Setting Up and Operating on MongoDB

simplilearn

A Day in the Life of a MERN Stack Developer

In this sprint, Joe handles an important project of a telecom company where he must work with MongoDB.

He has to create a database of call records for the telecom company and write a program to analyze them.

In this lesson, we will learn how to solve this real-world scenario to help Joe complete his task effectively and quickly.

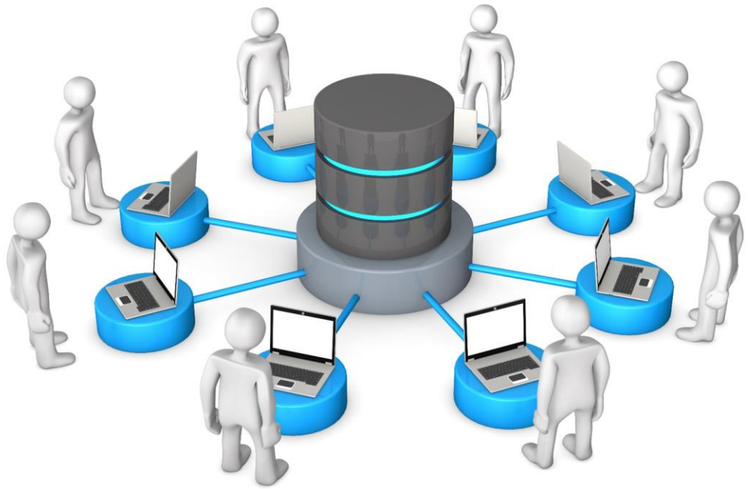
Learning Objectives

By the end of this lesson, you will be able to:

- List the differences between RDBMS and NoSQL
- Explain MongoDB structure
- Demonstrate scaling, replication, and memory management
- Work with relationships in MongoDB
- Perform different operations on MongoDB

Introduction to NoSQL Database

Introduction to Database



- Database is an organized collection of structured data
- The main purpose of a database is to store, retrieve, and manage data
- It is controlled by Database Management System (DBMS)
- Data is organized in rows, columns, tables, and indexes to make it easily accessible

Database categories

OLTP

- Online Transaction Processing
- Relatively standardized and simple queries which return relatively few records
- Also known as RDBMS
- High processing speed

OLAP

- Online Analytical Processing
- Complex queries involved
- Processing speed depends on the amount of data

NewSQL

- Also known as Not only SQL (NoSQL)
- Unstructured database handling

SQL

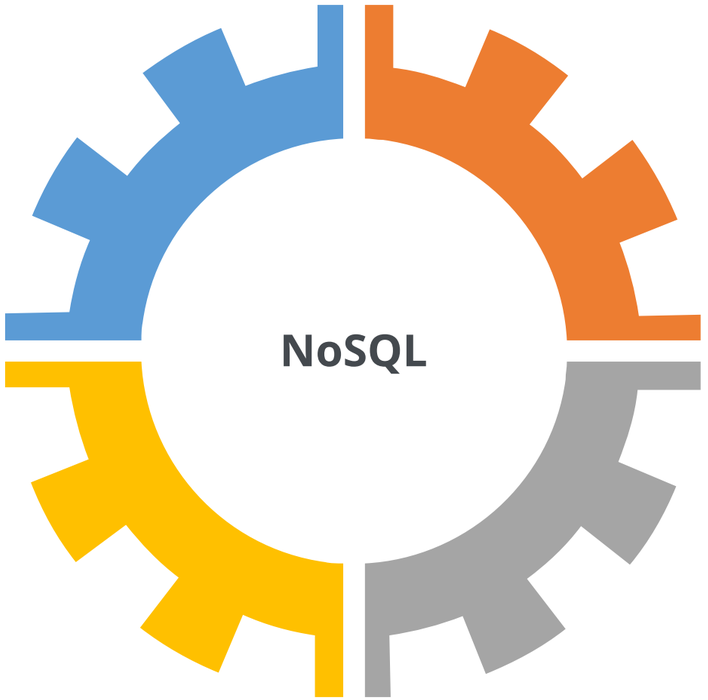
Structured query language is the standard language to deal with relational database

It is used to create, insert, search, update, and delete database records

It is used by database administrators and developers to write data integration scripts

NoSQL

It stores frequently demanded data into caches for a quick query execution



It is used to manage huge sets of unstructured data

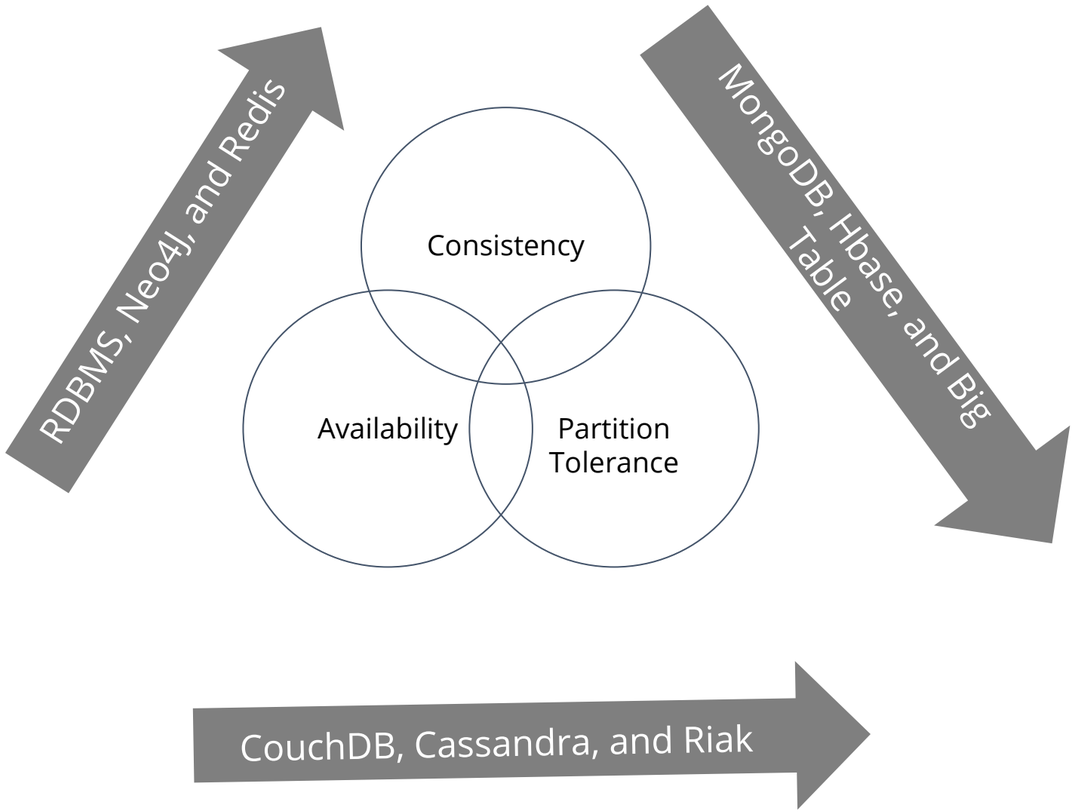
It supports auto replication by default

It supports sharding where large databases are partitioned into small, fast, and easily manageable ones

Difference between RDBMS and NoSQL

Properties	RDBMS or SQL	NoSQL
Developed in	1970	2000
Storage	Structured way of storing data It stores real-time data Short-term storage	Unstructured way of storing data It stores all application data Long-term storage
Schema	Fixed	Dynamic
Scaling	Vertical	Horizontal
Property followed	ACID	BASE
Examples	Oracle, MySQL	MongoDB, CouchDB
Uses	ATM or Retail transactions	Handling Big Data

CAP Theory



- CAP stands for Consistency, Availability, and Partition Tolerance
- For a NoSQL database, theoretically, it is not possible to fulfill all three requirements
- CAP theory provides the basic requirement for a distributed system to follow two out of three, C, A, or P

Advantages of NoSQL

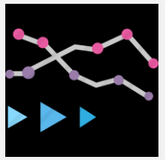
Volume

- Data at rest
- Large amount of data to process

Velocity

- Data in motion
- System will take milliseconds to seconds to respond to data streaming

Variability



- Data in many forms
- Structured data, unstructured data, and text

Veracity

- Data in doubt
- Uncertainty in data because of ambiguities and deceptions

Key Features of NoSQL

It handles semi-structured and structured data

It supports huge number of concurrent users

It is always available with no downtime

Types of NoSQL

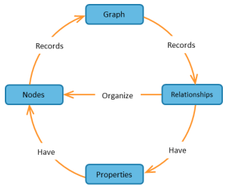
Document database

- Examples: MongoDB, CouchDB, and Cloudant

Key-value stores

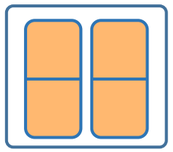
- Examples: Coherence, Redis, and Memcached

Graph stores



- Examples: Neo4J, Oracle NoSQL, and HyperGraph DB

Wide-column stores

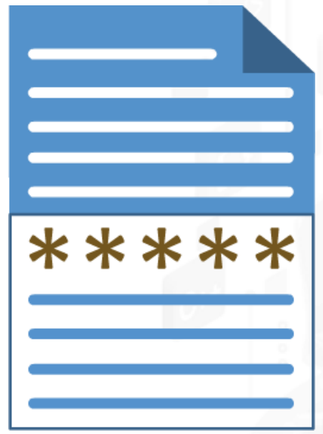


- Examples: Cassandra, Hbase, and Big Table

Document Database

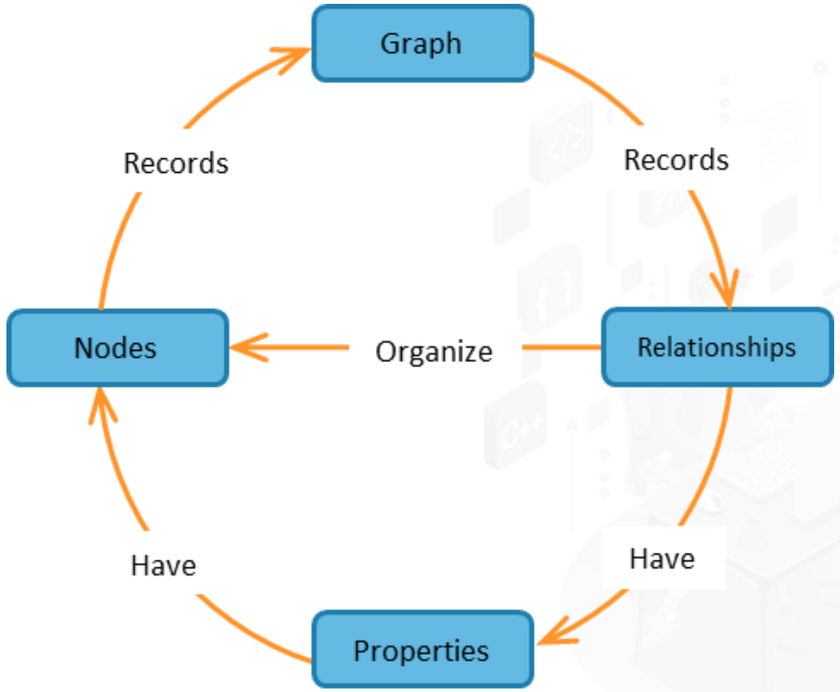
Based on the concept of documents, a document database:

- Stores and retrieves various documents in JavaScript Object Notation (JSON), Extensible Markup Language (XML), and Binary JavaScript Object Notation (BSON)
- Consists of maps, structures, and scalar values
- Stores documents in the value part of a key-value store



Graph Stores

A Graph database makes relationships readily available for any join-like execution allowing quick access of millions of connections. It lets you store data and its relationships with other data in the form of nodes and edges.



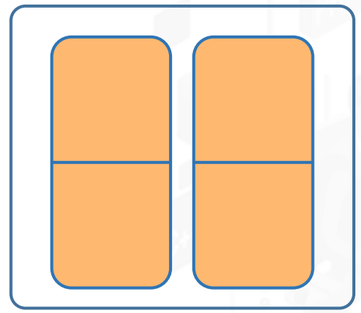
Key-Value Stores

Key-value stores are the simplest NoSQL databases that:

- Store every single item in the database as an attribute name (key) together with its value
- Are ideal for handling Web scale operations that need to scale across thousands of servers and millions of users with extremely quick and optimized retrieval

Wide-Column Stores

- Column-based databases store data in column families as rows
- These rows contain multiple columns, each associated with a row key
- In a column-based database:
  - The key identifies the rows
  - Rows do not need to have the same columns
- A column-based database reads and writes data to and from hard disk storage to quickly return a query. It allows you to access individual data elements as a group in columns

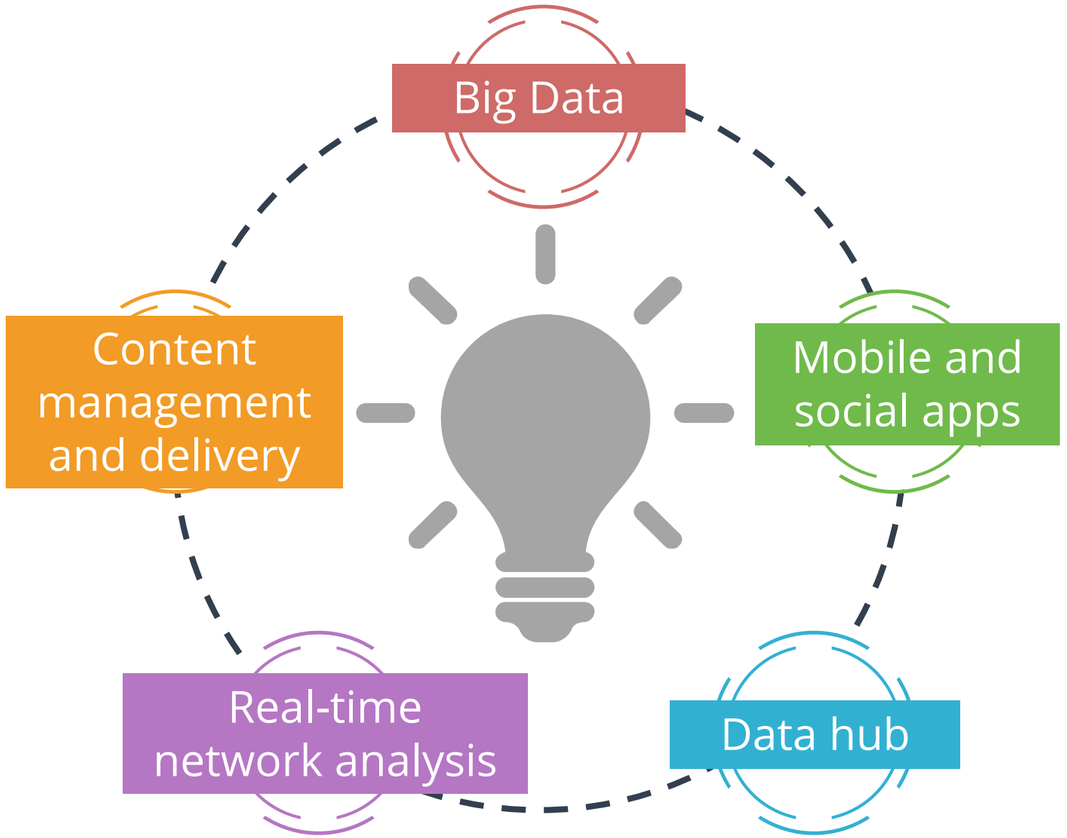


MongoDB: Database for Modern Web

Introduction to MongoDB

- It is a document based, non-relational database
- MongoDB database is written in C++ language
- It provides:
  - High performance
  - High availability
  - Easy scalability

Uses of MongoDB



MongoDB Users

Google

NOKIA

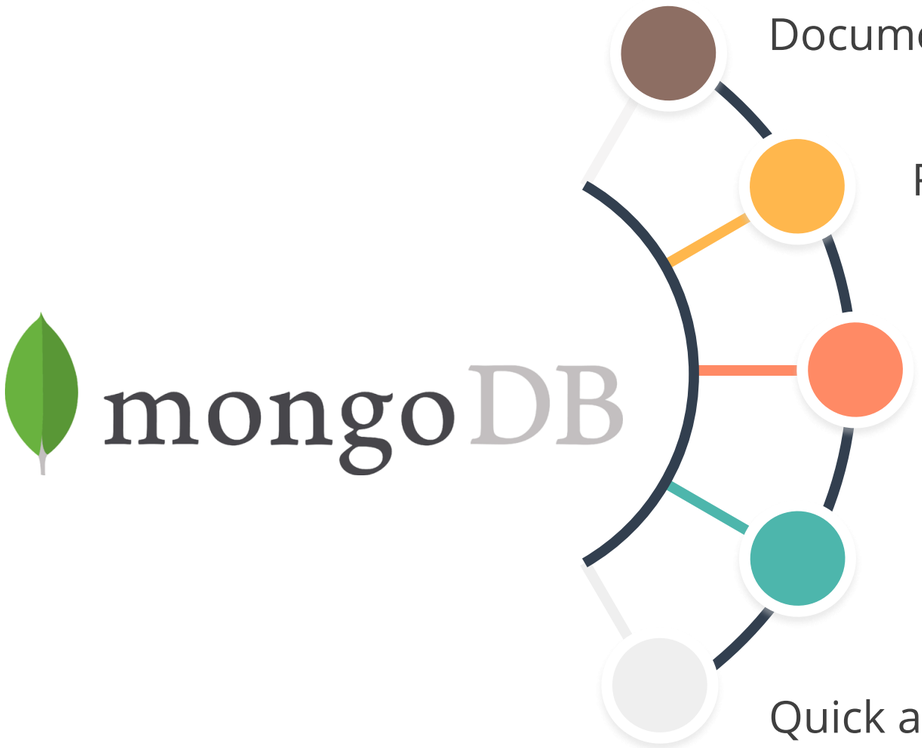
Adobe

Forbes

SAP

MetLife

Why Use MongoDB

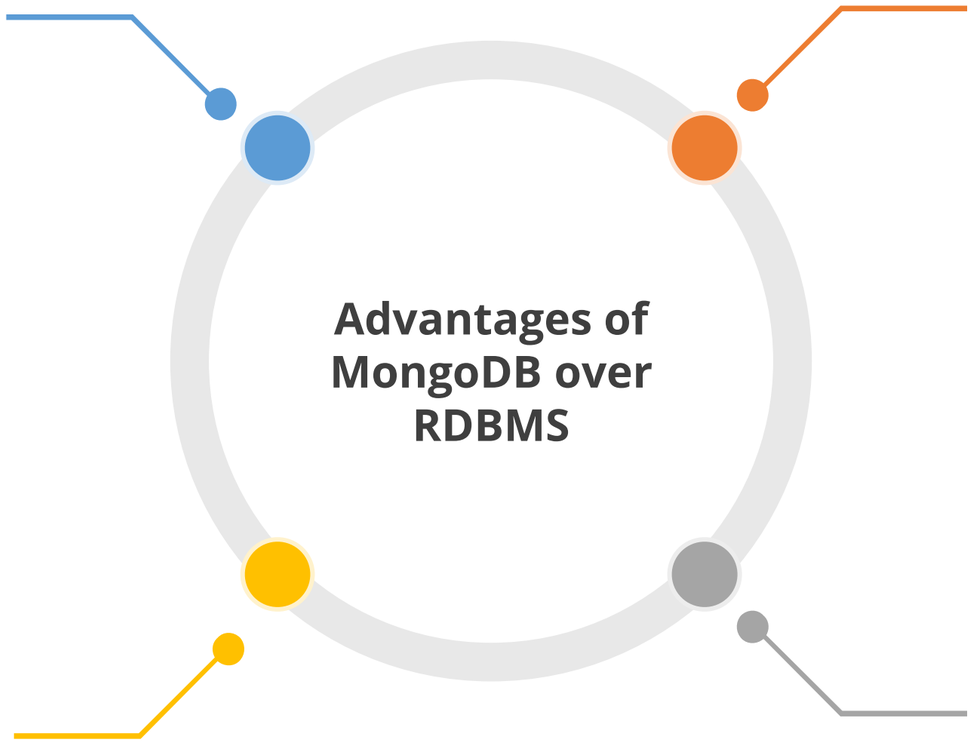


- Document-oriented storage
- Rich and dynamic query support
- Replication and high availability
- Auto-sharding
- Quick access to data using internal memory

Advantages of MongoDB over RDBMS

Enables quick access to data using internal memory

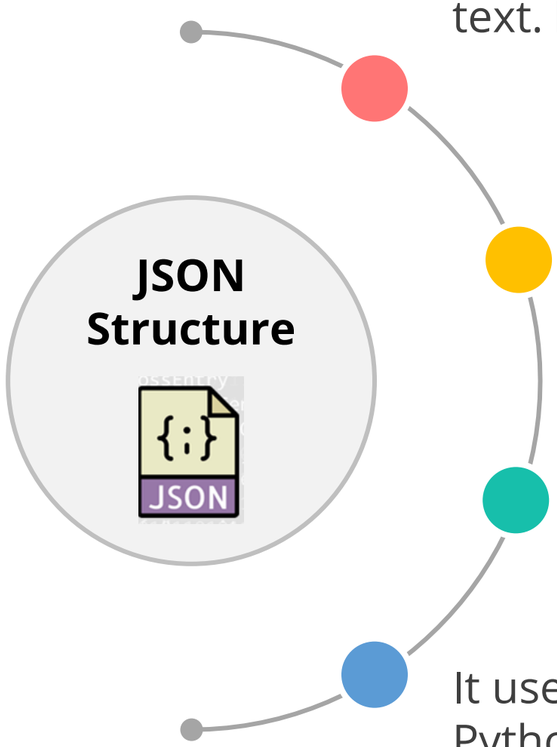
Is schema-less



Is easy to scale and tune and structure of a single object is clear

Does not require complex joins

JSON Structure



JavaScript Object Notation (JSON) is a standard file format that uses human-readable text. It can be easily transmitted from one system to another.

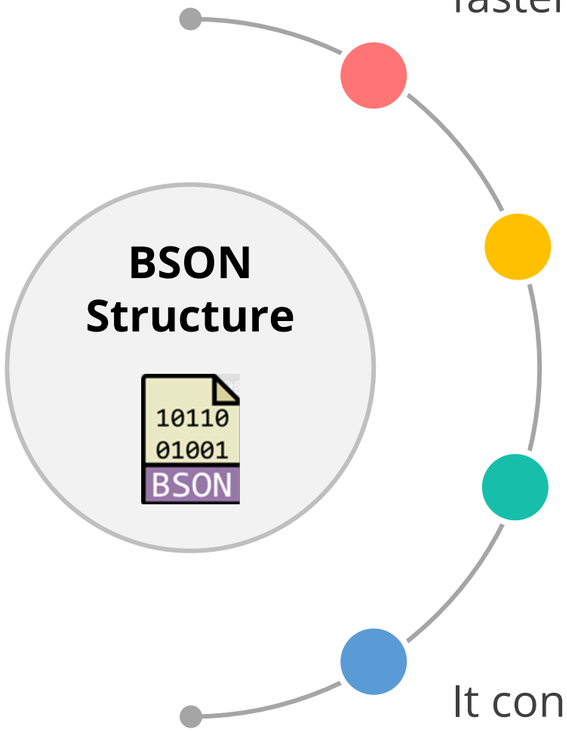
It is used by web services and APIs to provide public data.

It has two things, object and array. An object is a collection of key-value pairs and an array is an ordered list of values.

It uses conventions of the C-family languages: Java, JavaScript, Perl, and Python. It consumes less space.

BSON Structure

BSON is nothing but Binary JSON. It is not in a readable format. BSON is faster than JSON.



It is used by MongoDB while storing documents in collections.

It is easy for machines to parse and generate.

It consumes more space.

BSON Properties

Lightweight

- When used over the network, BSON keeps the overhead involved in processing extra header data to a minimum

Traversable

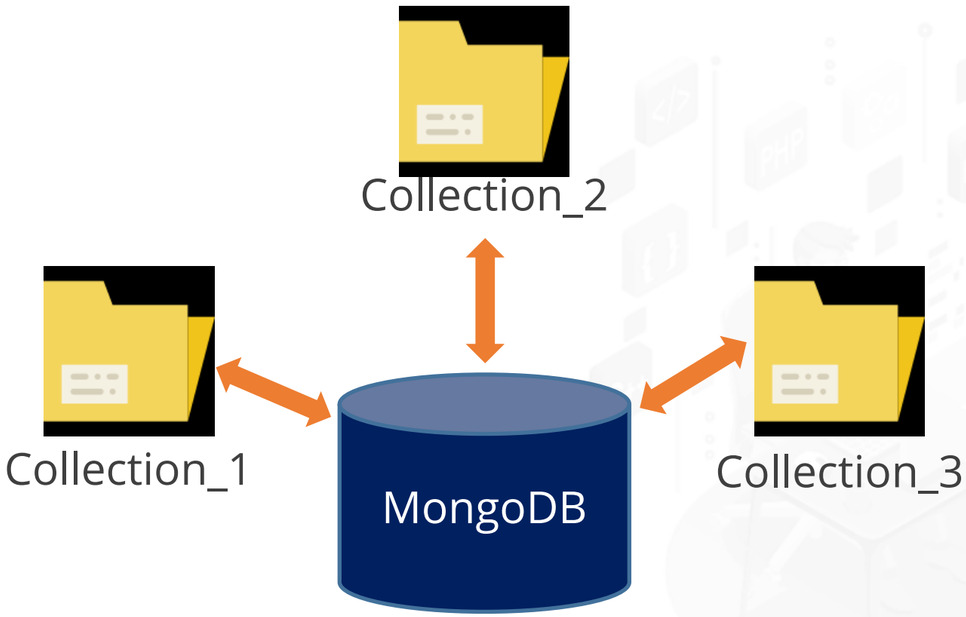
- It is designed to traverse easily across network. This helps in its role as the primary data representation for MongoDB

Efficient

- It uses C data types that allow easy and quick encoding and decoding of data

MongoDB Structure

- A single MongoDB has several databases
- A database is a container to store collections
- A collection is a group of similar or related documents within a single database
- A document can have multiple fields



MongoDB GUI Tools

There are various MongoDB tools that are used to deal with NoSQL queries. A few examples are listed below:

1. Studio 3T
2. Robo 3T
3. MongoDB Compass
4. NoSQL Booster
5. NoSQL Manager
6. Cluster Control and many more.

Studio 3T

Studio 3T is a popular MongoDB GUI tool used for handling MongoDB queries and operations. This tool allows the users to work with shards and replica sets.

Features of Studio 3T:

- Has rich query autocompletion with IntelliShell
- Uses SQL with INNER and OUTER joins to query MongoDB
- Generates driver-code from SQL or mongo shell
- Secures connections for single MongoDB instances and replica sets
- Effortlessly compares and synchronises data
- Builds aggregation queries stage by stage

Robo 3T

- Robo 3T is a light weight and open-source tool that has cross platform support that provides both shell and GUI interaction.
- It has an asynchronous and non-blocking UI.

MongoDB Compass

- MongoDB compass is the MongoDB GUI tool used to work with MongoDB queries and operations.
- It analyses documents and displays rich structures inside the GUI.
- Features of MongoDB compass are:
  - Allows visual exploration of data
  - Allows easy management of indexes
  - Makes aggregation easy using intuitive UI
  - Allows to view query performance
  - Follows a better approach in CRUD that makes it easier to interact with the data

Setting Up MongoDB

Duration: 60 min.

Problem Statement:

You are given a project to set up a MongoDB environment.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to set up a MongoDB environment:

1. Download the MongoDB MSI installer package
2. Install MongoDB with the installation wizard
3. Create a data folder to store databases
4. Set up shortcuts for Mongo and Mongod

Setting Up MongoDB

Duration: 50 min.

Problem Statement:

You are given a project to set up a MongoDB environment.

UNASSISTED PRACTICE

Unassisted Practice: Guidelines

Steps to set up a MongoDB environment:

1. Download the MongoDB MSI installer package
2. Install MongoDB with the installation wizard
3. Create a data folder to store databases
4. Set up shortcuts for Mongo and Mongod

MongoDB as a Document Database

MongoDB as a Document Database

The data model in MongoDB is document based, which does not confirm to any prespecified schema. Differences between a relational database and MongoDB are as follows:

Relational Database	MongoDB
Rows are stored in a table and each table has a strictly defined schema that specifies the permitted types and columns. If a row in a table requires an extra field, the entire table needs to be altered.	It groups documents into collections that do not follow any schema. Each individual document in a collection can have a completely independent structure.

A schema-less model lets you represent data with variable properties.

Transaction Management in MongoDB

Transactions in MongoDB are as follows:

- MongoDB supports single-phase commit at each document level
- A write operation in a single document is considered atomic in MongoDB even if the operation alters multiple embedded documents

A write operation is atomic, but the entire operation is not atomic; other operations may interleave.

Document Store: Example

Duration: 30 min.

Problem Statement:

You are given a project to demonstrate how to create a database in MongoDB.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate document store:

1. Set up MongoDB server and shell
2. Write a program to create a database and display the data present in the collection of the database

Scaling, Replication, and Memory Management

Scaling

Size of a data set is growing with advanced technology. This has created a demand for:

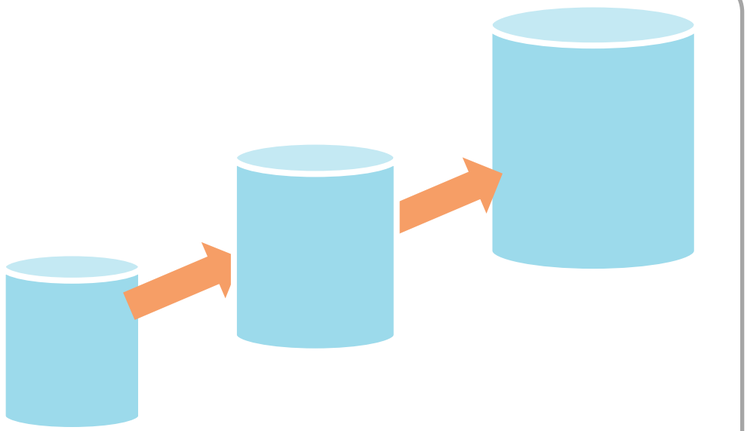
- More data storage with capacity to handle more data
- Scaling databases for size and performance

Scaling of a database

Scaling up vs. Scaling out

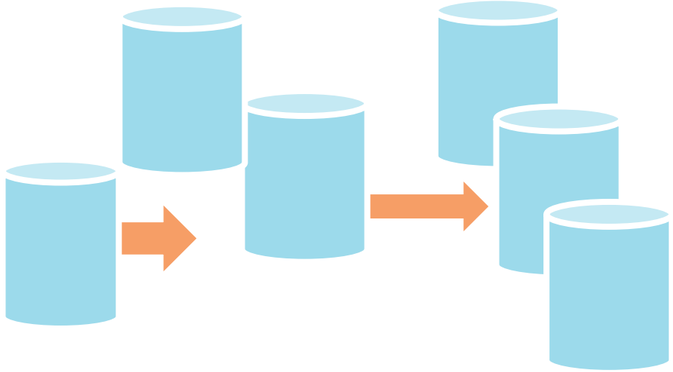
Scaling a database involves two choices:

Scaling up



- Easy but costly affair
- Even the most powerful database may not be able to manage growing volumes of data

Scaling out



- Extensible and cost-effective
- Commodity server can be added to increase storage and enhance performance

Vertical Scaling

A database can be scaled by upgrading hardware. The technique of enhancing a single-node hardware is called vertical scaling or scaling up. Vertical scaling is:

- Simple
- Reliable
- Cost-effective to some extent

! If your database is running on a physical hardware and the cost of a more powerful server is not permitted, consider horizontal scaling.

Horizontal Scaling

Horizontal scaling distributes a database across multiple machines. Advantages include:

- Commodity hardware is used
- Risk of failure get mitigated
- Failure is less disastrous because a single machine is only a small part of the entire system

MongoDB supports horizontal scaling and uses a range-based partitioning mechanism called **auto-sharding** which:

- Automatically manages data distribution across multiple nodes
- Allows addition of new nodes
- Is transparent to the client

Secondary Indexes

- MongoDB implements multiple secondary indexes as B-trees that can be optimized for range scan queries and queries with sort clauses
- MongoDB can create up to 64 indexes per collection and supports indexes, such as ascending, descending, unique, compound-key, and geospatial

MongoDB uses the same data structure for indexes as most RDBMSs.

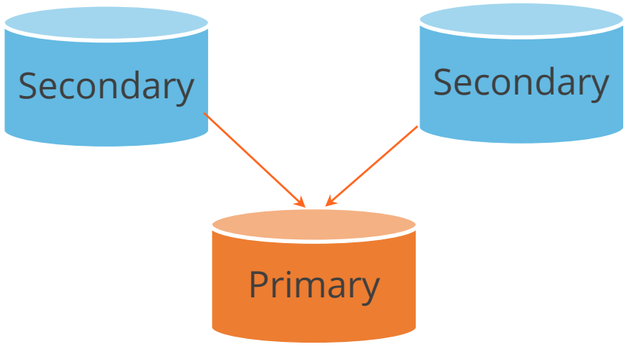
Replication

MongoDB uses replica sets to create database replication. A replica set performs the following actions:

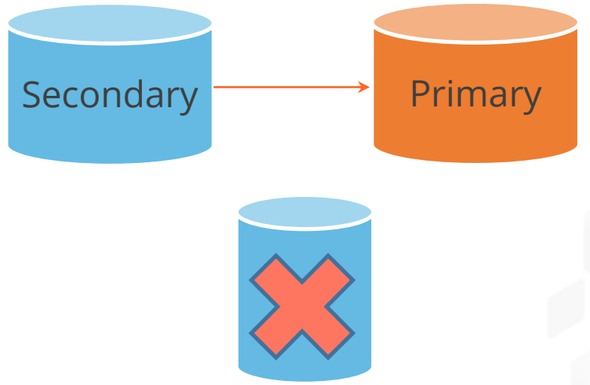
- Distributing data across various MongoDB nodes, also known as shards, for redundancy
- Automating failover when server or network outages occur
- Scaling database reads

A replica set contains one primary node and one or more secondary nodes. The primary node supports both read and write, whereas the secondary supports read.

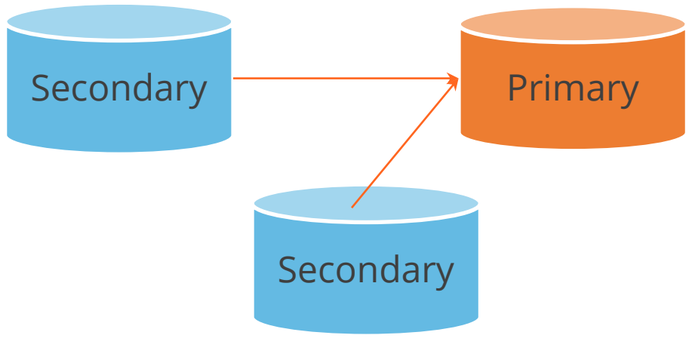
Replication



A working replica set



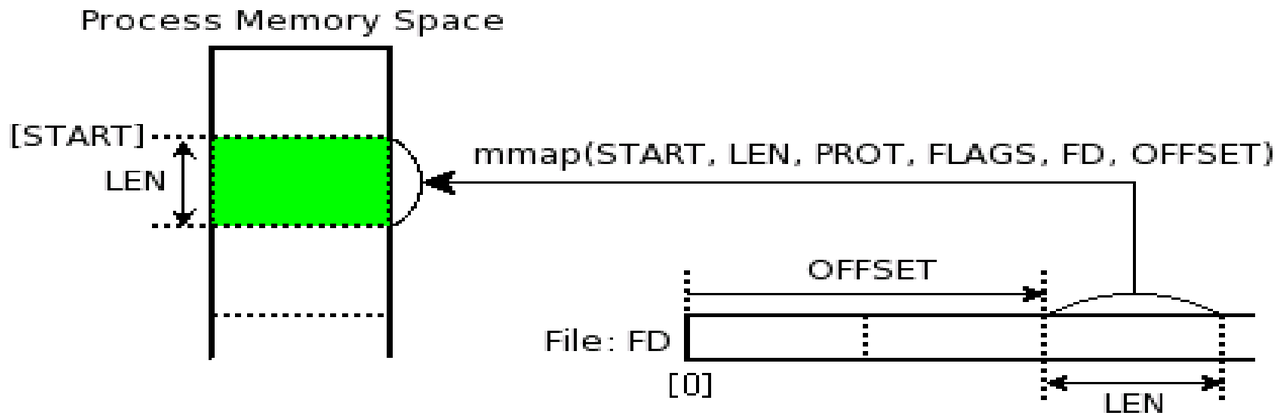
Secondary node is converted to primary when the primary fails



Original primary node comes back as secondary

Memory Management

MongoDB stores data in memory-mapped files and uses the entire system memory to operate fast. MongoDB allows its operating system to manage its memory which impacts its performances and operations.



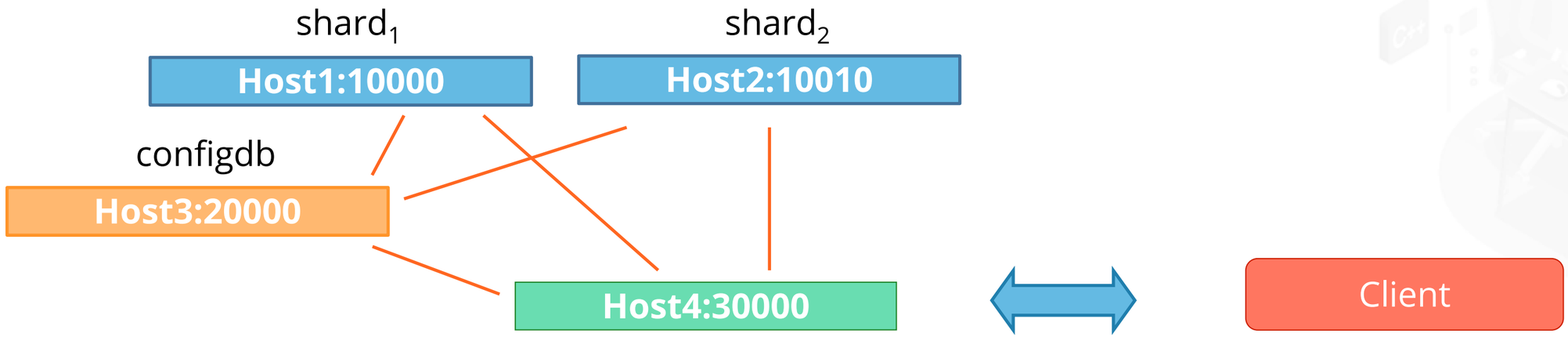
Replica Set

The replica set feature in MongoDB facilitates redundancy and failover with the following actions:

- When a master node of the replica set fails, another member of the set is converted to the master
- It can choose a master or a slave depending on whether you want a quick or a delayed consistency
- It keeps replicated data on nodes belonging to different data centers to protect the data in case of natural disasters

Auto-Sharding

- MongoDB uses sharding for horizontal scaling. For automatic sharding, choose a shard key that determines distribution of the collection data
- MongoDB is spread over multiple servers and performs load balancing and/or data duplication to keep the system up and running in case of hardware failure

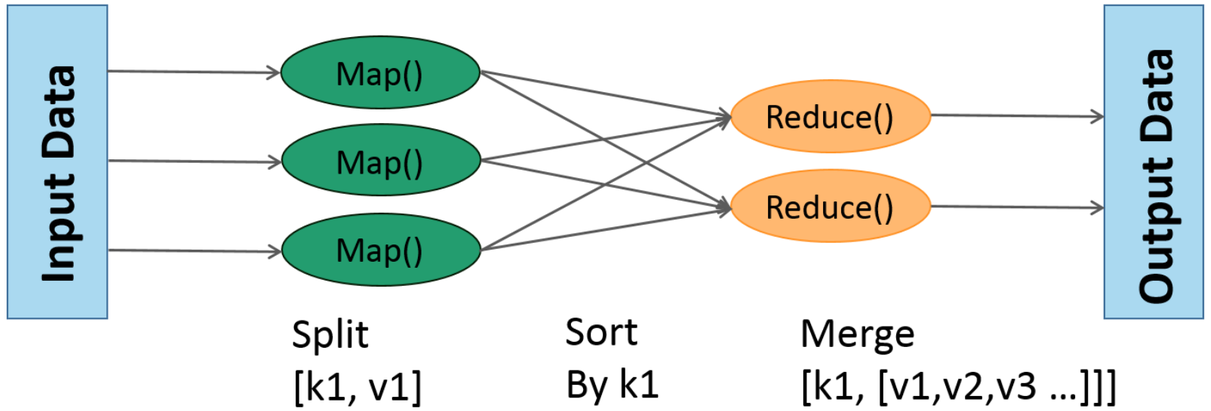


Aggregation and MapReduce

Aggregation operations analyze data sets and return calculated results.

A MapReduce operation consists of two phases:

- **Map**: Documents are processed, and one or more objects are produced for each input document
- **Reduce**: Outputs of the map operation are combined



MapReduce can define a query condition to select input documents and sort and limit the results.

Collection and Database

- A collection is a group of documents not restricted by any schema. MongoDB groups collections into databases. A single instance of MongoDB can host several independent databases.
- Developers and administrators find it difficult to handle different types of documents in the same collection. Developers need to ensure that their queries retrieve specific documents or their application code handle documents of different structure.

Core servers of MongoDB

The core database server of MongoDB can be run on an executable process called mongod or mongodb.exe on Windows.
A mongod process can be run on several modes:

- Replica set: Configurations comprise two replicas and an arbiter process that reside on a third server
- Per-shard replica sets: Auto-sharding architecture of MongoDB consists of mongod processes configured per shard replica sets
- Mongos: A separate routing server is used to send requests to the appropriate shard

Mongos send queries from the application layer and locate the data in the sharded cluster to complete their operations.

MongoDB Tools

MongoDB tools consist of the following:

- **The JavaScript shell:** The MongoDB command shell is a JavaScript-based tool used for administering a database and manipulating data

- **Database driver:** It provides an Application Program Interface (API) that matches the syntax of the language used

- **Command-line tools:**
  - mongodump and mongorestore: Standard utilities that back up and restore a database
  - mongoexport and mongoimport: Tools used to export and import JSON, comma-separated value (CSV) and tab-separated value (TSV) data
  - Mongosniff: A wire-sniffing tool used for viewing operations sent to a database
  - Mongostat: It provides helpful statistics, such as inserts, queries, updates, and deletes

Relationships in MongoDB

Data Models to Create Relations in MongoDB

Embedded Data Model

- In this method, BSON document is embedded within another document
- Queries run faster here than in documented reference data models

Documented Reference Data Model

- In this model, the BSON document is referred from another document
- This method reduces error and keep data consistency

Embedded Data Model

- Embedded documents store related data in a single document structure and thus, capture relationships between data. MongoDB allows denormalized data models to permit retrieval and manipulation of related data in a single database operation

- Embedded data models can be used when the following relationships exist between entities:
  - contains
  - one-to-many

Embedded data models update related data in a single atomic write operation.

Embedded Data Model: Example

For example:

```
{
        _id: “123”,
                coursedetail:
        {
                Topic:  “MongoDB ”,
                Duration : 24,

        }

        instructor:
                {
                name: “XYZ”,
                YOE: 5,
        }

}
```

Reference Data Model

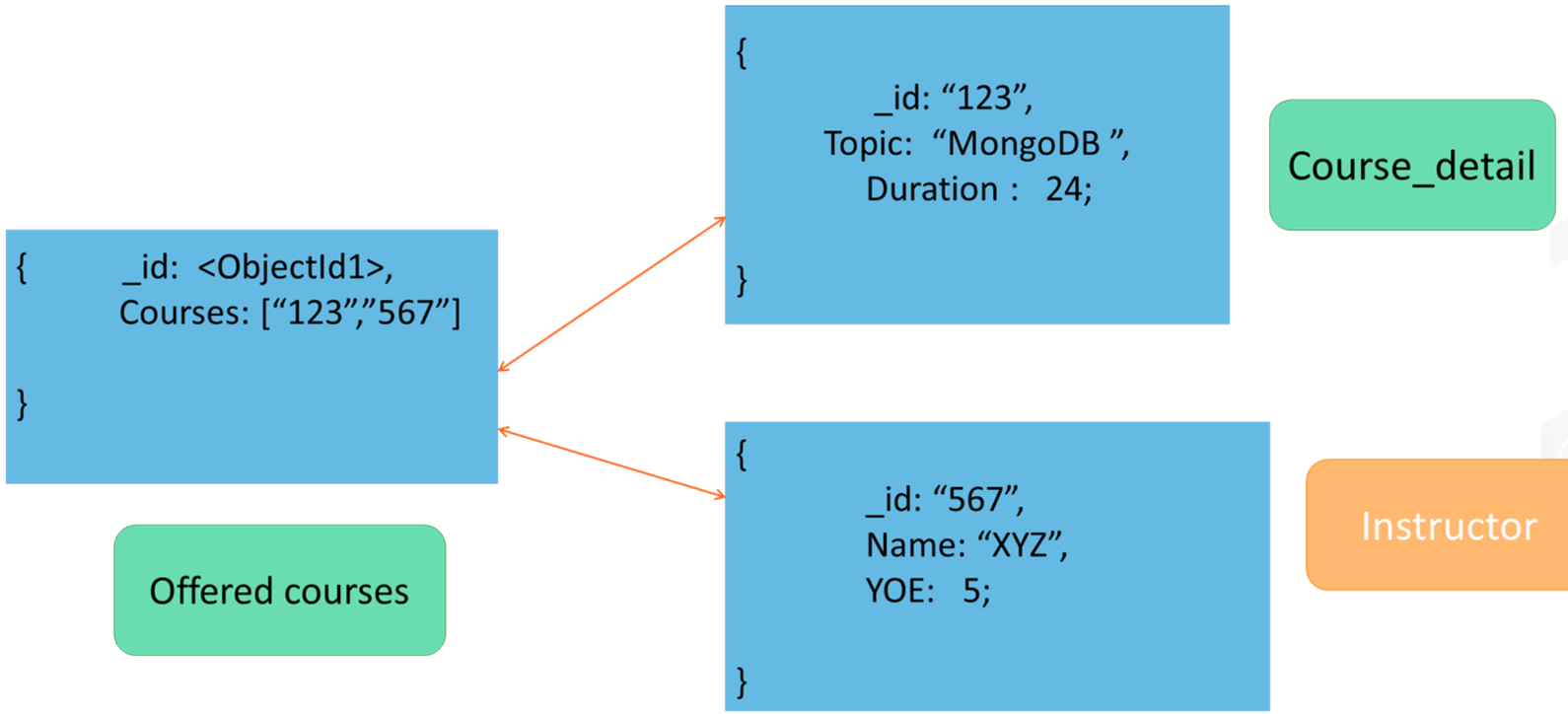
- Data model design requires two important things, structure of documents and representation of data relationships. References and embedded documents allow applications to represent data relationships

- References use links to store data relationships and applications use these references to access the related data

- These are normalized data models, which you can use:
  - When embedding document results in data duplication does not perform well
  - To represent complex many-to-many relationships
  - To model large hierarchical data sets

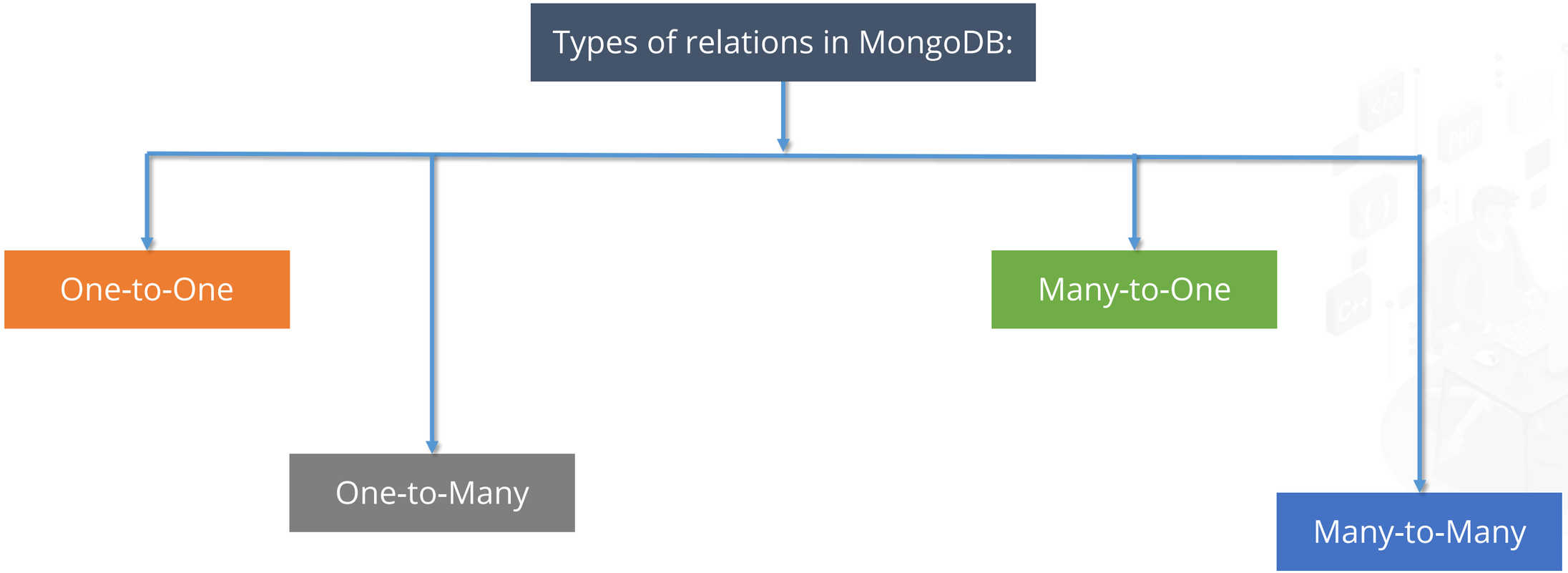
Normalized data models can send more database queries from client to the database server.

Reference Data Model: Example



Relations in MongoDB

Relations in MongoDB are ways to show how one document interacts with another.



One-to-One

In this relation, the parent document has one child, and the child has one parent.

For example:

```
db.employees.insert(
{
        _id : 2,
        empname : "XYZ",
        address : {
                street : "Bellandur Road",
                city : "Bangalore",
                state : "Karnataka",
                country : "India"
                }
        }
)
```

One-to-Many

In this relation, the parent document can have many child documents in it, but child document can have only one parent.

For example:

```
db.employees.insert(
{
        _id : 3,
        empname : “PQR",
        projects : [
        {
                project : "DEF",
                release : 2000,
                language : “Java"
        },
        {
                project : "ABC",
                release : 2013,
                language : “Python"
        }
        ]
        }
)
```

Many-to-Many

In this type of relation, two documents may have multiple relationships between each other.

For example:

```
db.employees.insert(
{
        _id : 8,
        empname : “LMN",
        roles:[“Program Manager”, “Employee”, “Lead”]        }
)
```

Operations in MongoDB

Data Modification in MongoDB

- Data modifications in MongoDB involve creating, updating, or deleting data. These operations modify the data of a single collection. The **insert()** method is used to insert a document into a MongoDB collection.
- An example of the Insert query is given below:

For example:

```
db.courses.insert({Name:“SimpliLearn",
Address:" 10685 Hazelhurst Dr, Houston, TX 77043, United States”,
Courses: [“Big Data”,”Python”,”Android”,”PMP”,”ITIL”],
Offices: [ “NYK”,”Dubai”,”BLR”]
 });
```

Batch Insert in MongoDB

A batch insert allows to store multiple documents in a database at a time. Characteristics of a batch insert:

- Sends a large number of documents to a database in a batch at a time
- Operates faster than other operations
- Reduces insert time by eliminating header processing activities
- Is used in applications to store server logs and sensor data
- Limits inserts to 16 MB in a single batch insert

Ordered Bulk Insert

MongoDB groups ordered list operations by their type and contiguity. Characteristics of ordered bulk insert include the following:

- It serially executes write operations
- If an error occurs during one write operation, MongoDB returns the remaining ones
- Each group of operations can have maximum 1000 operations
- On exceeding 1000 operations, MongoDB divides the group into smaller groups of 1000 or less

For example:

```
var bulk = db.items.initializeOrderedBulkOp();
        bulk.insert( { _id: 1, item: “pen", available: true, soldQty:700} );
        bulk.insert( { _id: 2, item: “pencil", available:false, soldQty:900} );
        bulk.insert( { _id: 3, item: “books", available: true, soldQty: 600 } );
        bulk.execute();
```

Unordered Bulk Insert

In an unordered operations list, MongoDB can execute in parallel in a nondeterministic manner.
When performing an unordered list of operations, MongoDB:

- Groups and reorders operations for enhanced performance
- Further splits the groups when the number of operations crosses 1000 in each group

For example:

```
var bulk = db.items.initializeUnorderedBulkOp();
        bulk.insert( { _id: 1, item: “pen", available: true, soldQty:700} );
        bulk.insert( { _id: 2, item: “pencil", available: false, soldQty:900} );
        bulk.insert( { _id: 3, item: “books", available: true, soldQty: 600 } );
        bulk.execute( );
```

Inserts: Internals and Implications

Process of executing an Insert operation is as follows:

- Language drivers convert the data structure into Binary JSON (BSON) before sending to the database
- The database looks for '**_id'** key and confirms that the document has not exceeded 16 MB
- The document is saved to the database without any changes

MongoDB does the following before sending data to a database:

- Checking for invalid data
- Checking UTF-8 compliance of strings
- Filtering unrecognized data types

MongoDB is not vulnerable to traditional injection attacks, because it does not perform any code execution on inserts.

Retrieving Documents

MongoDB queries define the criteria or conditions for document retrieval. A query may also include a projection that defines the fields that match the document fields to return. You can modify queries to impose limits, skips, and sort orders.

```
db.items.find( {available: true },{item:1,} ).limit(5)
```

Specify Equality Condition

The FindOne() method finds the first record in a document. The pretty() method helps get properly formatted results. The query given below selects all documents in a collection and displays the result in proper format.

```
db.items.find().pretty()
```

An empty ({}) query document selects all documents in the collection. The query given below finds all documents in a collection.

```
db.items.find( {} )
```

=

```
db.items.find()
```

To specify an equality condition, use the query document { <field>: <value>}. The query below retrieves all documents having the available field value as true.

```
db.items.find( {available: true } )
```

$in, $or , and AND Conditions

You can specify query conditions using the following query operators:

- **$in**: It queries a variety of values for a single key

```
db.items.find( {available : { $in: [true, false ] } } )
```

- **$or**: It queries similar values as $in operator does. However, it is recommended to use the $in operator when performing equality checks on the same field

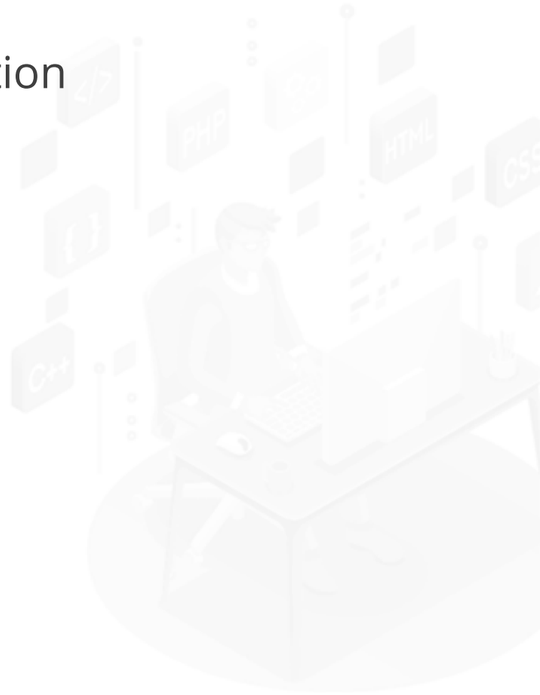
- **AND**: A compound query can specify conditions for more than one field in the collection's documents

```
db.items.find( {available: true, soldQty: { $lt: 900 } } )
```

$or Operator

- Returns a value true when any of its expressions evaluate to true or accepts any argument expressions
- Defines a compound query where each clause is joined with a logical OR conjunction

```
db.items.find({ $or: [ {soldQty : { $gt: 500 } }, { available:true } ] })
```



Specify AND/OR Conditions

Given below is an example of a query that selects all documents in a collection where:

- Value of the available field is true
- soldQty has a value greater than 200
- Value of the item field is “Book”

```
db.items.find({ available:true,$or: [ {soldQty : { $gt: 200 } }, {item: “Book” } ]})
```

The **“$not”** is a metaconditional operator. You can apply a **“$not”** to any other criteria. In the example given below, the **“$mod”** command queries the keys whose values, when divided by the first given value, show a remainder of the second value.

```
db.items.find({“_id" : {"$not" : {"$mod" : [4, 1]}}})
```

Regular Expression

- Regular expression string matches flexible items and performs case-insensitive matching.

```
db.items.find({item:/pe/i})
```

- You can modify the regular expression to search for any variation.

```
db.items.find({item: /Pen?/i})
```

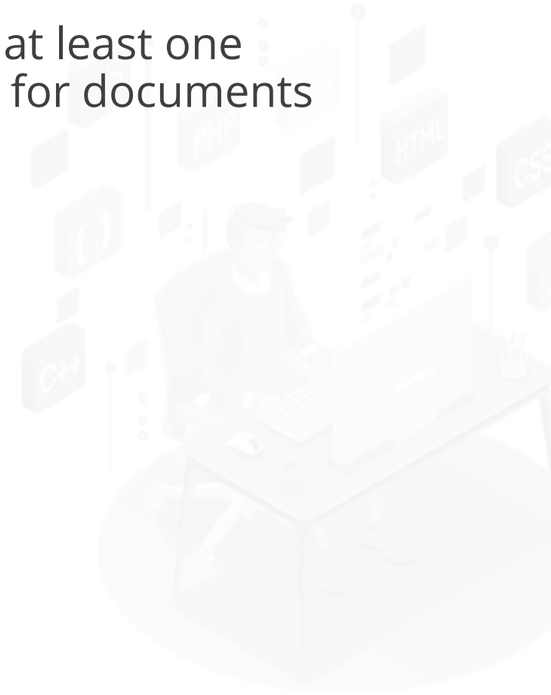
Array Exact Match

Equality matches can specify a single element in the array to match. If the array contains at least one element with the specified value, these specifications match. The example below queries for documents that contain an array, country_codes, which contains 5.

```
db.items.find( {country_codes: 5 } )
```

In the example below, the query uses the dot notation:

```
db.items.find( { 'country_codes.0': 1 } )
```



Array Projection Operators

MongoDB provides three projection operators: $elemMatch, $slice, and $. The operation given below uses the $slice projection operator.

```
db.items.find( { _id: 5 }, {country_codes : { $slice: 2 } } )
```

$elemMatch and $slice are used to return a subset of elements for an array key. The operation given below uses $elemMatch to get that document in which the array values, ($gte) and ($lte) of **country_codes** are set to 6.

```
db.items.find( {country_codes : { $elemMatch: { $gte: 3, $lte: 6 } } } )
```

$Where Query

$Where clause allows you to:

- Represent queries that key-value pairs fail to represent
- Perform any logical execution within a query
- Compare values of two keys in a document

```
> db.foo.insert({"apple" : 8, "spinach" : 4, "watermelon" : 4})
```

For example:

```
db.foo.find({"$where" : function () {
        ... for (var current in this) {
        ... for (var other in this) {
        … if (current != other && this[current] == this[other]) {
          ... return true;
        }} }
        .. return false;
        ... }});
db.foo.find({"$where" : "this.x + this.y == 10"})
> db.foo.find({"$where" : "function() { return this.x + this.y ==
10; }"})
```

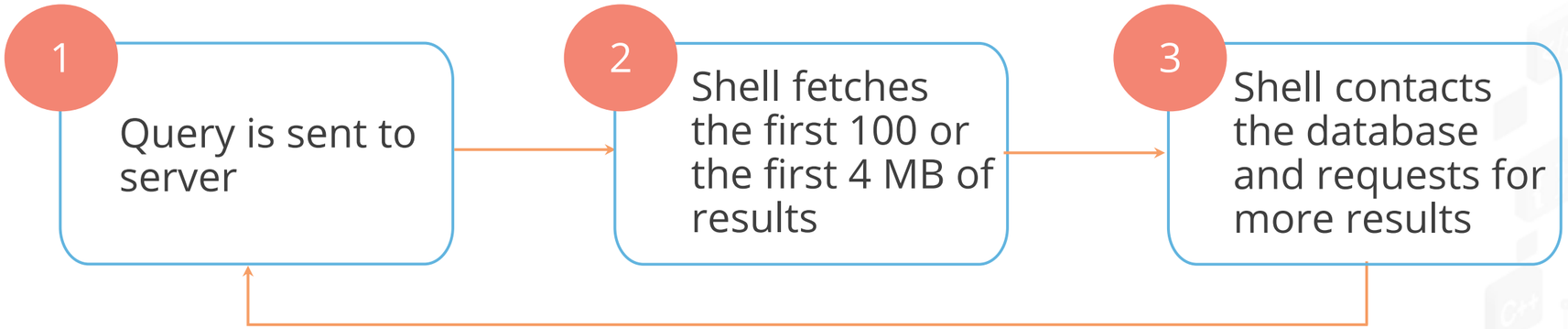
Cursor

Cursors allow you to control a query output by limiting the number of , skipping some, and sorting results. To create a cursor with the shell, add documents into a collection, perform a query on the documents, and then allocate the results to a local variable such as "var“.

For example:

```
var cursor = db.collection.find();
 while (cursor.hasNext()) {
... obj = cursor.next();
... // do stuff
... }
> var cursor = db.people.find();
> cursor.forEach(function(x) {
... print(x.name);
... });
db.c.find().skip(3)
db.c.find().sort({username : 1, age : -1})
```

Cursor



Cursor

Before sending a query to a database, you must include the following three options:

Limit

Set a limit to the number of results fetched. Example: To limit the number of documents found to three, use the query: **db.items.find().limit(3)**

Skip

Skip few documents from the result. Example: To skip the first three documents, use the query: **db.c.find().skip(3)**

Sort

Sort the results in ascending or descending manner. To sort results by name in the ascending and age in the descending manner, use the query: **db.c.find().sort({username : 1, age : -1})**

Pagination

To display 25 results sorted by price, from high to low, per page, perform the query given below:

```
db.stock.find({"desc" : “mobile"}).limit(25).sort({"price" : -1})
```

To display the Next Page for more results, use the second query given below :

```
db.stock.find({"desc" : “mobile"}).limit(25).skip(25).sort({"price" : -1})
```

Pagination: Avoiding Large Skips

Pagination lets you control the number of documents returned by the find() query. To display the first page of a query result in a descending chronological order, use the query given below:

```
var page1 = db.foo.find().sort({"date" : -1}).limit(100)
```

For pagination without using a skip, use the query given below:

For example:

```
var latest = null;

// display first page

while (page1.hasNext()) {

latest = page1.next();

display(latest);

}

// get next page

var page2 = db.foo.find({"date" : {"$gt" : latest.date}});

page2.sort({"date" : -1}).limit(100);
```

Advance Query Option

Following options help in adding arbitrary options to queries:

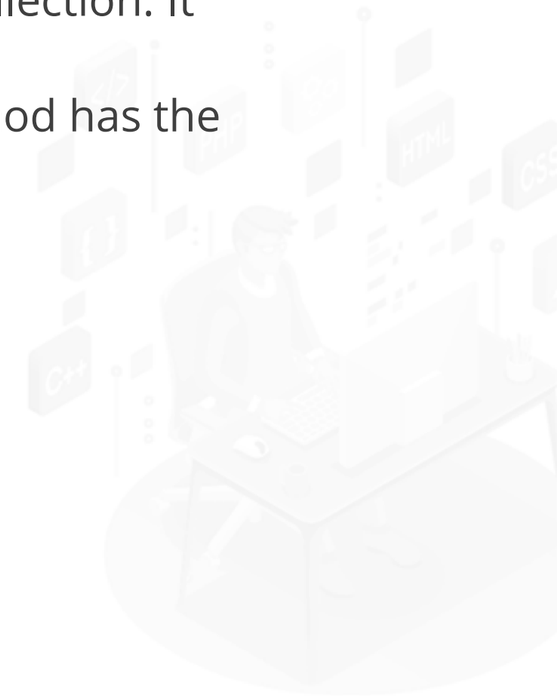
$maxscan: Integer	Specifies the maximum number of documents that are scanned for a query
$min: Document	Starts the criteria for querying
$max: Document	Ends the criteria for querying
$hint: Document	Tells the server which index to use for the query
$explain: Boolean	Does not perform the actual query but explains how the query will be executed
$snapshot: Boolean	Forces the query to use the index of the _ID field

Update Operation

The db.collection.update() method in MongoDB modifies existing documents in a collection. It identifies the documents that need update and options that affect their behavior.

Operations performed by an update are atomic within a single document. This method has the following parameters:

- An update condition
- An update operation
- An options document



$set

MongoDB provides update operators such as $set to modify values to change a field value. You need to perform the following steps to use a $set modifier:

1. Use update operators to change field values

```
db.items.update({ item:"Book"},{$set: {category: 'NoSQL',details: {ISDN: "1234",publisher:"XYZ"}}});
```

2. Update an embedded field

```
db.items.update({ item: “Pen" },{ $set: { "details.model": "14Q2" } })
```

3. Update multiple documents

```
db.items.update({ item: “Pen" },{$set: { category: "stationary" },$currentDate: { lastModified: true }},{ multi: true })
```

$unset and $inc Modifiers

- A $inc operator is used for incrementing and decrementing numbers. The query given below increments the value of the soldQty field by 1 for “Pencil”.

```
db.items.update({“item" : “Pencil"}, {"$inc" : {“soldQty" : 1}})
```

- The value of the $inc key must be a number, because you cannot increment a non-numeric value. To remove the soldQty field for all documents where its value is greater than 700, use the query given below:

```
> db.items.update( {soldQty: { $gt: 700} },{ $unset: {soldQty: “1000" } },{ multi: true })
```

$push Function

The **$push** function allows you to add new elements to an array field. The $push function does the following:

- Adds an element at the end of an array if the array already exists
- Creates an array field and inserts the element into that field

The query given below allows you to use the $push function to an ingredients key:

```
db.user.update({“item" : “Pencil”}, {$push : {“ingredients" :  {“wood":"California cedar",“graphite":“mixture of natural graphite and chemicals"}}})
```

Positional Array Modifications

You can modify array values in two ways:

- Changing the position
- Using the position operator ($)

To increment the value of **votes** field for the first comment in a blog post, use the command given below:

```
db.blog.update({"post" : post_id},{"$inc" : {"comments.0.votes" : 1}})
```

Positional operator ($) helps identify arrays matching the query document and updates them. The command given below shows how a positional operator updates an author's name in an existing document for the blog collection.

```
db.blog.update({"comments.author" : "John"}, {"$set" : {"comments.$.author" : "Jim"}}))
```

Upserting Documents

An upsert is a special kind of update that:

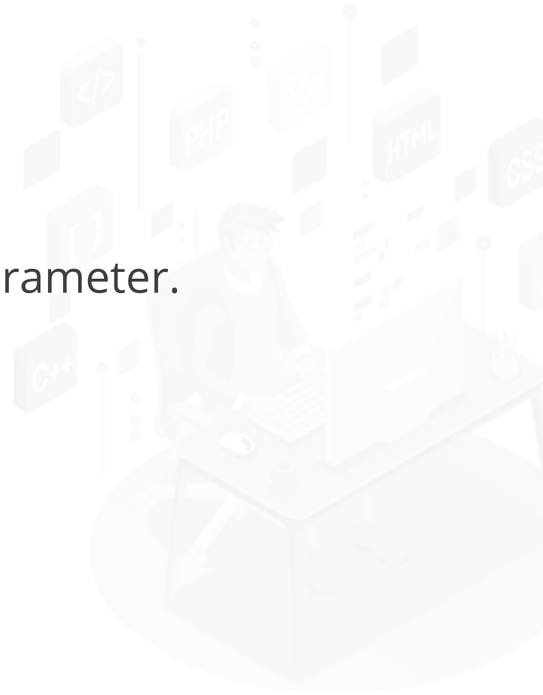
- Updates a document if it matches an update criterion
- Inserts new documents into the collection if no matching criteria is found

The command given below is an example of upsert operation.

```
db.items.update({“item" : “Bag"}, {"$inc" : {“soldQty" : 1}}, {upsert:true})
```

To update all documents that match the query criteria, you can use true as the fourth parameter.
Below is an example of a multi-update command.

```
db.users.update({item: "10/22/1978"},{$set : {gift : "Happy Birthday!"}}, false, true)
```



Removing Documents

- To remove data from a collection, use the command given below:

```
db.courses.remove()
```

- To remove documents from the items collection where the value for "item" is ”Bag”, use the command given below:

```
db.items.remove({“item" : “Bag”)
```

- To remove a single document, call the remove() method with two parameters:

```
db.items.remove({" item" : “Bag},1)
```

- To delete all documents from a collection, use the drop() method:

```
db.courses.drop()
```

Operations in MongoDB

Duration: 90 min.

Problem Statement:

You are given a project to demonstrate different operations performed on MongoDB.

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to demonstrate operations in MongoDB:

1. Create a JavaScript project in your IDE
2. Write a program to demonstrate operations performed on MongoDB
3. Initialize the .git file
4. Add and commit the program files
5. Push to code to your GitHub repository

Key Takeaways

- Database is an organized collection of structured data used to store, retrieve, and manage data.
- NoSQL is used to manage large amount of unstructured data.
- MongoDB provides high performance, high availability, and high scalability.
- MongoDB relationships can be created either by using embedded data model or by document reference data model.
- Operations like creation, updation, and deletion can be performed on MongoDB to modify the data.

Call Record Analysis

Problem Statement:

Duration: 60 min.

A telecom company has customers all over the world. Now, the government has introduced a new rule that all telecom companies should store call records of their customers. The company has already stored all the relevant data in Excel, for their analytics team. Now, they have hired you as a programmer to write a program that can help them store the same data in MongoDB.

LESSON-END PROJECT