Mongo DB Fundamentals

1. **Table of Content** 
   1. What is Upsert

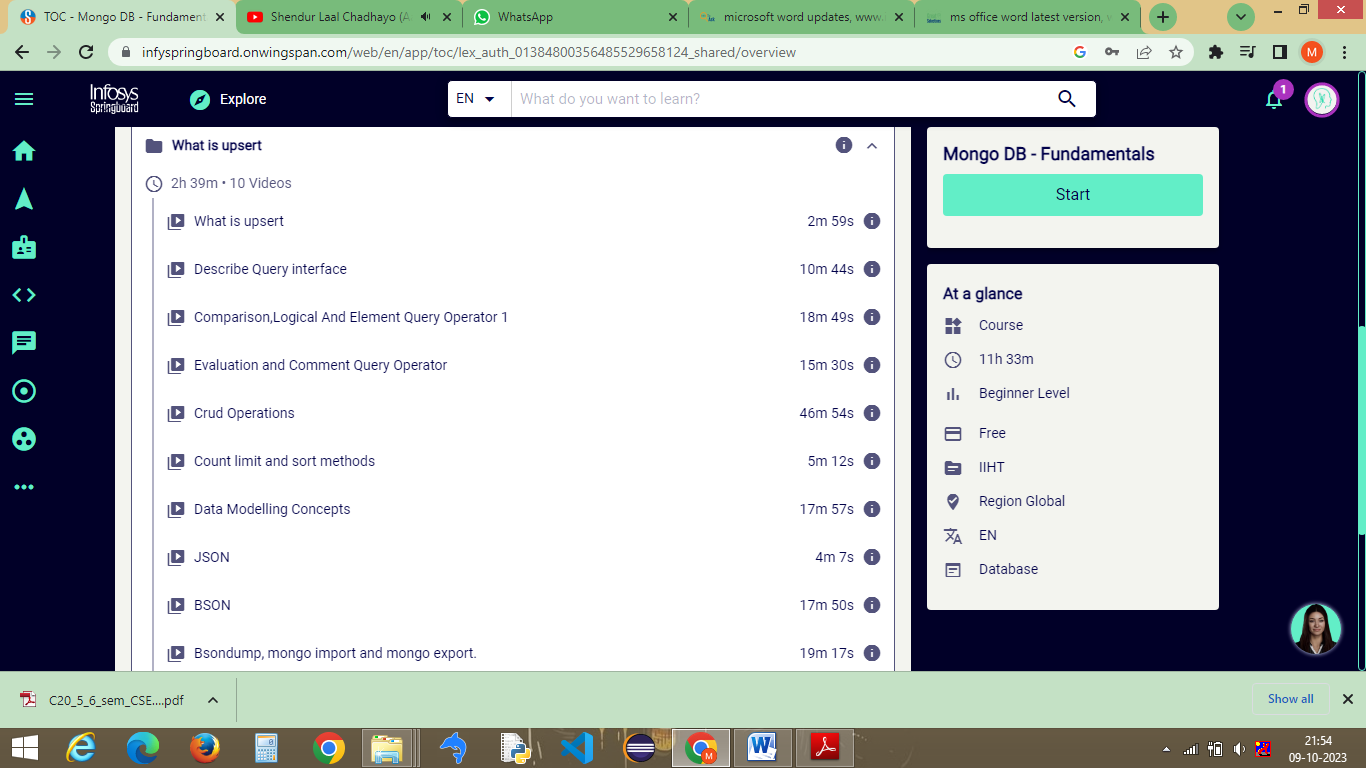


Fig 1.1 : Content of Upsert

* 1. Data Type

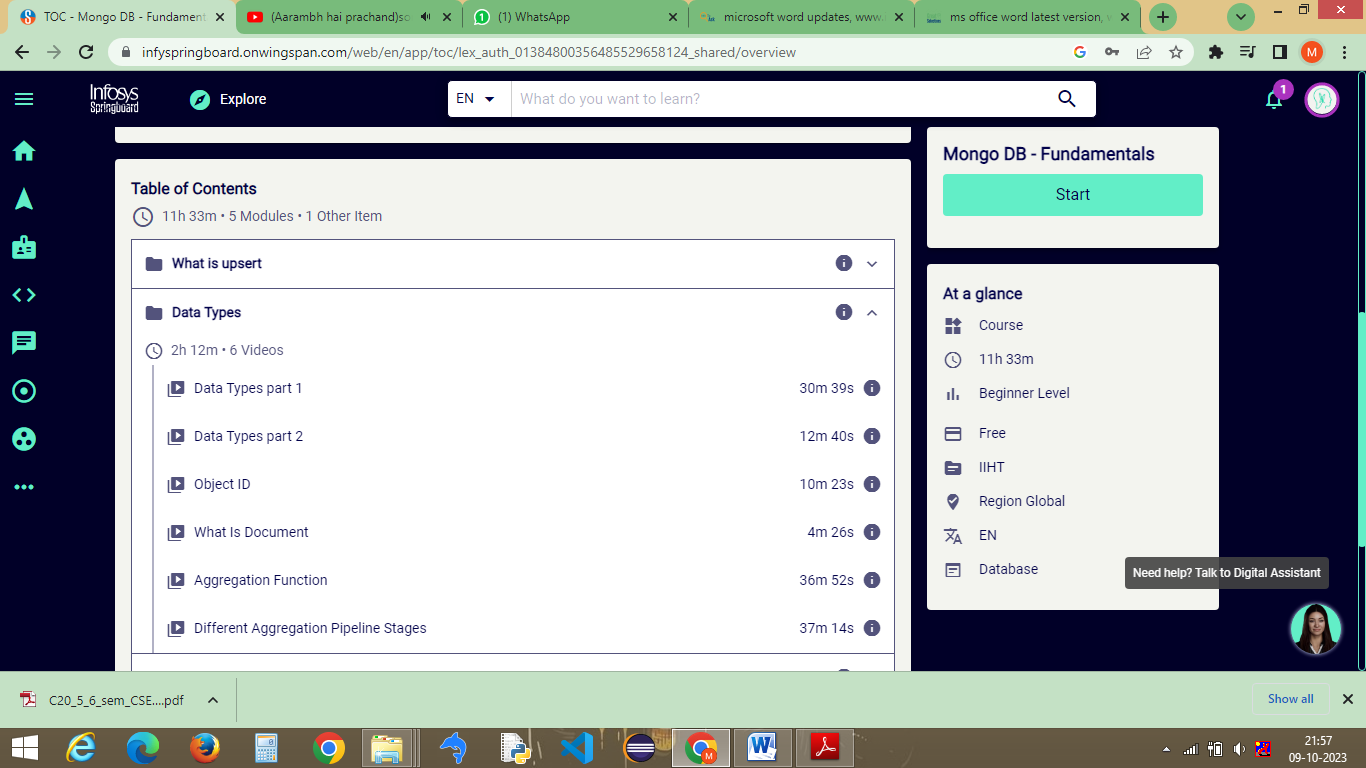


Fig 1.2 : Content of Data type

1.3 Indexes The Keys to Speed

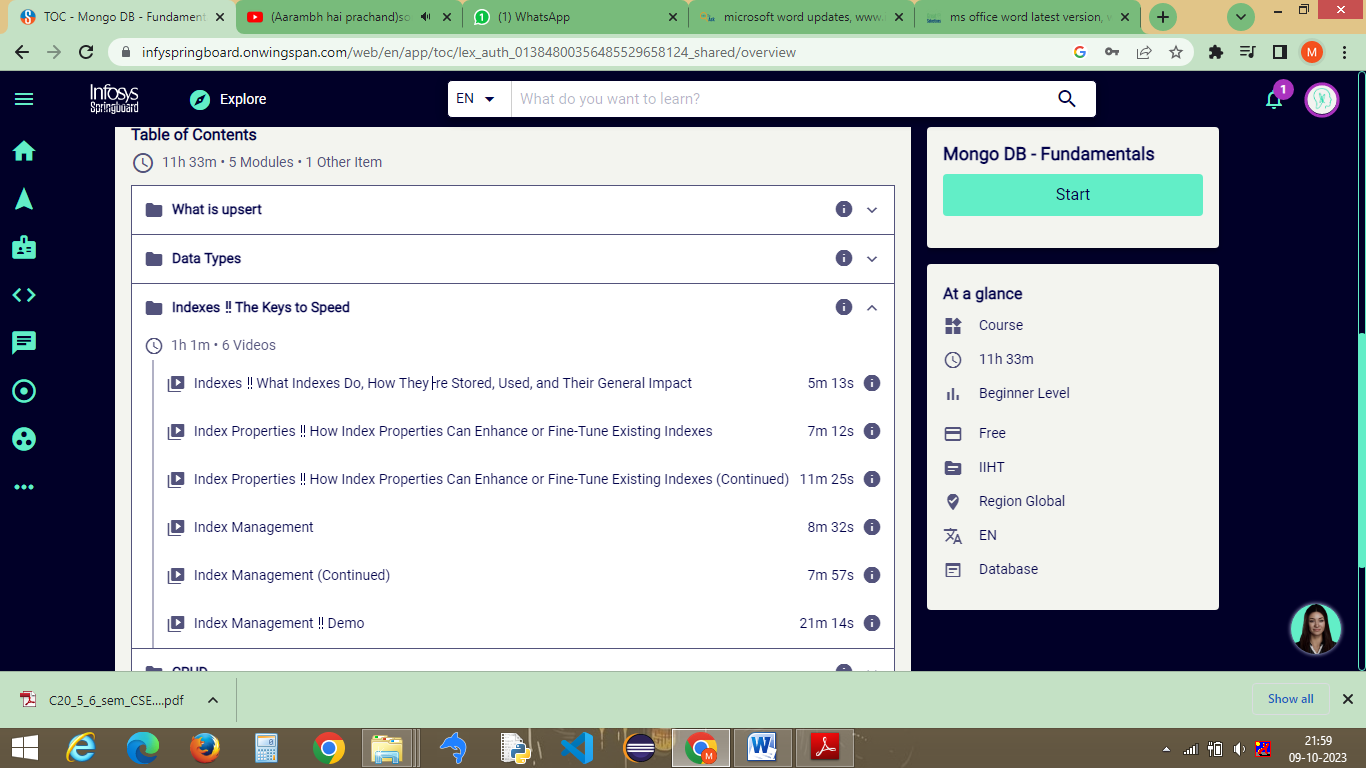
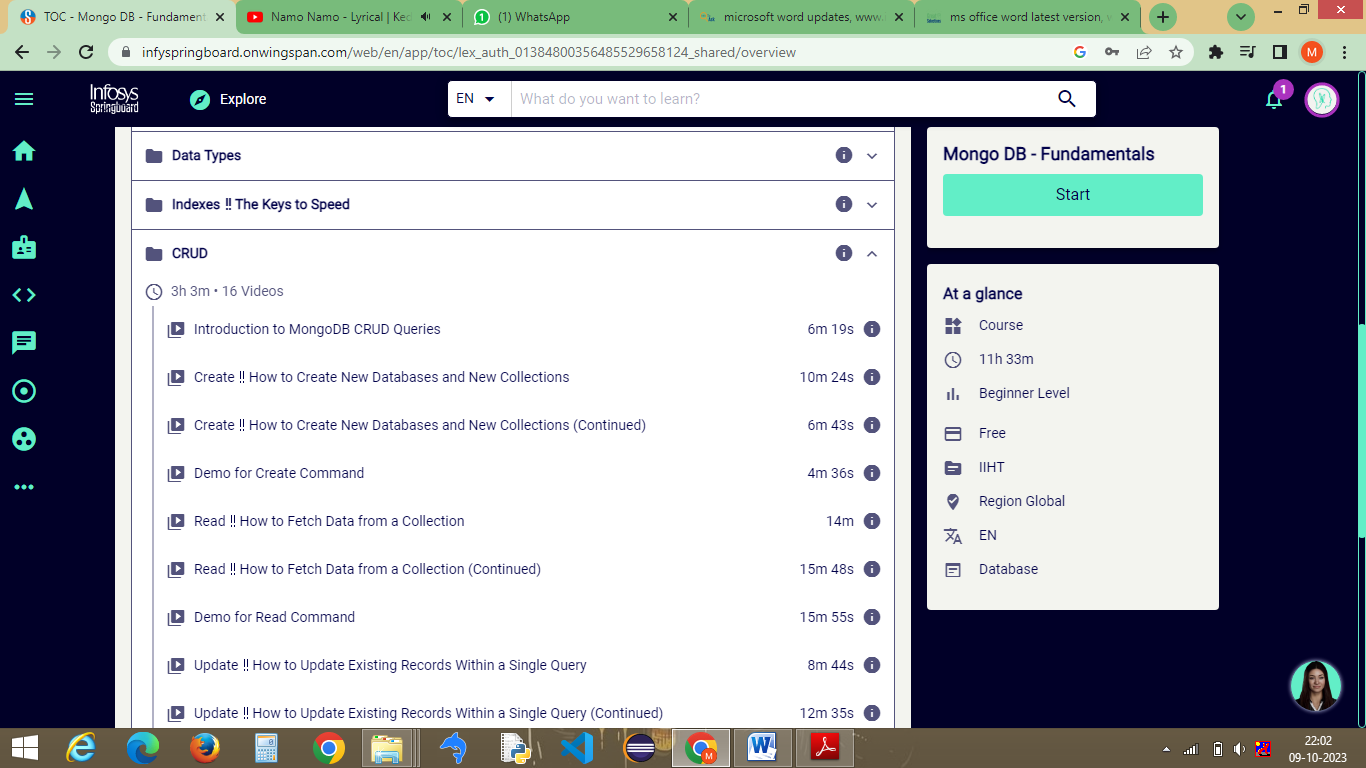


Fig 1.3 : Content of Indexes The Keys to Speed

* 1. CRUD



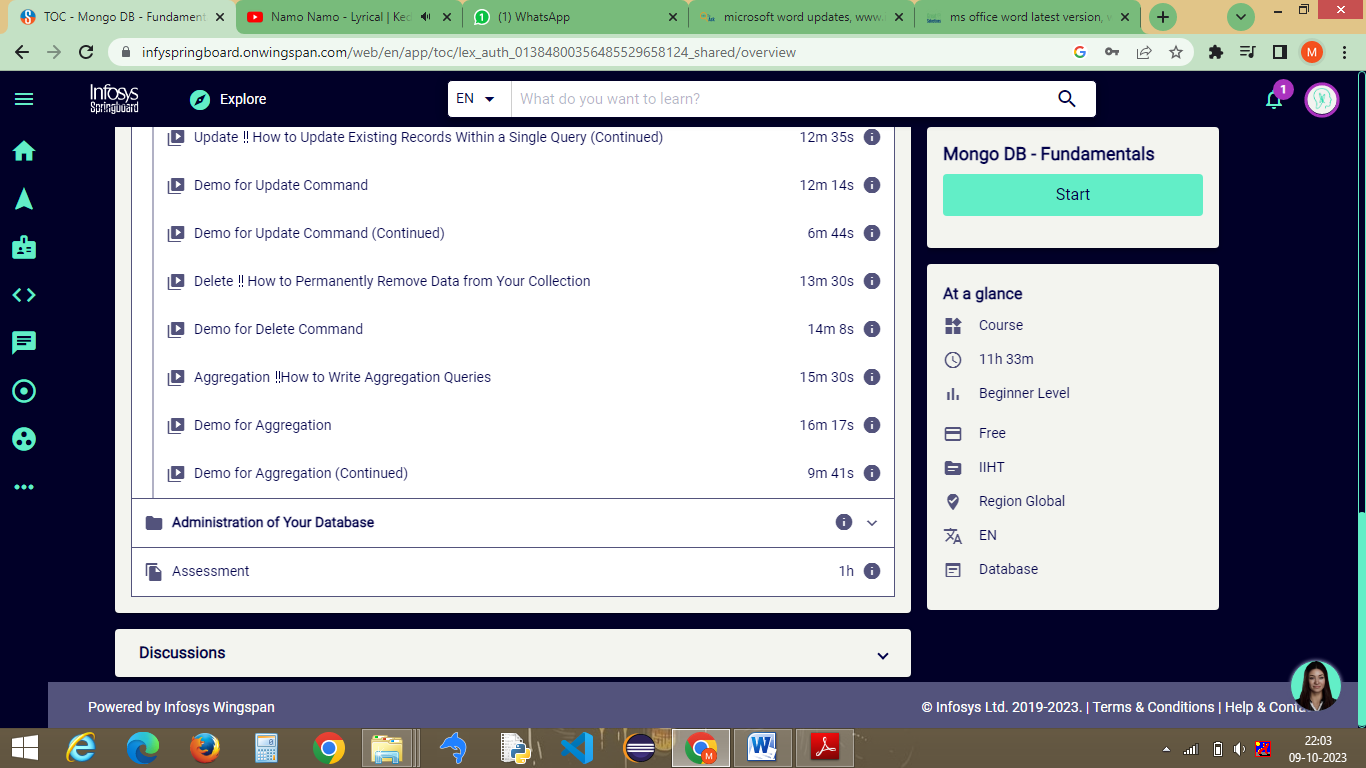


Fig 1.4 : Content of CRUD

* 1. Administration of your Database

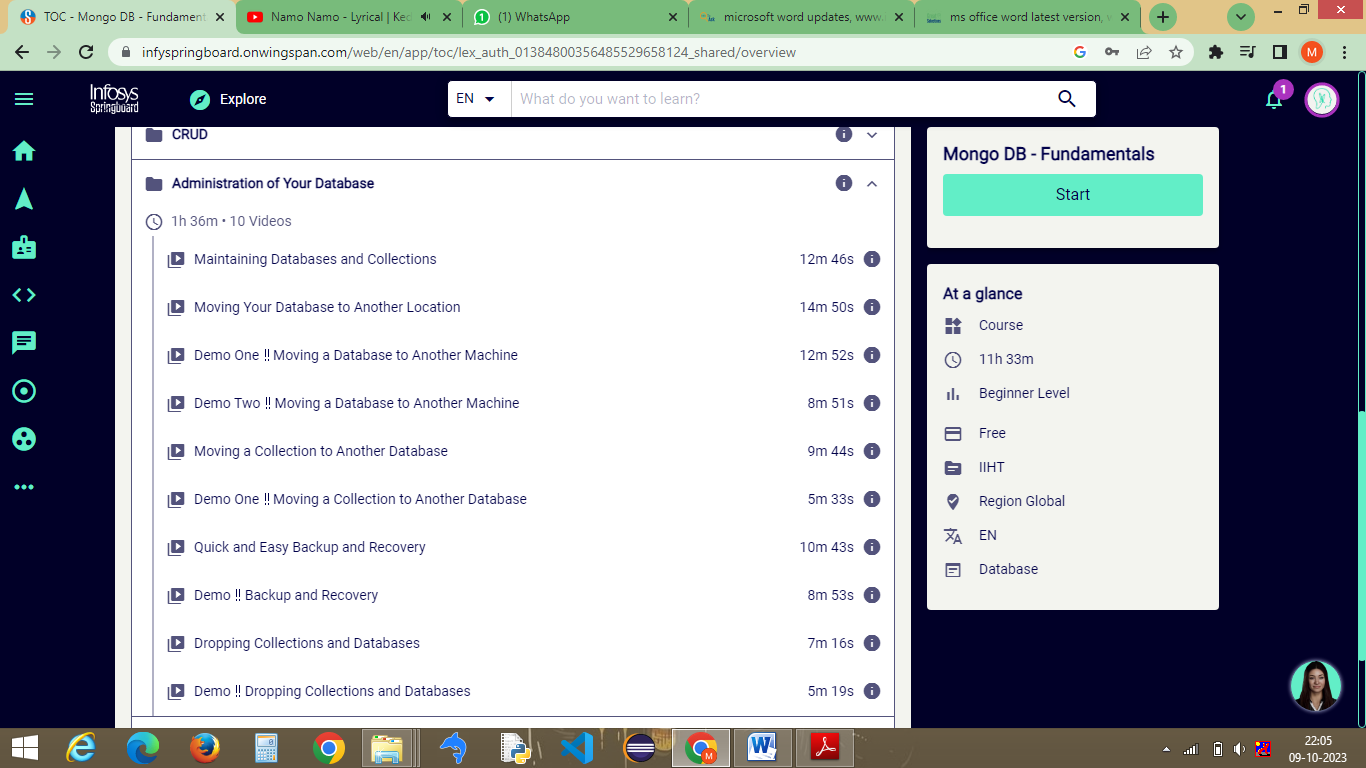


Fig 1.5 : Content of Administration of your Database

* 1. Assessment

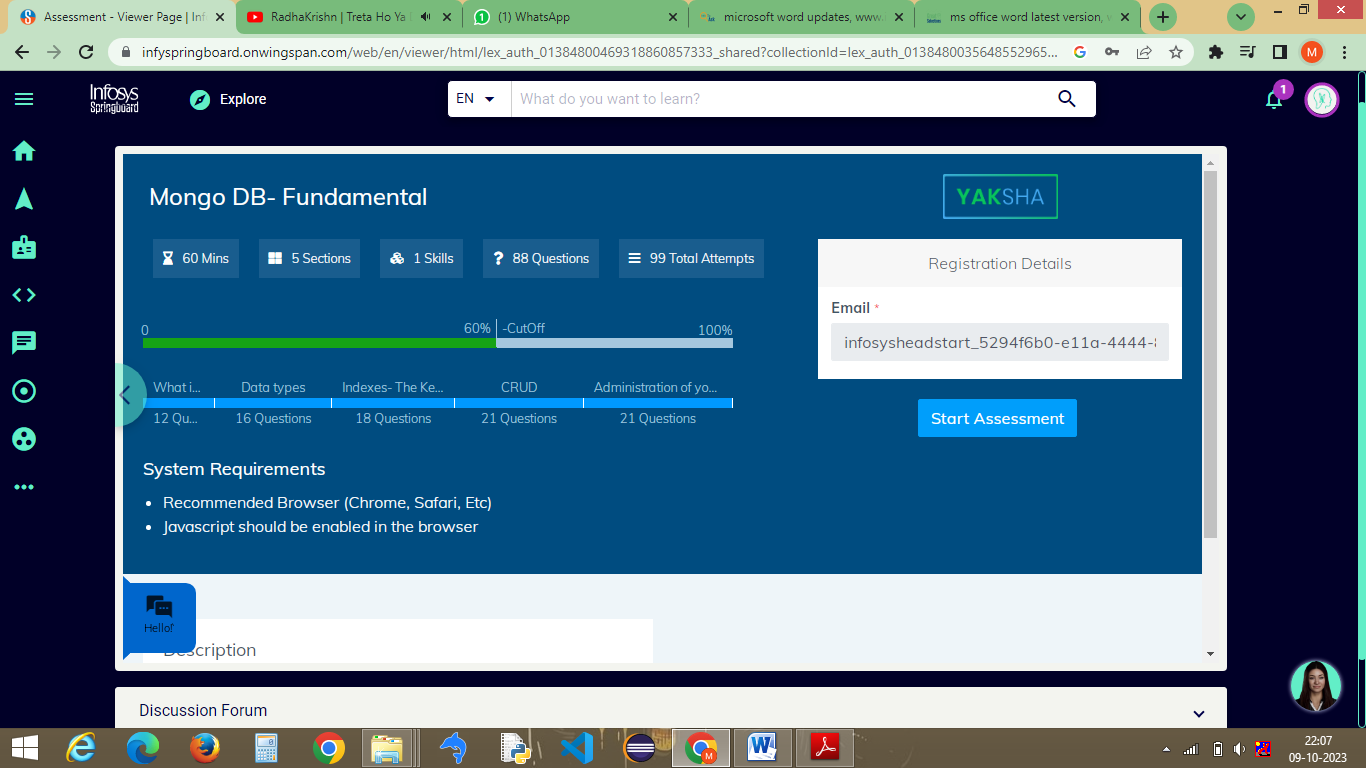


Fig 1.6 : Assessment Requirements

**2 Upsert:**

* 1. What is Upsert?

>>> In MongoDB, "upsert" is a combination of the words "update" and "insert." It is an operation that allows you to update a document in a collection if it exists or insert it if it does not exist. In other words, an upsert is a way to ensure that a document with a specific set of criteria either gets updated or created if it doesn't exist.

The upsert operation typically involves two main components:

Query Criteria: You specify the criteria that determine whether a document should be updated or inserted. If there is an existing document in the collection that matches these criteria, MongoDB will update it. If not, a new document will be inserted.

Update Data: You provide the data that you want to update the document with (if it exists) or insert as a new document (if it doesn't exist).

* 1. Describe Query Interface.

>>> In MongoDB, the "query interface" typically refers to the methods and operations used to retrieve data from a MongoDB database. MongoDB provides a rich set of query capabilities to search for and retrieve documents from collections. The primary way to perform queries in MongoDB is through the use of the find method, which is used to search for documents in a collection that match a specified query criteria.

* 1. Comparison, logical and element query operator 1

>>> In MongoDB, query operators are used in conjunction with the find method to filter and retrieve documents from a collection based on specific criteria. There are three main types of query operators: comparison operators, logical operators, and element operators.

Comparison Operators : Comparison operators are used to compare fields with a specified value.

$eq: Matches values that are equal to a specified value.

$ne: Matches values that are not equal to a specified value.

$gt: Matches values that are greater than a specified value.

$lt: Matches values that are less than a specified value.

$gte: Matches values that are greater than or equal to a specified value.

$lte: Matches values that are less than or equal to a specified value.



Fig 2.1 : Comparison Query Operators

Logical Operators: Logical operators allow you to combine multiple conditions in a query.

$and: Joins query clauses with a logical AND, and returns documents that meet all conditions.

$or: Joins query clauses with a logical OR, and returns documents that meet at least one condition.

$not: Inverts the effect of a query expression.

$nor: Joins query clauses with a logical NOR, and returns documents that do not meet any condition.

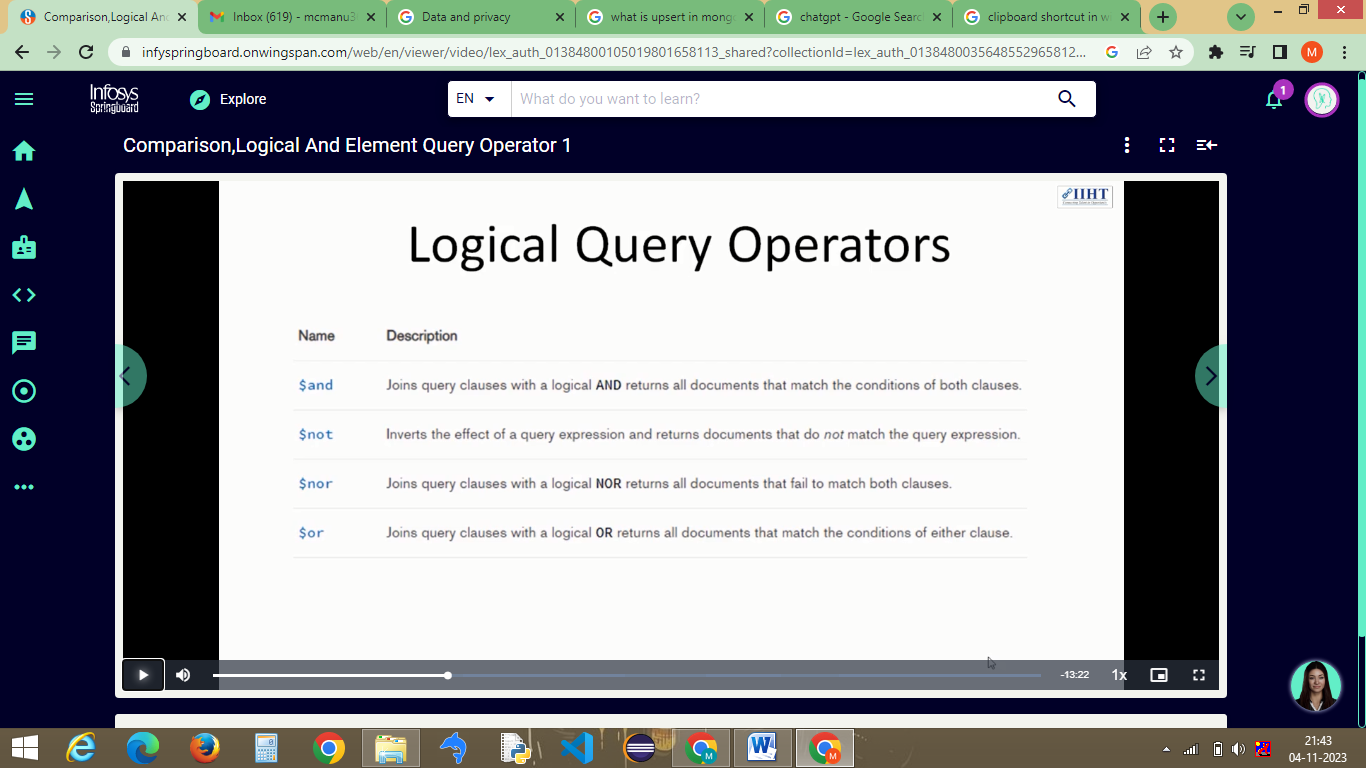


Fig 2.2 : Logical Query Operators

Element Operators: Element operators allow you to query documents based on the existence or type of fields.

$exists: Matches documents that contain a specific field.

$type: Matches documents where a field's type matches the specified BSON data type.



Fig 2.3 : Element Query Operators

2.4 Evaluation and comment query operator

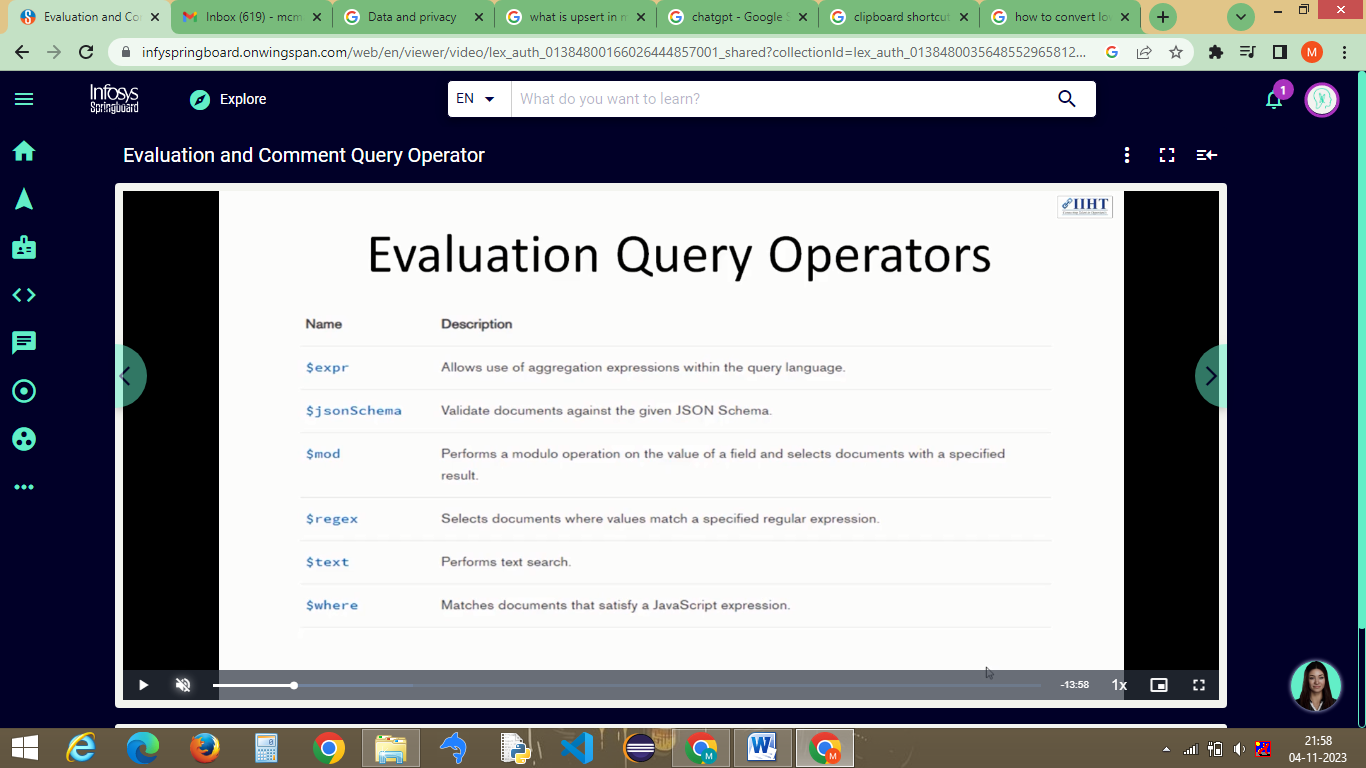


Fig 2.4 : Evaluation Query Operators

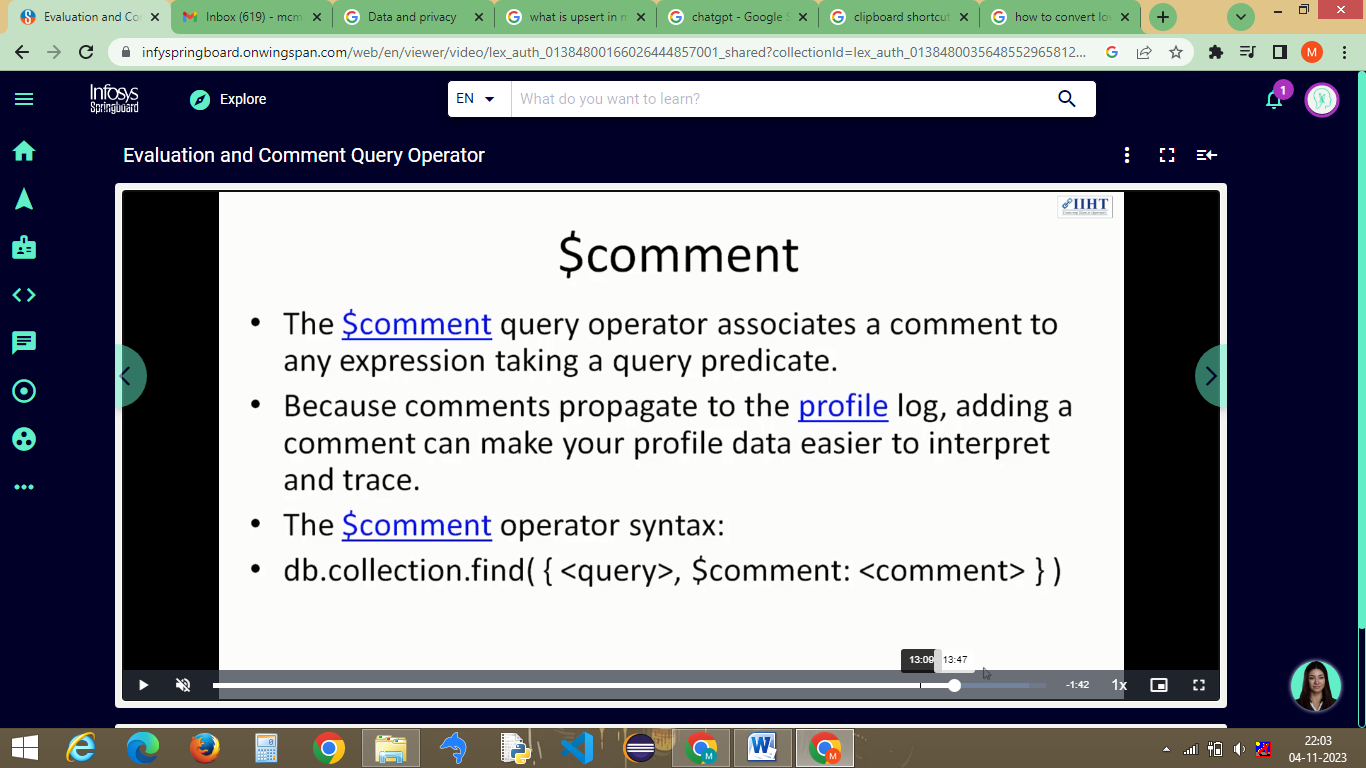


Fig 2.5 : Comment Query Operator

* 1. Crud Operations

>>> CRUD operations are fundamental operations used to interact with a database, and the acronym CRUD stands for Create, Read, Update, and Delete. These operations are common in database management and data manipulation.

* 1. Count limit and Sort Methods

>>> In MongoDB, you can use the count method to count the number of documents in a collection that match a specified query. The count method has been deprecated in recent MongoDB versions and has been replaced by the countDocuments method.

The sort method is typically used in conjunction with the find method to retrieve and sort documents from a collection. Here's how you can use the sort method in MongoDB.

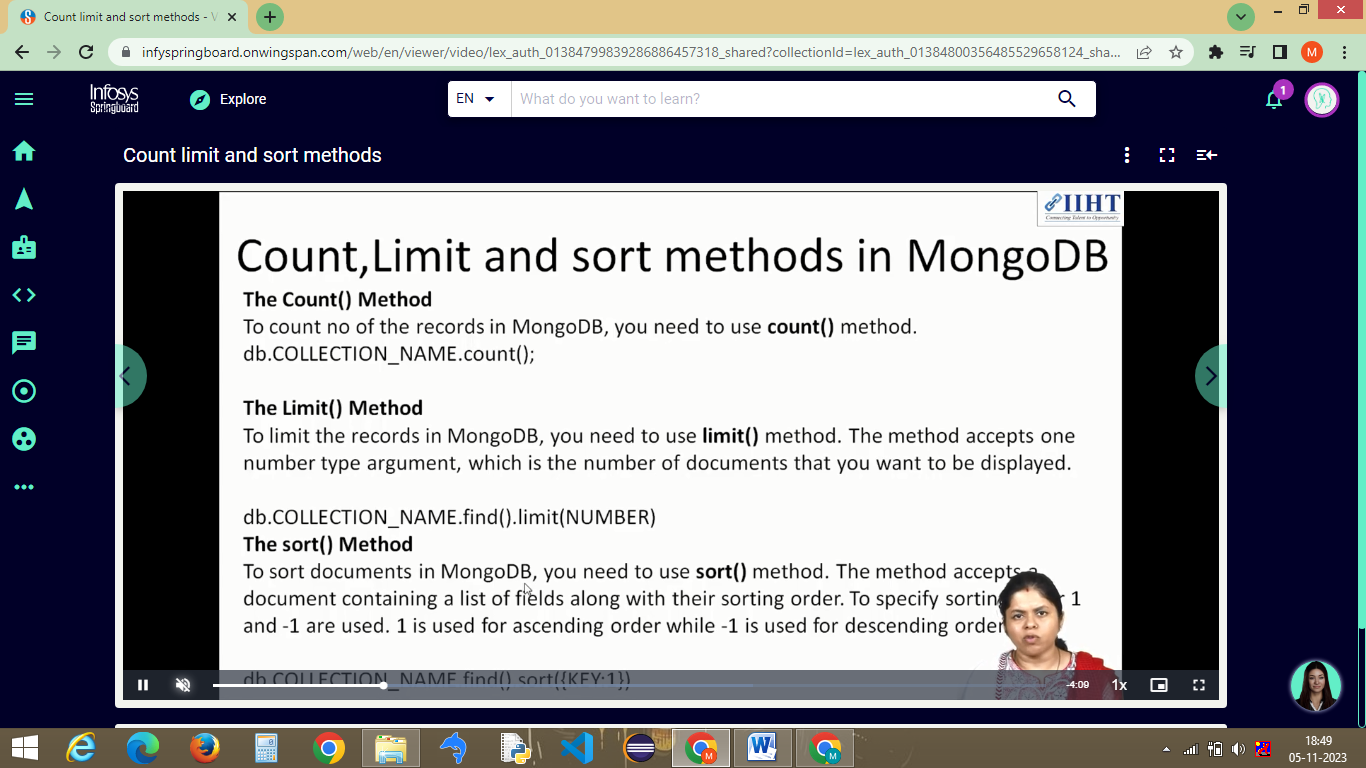


Fig 2.6 : Count Limit and Sort Methods

* 1. Data Modelling Concepts.

>>> Data modelling in MongoDB involves designing the structure of your documents, collections, and databases to efficiently store and retrieve data. MongoDB is a NoSQL database, and it uses a flexible schema approach, allowing you to model your data in various ways.

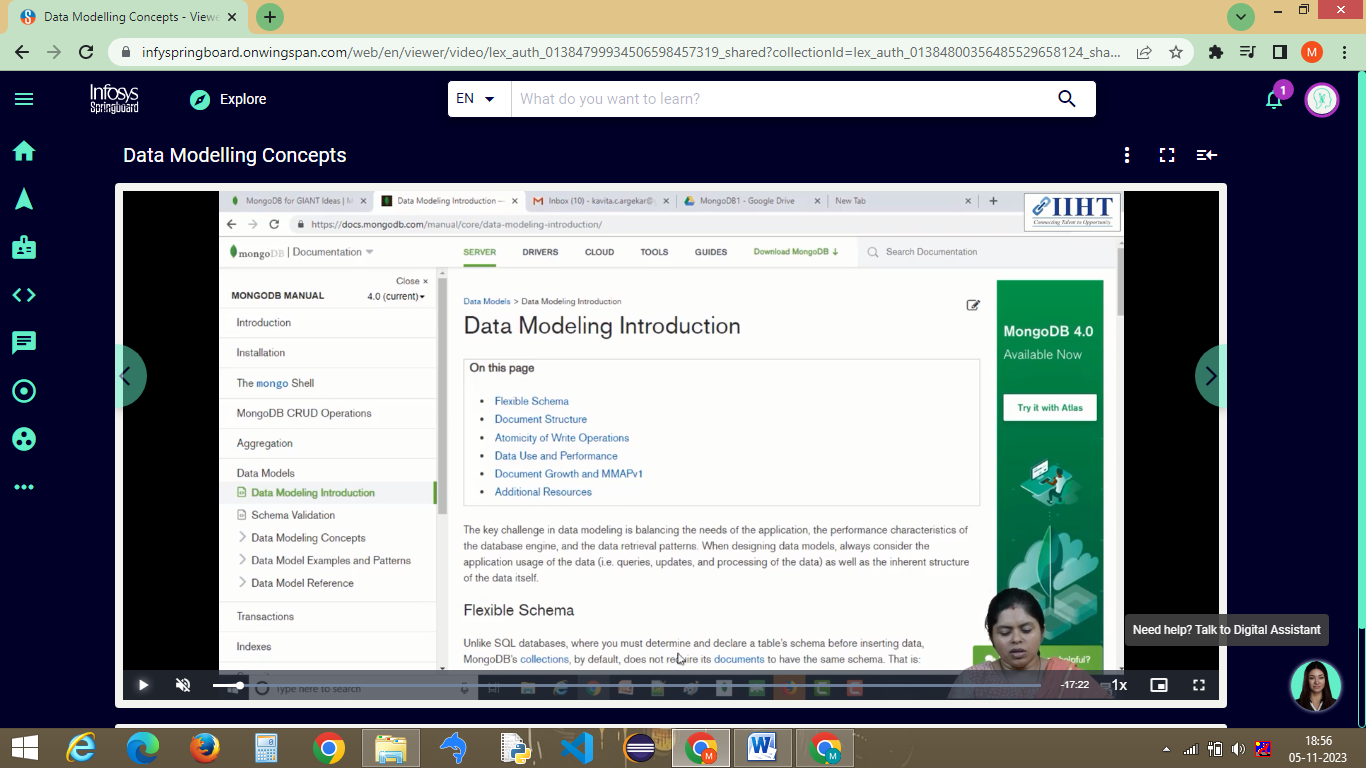


Fig 2.6 : Data Modelling Concepts

* 1. JSON.

>>> In MongoDB, JSON (JavaScript Object Notation) plays a significant role in the data storage and exchange process. MongoDB stores data in a format that closely resembles JSON, and it is known as BSON (Binary JSON). BSON is a binary representation of JSON that MongoDB uses for efficiency and to support various data types.

2.9 BSON.

>>> In MongoDB, BSON (Binary JSON) is a binary-encoded serialization format used to store and exchange data. BSON is the native storage format for data in MongoDB, and it extends the JSON (JavaScript Object Notation) format to provide more data types, support for binary data, and additional features for efficient data storage and retrieval. BSON is a compact, binary format that allows MongoDB to work efficiently with data, making it well-suited for a database system.

* 1. bsondump, mongodb import and mongodb export.

>>> In MongoDB, bsondump is a utility that allows you to view the contents of BSON data in a more human-readable format. You can use bsondump to convert BSON data to JSON format, which is easier to read and work with.

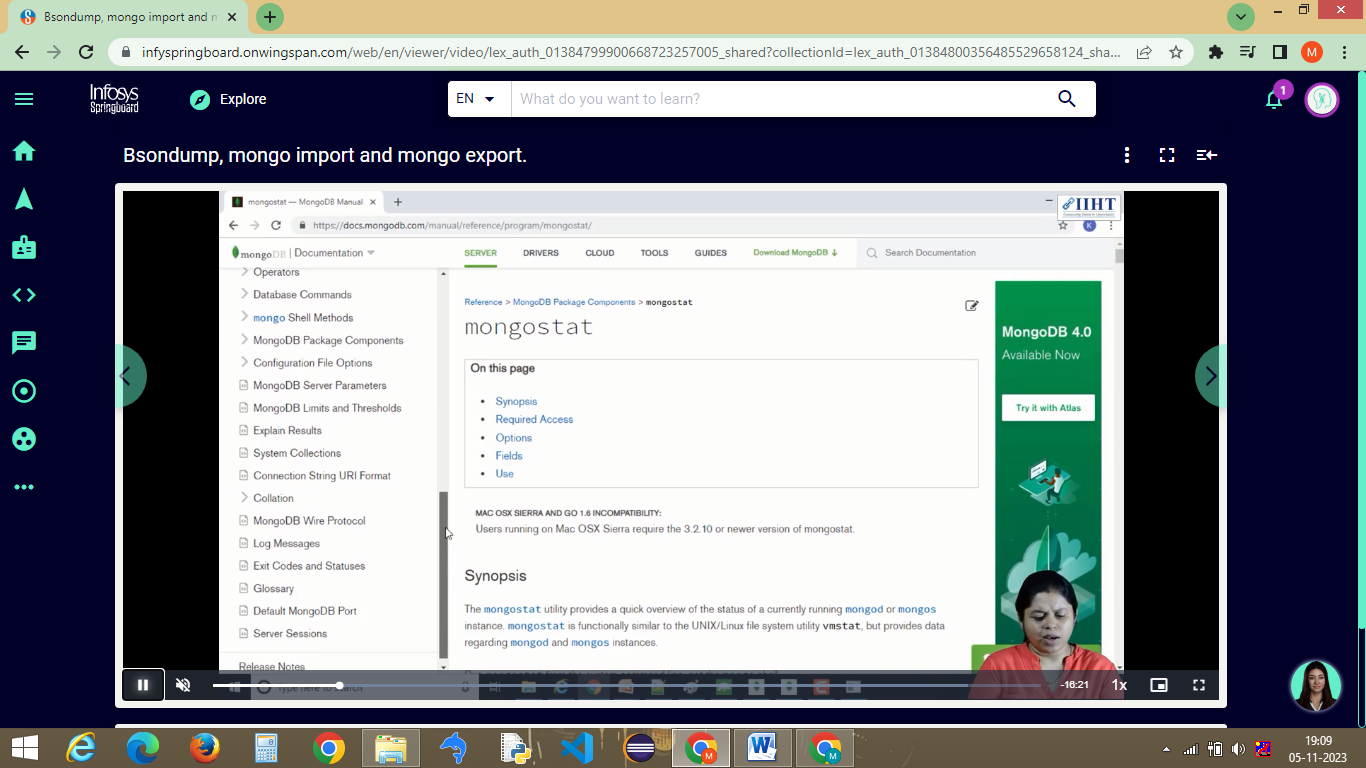


Fig 2.7 : Bsondump Introduction in Infosys Sprinboard

MongoDB provides tools for importing and exporting data in various formats. The most commonly used tools are mongoimport and mongoexport.

**mongoimport:**

mongoimport is used to import data into MongoDB from various file formats, including JSON, CSV, and TSV.

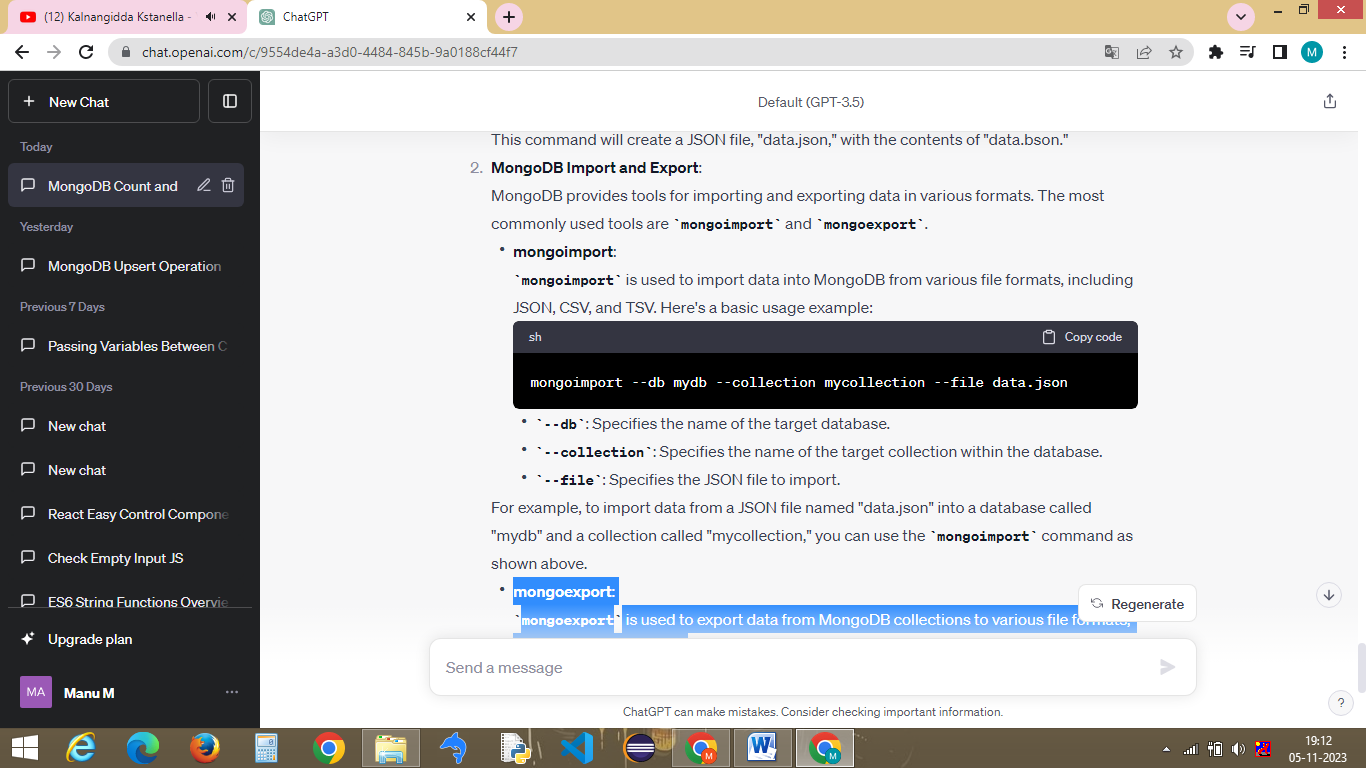


Fig 2.8 : Mongodb Import Syntax

**mongoexport:**

mongoexport is used to export data from MongoDB collections to various file formats, including JSON and CSV.

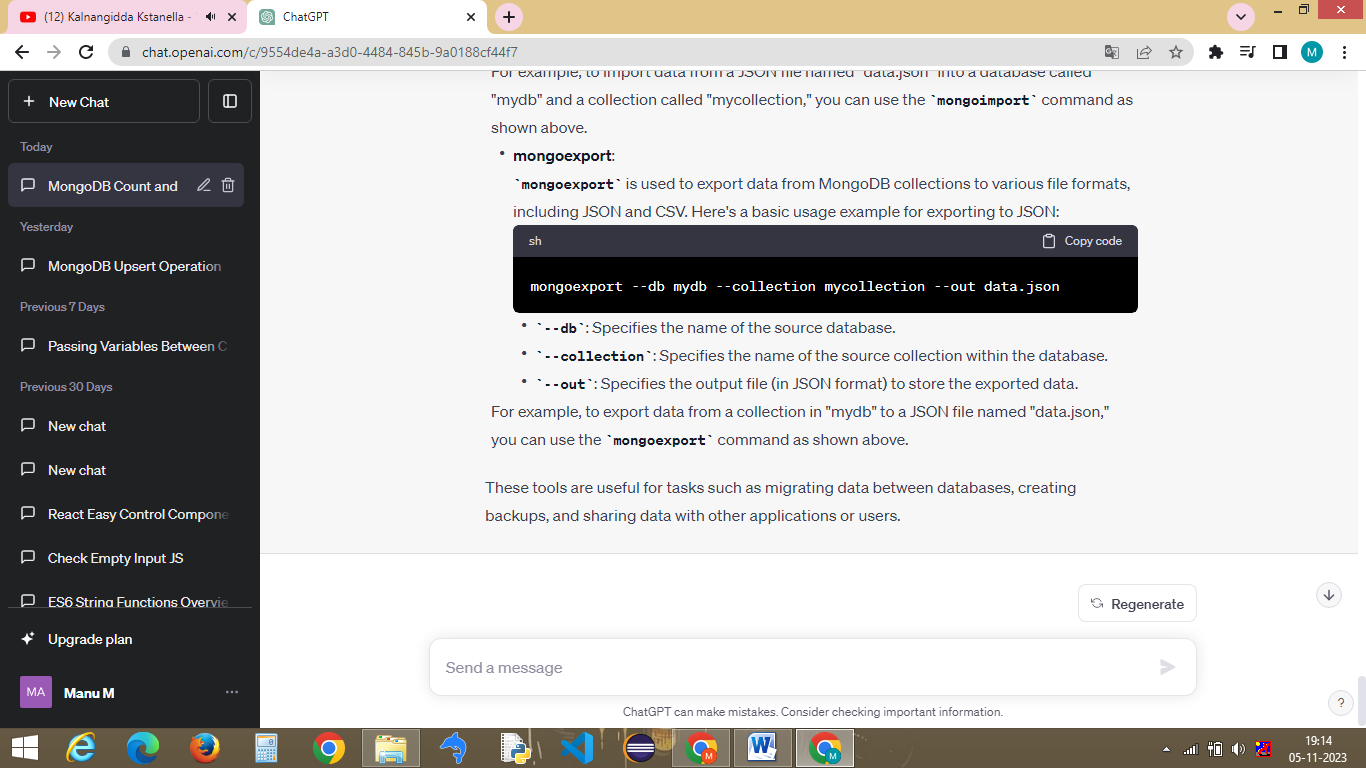


Fig 2.9 : Mongodb Export Syntax

1. **Data Types**

3.1 Data Types

>>>

1. String: This data type is used to store textual data, such as names, addresses, and descriptions.

2. Integer: MongoDB supports 32-bit and 64-bit integer data types, which can store whole numbers.

3. Double: Double data types are used to store floating-point numbers, which can represent decimal values.

4. Boolean: This data type can store true or false values.

5. Object: MongoDB allows you to store embedded documents (objects) within a document. These embedded objects can have their own fields with various data types.

6. Array: Arrays are used to store lists of values. You can have arrays of various data types, including strings, integers, and other arrays.

7. Date: MongoDB provides a specific data type for working with dates and times.

8. ObjectId: ObjectId is a unique identifier for documents in a collection. It is a 12-byte hexadecimal number, typically generated by MongoDB when a document is inserted.

9. Binary Data: MongoDB can store binary data in the form of Binary Large Objects (BLOBs) or GridFS for larger files.

10. Regular Expression: MongoDB allows you to store regular expressions as data types.

11. Null: Represents a null or empty value.

12. Undefined: Represents a field with an undefined value.

13. MinKey and MaxKey: Special values used to compare and sort data.

14. Timestamp: Used for internal MongoDB operations and replication.

15. Symbol: Symbol data type is used to store symbol data.

3.2 ObjectID

>>> In MongoDB, an ObjectId (short for "Object Identifier") is a 12-byte identifier typically used as the primary key for documents within a collection. ObjectId is a special data type used to uniquely identify documents.

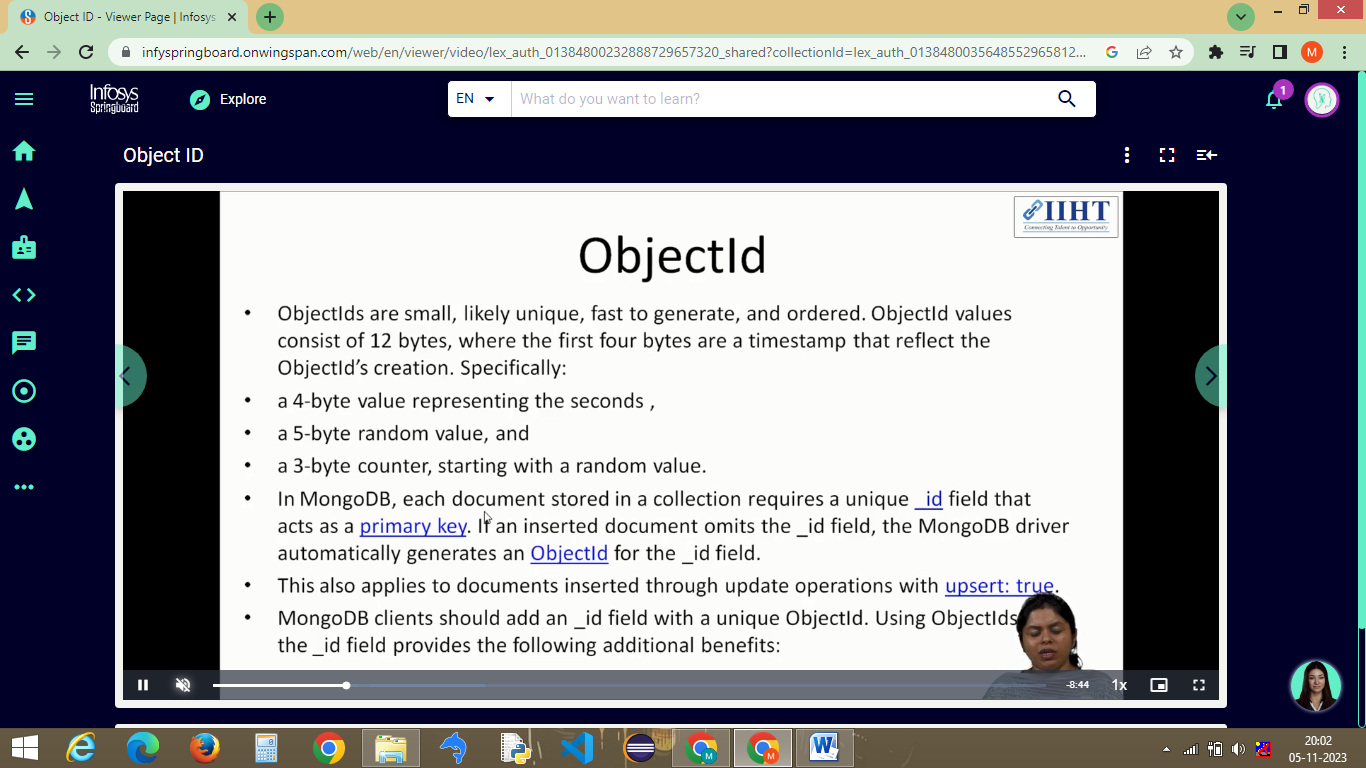
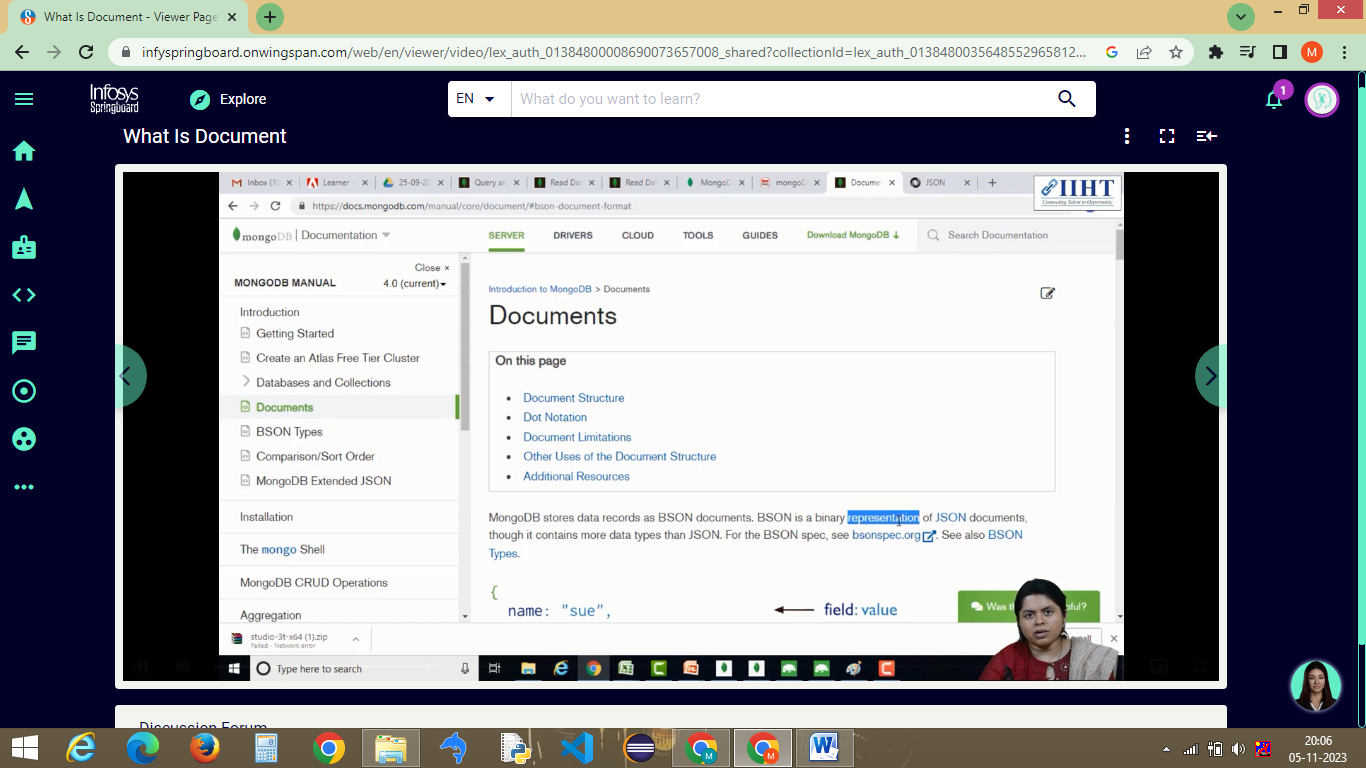


Fig 3.1 : Object ID

3.3 What is Documents

>>> In MongoDB, a "document" is the basic unit of data storage and retrieval. It is a data structure used to represent and store information in a flexible, schema-less format. Documents are the equivalent of rows or records in traditional relational databases. MongoDB is a NoSQL database, and it uses a document-oriented data model, which means that data is stored as collections of documents.



3.3 Aggregation Functions

>>> MongoDB provides a powerful aggregation framework that allows you to perform complex data transformations, calculations, and analysis on your data. The aggregation framework is used to process and manipulate data in a flexible way, often similar to SQL's GROUP BY and aggregate functions.



Fig 3.3 : Aggregation in MongoDB

3.4 Different Aggregation Pipeline Stages

>>> In MongoDB, the aggregation pipeline is a framework for processing and transforming documents in a collection. The pipeline consists of multiple stages, each of which performs a specific operation on the data. You can chain these stages together to perform complex data manipulations. Some of the most common aggregation pipeline stages are:

$match: Filters documents based on specified criteria, effectively acting as a filtering stage.

$project: Reshapes documents, selecting specific fields, and computing new fields using expressions. This is similar to the SELECT clause in SQL.

$group: Groups documents together and performs aggregation operations on those groups. You can use operators like $sum, $avg, $min, $max, and others within $group to compute aggregated values.

$sort: Sorts the documents based on one or more fields, either in ascending (1) or descending (-1) order.

$limit: Limits the number of documents in the result set to a specified number.

$skip: Skips a specified number of documents and returns the remaining ones.

$unwind: Deconstructs arrays within documents, creating multiple copies of the document with each array element as a separate document.

$lookup: Performs a left outer join between documents from two collections based on a common field, allowing you to combine data from multiple collections in your aggregation.

$addFields and $set: These operators add new fields to documents, either by specifying a constant value or using expressions.

$replaceRoot and $replaceWith: These operators allow you to promote a specified subdocument to be the new root document or replace the current root document entirely.

$out: Writes the result of an aggregation pipeline to a new collection.

$facet: Allows you to perform multiple independent aggregations on the same set of documents and return the results as an array.

$redact: Applies field-level access control to documents, allowing you to restrict the fields visible in the output.

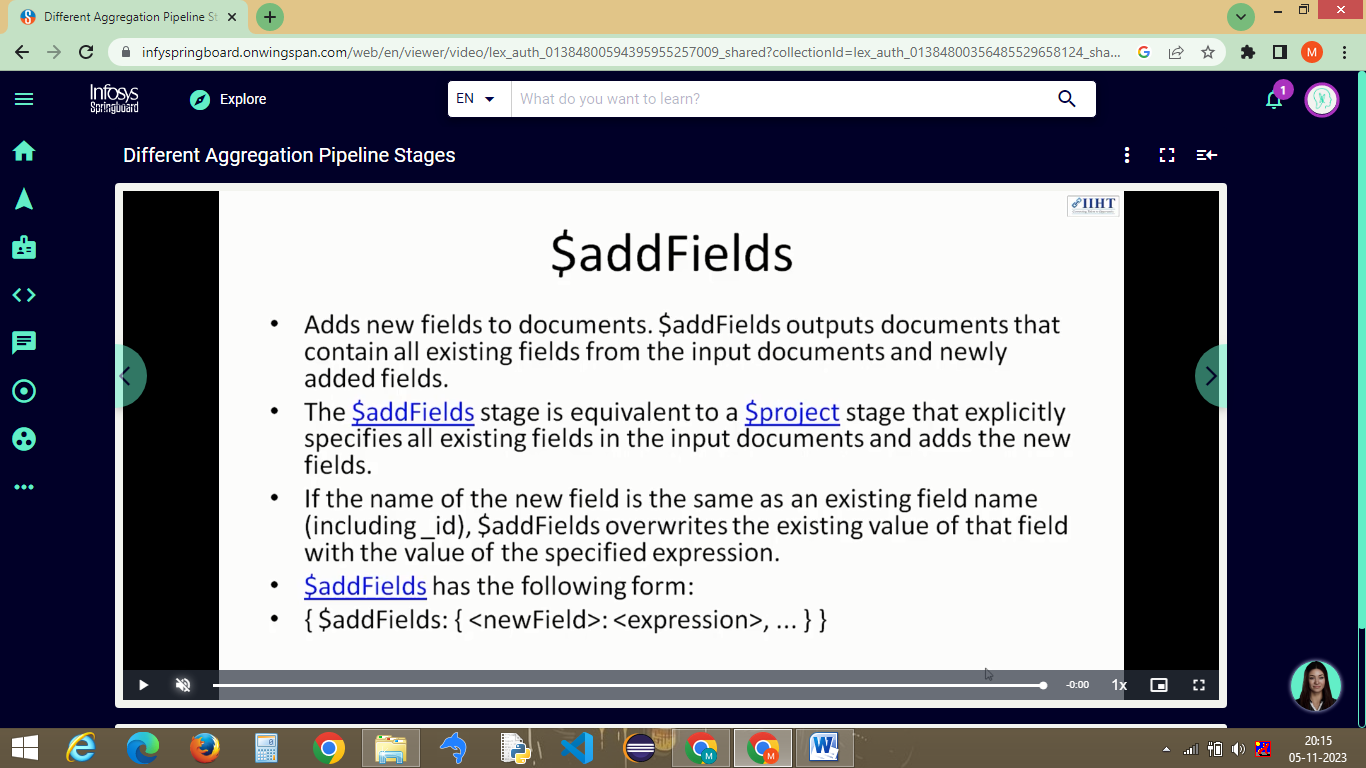
$bucket and $bucketAuto: These stages group documents into "buckets" based on specified criteria, such as ranges of values, and perform aggregations within those buckets.

$sample: Randomly selects a specified number of documents from the input stream.

$graphLookup: Performs recursive graph searches on hierarchical data using a specified field's relationship.

$lookup: Performs a left outer join between documents from two collections based on a common field, allowing you to combine data from multiple collections in your aggregation.

Custom Expressions: You can use custom JavaScript expressions in the $addFields, $project, and other stages to perform complex calculations.



1. **Indexes The Keys to Speed**

4.1 What indexes do, how they re-stored, used and their general impact

>>> What indexes do.

Improving Query Performance: Indexes provide a way to quickly access specific data in a collection, allowing queries to return results much faster. Without indexes, MongoDB would need to perform a collection scan, which can be slow for large datasets.

Enforcing Uniqueness: You can create unique indexes to ensure that values in a field (or a combination of fields) are unique across documents in a collection. This is especially useful for maintaining data integrity.

Supporting Sort Operations: Indexes also support sorting operations, making it faster and more efficient to retrieve data in a specific order.

How Indexes are Stored:

Indexes in MongoDB are stored in data structures called B-trees. A B-tree is a balanced tree structure that allows for efficient searching and retrieval of data. Each index in MongoDB contains a mapping of indexed field values to the location of the corresponding documents in the collection.

How Indexes are Used:

When you run a query in MongoDB, the query planner may use one or more indexes to locate and retrieve the desired documents. The query planner considers the available indexes, the query's filter conditions, and sort orders to determine the most efficient way to execute the query. It aims to minimize the number of documents that need to be scanned and returned.

General Impact of Indexes in MongoDB:

Faster Query Performance: The primary benefit of indexes is improved query performance. Properly designed and well-utilized indexes can significantly speed up data retrieval.

Reduced Scanning: Indexes reduce the need for full collection scans, which can be time-consuming and resource-intensive, especially in large collections.

Data Integrity: Unique indexes ensure that certain fields (or combinations of fields) have unique values, helping to maintain data integrity.

4.2 How index properties can enhance or fine-tune existing indexes

>>>

Index Types: Single Field Indexes: These indexes are created on a single field in a document and are suitable for queries that filter, sort, or perform operations on that field.

Compound Indexes: These indexes are created on multiple fields and are useful for queries that involve multiple fields in the filter or sort conditions. The order of fields in a compound index is important and should match the query patterns.

Index Direction: By default, indexes are created in ascending order. You can specify a descending order for an index if your queries frequently require descending sorts on a field.

Sparse Indexes: Sparse indexes only include entries for documents that have the indexed field. This can save storage space when dealing with sparsely populated fields and improve index performance. Use the {sparse: true} option when creating an index to make it sparse.

TTL (Time-To-Live) Indexes: TTL indexes are used to automatically remove documents from a collection after a specified time period. They are useful for data that has a time-based expiration, such as logs or temporary data.

Partial Indexes: Partial indexes are created with a filter expression that defines which documents should be included in the index. This allows you to create indexes on a subset of documents based on specific criteria.

Collation: You can specify a collation for an index to control case-insensitive or accent-insensitive sorting and comparisons. This is useful for internationalization and localization requirements.

Geospatial Indexes: For geospatial data, you can create geospatial indexes (e.g., 2dsphere or 2d indexes) to optimize queries related to location-based data.

Text Indexes: Text indexes are used for text search queries and can be customized with various options for language-specific behavior.

Background Index Builds: You can create indexes in the background to avoid locking the collection during the index creation process. This can help prevent disruptions to your application's normal operation.

Index Size and Memory: Consider the trade-off between the size of indexes and available memory. Very large indexes may not fit entirely in memory, leading to slower query performance. You can use the collMod command to set index storage options.

Query Analysis: Regularly analyze query performance to identify which queries benefit from indexes and which indexes can be enhanced or dropped. You can use the explain() method to analyze query plans and identify index usage.

Index Optimization: Monitor the performance of your queries and consider optimizing indexes by adding, removing, or modifying them based on actual usage patterns.

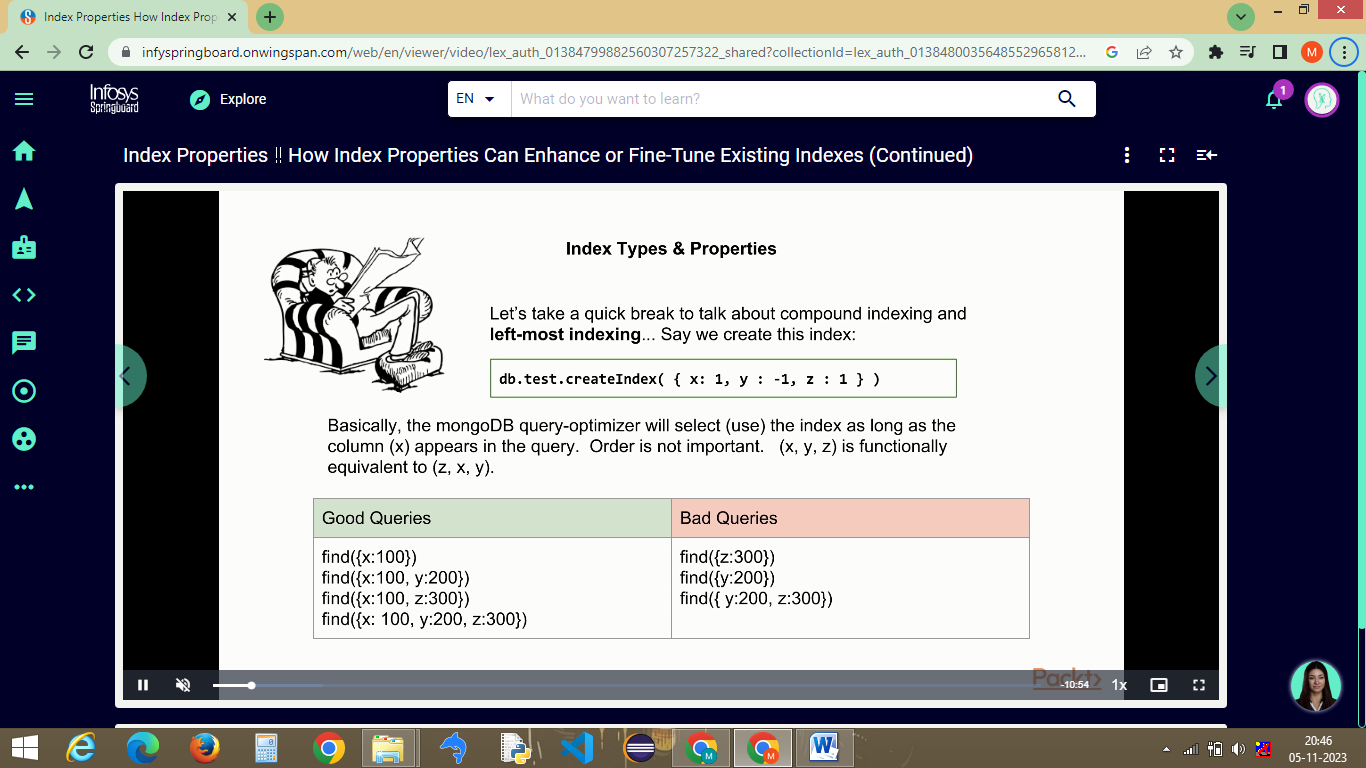


Fig 4.1 : Index properties

4.3 Index Management

>>> Index management is a critical aspect of database administration in MongoDB. Properly managing indexes can have a significant impact on query performance, while inefficient or unnecessary indexes can lead to increased storage and maintenance costs.

Index Creation: Create indexes based on the specific queries your application needs to optimize. Use the createIndex() method or the ensureIndex() method to create indexes on the desired fields in your collections. Consider the type of indexes (single field, compound, geospatial, text, etc.) that best suit your query patterns.

Index Analysis: Regularly analyze query performance to identify which queries are being used, and which indexes are being utilized. You can use the explain() method to understand the query execution plan and determine if indexes are effective.

Index Optimization: Consider removing unused or unnecessary indexes to reduce storage overhead and improve write performance. An index that's rarely used may not be worth maintaining. Use the dropIndex() method to remove indexes.

Index Monitoring: Use tools or database monitoring solutions to keep an eye on the state of your indexes. MongoDB provides server status information that can be used to monitor index usage.

Index Maintenance: Indexes need to be maintained as data is inserted, updated, or deleted. MongoDB automatically manages index maintenance for the most part, but you should be aware of the implications of index maintenance on write performance.

Background Index Building: Consider building indexes in the background to avoid locking the collection during the index creation process. This can be particularly important for large collections or collections with high write activity.

Index Storage Options: Adjust index storage options to control the size and memory usage of indexes. You can use the collMod command to set index storage options, such as setting a maximum index size or specifying storage options.

Compound Index Order: Pay attention to the order of fields in compound indexes. The order should match the query patterns to maximize index efficiency. Experiment with different index orderings if needed.

Index Compression: Consider enabling index compression to reduce the storage requirements of indexes while maintaining query performance.

Index Rebuilds: If you need to change the options or properties of an existing index, you may need to rebuild the index using the reIndex() method.

Index Naming: Give meaningful names to your indexes to make it easier to identify their purpose and usage.

Index Locks: Be aware that index builds and maintenance can introduce locks on the collection, impacting the concurrency of write operations. Plan for maintenance windows if necessary.

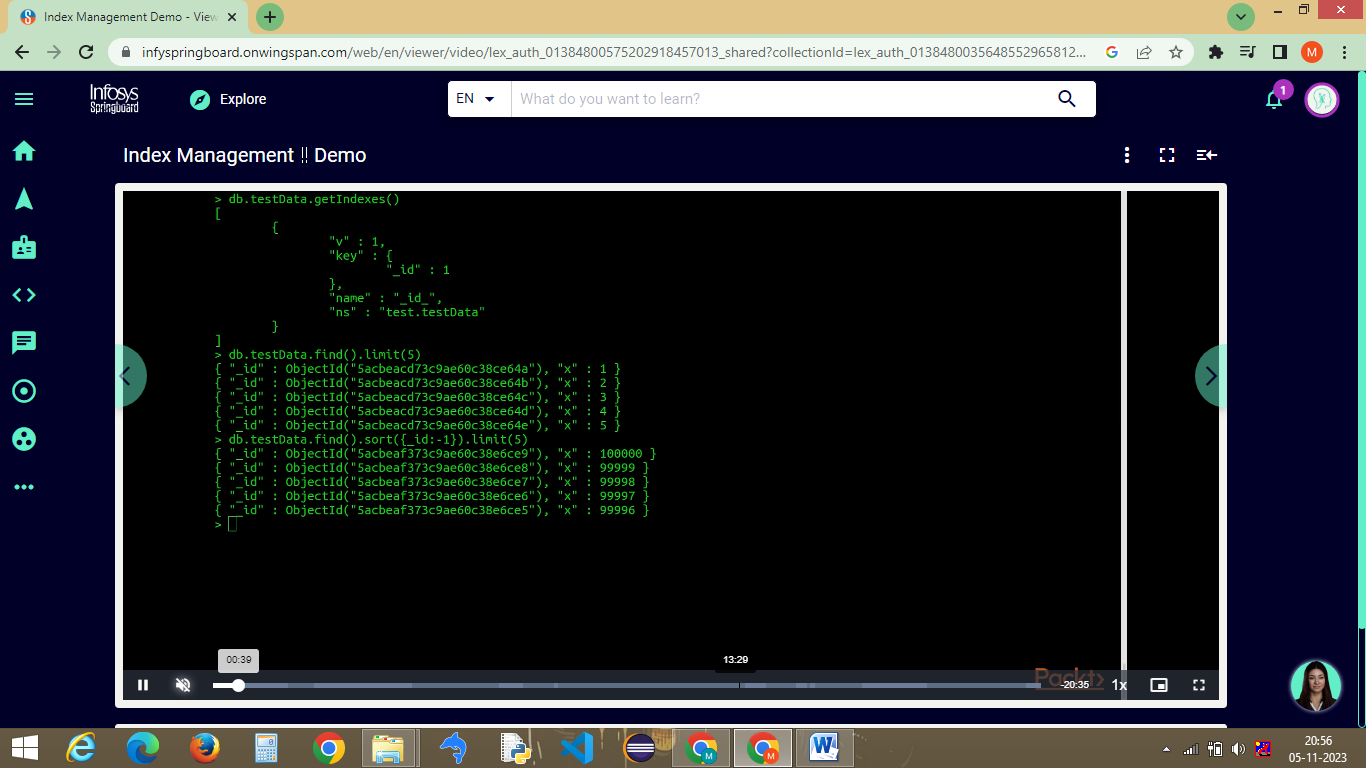


Fig 4.2 : Index Management

1. **CRUD**

5.1 Update !! How to Update Existing Records within a single query

>>> In MongoDB, you can update existing records using the update() or updateMany() method depending on whether you want to update a single document or multiple documents within a collection.

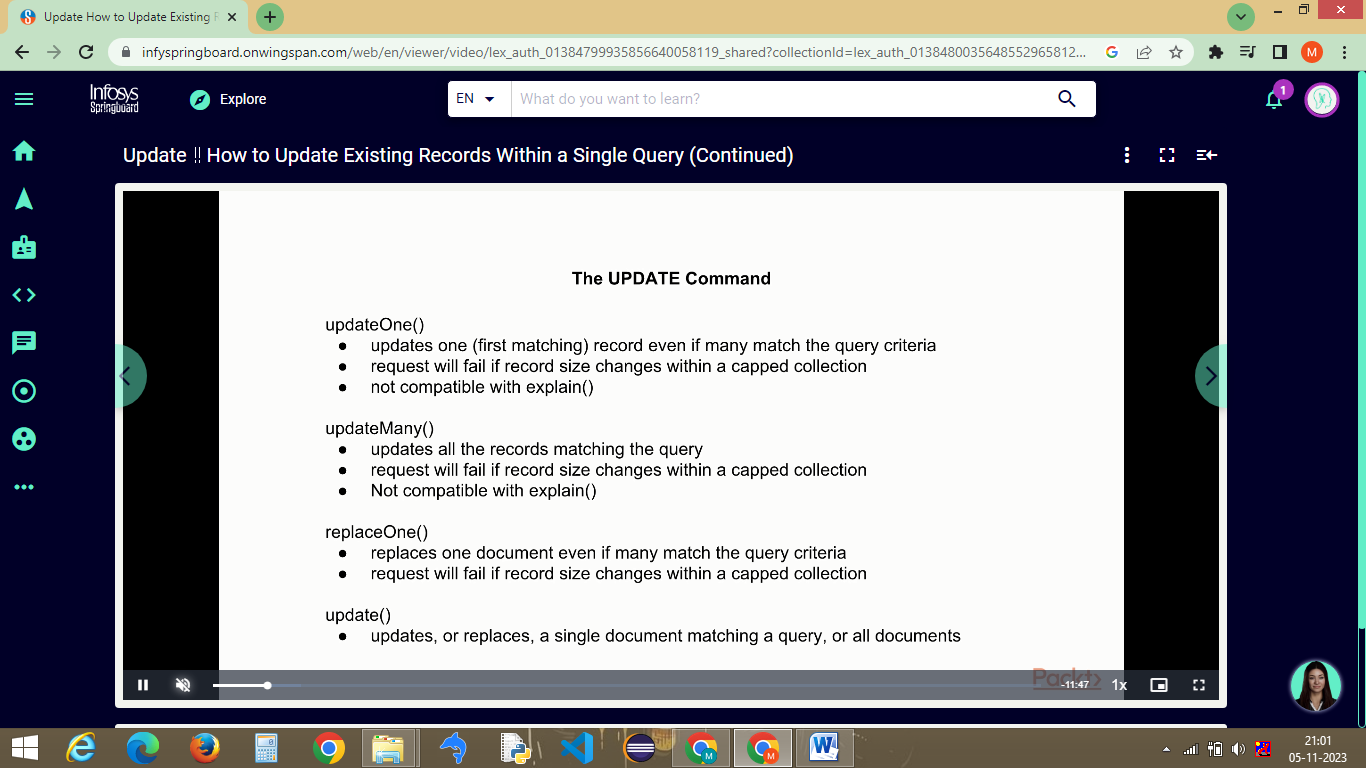


Fig 5.1 : Update commands

5.2 How to Permanently remove Data from yor collection.

>>> To permanently remove data from a collection in MongoDB, you can use the deleteOne() or deleteMany() methods to delete one or multiple documents, respectively. These operations remove documents from the collection, and the data is permanently deleted.

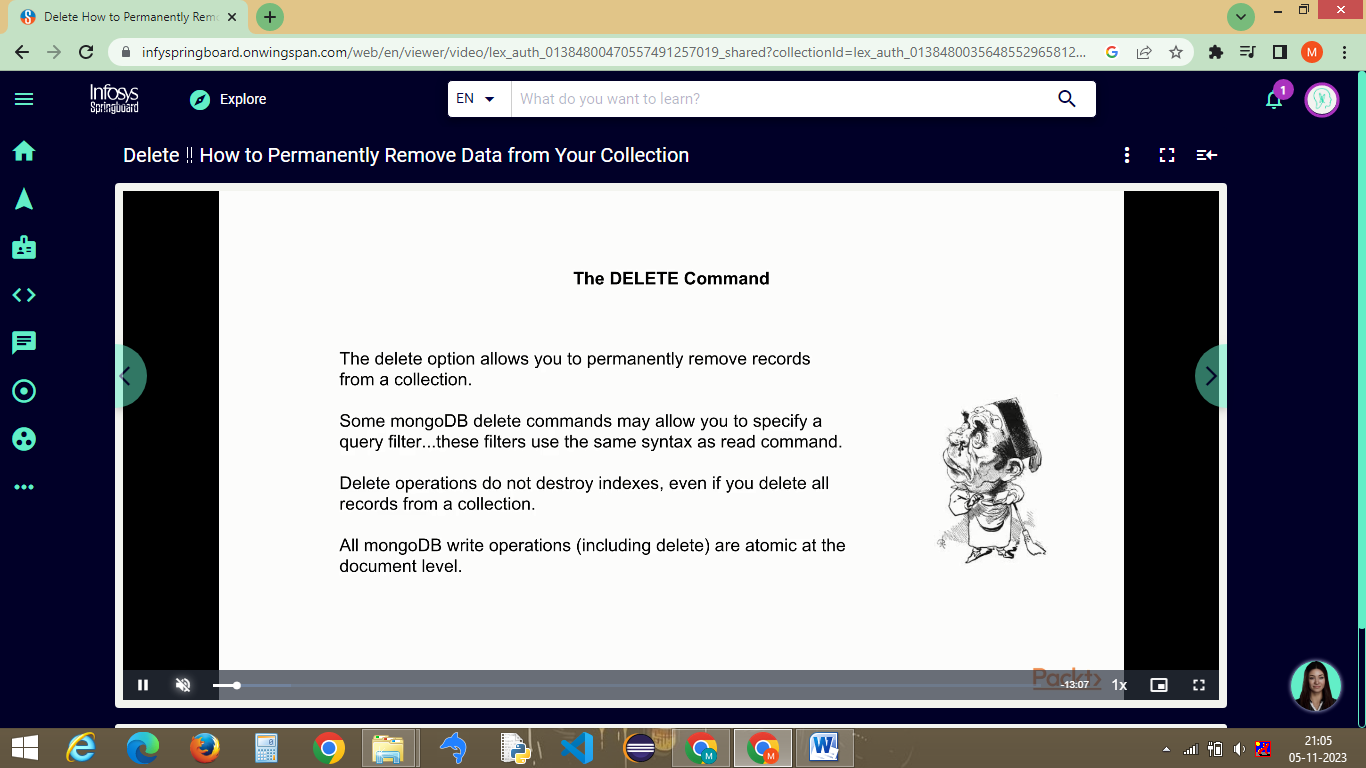


Fig 5.2 : Delete a data from a collection

5.3 Delete command