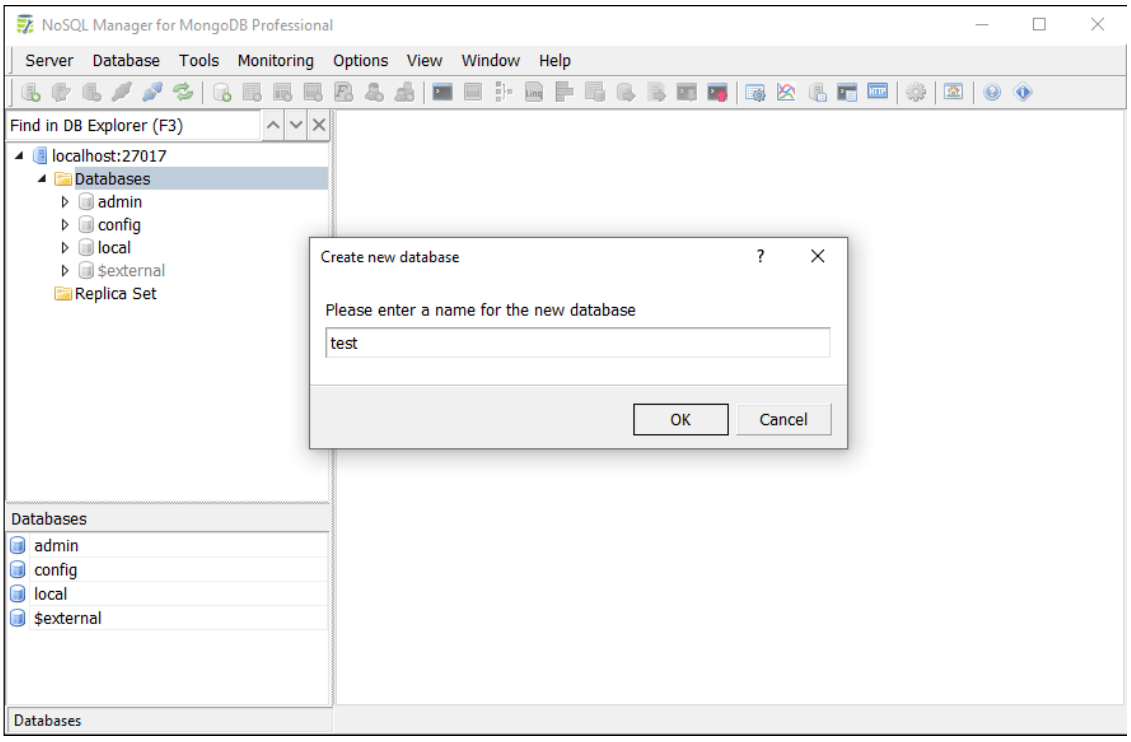


Connect to your instance in NoSQL Manager. Now you can add, edit and remove users and roles.

4. Create a user with specific privileges.

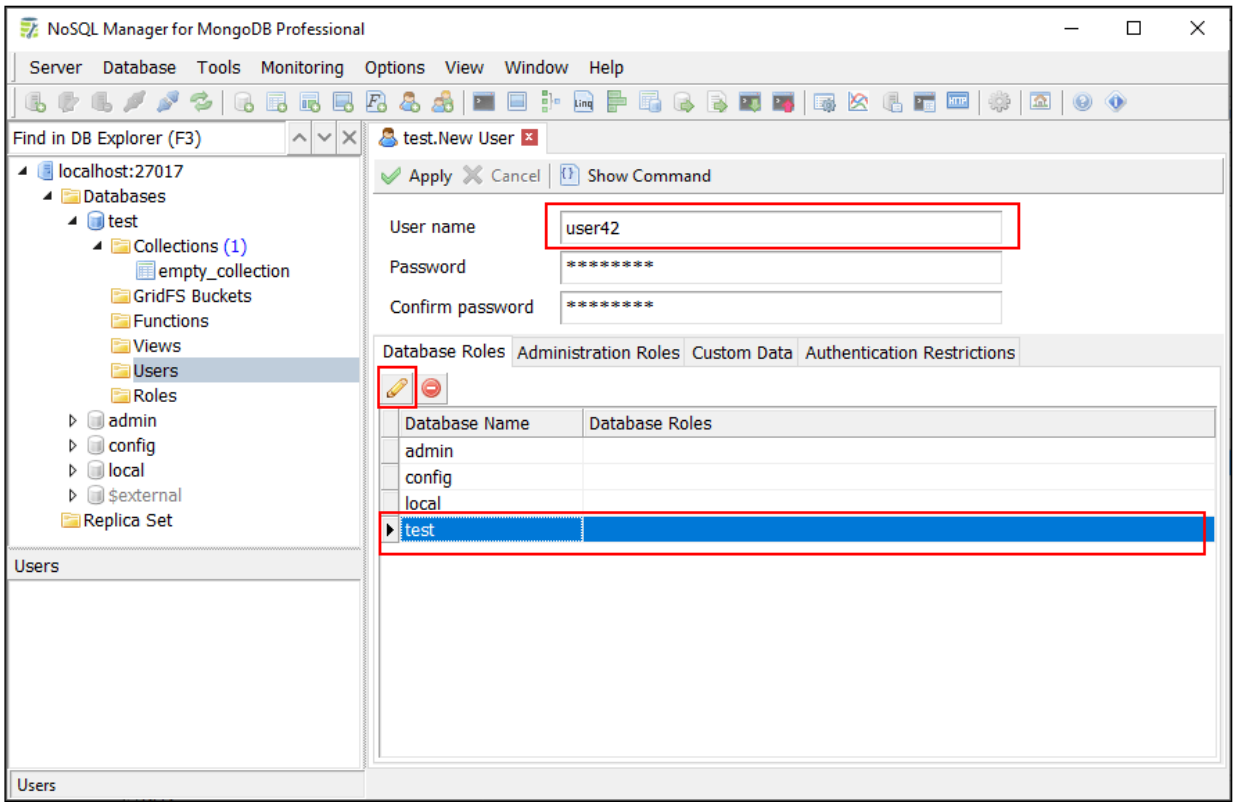
In this example we will create a limited user user42 that has read-only access to the test database only.

1. First, create the test database.

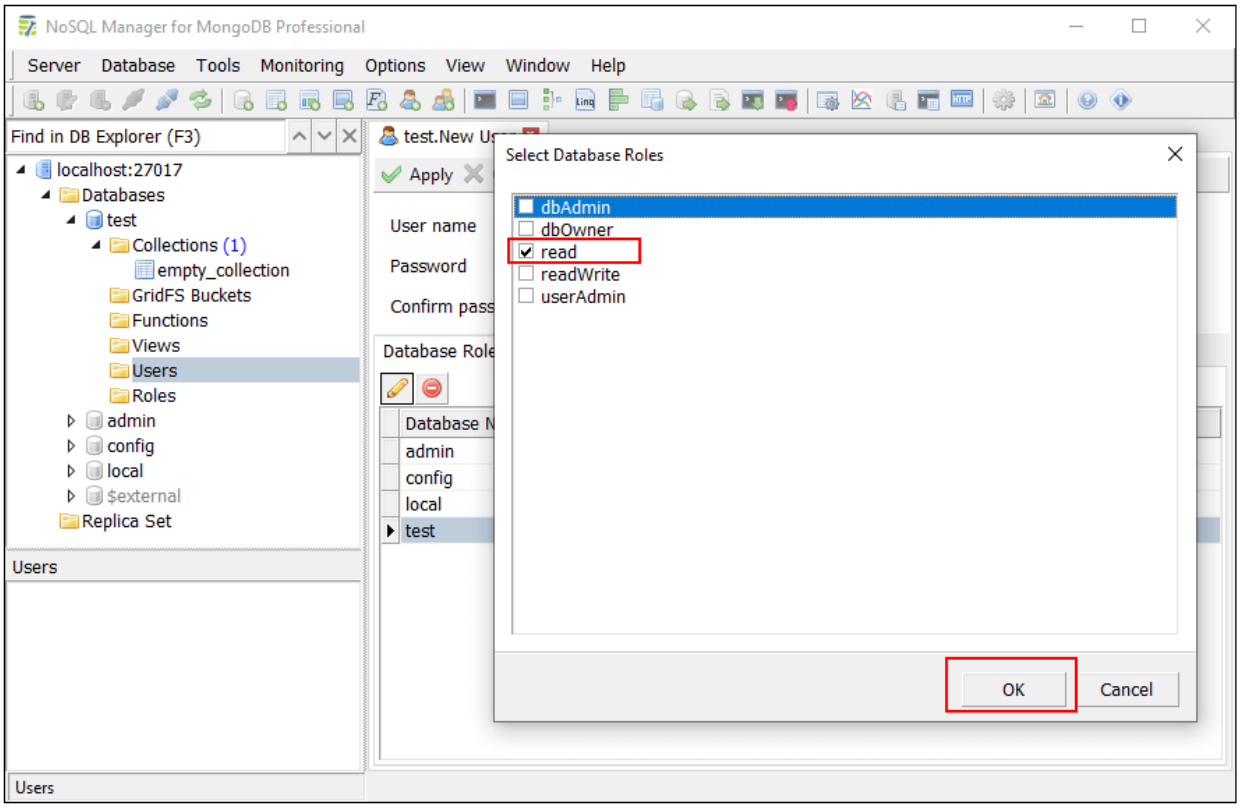


2.Next, connect to the test database and create a user.

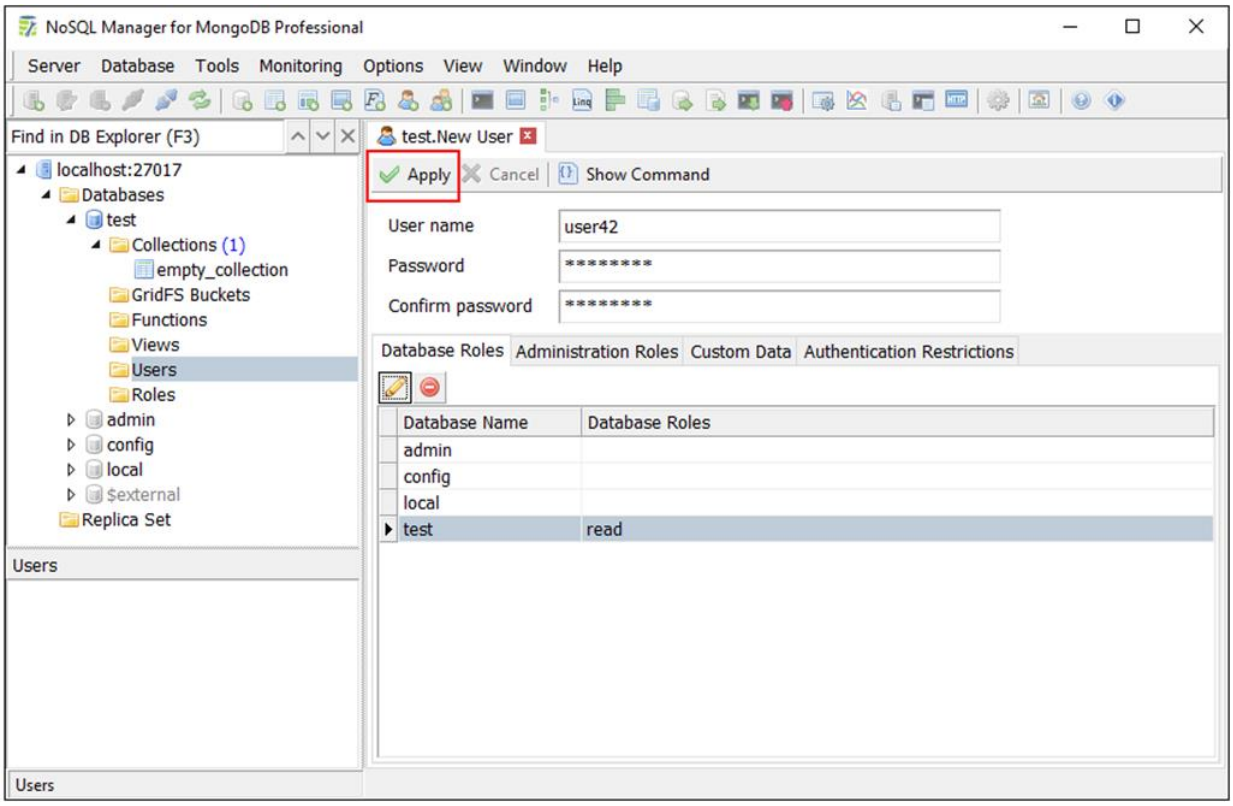
3.Specify the user name and password, select the test database on the Database Roles tab and click Edit Database Permission button.



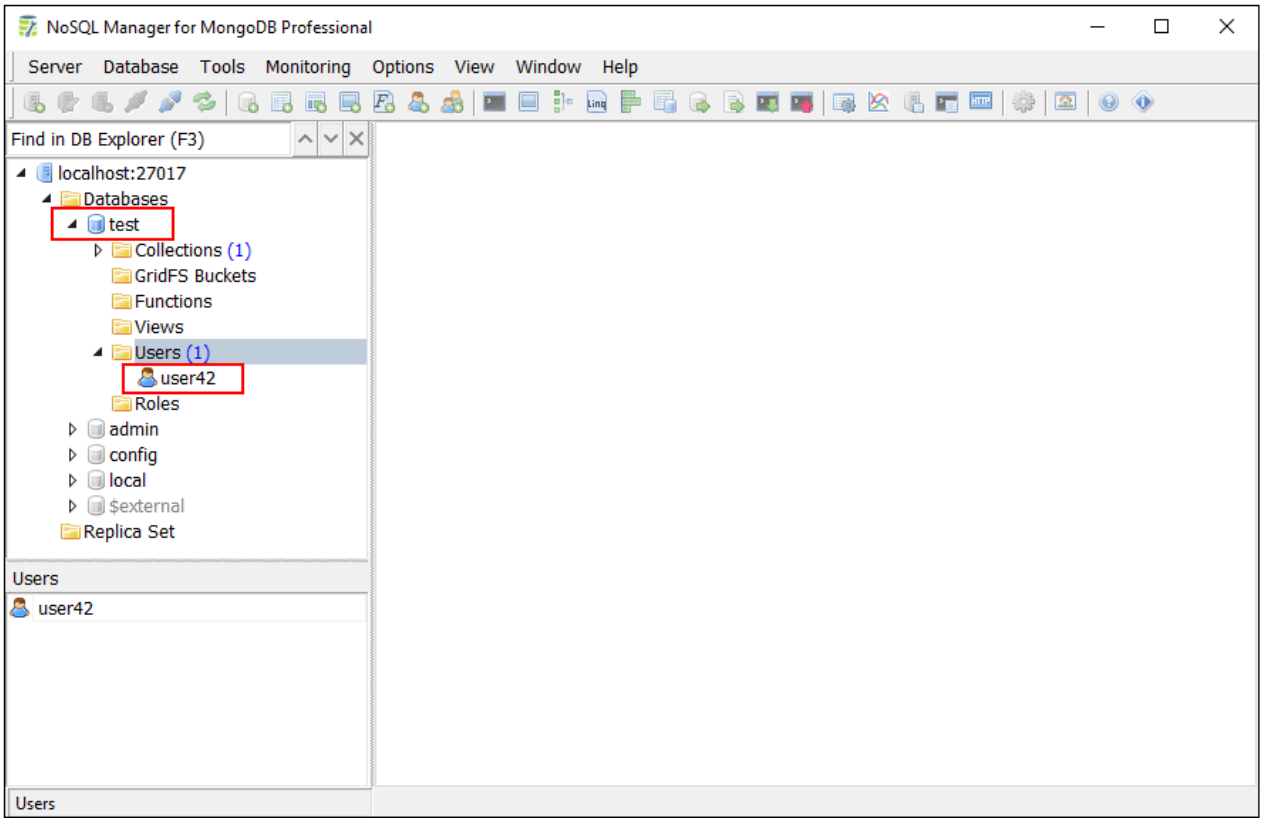
4.Check on the read role and click OK button.



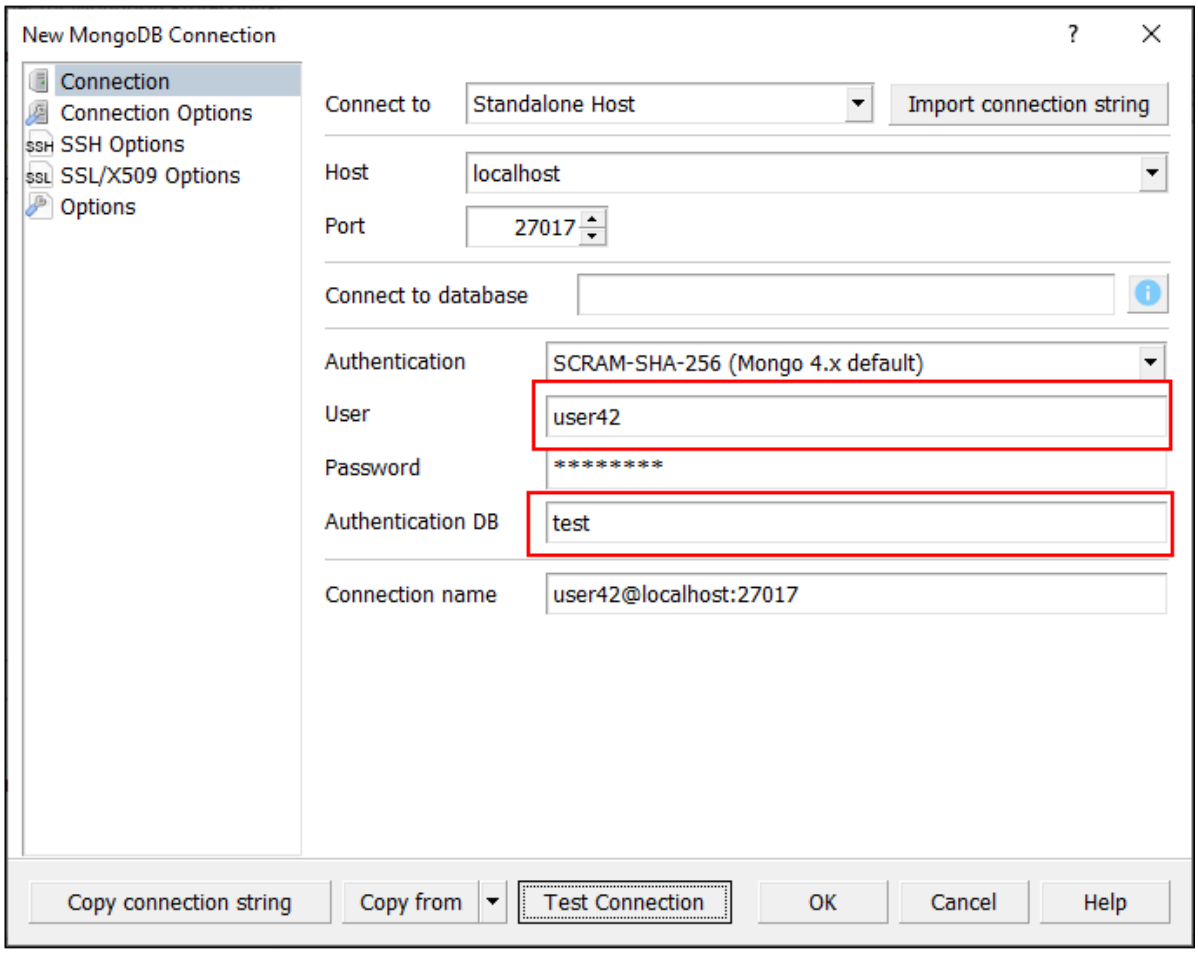
5. Click Apply to save user to the database.



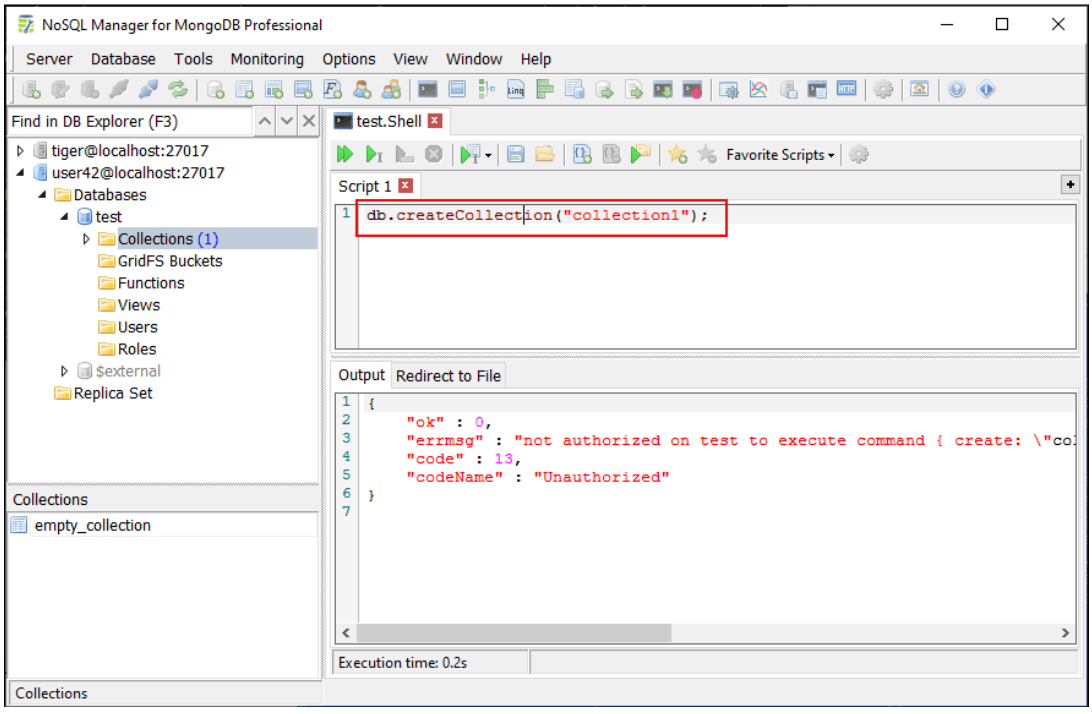
6. So, we have just created a limited user user42 in the test database.



7. Create a new MongoDB connection to test this user.



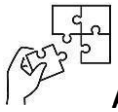
8. Next, connect to the MongoDB instance with the user42 user and try to execute any command that requires an extra privilege. Try to create a collection, for example.



Points to Remember

- Authentication: Use authentication to validate a user's identity and ensure that only trusted users can access the database.
- Encryption: Use encryption to provide high-level access control for users
- Auditing: Monitor and record user database actions, including those performed by administrators.

- **Steps for creating users in MongoDB are as follows:**
  - ✓ First, you need to create database administrator in NOSQL Manager
  - ✓ Enable authentication for your mongoDB instance.
  - ✓ Switch to the database administrator for user in the NoSQL Manager
  - ✓ Create user with specific privileges.



Application of learning 4.1.

Fin Serve, a financial services provider, wants to improve its database user management in MongoDB implementation to ensure data security and operational efficiency. You are assigned or tasked to create users, identify unique credentials to be used for each user and then assign the role-based access controls for each user.

Indicative content 4.2: Securing database

Duration: 3 hrs

Theoretical Activity: 4.2.1: Description of database security

Tasks:

1: Answer the following questions:

Define the following terms:

i. User authentication
ii. RBAC
iii. Auditing
iv. Describe types of data encryption

2: Write your findings on Papers or flipcharts

3: Present your findings to the trainer or classmates

4: Pay attention to the trainer's clarification and ask question where necessary.

5: Read the key readings 4.2.1

Key readings 4.2.1.: Description of database security

✓ **Mongo database security**

Securing a MongoDB database is crucial for protecting sensitive information and maintaining the integrity of your data. Below is a step-by-step guide covering essential security practices, including enabling access control, configuring role-based access control, data encryption, auditing, and backup strategies.

Enable Access Control and Enforce Authentication

Access control ensures that only authorized users can access your database. MongoDB provides various authentication mechanisms, including SCRAM, x.509, and LDAP.

✓ **Steps to Enable Access Control**:

- **Edit the MongoDB Configuration File** (usually mongod.conf):

  - Find the security section and add or uncomment the following line to enable authentication:

```
security:
  authorization: "enabled"
```

- **Restart MongoDB** to apply the changes:

  sudo systemctl restart mongod

- **Create an Admin User** (if not already created):

```
use admin;
db.createUser({
  user: "admin",
  pwd: "secureAdminPassword",
  roles: [{ role: "userAdminAnyDatabase", db: "admin" }]
});
```

- **Connect with Authentication**:

  mongosh -u admin -p secureAdminPassword --authenticationDatabase admin

✓ **Configure Role-Based Access Control (RBAC)**

Role-based access control allows you to define specific permissions for users based on their roles.

❖ **Steps to Configure RBAC**:

- **Create Users with Specific Roles**:

```
use FitTrack;

db.createUser({
  user: "appUser",
  pwd: "securepassword",
  roles: [
    { role: "readWrite", db: "FitTrack" },
    { role: "dbAdmin", db: "FitTrack" }
  ]
});
```

- **Modify User Roles** as needed using updateUser():

```
db.updateUser("appUser", {
  $set: {
    roles: [{ role: "read", db: "FitTrack" }]
  }
});
```

✓ **Data Encryption and Protect Data**

Data encryption helps protect sensitive information both at rest and in transit.

Steps for Data Encryption:

- **Enable TLS/SSL for Data in Transit**:
  - Obtain a TLS certificate and configure it in the mongod.conf:

```
net:
  ssl:
    mode: requireSSL
    PEMKeyFile: /path/to/your/certificate.pem
```

- **Enable Encrypted Storage Engine for Data at Rest**:
  - Use the WiredTiger storage engine and enable encryption in the mongod.conf:

```
storage:
  wiredTiger:
    engineConfig:
      encryption: "enabled"
```

- **Restart MongoDB** to apply the encryption settings.

✓ **Audit System Activity**

Auditing helps track access and modifications to the database, providing visibility into user actions.

Steps to Enable Auditing:

- **Edit the MongoDB Configuration File**:
  - Enable auditing in the mongod.conf:

```
auditLog:
  destination: file
  path: "/var/log/mongodb/auditLog.json"
  format: JSON
```

- **Restart MongoDB** to start logging audit events.
- **Review Audit Logs**:
  - Use a JSON viewer or command-line tools to analyze the logs:

```
cat /var/log/mongodb/auditLog.json | jq .
```

✓ **Perform Backup and Disaster Recovery**

Implementing a robust backup and disaster recovery plan ensures that you can recover data in case of failure.

Steps for Backup and Recovery:

- **Use mongodump for Backups**:

  mongodump --db FitTrack --out /path/to/backup/directory

- **Restore from Backup using mongorestore**:

  mongorestore --db FitTrack /path/to/backup/directory/FitTrack

- **Automate Backups**:
  - Use cron jobs or scheduling tools to automate the backup process. For example, to back up daily at midnight:

  0 0 * * * /usr/bin/mongodump --db FitTrack --out /path/to/backup/directory/$(date +\%F)

- **Test Disaster Recovery Procedures**:
  - Regularly test your backup and restore procedures to ensure they work as expected and data can be recovered quickly.

Practical Activity 4.2.2: Performing NoSQL backup

Task:

1: Read the key readings 4.2.2

2: Go to the computer lab to perform dataSafe Corp implemented as robust backup strategy utilizing MongoDB's native tools, geographical distribution, and oplog-based replication to safeguard unstructured data and ensure business continuity

3: Present the steps to perform backup in NoSQL MongoDB

4: Referring to the steps provided in tasks2, encrypt database in NoSQL in Mongo DB

5: Present your work to the trainer or classmates

6: Ask questions for clarification where necessary.

Key readings 4.2.3: Performing NoSQL backup

✓ **Perform backup and recovery**

Performing backup and disaster recovery in NoSQL involves creating periodic copies of your NoSQL database data. typically through snapshotting or export/import methods, to a separate storage location, allowing you to restore the database to a previous state in case of hardware failure, data corruption, or other disruptions, ensuring business continuity by minimizing downtime and data loss; key aspects include choosing the appropriate backup strategy based on your data volume, access requirements, and recovery time objectives (RTO), and setting up automated backup schedules with proper retention policies to maintain multiple versions of your data.

Key points about NoSQL backup and disaster recovery:

- Backup methods: Snapshot backups: Creating a point-in-time copy of the entire database, often considered the most reliable method for consistency.
- Export/Import backups: Exporting data to a file format and then importing it to a new database if needed.
- File system backups: Backing up the database files directly from the file system, but may not capture data consistency.
- Considerations for NoSQL backups: Data distribution: NoSQL databases often distribute data across multiple nodes, requiring a strategy to capture all data consistently.
- Replication: Leverage built-in replication features to create redundant data copies for improved availability.
- Backup frequency and retention: Determine how often to back up data and how long to retain backups based on your recovery point objective (RPO).
- Disaster recovery strategies: Failover clusters: Setting up a secondary cluster in a different location to quickly switch to in case of a primary site failure.
- Warm standby: Maintaining a partially active replica of the database that can be quickly brought online in an emergency.
- Cloud-based backups: Utilizing cloud storage services for offsite backups and disaster recovery capabilities.

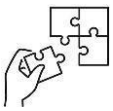
Steps for performing a NoSQL backup:

- **Choose a backup method:** Select the most suitable backup method based on your database type (MongoDB, Cassandra, etc.) and desired data consistency.
- **Configure backup schedule:** Set up automated backups at regular intervals to capture changes in your data.
- **Create snapshots or exports:** Use the provided database tools to create snapshots or export data to a backup location.
- **Monitor backup process:** Regularly monitor backup jobs to ensure successful completion and address any issues.

- **Test restore process:** Periodically perform test restores to verify data integrity and identify potential problems with your backup strategy**.**

Points to Remember

- While describing database security, take into consideration the following elements:
  - ✓ Choose the backup methods
  - ✓ Configure backup schedule
  - ✓ Create snapshots or exports
  - ✓ Monitor backup process
  - ✓ Test restore process
- Steps for performing backup are:
  - ✓ Choose the backup methods
  - ✓ Configure backup schedule
  - ✓ Create snapshots or exports
  - ✓ Monitor backup process
  - ✓ Test restore process



Application of learning 4.2.

ecBank Tech is strengthening its database security by implementing multi-factor authentication, role-based access control, data encryption, an audit system, and a robust backup and disaster recovery plan. You are tasked to help back up MongoDB of SecBank

Indicative content 4.3: Deployment of database

Duration: 4 hrs

Theoretical Activity 4.3.1: Description of database deployment

Tasks:

1: Answer the following questions:

i. What do you understand by deployment?
ii. State deployment options
iii. state characteristic of deployment options used in NoSQL Database
iv. Identify MongoDB cluster architectures

2: Write your findings on papers or flipcharts

3: Present your findings to the trainer or classmates

4: Pay attention to the trainer's clarification and ask questions where necessary.

5: Read the key readings 4.3.1

Key readings 4.3.1.: Description of database deployment

✓ Applying deployment Options

When deploying a NoSQL database, you can choose between "On-Premises" where the database runs on your own physical hardware, "Cloud" where it is hosted on a remote cloud provider's infrastructure, or "Hybrid" which combines elements of both, allowing you to leverage the benefits of both on-premises control and cloud scalability depending on your specific needs; all while maintaining the flexible, non-relational data structure characteristic of NoSQL databases.

On-Premises:

- Control: Provides full control over hardware, security, and network configuration, ideal for highly sensitive data or strict compliance requirements.
- Customization: Ability to tailor the database environment to specific needs by managing hardware upgrades and software installations.

- Cost considerations: Requires upfront investment in hardware and maintenance staff, potentially higher operational costs compared to cloud.

Cloud:

- Scalability: Easily scale up or down database capacity on demand based on application usage, without the need for manual hardware management.
- Cost-efficiency: Pay-as-you-go model can reduce costs, especially for applications with fluctuating data demands.
- High availability: Cloud providers offer robust disaster recovery features and redundancy across multiple data centers.

Hybrid:

- Data locality: Store frequently accessed data on-premises for faster access while utilizing the cloud for large data storage or off-peak processing.
- Data migration flexibility: Gradually move data to the cloud while maintaining on-premises access to critical information.
- Cost optimization: Leverage the benefits of both cloud and on-premises infrastructure to optimize costs based on specific application needs.

Considerations when Choosing a Deployment Option:

- **Data sensitivity: f**or extremely sensitive data, on-premises may be preferred due to greater control over security.
- **Application requirements:** Consider the expected data volume, scalability needs, and performance demands of your application.
- **Budget constraints:** Evaluate the upfront costs of hardware versus the pay-as-you-go cloud model.
- **Examples of NoSQL Databases suitable for various deployment options:**
- **On-Premises:** MongoDB, Couchbase Server, Cassandra
- **Cloud:** Amazon DynamoDB, Azure Cosmos DB, Google Cloud Spanner
- **Hybrid:** A combination of on-premises MongoDB with cloud-based data storage on AWS S3

- ✓ Identify mongoDB Cluster Architectures

MongoDB Cluster Architectures

MongoDB offers several cluster architectures to accommodate different scalability and availability requirements.

Three main types:

Single-Node Cluster

Description: A single MongoDB instance running on a single server.

- Advantages:
- Simple to set up and manage.
- Suitable for small-scale applications with low data volumes.
- Disadvantages:
- Limited scalability and availability.
- Single point of failure.

Replica Set

Description: A group of MongoDB instances that maintain a consistent replica of the same dataset.

- **Advantages:**
- Improved availability through automatic failover.
- Data redundancy for disaster recovery.
- Read scaling through secondary members.
- **Disadvantages:**
- Increased complexity compared to a single-node cluster.
- Limited write scalability.
- **Components:**
- Primary: The instance that handles write operations and acts as the authoritative source of data.
- Secondary: Read-only replicas that maintain consistency with the primary.
- Arbiter: Non-voting members used for tie-breaking during elections.
- **Sharded Cluster**

Description: A distributed system that horizontally scales MongoDB across multiple servers by partitioning data across shards.

- **Advantages:**

- Exceptional scalability for handling large datasets.
- Improved read and write performance.
- High availability through redundancy across shards.

- **Disadvantages:**

- Increased complexity and management overhead.
- Requires careful sharding key design for optimal performance.

- Components:

- Shard: A standalone MongoDB instance responsible for storing a subset of the data.
- Config Server: Stores configuration information about the sharded cluster, including shard assignments and routing rules.
- Mongos: Query routers that act as a single entry point for client applications.

Choosing the Right Architecture

- **The optimal cluster architecture depends on factors such as:**

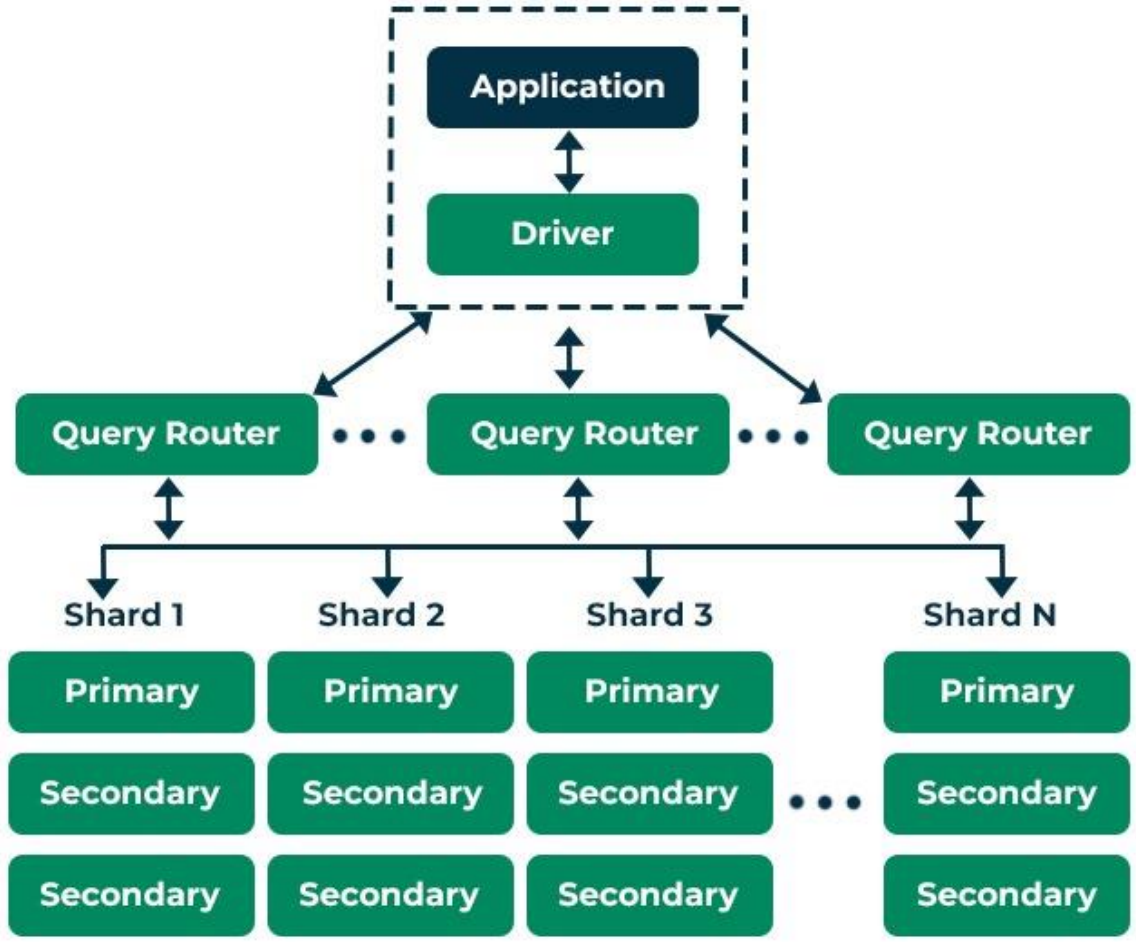
- Data Volume: The amount of data to be stored.
- Read and Write Patterns: The expected frequency and intensity of read and write operations.
- Availability Requirements: The need for high availability and disaster recovery.
- Scalability Needs: The potential for future growth and scalability.
- Complexity Tolerance: The organization's ability to manage a more complex system.

✓ **Scaling MongoDB with Sharding**

Sharding is a horizontal scaling technique in MongoDB that allows for automatic scaling of a database across multiple nodes and regions. Sharding is a way to distribute a large database into smaller pieces, called shards, and spread them across multiple machines.

Sharding can help with:

Scalability



Sharding allows for near-limitless scaling to handle large data sets and intense workloads.

Performance

Sharding allows read and write operations to be parallelized, which improves overall system performance.

High availability

Sharding, combined with replication, ensures that data is redundant, providing high availability and fault tolerance.

Some disadvantages of sharding include:

- Additional complexity: Sharding can increase the complexity of infrastructure and maintenance.
- Risk of duplicated or lost data: Sharding can introduce a risk of duplicated or lost data.

- Additional system traffic and storage requirements: Sharding can increase system traffic and storage requirements.

Introducing MongoDB Atlas

MongoDB Atlas is a Database as a Service (DbaaS), provided by the team behind MongoDB. It is a fully automated service with minimal to no configuration. Additionally, you have an option to deploy MongoDB instances in the any of the top three cloud providers, which are AWS, Azure, or Google cloud. It is an easy-to-use cloud-based service, which was released in 2016 and has been battle tested since. It is used and loved by both start-ups and many well-established enterprises like Invision, Ebay, Adobe, and Google.

Feature Rich Deployment

Although MongoDB Atlas is fully automated, it provides a very feature rich deployment. The moment we create a MongoDB instance, the Built-in replication kicks and our data is now stored at multiple locations. It is always available, even when the master is down.

- It provides us Automated Security features, through which we can keep an eye on who is using our data and keep the bad actors out.
- It provides a good option for automatic backup and recovery. Even if our data is corrupted, we have a dependable recovery option.
- Through the dashboard, we get a lot of information using which, we can monitor everything and decide when to upgrade our plan.
- Getting Started with Atlas

Practical Activity 4.3.2: Deploying NoSQL database

Task:

1: Read the key readings 4.3.2

2: Go to the computer lab to deploy NoSQL mongoDB with deployment options, mongoDB cluster architectures and sharding.

3: Present the steps to deploy NoSQL MongoDB

4: Referring to the steps provided in tasks 2, deploy NoSQL MongoDB

5: Present your work to the trainer or classmates.

6: Ask questions for clarification where necessary.

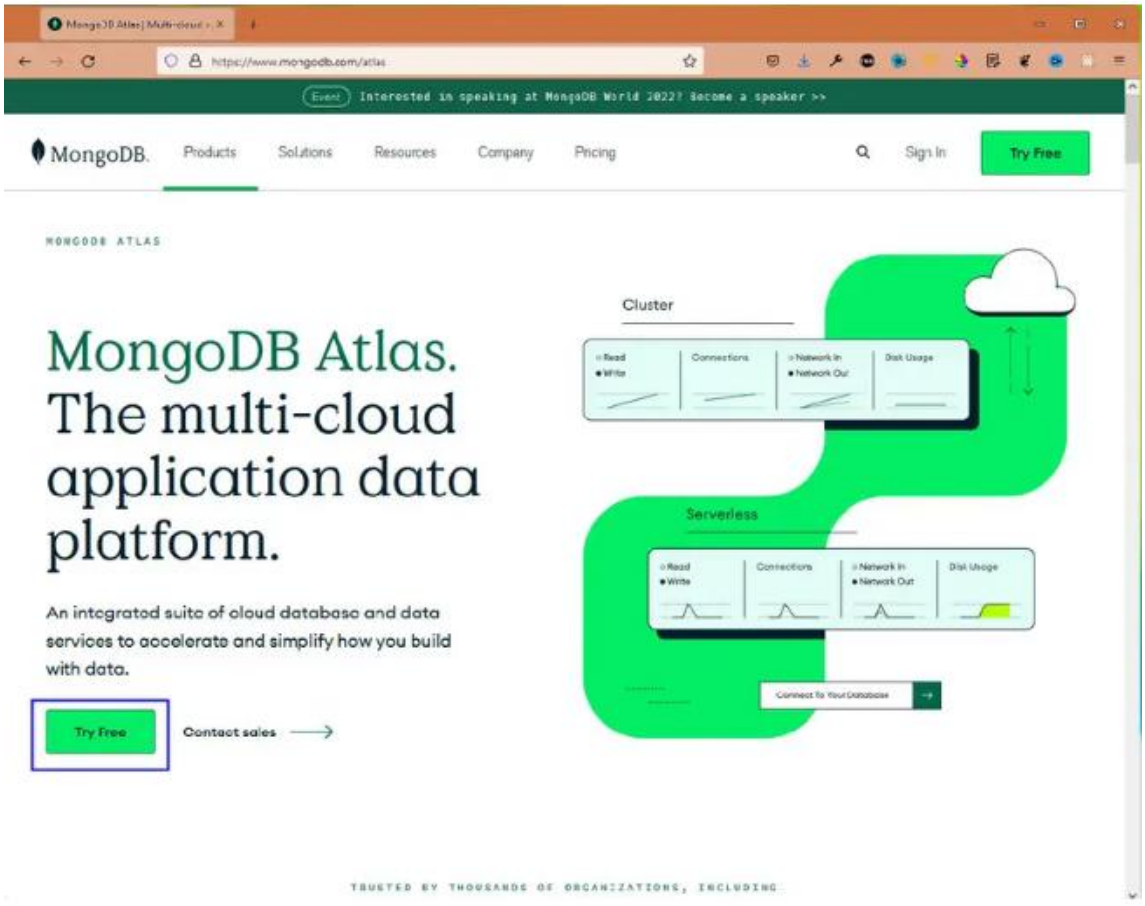
Key readings 4.3.2: Deploying NoSQL mongoDB

✓ **Deploying MongoDB**

- MongoDB is completely open -source and free to use, but for deployment, we generally need to take the paid route. We can also download the community edition of MongoDB locally and use it through the command line or the nice graphical interface of MongoDB Compass.

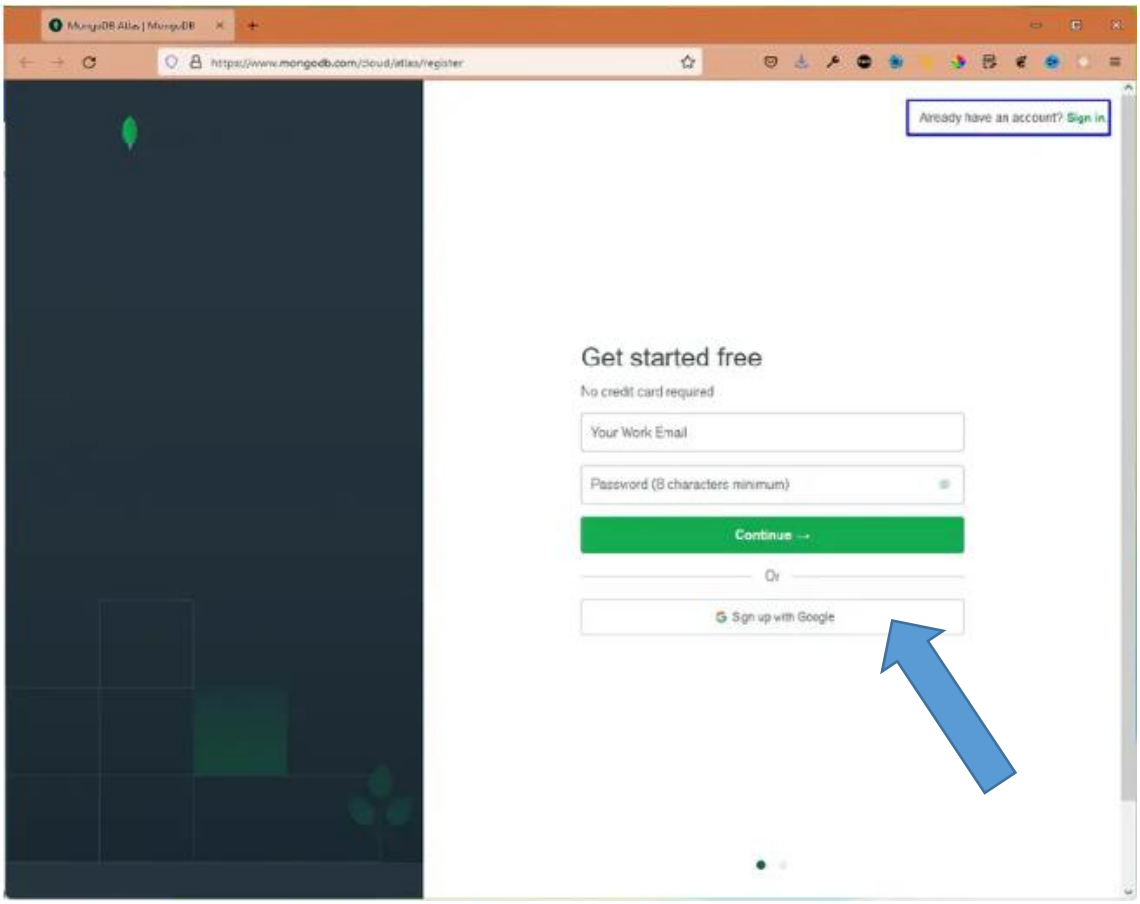
- For deployment, we need a Linux based server. We can either use our own server or deploy it in any of the available, professionally managed cloud services. Three popular options are to deploy in a linode server, Heroku, or AWS.

Getting Started with Atlas

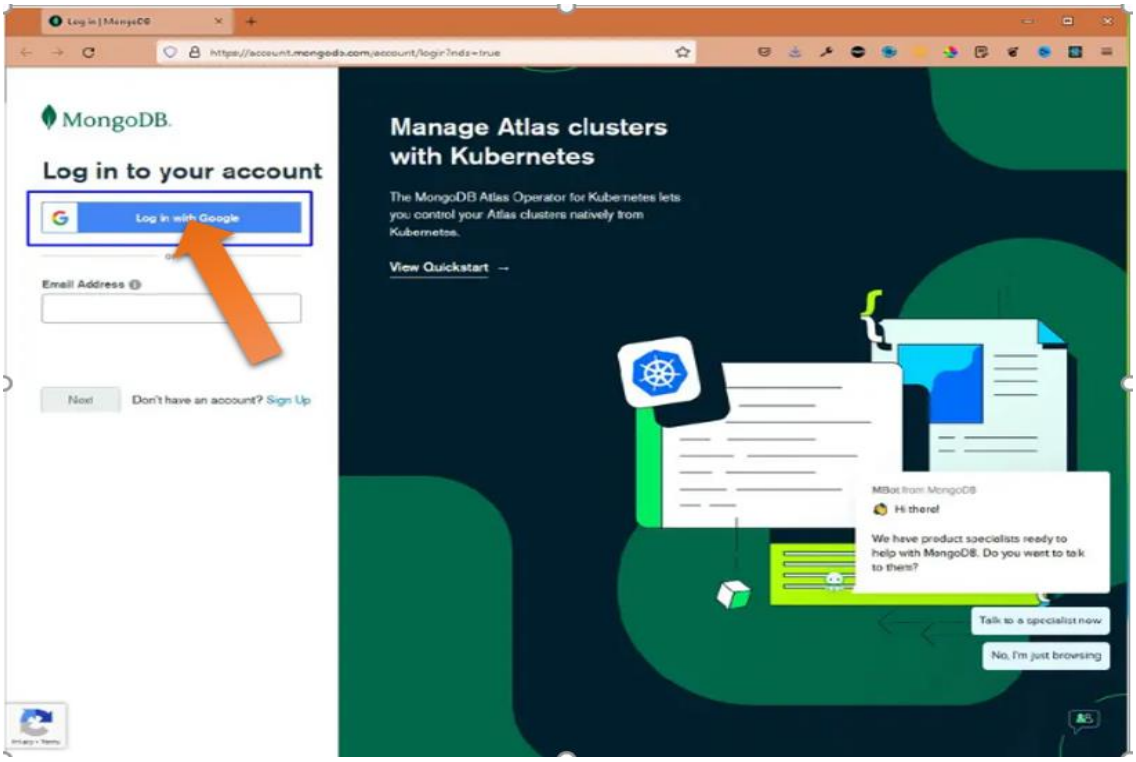


Sign up or Sign in

In the next page, it will ask you to Sign up or Sign in. You can also use your google account to do so.

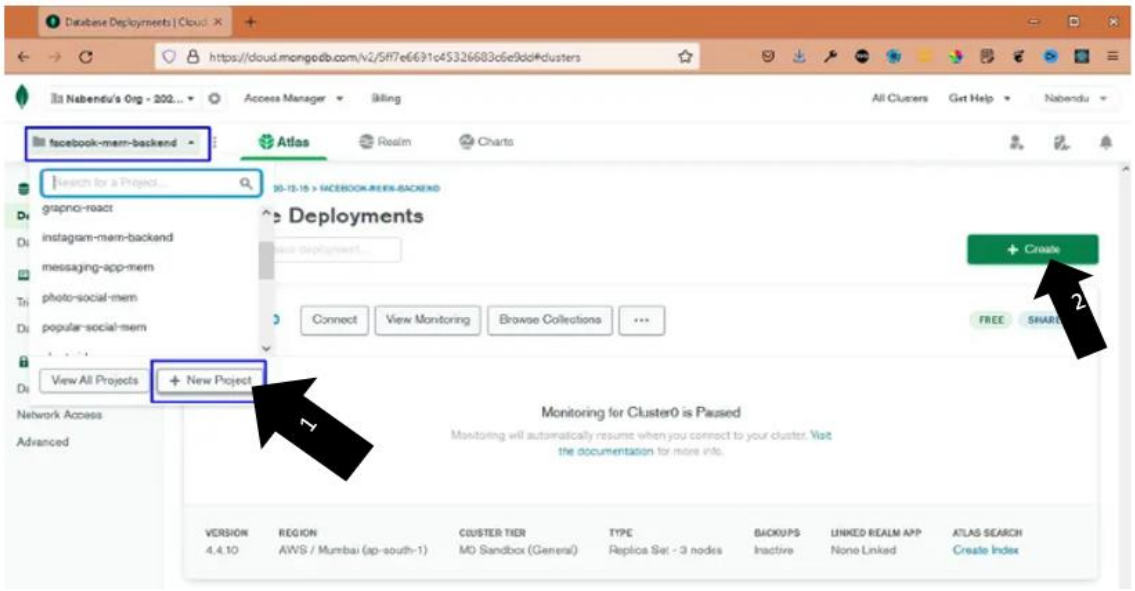


Since I already have an account, I clicked on the 'Sign in' option and the following page came up.



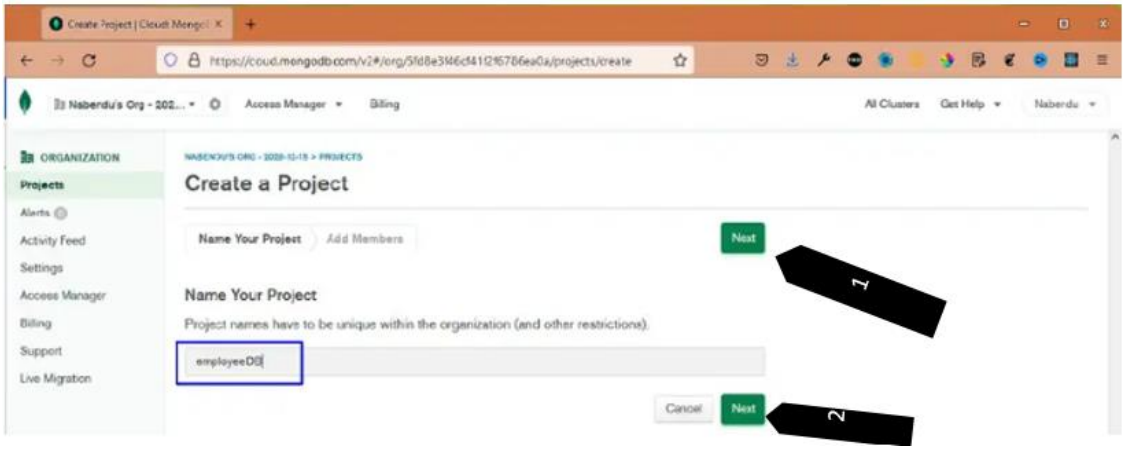
Create New Project

If you already have a project on MongoDB Atlas, you will be taken to the project on which you last worked. Here, you need to click on the project, and then in the pop-up, click on New Project.



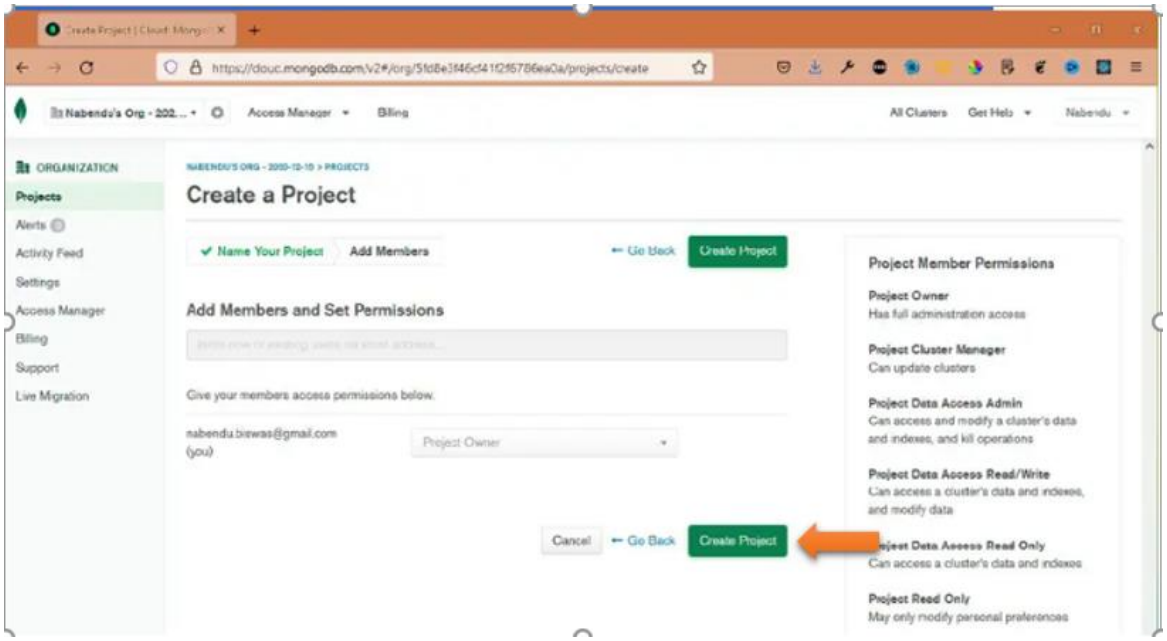
Accessing database

Then, it will ask us to give the project a name. I have given the name ‘employees’.

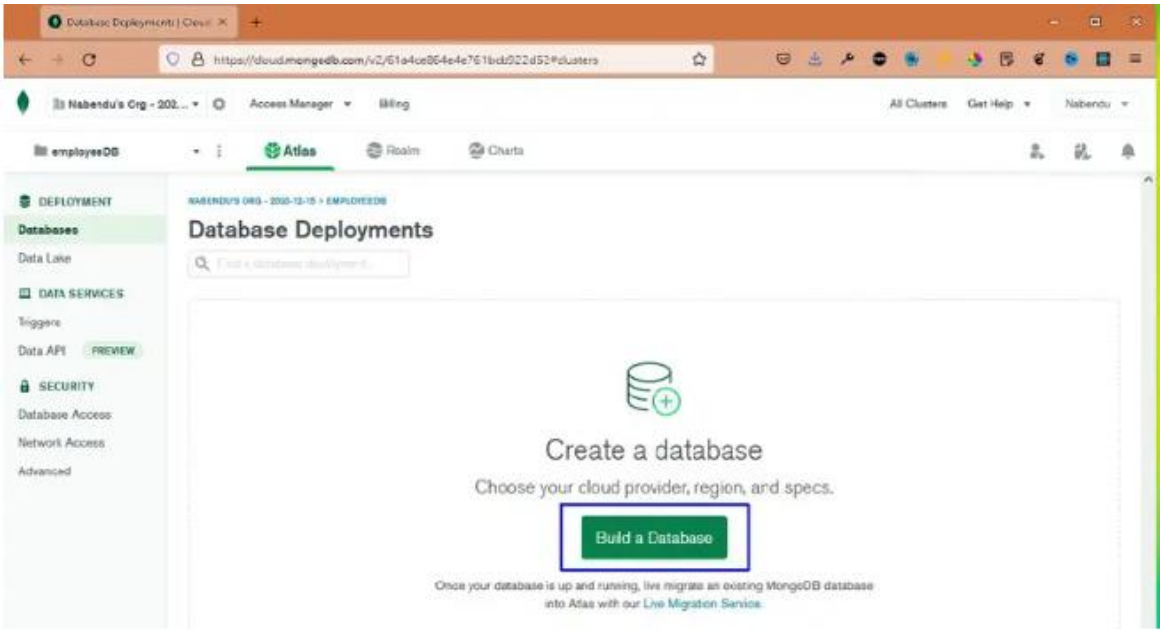


Deployment options

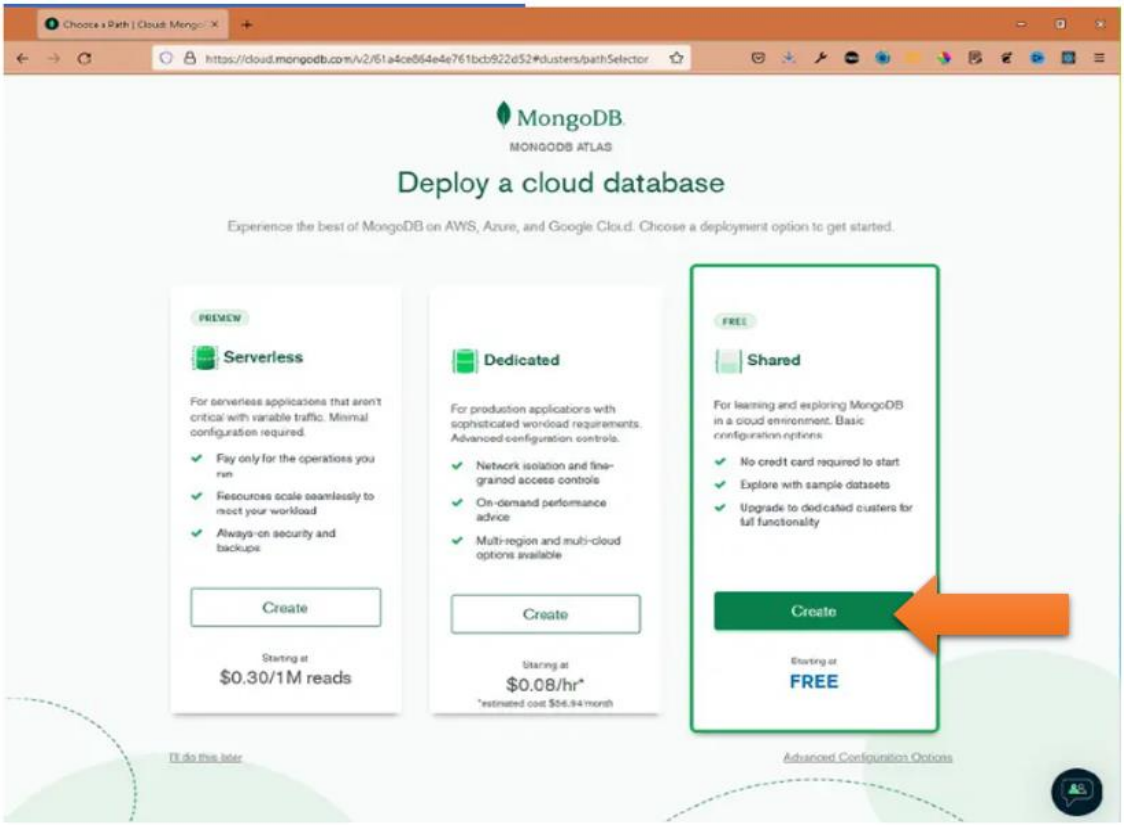
In the next screen, it will ask you to give access to members. I have given access to the existing users. After that, you need to click on Create Project button.



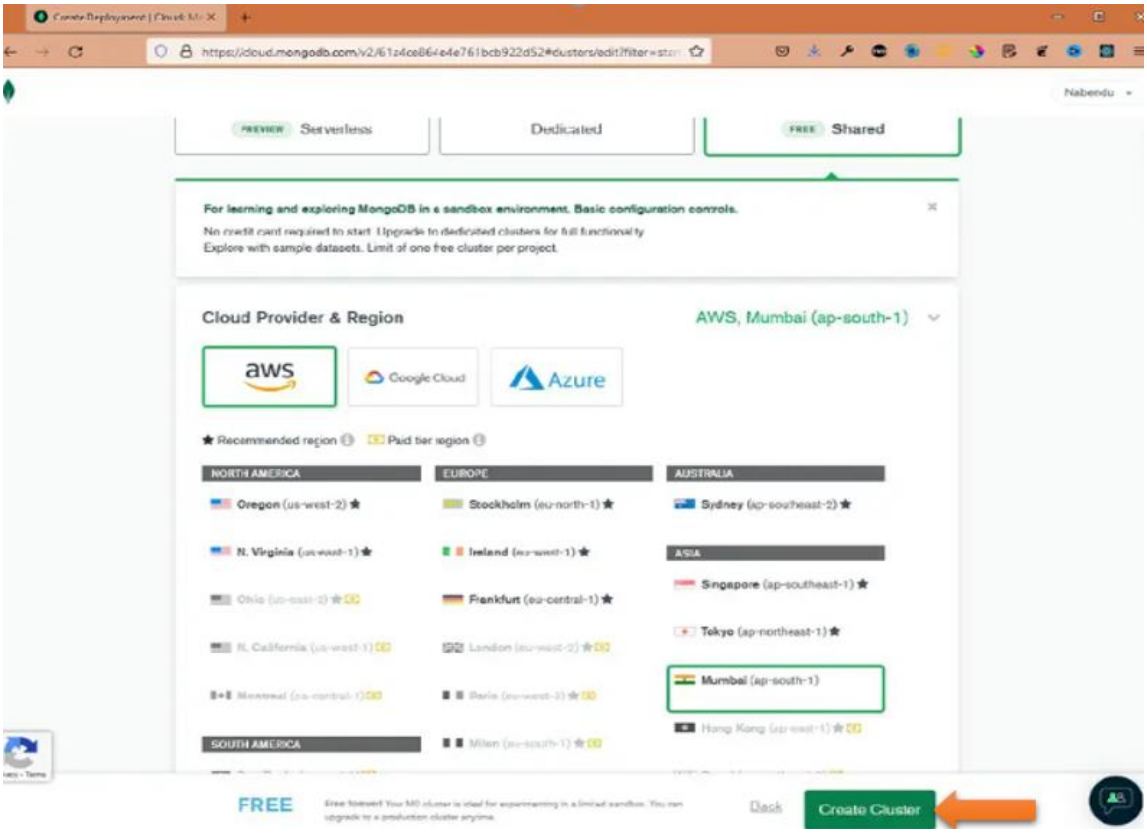
In the next page, click on the big Build a Database button to create your database.



It will then give you three options to start with. Here, I am going for a Shared server, which is free. Notice that you also have the option of a Dedicated server, and you should use this for production of apps.

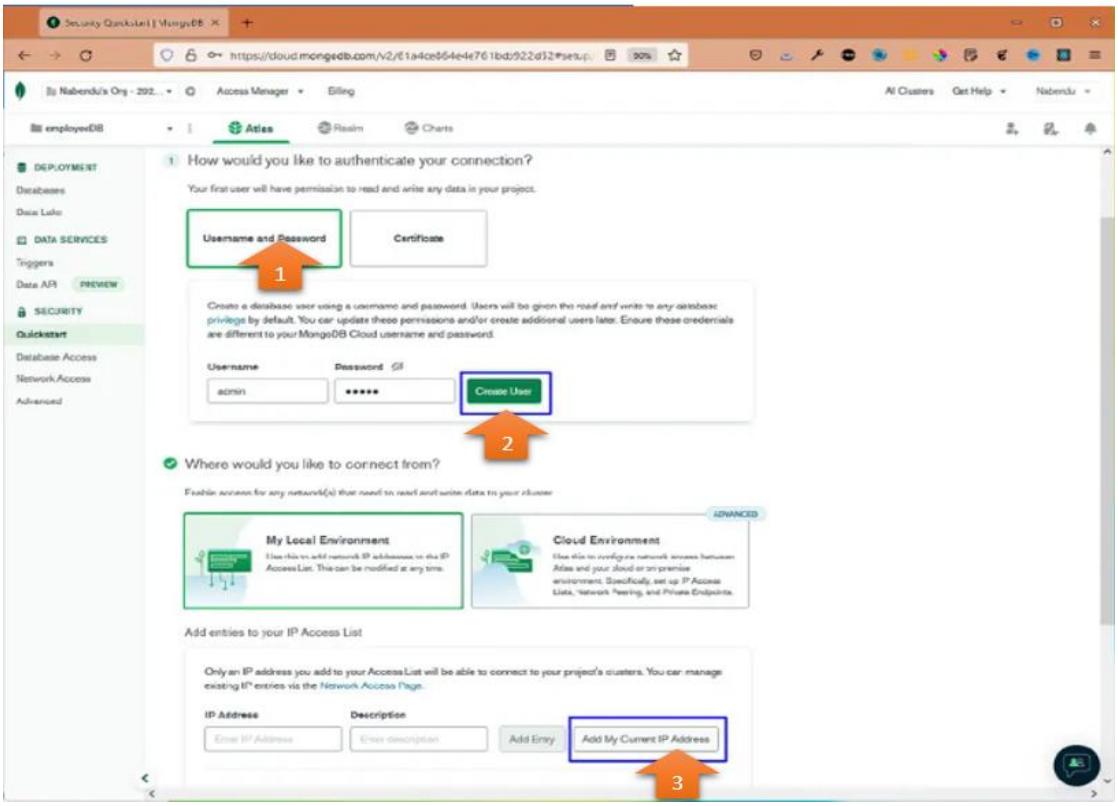


Now, it will ask you to choose a cloud provider and the region for the server. Choose the server that is nearest to your user base, as the lag will be minimal. Click on the Create Cluster button.

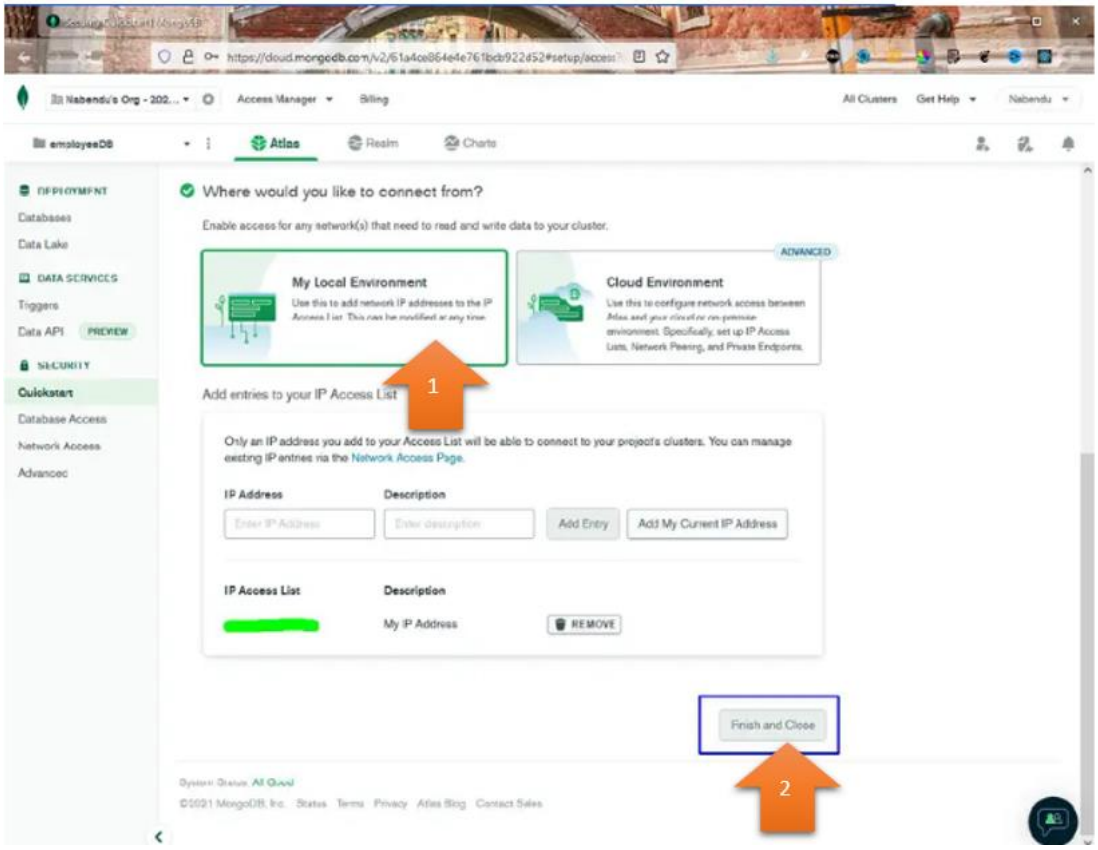


Next, it you will ask for a username and password. You should remember this, as you'll need it to connect through the NodeJS application. After providing the username and password, click on the Create User button.

You also need to give the IP address and for your testing project. You should then click on the Add My Current IP Address button.

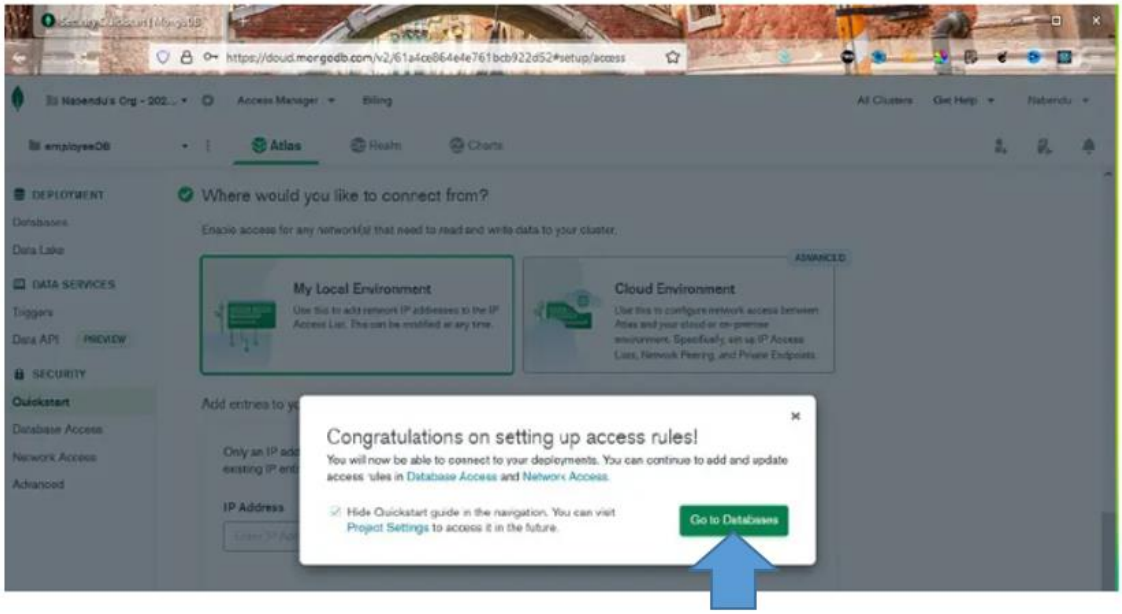


After that, scroll down a bit and click on the Finish and Close button.

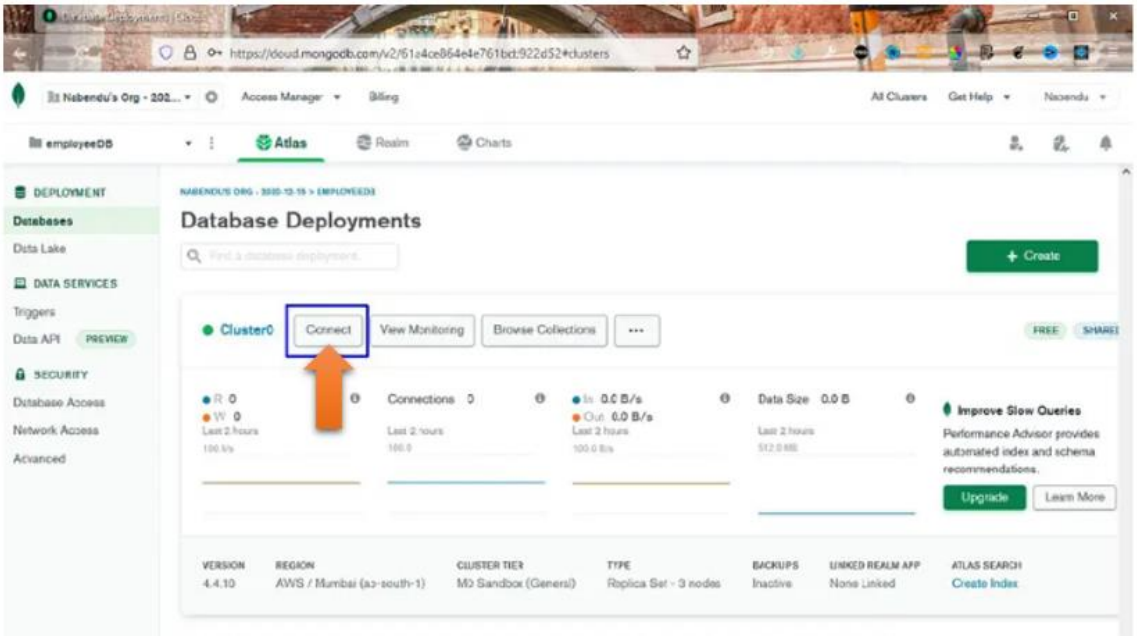


On successful creation of the user and IP address, you will get this pop-up. Click on the

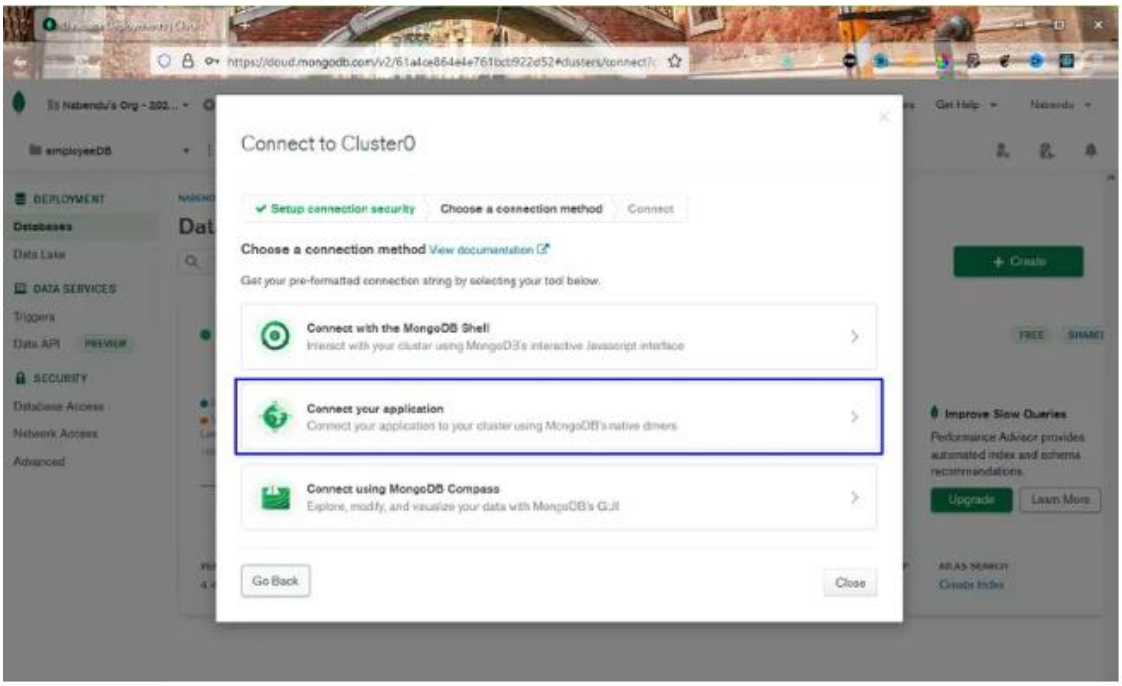
Go to Databases button.



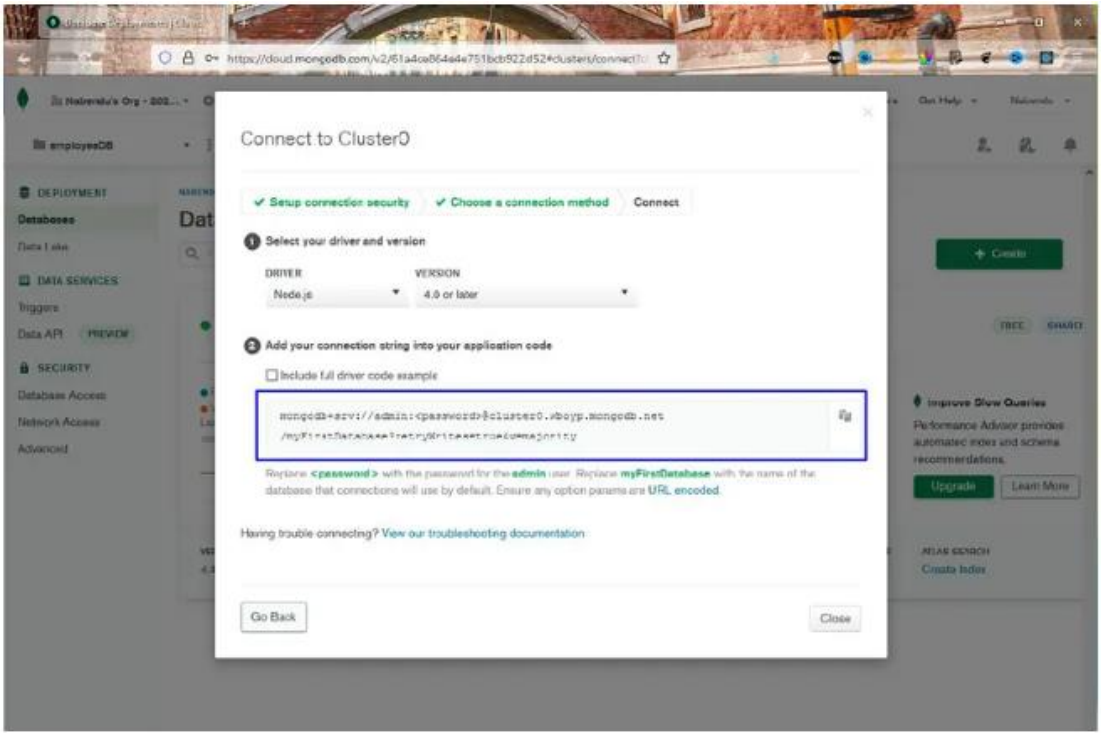
Now, you will be taken to below screen, which shows your cluster. Here, click on the Connect button.



A pop-up will appear. Click on the Connect your application option in the middle.



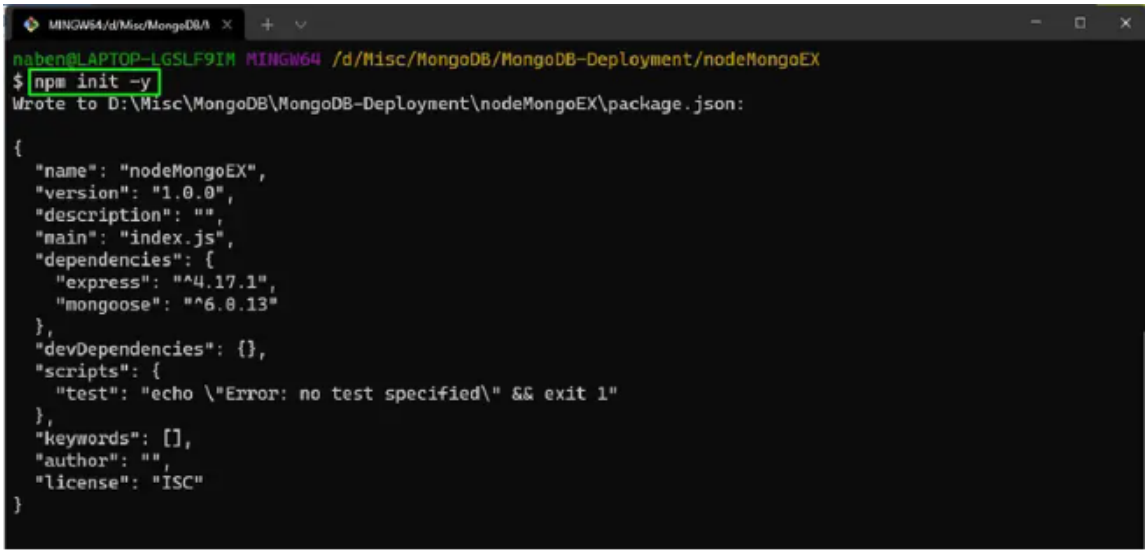
Now, you will get the connection string and you can copy it. You will need it to connect your NodeJS application next to the MongoDB database.



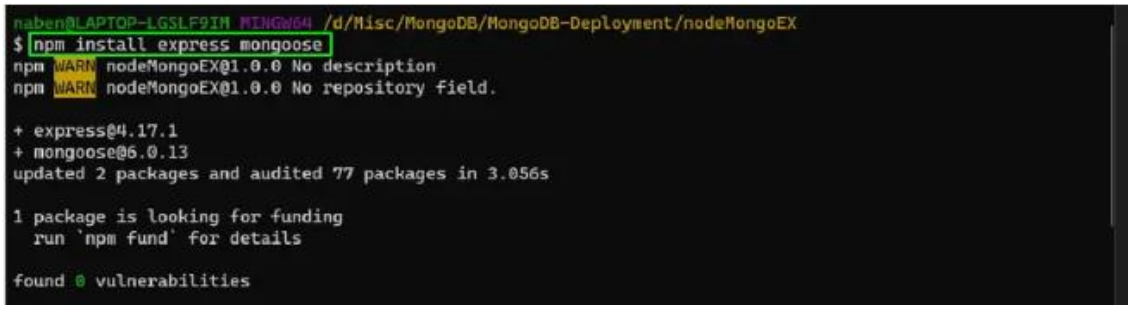
Connecting to Atlas

You will now connect a simple NodeJS app to your newly created Atlas database. You will create a simple app with NodeJS and express by first creating a folder and then changing to it.

Now, you will create an empty node app by giving the command npm init –y.



You will then install the mongoose and express package in it. Mongoose is a npm module, which is required to connect NodeJS app to a mongodb database. And express is used in the NodeJS app to make the programming much easier.



Steps for deploy NoSQLmongoDB

1) In MongoDB Cloud Manager, go to the Deployment page for your project.

a. If it is not already displayed, select the organization that contains your desired project from the organizations menu in the navigation bar.

b. If it's not already displayed, select your desired project from the Projects menu in the navigation bar.

c. If the Deployment page is not already displayed, click Deployment in the sidebar.

d. The Deployment page displays

2) Go to the *Processes* page.

a. Click the Processes tab for your deployment.

b. The Processes page displays

3) Choose New Standalone.

a. Click the Add dropdown menu.

b. Select New Standalone.

4. Configure the standalone MongoDB instance.

5. Set any Advanced Configuration options for the standalone MongoDB instance.

In the Advanced Configuration Options section, add any additional runtime options you want to set for your MongoDB deployment.

a. To add an option:

b. Click Add Option.

c. Select a Startup Option.

d. Set an acceptable value for that Startup Option.

4) Click Apply.

5) Click Review & Deploy to review your changes.

6) Click Confirm & Deploy to deploy your changes.

Otherwise, click Cancel and you can make additional changes.

Points to Remember

- Deployment option are on-premises, cloud and hybrid.
- Types of mongoDB cluster architectures are single-node cluster, Replica set and sharded cluster.
- **Steps of deploying NoSQL MongoDB**

✓ In MongoDB cloud manager, go to the deployment page for your project.

✓ Go to the processes page

✓ Choose new standalone

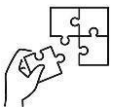
✓ Configure the standalone MongoDB instance

✓ Set any advanced configuration options for the standalone MongoDB instance

✓ Click apply

✓ Click review & deploy to review your changes

✓ Click confirm & deploy to Deploy your changes



Application of learning 4.3.

TechMart, an online retail platform, shifted from an on-premises database to a hybrid deployment using a cloud-based MongoDB solution. They initially used a single-node MongoDB architecture, but later upgraded to a replica set architecture for data redundancy. They adopted a sharded cluster architecture to improve query performance and support their rapid business growth. You are requested to deploy the database of TechMart.

.

Learning outcome 4 end assessment

Theoretical assessment

Section A: Read the statement carefully, then circle the letter corresponding to the correct answer based on the statement given

1. What is the primary role of database users?

a) To create backups
b) To manage database performance
c) To access and manipulate data
d) To configure network settings

2. Which command is used to create a new user in MongoDB?

a) db.createUser()
b) db.addUser()
c) db.newUser()
d) db.insertUser()

3. What is the purpose of roles in MongoDB?

a) To create collections
b) To define user permissions and privileges
c) To enhance database performance
d) To manage database backups

4. Which of the following is a method to enforce authentication in a database?

a) Using indexes
b) Setting up role-based access control
c) Implementing data encryption
d) Creating a user account with a password

5. What does role-based access control (RBAC) in MongoDB allow administrators to do?

a) Back up data automatically
b) Create indexes for faster queries
c) Assign specific roles to users based on their job functions
d) Encrypt sensitive data

6. Which of the following methods helps protect sensitive data in a database?

a) Implementing database sharding
b) Enabling data encryption
c) Creating backup copies
d) Monitoring database performance

7. What is the purpose of auditing system activity in a database?

a) To improve query performance
b) To track changes and access to the database
c) To back up data regularly
d) To manage user roles

8. Which deployment option involves hosting the database on physical servers owned by the organization?

a) Cloud
b) Hybrid
c) On-Premises
d) Remote

9. Which MongoDB cluster architecture consists of a primary node and multiple secondary nodes?

a) Single-Node
b) Replica Set
c) Sharded Cluster
d) Hybrid Cluster

10. What is the primary benefit of scaling MongoDB with sharding?

a) Improved data security
b) Enhanced data visualization
c) Increased database availability and performance
d) Simplified user management

Section B: Read the statement below and Answer by True for the correct statement Or False for wrong statement.

1) The primary role of database users is to manage server hardware and configuration.

2) A user in MongoDB can be created using the command db.createUser() with specific roles assigned.

3) Roles in a database are used to assign privileges and control what actions users can perform.

4) Enforcing authentication in a database is optional and does not significantly impact security.

5) Role-Based Access Control (RBAC) allows different users to have different permissions based on their roles within the organization.

6) Data encryption ensures that sensitive information in the database is protected from unauthorized access.

7) Auditing system activity in a database is only necessary during the initial setup and not needed afterward.

8) On-premises deployment of a database means that the database is hosted on the organization's physical servers.

9) A MongoDB sharded cluster allows for horizontal scaling by distributing data across multiple servers.

10) A replica set in MongoDB consists of only one primary node and no secondary nodes.

Section C: Read clearly the questions and answer them

1. What is the primary responsibility of database users?

2. Which command is used to create a new user in MongoDB?

3. How are roles and privileges managed in a database?

4. What is the purpose of enabling access control in a database?

5. What is Role-Based Access Control (RBAC)?

6. Why is data encryption important in database security?

7. What is the significance of auditing system activity in a database?

8. What does an on-premises database deployment involve?

9. Describe the difference between a replica set and a sharded cluster in MongoDB.

10. How does sharding improve the performance of a MongoDB database?

Practical assessment

The IT department at a mid-sized financial company migrates to MongoDB, managing user roles and privileges. They use Role-Based Access Control, data encryption, and system activity auditing to ensure security. They deploy a hybrid architecture, using replica sets for high availability and sharded clusters for data scalability. Sharding is crucial for efficient data distribution. As full stack developer, you are tasked to perform the following activities:

a) Identify the roles and privileges based on users
b) Create Role-based access control on users
c) Perform data encryption accordingly.
d) Perform data recovery and prevent disaster
e) Audit database to ensure security.

References:

/mongodb-role-based-access-control/. (n.d.). Retrieved from ww.bmc.com: https://www.bmc.com/blogs/mongodb-role-based-access-control/

access-control-101-a-comprehensive-guide-to-database-access-control. (n.d.). Retrieved from https://satoricyber.com: https://satoricyber.com/access-control/access-control-101-a-comprehensive-guide-to-database-access-control/

Batelho, A. S. (2023, March). *MongoDB*. Retrieved from www.techtarget.com: https://www.techtarget.com/searchdatamanagement/definition/MongoDB

database_concepts.htm. (n.d.). Retrieved from https://docs.exasol.com: https://docs.exasol.com/db/latest/database_concepts.htm

manage-users-and-roles/. (n.d.). Retrieved from https://www.mongodb.com: https://www.mongodb.com/docs/manual/tutorial/manage-users-and-roles/

mongodb-from-basics-to-deployment-on-kubernetes-c1ced7143a6c. (n.d.). Retrieved from https://medium.com/: https://medium.com/@tanmaybhandge/mongodb-from-basics-to-deployment-on-kubernetes-c1ced7143a6c

ore/authorization/. (n.d.). Retrieved from www.mongodb.com: https://www.mongodb.com/docs/manual/core/authorization/

RTB | RWANDA TVET BOARD

October 2024