MongoDB Developer and Administrator

Certification Training

7277 Learners

VIEW COURSE NOW!

8 Chapters

MongoDB - A Database for the Modern Web Tutorial

Welcome to the second chapter of the MongoDB tutorial (part of the MongoDB Developer and Administrator Course). This chapter provides an introduction to the MongoDB database and its key features.

Let us explore the objectives of this lesson in the next section.

Objectives

After completing this lesson, you will be able to

- Identify the key features of MongoDB
- Explain MongoDB's core server and tools
- Explain how to install MongoDB on Windows and Linux computers
- Identify the steps to start the MongoDB server
- Identify the data types available in MongoDB
- Identify the schema design and data modeling techniques in MongoDB

Let us begin with understating what MongoDB is in the next section of this tutorial.

What is MongoDB?

MongoDB is a document-based database. The basic idea behind shifting from relational data model to a new data model is to replace the concept of a 'row' with a flexible model, the 'document'.

The document-based approach allows embedded documents, arrays, and represents a complex hierarchical relationship using a single record. This is how developers using object-oriented languages want to represent their data.

MongoDB is schema-free, the keys used in documents are not predefined or fixed. Without a fixed schema, massive data migrations have become unnecessary. While migrating data to MongoDB, any issues of new or missing keys can be resolved at the application level, rather than changing the schema in the database. This offers developers the flexibility of working with evolving data models.

In the next section, we will discuss JSON, a data format used in MongoDB.

JSON

In computing, JavaScript Object Notation or JSON is an open-standard file format that uses human-readable text to transmit data objects consisting of

- JSON is language independent.
- JSON uses conventions of the C-family of languages, such as C, C++ (Pronounce as C++), and C# (Pronounce as C hash), Java, JavaScript, Perl, and Python. Because of these properties, JSON makes an ideal data-interchange language. The main format used on the modern Web is XML.
- JSON supports the basic data types, such as numbers, strings, boolean values, arrays, and hashes.

We will discuss JSON structures in the next section of this tutorial.

JSON Structure

JSON is built on the following two structures

1. A collection of name/value pairs, such as an object, record, struct, dictionary, hash table, keyed list, or associative array.
2. An ordered list of values, such as an array, vector, list, or sequence.

Typically, all modern programming languages support these universal data structures. In JSON, an object is considered as a set of name/value pairs which does not follow any order. Typically, an object is presented between a left brace and right brace. Each name in the object is followed by a colon and a comma separates the name/value pairs.

```
{
"_id" : 1,
"name" : { "MongoDB"},
```

```
"customers" : [ "Metlife", "OTTO", "Expedia", "ADP" ],
"applications" : [
{ "name" : "Forward",
"domain" : "e-commerce"
```



```
"domain" : "content management"
}
]
}
```

In the example of JSON given above, the document has the ID field which contains unique numbers to identify the document name field as ‘MongoDB’ string, ‘customers’ field as an array to represent customers of MongoDB. It also has an embedded document called ‘application’ that contains information about a different kind of application built using MongoDB.

In the next section of this tutorial, we will discuss BSON.

BSON

Binary JSON or BSON is a binary serialization format which is used for storing documents and making remote procedure calls in MongoDB. Similar to JSON, BSON allows embedding of documents and arrays within other documents and arrays. In addition, BSON supports extensions that represent data types that are not part of JSON.

For example, BSON has a Date type and a BinData type.

BSON has the following three characteristics.

Lightweight

BSON is lightweight. When used over the network, BSON manages to keep the overhead involved in processing extra header data to a minimum.

Traversable BSON

Traversable BSON is designed to traverse easily across a network. This is a vital property in its role as the primary data representation for MongoDB.

Efficient BSON

Efficient BSON uses C data types that allows easy and quick encoding and decoding of data.

MongoDB can build indexes inside BSON objects and match objects against query expressions on both top-level and nested BSON keys.

We will discuss the structure of MongoDB in the next section.

MongoDB Structure

The table given here depicts the various SQL (pronounce as a sequel) terminology and concepts and the corresponding MongoDB terminology and concepts.

RDBMS	MongoDB
Database	Database
Table, View	Collection
Row	Document (JSON, BSON)
Column	Field
Index	Index
Join	Embedded Document
Foreign Key	Reference
Partition	Shard

In MongoDB, the table and view structure are known as collections.

- Typically, these collections group documents that are structurally or conceptually similar.
- MongoDB allows embedding of the related document to one another. These documents are used to query related data

which are similar to the concept of partition in Relational Database Management Systems or RDBMS.

- After grouping structurally similar documents into the collection, MongoDB groups different collections into databases. For instance, a single instance of MongoDB is capable of hosting multiple independent databases.

> Ideally, you should store data related to a single application at one database. You need separate databases when storing multiple application or users data on the same MongoDB server.

In the next section, we will examine a document store example.

Document Store Example

An example of an embedded document is given here. In this example, a document is stored in the collection user for a person called Mark Tailor, who is 35 years old and has interest in long distance running and mountain biking. This document also contains an embedded document called favorites that displays the data about Mark Tailor's favorite color and sports.

```
> db.user.findOne({age:35})
{
"_id" : ObjectId("5224e0bd52..."),
"first" : "Mark",
"last" : "Tailor",
"age" : 35,
"interests" : [
"long distance running",
"Mountain Biking ]
"favorites": {
```

```
"color": “Yellow",
"sport": “Boxing"}
```


MongoDB as a Document Database

The data model in MongoDB is document-based. Although documents provide rich structure, they need not conform to any pre-specified schema.

The 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 tabled needs to be altered.	Group documents into collections that do not follow any schema. Each document in a collection can have a completely independent structure.

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

In the next section, we will discuss Transaction Management in MongoDB.

Transaction Management in MongoDB

Transactions in MongoDB are as follows:

- MongoDB only supports a 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. However, the entire operation is not atomic and other operations may interleave.

We will discuss Easy Scaling in the next section.

Easy Scaling

remarkable pace. Advances in sensor technology, the popularity of internet-connected handheld devices have created a demand for more data storage with more databases capabilities.

Even small-scale applications generate more data than many databases are capable of handling. As the amount of data that needs to be stored are growing, developers face the challenge of scaling their databases. Scaling is the core feature of MongoDB.

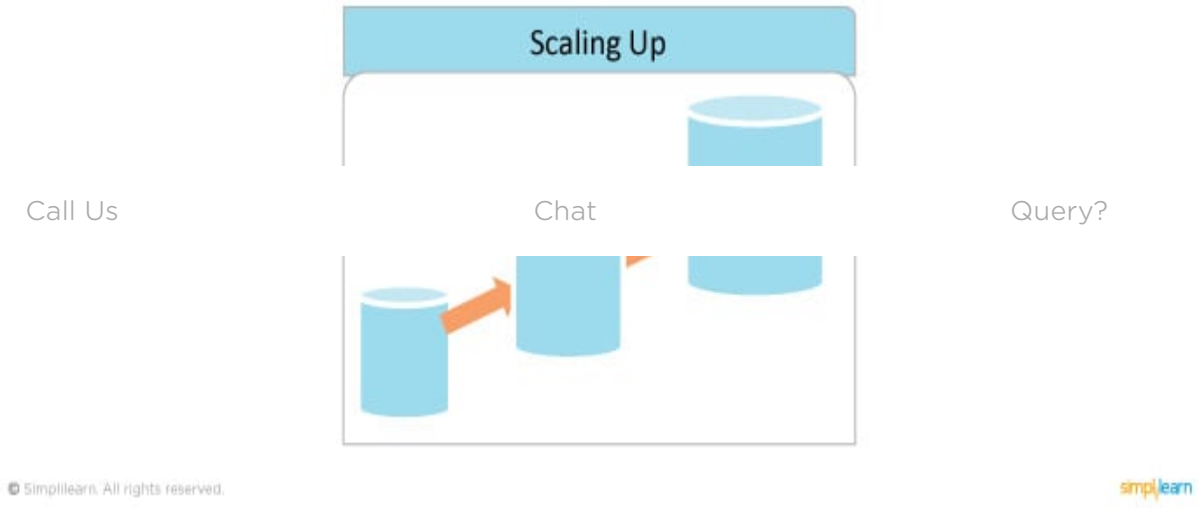
You can add a new machine to the existing MongoDB cluster to expand its capacity.

In the next section, we will discuss how to scale up or scale out a database.

Scaling Up vs. Scaling Out

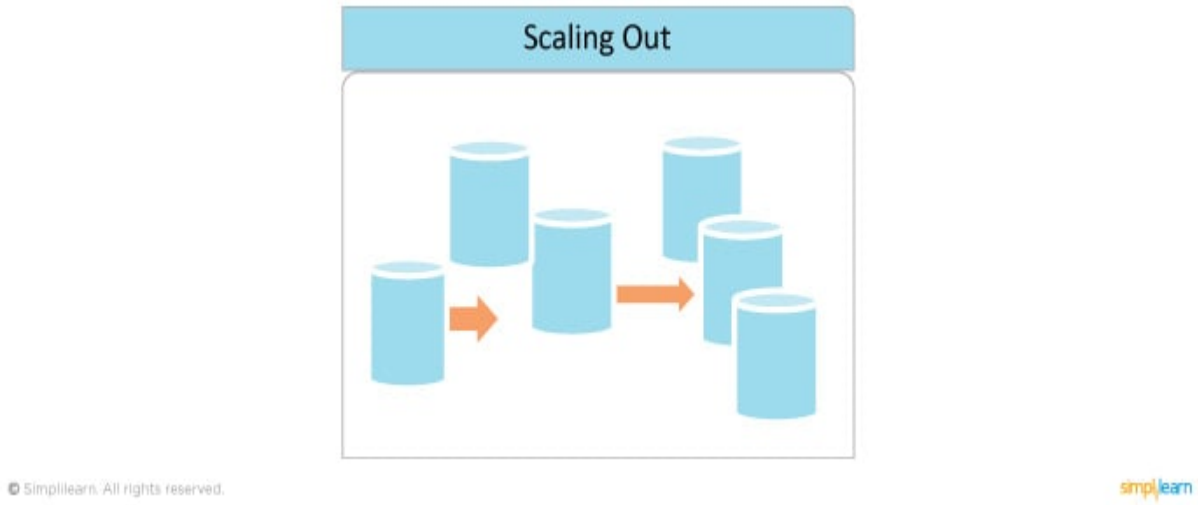
Scaling a database involves two choices:

- Scaling up or getting a bigger database



Scaling up is an easy but costly affair. Large databases are very expensive and eventually, a time may come when the purchased database too will be unable to handle the growing volumes of data and a more powerful database is required.

- Scaling out or partitioning data across multiple databases



It is both extensible and cost-effective to scale out. You can buy

commodity servers and add them to your cluster, thus add storage space and increase performance.

In the next section, we will discuss scale up or vertical scaling.

Vertical Scaling

You can easily scale a database by upgrading the hardware. If your application is running on a single node, you can add some combination of disk input output per second or IOPS (Pronounce as I-O-P-S), memory, and CPU to enhance the database performance. 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 you are running on a physical hardware, you may face a situation where the cost of a more powerful server is not permitted. In such cases, you need to consider horizontal scaling, or scaling out.

In the next section, we will learn about horizontal scaling.

Horizontal Scaling

In horizontal scaling, the database is distributed across multiple machines.

The advantages of horizontal scaling are as follows

- Commodity hardware for a horizontally scaled architecture is used. Thus, the costs of hosting the total data are reduced.
- Mitigates the risk of failure.
- Failure may not impact your business if a copy of the data is saved on a replicated slave. However, failure of a single server may bring down the

entire system. On the other hand, the failure in a horizontally scaled architecture is less disastrous because a single machine is only a small part of the entire system.

- Automatically manages data distribution across multiple nodes.
- Can easily add new nodes. When a failure of the master node occurs, it does the failover to another replica-set node.
- Is transparent to the client. The client does need to be aware whether it is single shard or a set of shards.

In the next section, we will discuss the features of MongoDB.

Features of MongoDB

Some of the key features of MongoDB are as follows:

- **Ad hoc queries**: MongoDB allows performing of search functions by field, range queries, and regular expression searches. Queries can return specific document fields and may include user-defined JavaScript functions.
- **Querying**: For document retrieval, MongoDB uses rich query language. It also allows you to write any complex condition to retrieve documents.
- **Fast In-Place Updates**: MongoDB allows you to choose write semantics, enable journaling, and thus control the speed and durability. All writes are sent across a Transmission Control Protocol or TCP socket and do not require a database response. To get a response, use the special safe mode of the drivers and perform a write. This generates a response acknowledging the receipt of the writer with no errors.

- **Server-side JavaScript execution**: MongoDB uses JavaScript in queries and aggregation functions such as MapReduce, which are sent to the database for execution.
- **Capped collections**: Capped collections are fixed sized collections

and once the specified size has been reached, behaves like a circular queue.

In the next section, we will discuss secondary indexes.

Secondary Indexes

MongoDB allows implementation of multiple secondary indexes as B-trees. These B-tree indexes can be optimized for range scan queries and queries with sort clauses.

MongoDB lets you create up to 64 indexes per collection. It supports indexes, such as ascending, descending, unique, compound-key, and geospatial. MongoDB uses the same data structure for indexes as most RDBMSs.

In the next section, we will learn about replication.

Replication

MongoDB creates database replication using a topology called replica set.

A replica set:

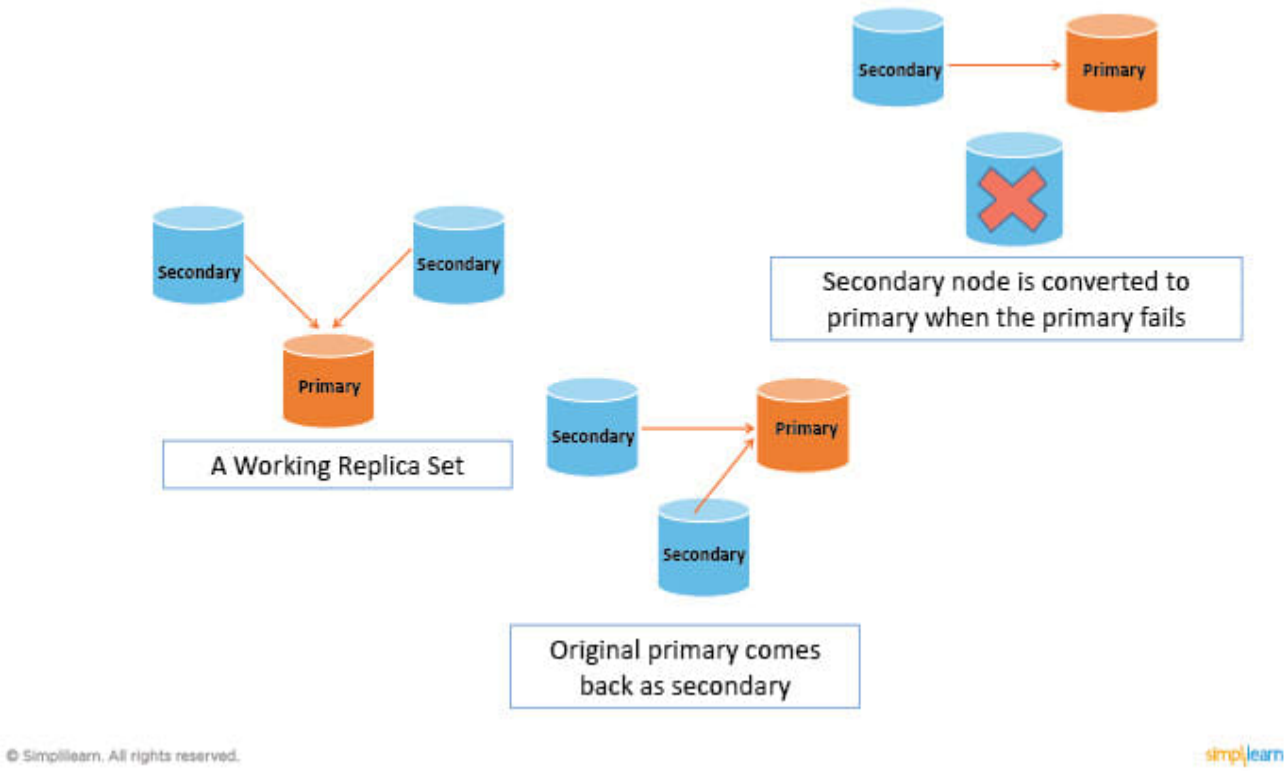
- Distributes data across various MongoDB nodes called shards for redundancy.
- Automates failover when a server or network outages occur.
- Scales database reads.

For example, if your application is read intensive, you can spread database reads across various nodes in the replica set cluster. Typically, a replica set contains one primary node and one or more secondary nodes. Similar to the

master-slave replication, the primary node of a replica set supports both reads and writes. However, the secondary nodes support read-only.

We will continue with Replication in the next section.

Schematic Representation of Replication



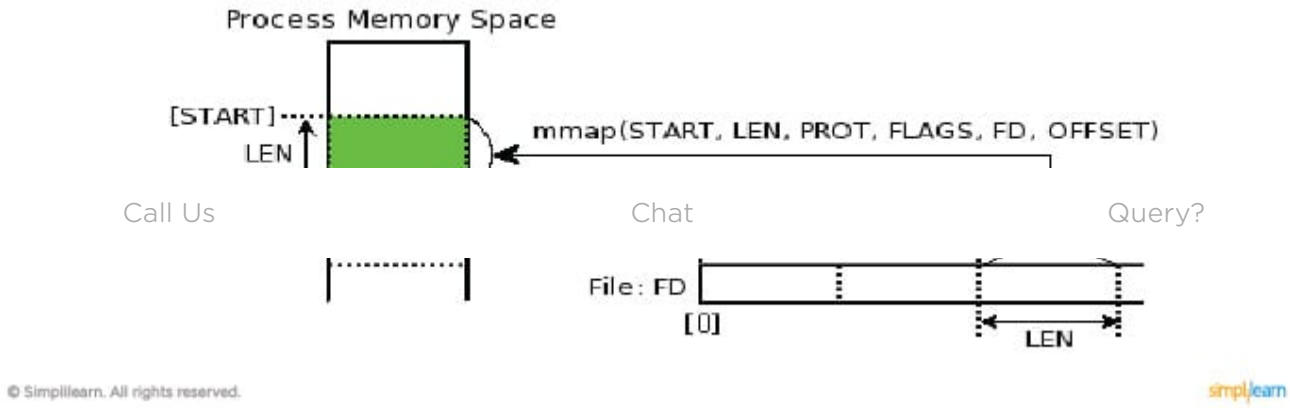
The image above depicts a working replica set. When the primary node fails, the cluster picks a secondary node and automatically converts it to the primary. When the failed primary is restored, it acts as a secondary.

We will discuss Memory management in the next section.

Memory Management

MongoDB store the data in memory mapped files. By default, MongoDB uses all the system memory for these mapped files. This is the reason why MongoDB operations are fast. MongoDB allows its operating system to manage its memory. This design impacts its performances and operations.

All extents are mapped to memory using mmap methods.



In the image above, this mapping is depicted by the solid black lines connecting the mapped disk blocks to the virtual memory pages.

In the next section, we will discuss Replica Set.

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 replica set is converted to the master.
- Allows choosing a master or slave depending on whether you want a strong or delayed consistency.
- Keeps replicated data on the nodes belonging to different data centers to protect the data in the case of natural disaster.

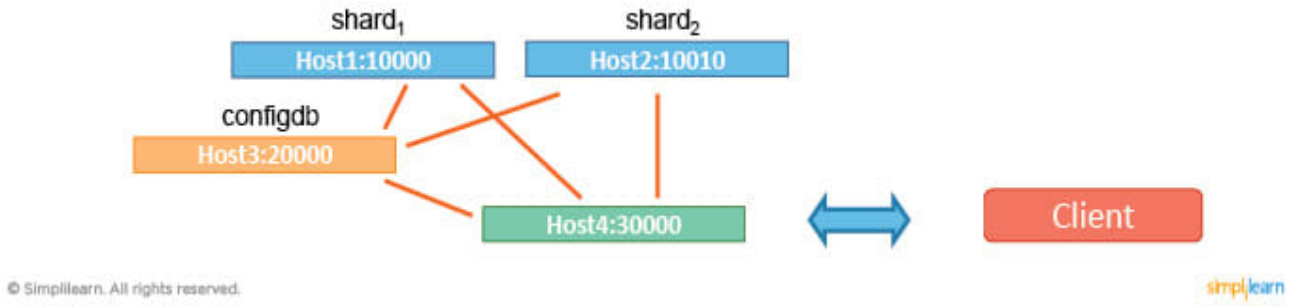
In the next section, we will discuss Auto-Sharding.

Auto-Sharding

MongoDB uses sharding for horizontal scaling. A shard is a master node with one or more slaves. For automatic sharding, choose a shard key that determines the distribution of the collection data. The shard key splits the data into ranges and distributes them across multiple shards.

MongoDB is spread over multiple servers and performs load balancing and/or data duplicating to keep the system up and running in case of hardware failure.

added to a running database.



The diagram given above represents documents distributed to different shards of the MongoDB cluster.

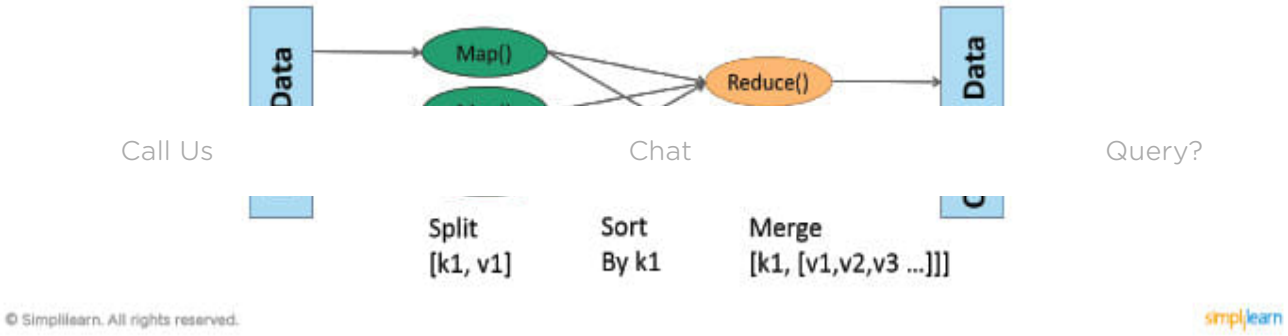
In the next section, we will discuss Aggregation and MapReduce.

Aggregation and MapReduce

Aggregation is operations used to analyze data sets and return calculated results. In MongoDB, you have a rich set of operations to perform aggregate calculation by analyzing the data sets.

MapReduce is typically used for operations, such as batch processing of data and aggregation. MongoDB uses MapReduce operations to perform data aggregation.
A MapReduce operation consists of two phases:

- Map: Documents are processed and one or more objects are produced for each input document.
- Reduce: The outputs of the map operation are combined.



Optionally, there can be an additional stage to make final modifications to the result. Similar to other aggregation operations, MapReduce can define a query condition to select the input documents, and sort and limit the results.

In the next section, we will discuss collection and databases.

Collection and Database

A collection is a group of documents. By comparing a document in MongoDB to a row in a relational database, the collection can be found to be similar to a table. Collections are not restricted by any schema.
Therefore, documents within a single collection can have any shape.

MongoDB groups collections into databases and a single instance of MongoDB can host several databases, each with a completely independent database.

Having different types of documents in the same collection can be cumbersome for developers and administrators. Developers need to ensure that their queries retrieve specific documents or their application codes handle documents of different structure. A good practice is to store data of a single application in the same database.

In the next section, we will discuss Schema Design and modeling.

Schema Design and Modeling

The collections in MongoDB allows flexibility in the document structure. This enables you to map a document to an entity or an object easily. Typically, all documents in a collection share a similar structure.

- The needs of the application.
- The performance capability of the database engine.
- Data retrieval patterns

When designing a data model, you need to consider the application usage of the data, such as queries, updates, and data processing along with its inherent structure.

In the next section, we will discuss reference data models.

Reference Data Model

When designing data models for MongoDB applications, two things are important:

- The structure of documents and
- Representation of the data relationships.

References and embedded documents are the two tools that allow applications to represent data relationship.

References use links from other documents to store relationships between data. Applications use these references to access the related data.

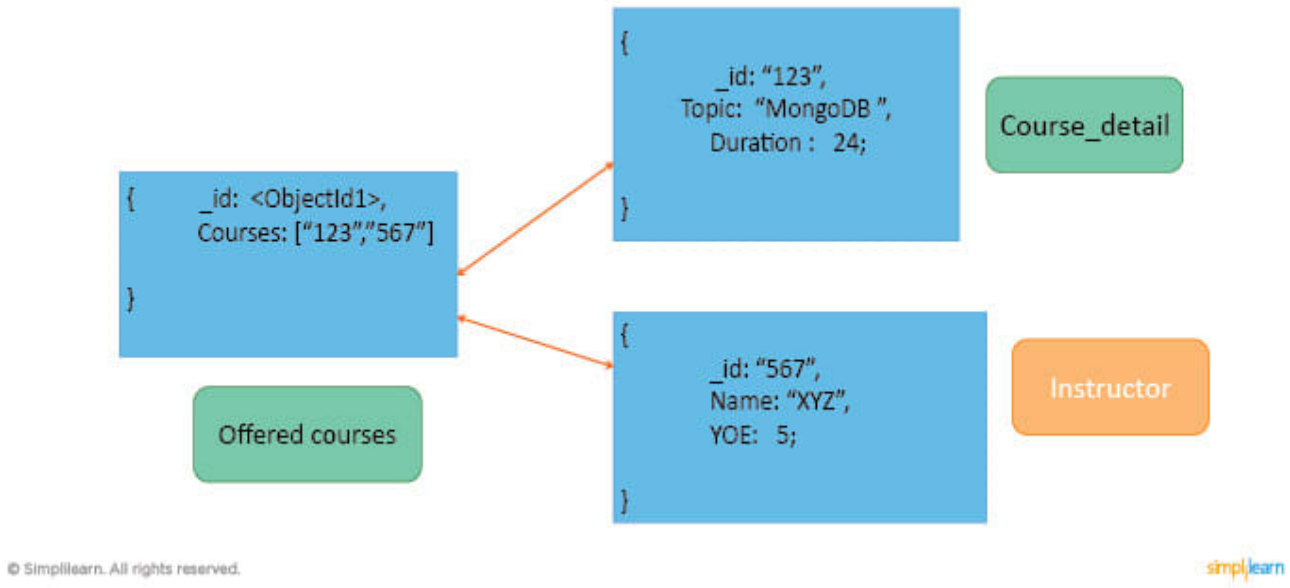
These are normalized data models, which you can use:

- When embedding document results in data duplication does not give sufficient performance benefit.
- To represent complex many-to-many relationships.

- To model large hierarchical data sets. References provide more flexibility than embedding. However, client-side applications must trigger follow-up queries to resolve references.

We will review an example of Reference data model in the next section.

Reference Data Model Example



The diagram provided above is an example of a reference data model, which consists three different collections named Offeredcourses (Pronounce as a single word), course_detail (Pronounce as course underscore detail), and instructor.

Each collection is dependent on the other collections. For example, if offered courses are dependent on course_detail and instructor, then you need to create three different collections and provide the reference of course_detail and instructor in the offerdcourses collection by storing their respective id field.

In the next section, we will discuss embedded data model.

Embedded Data Model

thus capture relationships between data. MongoDB allows embedding of document structures in a field or array within a document. Thus, these denormalized data models 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 In these relationships, the 'many' or child documents are viewed in the context of the parent documents.

Typically, embedding provides good read performance. It also allows request and retrieval of related data in a single database operation. Embedded data models allow updating of related data in a single atomic write operation.

In the next section, we will focus on an example of the embedded data model.

Embedded Data Model Example

The diagram given in the section is an example of an embedded data model. In this diagram, the outermost document offered course contains two embedded document course detail and instructor. Course detail has the fields, Topic, and duration while instructor has the fields name and year of experience or YOE.

```
{
_id: "123",
coursedetail:
{
Topic: "MongoDB ",
```

```
Duration : 24,
}
instructor:
{
```



```
}
}
```

In the next section, we will focus on data types.

Data Types

MongoDB supports other data types while retaining JSON's essential key/value pair characteristic. How values of each type are represented depends on the language used.

Following is a list of the commonly supported data types:

- **Null**: Used to represent both a null value and a nonexistent field {"x": null}
- **Undefined**: Used in documents {"x" : undefined}
- **Boolean**: Used for the values 'true' and 'false' {"x" : true}
- **32-bit integer**: Cannot be represented on the shell.
- **64-bit integer**: The shell cannot represent these.
- **64-bit floating point number**: All numbers in the shell will be of this type.
- **Maximum value**: Contains a special data type that represents the largest possible value. The shell does not support this type.
- **Minimum value**: Contains a special data type that represents the smallest possible value. The shell does not support this type.
- **ObjectId**: Unique, fast to generate and ordered, these consists of 12 bytes where the first four bytes represent the ObjectId creation time.
- **String**: Are UTF-8 compliant. When serializing and deserializing BSON, programming languages convert language strings to UTF-8 format.

- **Symbol**: Not supported by the shell. When the shell gets a symbol, it converts it into a string.
- **Timestamps**: BSON offers a special timestamp for internal MongoDB use

- **Date**: A 64-bit integer that denotes the number of milliseconds since the UNIX epoch.
- **Regular expression**: Documents contain Javascript's regular expression syntax. {"x" : /simplilearn/i}
- **Code**: Documents can contain JavaScript code. For example: {"x" : function() { /* ... */ }}
- **Binary data**: A string of arbitrary bytes that cannot be manipulated from the shell.
- **Array**: Sets or lists of values can be represented as arrays. For example: {"courses" : ["PMP", "Cloud", " MongoDB "]}
- **Embedded document**: Documents can contain entire documents, embedded as values in a parent document. For example: {"course_duration" : {"MongoDB " : "24 Hrs"}}

In the next section, we will discuss the core servers of MongoDB.

Core Servers of MongoDB

The core database server of MongoDB can be run as an executable process called mongod or mongodb.exe on Windows.
The mongod process receives a command to run the MongoDB server over a network socket through a custom binary protocol. The data files for a mongod process are stored by default in the directory /data/db (read as slash data slash D-B).

A mongod process can be run in several modes:

- Replica set: Configurations comprise two replicas and an arbiter process that reside on a third server.

- Per-shard replica set: The auto-sharding architecture of MongoDB consist of mongod processes configured as per-shard replica sets.
- Mongos: A separate routing server is used to send requests to the appropriate shard

Mongos queries from the application layer and locates the data in the sharded cluster to complete these operations. A mongos instance is identical to any MongoDB instance.

In the next section, we will discuss the MongoDB tools.

MongoDB's Tools

MongoDB Tools consists of the following:

1. JavaScript shell
2. Database drivers
3. Command-line tools.

The JavaScript shell

The command shell in MongoDB is a JavaScript-based tool. It is used to administer the database and manipulate data. A mongo executable loads the shell and connects it to a specified mongod process. In addition to inserting and querying data, the shell allows you to run administrative commands.

Database drivers

The MongoDB drivers are easy to use. It provides an Application Program Interface or API that matches the syntax of the language used while maintaining uniform interfaces across languages. 10gen (pronounce as ten gen), a company behind MongoDB supports drivers for C, C++, C#, (pronounce as C, C plus plus C hash) Erlang, Haskell, Java, Perl, PHP, Python, Scala, and Ruby.

Command-line tools

MongoDB contains the following command-line utilities:

- Mongodump and Mongorestore (Pronounce as Mongo dump and mongo store) are standard utilities that help backup and restore a database. Mongodump can save the data and the BSON format of MongoDB and thus,

- Mongoexport and Mongoimport are used to export and import JSON, comma separated value or CSV, and Tab Separated Value or TSV data. These tools help you get data in widely supported formats. You can use mongoimport for initial imports of large data sets. However, you need to adjust the data models for best results. In such a case, you can use a custom script to easily import the data through one of the drivers.
- Mongosniff is a wire-sniffing tool used for viewing operations sent to the database. This tool translates the BSON that is transmitted to human-readable shell statements.
- Mongostat is similar to iostat (pronounce as I-O-stat). Mongostat provides helpful statistics, including the number of operations per second, for example, inserts, queries, updates, deletes, and so on. It also provides information, such as the amount of virtual memory allocated and the number of connections to the server.

In the subsequent section, we will learn how to install and start MongoDB on Linux and Windows.

Use Cases

Some use cases of MongoDB are:

- Personalization: Allows personalization of customer experience in real time to predict the wants and needs of customers.
- Mobile: Allows scaling of mobile applications to cater to millions of online users.
- Internet of Things (IoT): Manages huge volumes of data generated by IoT. Allows building of your own IOT suite to manage big data on your own.
- Real-time Analytics: Allows real-time data analysis, minute by minute and second by second.

- Web Application: Helps Web application manage data efficiently through rich data structures, such as documents.
- Content Management: Allows storing and presenting of any type of content, build new features, incorporate all kinds of data in a single database

highly personalized and smooth shopping experience.

- Single View: Allows building of a single view of all your business data.

Summary

Here is a quick recap of what was covered in this lesson:

- MongoDB is a document-based database that represents a complex hierarchical data relationship using embedded document model or using reference model.
- MongoDB uses JSON as the data format, which is based on a collection of name/value pairs and an ordered list of values
- A collection in MongoDB is a table and view structure that groups structurally or conceptually similar documents.
- MongoDB supports auto-sharding to manage data distribution across multiple nodes.
- MongoDB uses replica sets to create redundancy and automates failover when a server or network outages occur.

Conclusion

This concludes the lesson MongoDB: A database for the modern web. In the next chapter, we will discuss CRUD Operations in MongoDB.

Find our MongoDB Developer and Administrator Online Classroom training classes in top cities:

Name	Date	Place	
MongoDB Developer and Administrator	3 Aug -25 Aug 2019, Weekend batch	Your City	View Details

MongoDB Developer and Administrator	Weekend batch		Details
MongoDB Developer and Administrator	28 Sep -20 Oct 2019, Weekend batch	Bangalore	View Details

Related Courses

Big Data Hadoop and Spark Developer

Learner Reviews

"Great effort by Simplilearn. I really appreciate it. I didn't get bored of any session like offline train..."

Nagarjuna D N

Related Articles

3 Reasons Why Your Big Data Career Needs a MongoDB Certification

Call Us Chat Query?

Refer and Earn

Company

About us

Our team

Careers

In the media

Alumni speak

Contact us

Help & support

Work with us

Become an instructor

Blog as guest

Become an affiliate

Discover

Resources

Tutorials

Career data labs

Simplilearn community

RSS feed

For Businesses

Corporate training

Learn On the Go!

Get the Android App

Get the iOS App