SIT378 MongoDB

Ontline

1. Install MongoDB (Mac)
2. Basic operations of MongoDB
3. What is MongoDB
4. Document oriented database
5. Reference list
6. Install MongoDB (Mac)

Install MongoDB Community Edition

prerequisite

Make sure your system meets each of the following prerequisites: You only need to run each prerequisite step once on your system.

Install Xcode Command Line Tools

Homebrew requires the Xcode command line tools from Apple's Xcode.

• Install the Xcode Command Line Tools by running the following command in the macOS Terminal.

xcode-select --install

Install Homebrew

macOS does not include the Homebrew brew package by default.

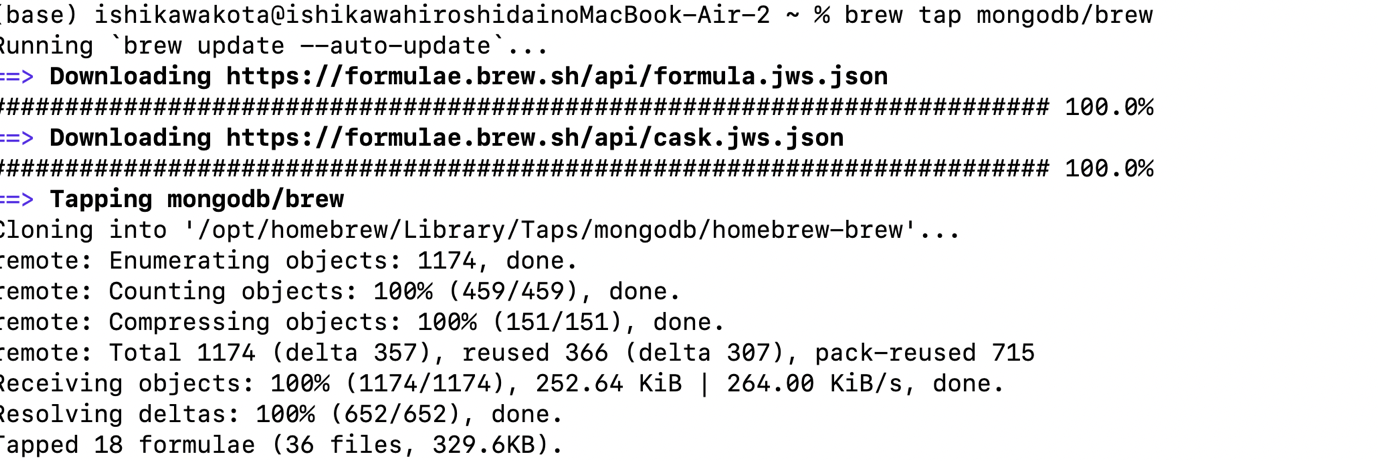
* + Homebrew installation instructions to install brew using the official.

Install MongoDB (6.0) Community Edition

To install MongoDB Community Edition using Homebrew's brew package manager, follow the steps below.

Tap MongoDB homebrew tap. To download the official Homebrew formula for MongoDB and database tools, run the following command in your macOS Terminal:

brew tap mongodb/brew



1. If you have already done this in a previous installation of MongoDB, you can skip this step.

2. To update Homebrew and all existing formulas:

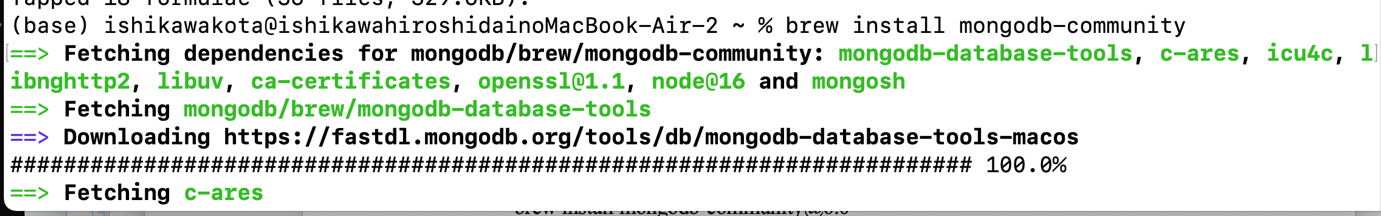
brew update

3. To install MongoDB, run the following command in the macOS Terminal application.

brew install mongodb-community@6.0

or

brew install mongodb-community



1. Basic operations of MongoDB

If you want the mongdb background process to start automatically even after a reboot, run the brew services start command.

brew services start mongodb-community

If you want to stop the service, execute the brew services stop command below.

brew services stop mongodb-community

Text, letter

Description automatically generated

Connection to MongDB

To connect to MongoDB, MongoDB must be started.

brew services start mongodb-community

mongo

There is something I want you to be careful about here. Mongo has been replaced by mongosh since Ver6.0. So mongo is no longer available.

Graphical user interface, text, application

Description automatically generated

After running this command go to the shell (our server) where you ran the mongod command. At the end you will see the message "Connection accepted". That means our installation and configuration was successful!

Just run it in mongo shell.

db

1. Open a command prompt and type mongod to start the MongoDB server.

As a result of verification, even if you do not use mongod, you can use it by entering the following command after brew start services mongodb-community.

2. Open another shell and type mongo to connect to the MongoDB database server. Please use mongosh for ver6.0 or higher as a note

1. Find the current database you are in

db

A picture containing graphical user interface

Description automatically generated

2. List database

show databases

or

show dbs

Text

Description automatically generated

3. Go to a specific database

use <your\_db\_name>

Text

Description automatically generated with low confidence

4. Create a database

Let's talk about how to create a database in Mongo shell.

use <your\_db\_name>

In MongoDB server, if the database already exists, use that command to go to the database.

But if the database doesn't exist yet, MongoDB server will create it for you. Then navigate to it.

After creating a new database, when I run the show database or show dbs command, the newly created database does not show up. This is because it doesn't show up in the db list until it has data (documents).

5. Create a collection

Navigate to the newly created database using the use command.

There are actually two ways to create a collection. Let's look at both.

One way is to insert data into the collection.

db.myCollection.insert({"name": "john", "age": 22, "location": "colombo"})

This will create the collection myCollection even if the collection doesn't exist. Then insert a document with name and age. These are uncapped collections.

The second method is as follows.

* 1. Creating an uncapped collection

db. createCollection("myCollection")

2.2 Creating capped collections

db.createCollection("mySecondCollection", {capped : true, size : 2, max : 2})

This way you create a collection without inserting data.

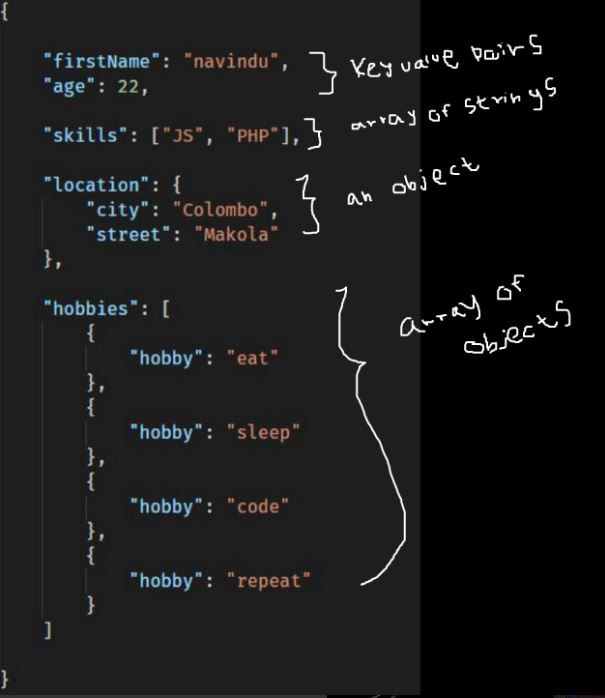
The "capped collection" has a maximum number of documents that prevents document overflow.

In this example, we enabled capping by setting the value to true.size: 2 means a limit of 2 megabytes, max: 2 sets the maximum number of documents to 2.

If you try to insert more than two documents into mySecondCollection and use the find command (which we'll get to in a moment), only the last inserted document is displayed. This does not mean that the first document has been deleted. It's just not displayed.

6. Insert data

You can insert data into a new collection or a previously created collection.



<https://www.freecodecamp.org/news/learn-mongodb-a4ce205e7739/amp/>

How to store data in JSON

There are three ways to insert data.

1. insertOne() is used to insert a single document only.

2. insertMany() is used to insert multiple documents.

3. insert() is used to insert the required number of documents.

Below are some examples.

• insertOne()

db.myCollection.insertOne(

{

"name": "navindu",

"age": 22

}

)

• insert many()

db.myCollection.insertMany([

{

"name": "navindu",

"age": 22

},

{

"name": "kavindu",

"age": 20

},

{

"name": "john doe",

"age": 25,

"location": "colombo"

}

])

The insert() method is similar to the insertMany() method.

Also note that I inserted a new property called location in John Doe's document. So using find we can see that only john doe has a property of location attached.

This can be an advantage when it comes to NoSQL databases such as MongoDB. Allows for scalability.

7. Query data

Here's how to query all the data from the collection:

db.myCollection.find()

If you want to see this data cleaner, just add .pretty() at the end. This will display the document in pretty-printed JSON format.

db.myCollection.find().pretty()

8. Documentation update

Let's say you want to update someone's address or age, how do you do that?

db.myCollection.update({age: 20}, {$set: {age: 23}})

The first argument is the field of the document to update. Here we specify age for simplicity. In production you can use something like the \_id field.

It's always better to use something like \_id to update unique rows. This is because multiple fields can have the same age and name. So updating one row affects all rows with the same name and age.

9. Delete documents

As I said earlier, when updating or deleting a document, just specify the \_id, not just the name, age and location.

db.myCollection.remove({name: "navindu"});

10. Deleting Collections

db.myCollection.remove({});

Note, this is not equal to the drop() method. The difference is drop() is used to remove all the documents inside a collection, but the remove() method is used to delete all the documents along with the collection itself.

Check collection

You can check the collections of the connected database with the show collections command. A collection can be dropped with db.collection\_name.drop().

> show collections

users

// remove collection

> db.users.drop()

true

Delete document

Use deleteOne to delete one document. Check the entire collection with find to see if the specified document has been deleted.

> db.users.deleteOne({ "name" : "Kevin"})

{ "acknowledged" : true, "deletedCount" : 1 }

> db.users.find()

{ "\_id" : ObjectId("5d941eb320153c73fb1b1ad0"), "name" : "John Doe", "age" : 30 }

Documentation update

Name is John Doe and age is changed from 30 to 45. Execute the find command before and after update to check whether the update has been performed normally.

For update, use $set as below.

> db.users.find().pretty()

{

"\_id" : ObjectId("5d941eb320153c73fb1b1ad0"),

"name" : "John Doe",

"age" : 30

}

> db.users.update({"name":"John Doe"},{$set: {"age":45}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.users.find().pretty()

{

"\_id" : ObjectId("5d941eb320153c73fb1b1ad0"),

"name" : "John Doe",

"age" : 45

}

1. What is MongoDB

MongoDB is one of the most popular unstructured database management systems that can store large amounts of data. It is a document-oriented database system belonging to the NoSQL (non-SQL) family. Here data and records are stored as documents that behave like JSON objects. A document is a combination of key-value pairs that form the basic unit of data in MongoDB. This database system went live in the mid-2000s.

What is NoSQL and why should I use NoSQL?

NoSQL, short for Not Only SQL or non-SQL, is an unstructured database that helps store and retrieve data. In 1998 Carl Strozz introduced NoSQL. He models data in ways other than table relationships. This means that such databases do not have a fixed schema, but are explicitly intended for distributed data requiring huge data storage. We use NoSQL databases for real-time web apps, mobile apps, big data, and more. Websites like Google, Twitter, Amazon, Facebook, and Instagram collect terabytes of data every day.

Previously, web applications were simple and did not generate such huge amounts of data. However, the advent of big companies like Facebook, Google, and Amazon generated massive amounts of data, which made NoSQL databases popular. Traditional RDBMS (like SQL) store and retrieve text data using simple queries. However, NoSQL database management systems employ a wide range of file systems that store structured, unstructured, semi-structured, and polymorphic data.

Features of NoSQL

* + NoSQL databases do not follow the relational model.
  + They are schema-free or do not follow any particular schema.
  + NoSQL renders heterogeneous data structures (graphs, trees, column families, key-value pairs, documents, etc.) in the same domain.
  + Data is not stored flat in rows and columns (tables).
  + NoSQL does not require data normalization and object-relational mapping.
  + NoSQL does not require the setting of complex concepts such as joins, referential integrity and ACID properties.

Who Should Use MongoDB?

Developers who want to work with structured, semi-structured, or unstructured data should use MongoDB for their applications. Those interested in big data analytics can also use MongoDB. Again, if your application's data requires agility, scaling, and high performance, MongoDB is the perfect solution.

It supports a wide range of use cases from real-time exploratory and predictive analytics to parallel data processing. MongoDB can provide high-performance data storage even when spread across multiple servers.

When should I use MongoDB and when not?

MongoDB is a great database, but there are times when you should use it and times when you shouldn't. It is not universally applicable. Like any tool, it has limitations.

MongoDB works best with unstructured data, making it ideal for big data systems, MapReduce applications, news site forums, and social networking applications.

Use MongoDB when:

* + You are using cloud computing. MongoDB is perfect for cloud computing. Cloud-based storage should easily distribute data across multiple servers, which is a perfect fit for MongoDB's nature.
  + Data should be accessible quickly and easily. Use MongoDB when running performance-critical applications. MongoDB provides high data availability and provides immediate and automatic data recovery.
  + No database administrator. Perhaps you have a small business or are starting a startup but don't yet have the resources to recruit a full-time database administrator. But MongoDB is low maintenance, so the lack of admins isn't that painful.
  + I have a lot of unstructured data. MongoDB (and NoSQL databases in general) have no restrictions on the data types that can be stored.
  + We use Agile methodology for development. Relational databases are not agile and will slow you down. A database like MongoDB, on the other hand, does not require the level of preparation that its relational counterpart requires.
  + There is a schema problem. If you have unstable or undefined schemas, use MongoDB.
* But MongoDB is not a panacea. When not to use non-relational databases.
  + ACID compliance is required. In this case, ACID is an acronym for atomicity, consistency, isolation, and durability. Applications that require database-level transactions (such as a financial institution's core banking system) must be ACID compliant.
  + Work with stored procedures. Unfortunately MongoDB has no provision for stored procedures.
  + Your data is immutable and structured. If your business isn't experiencing explosive growth and your data is consistent, you don't need a database like MongoDB.

Comparing MongoDB to other databases

With so many database management solutions available today, choosing the right one for your enterprise can be difficult. Here's a comparison of common solutions and some of the best use cases to help you decide.

MongoDB vs MySQL

MySQL (link exists outside of IBM) uses a structured query language to access stored data. This format uses schemas to create the database structure and tables as a way to standardize data types so that values are searchable and properly queried. A mature solution, MySQL is useful in a variety of situations, including website databases, applications, and commercial product management.

Due to its rigid nature, MySQL is preferable to MongoDB when data integrity and isolation are essential, such as when managing transactional data. However, MongoDB's less restrictive format and higher performance make it a better choice, especially when availability and speed are primary concerns.

MongoDB vs Cassandra

Cassandra (link is outside IBM) and MongoDB are both considered NoSQL databases, but have different strengths. Cassandra uses a traditional table structure with rows and columns, allowing users to maintain uniformity and durability when formatting data before compilation.

Cassandra has a SQL-like syntax, which can offer easier migration for businesses looking for NoSQL solutions. It also reliably handles deployment and replication without much configuration. However, it cannot match MongoDB's flexibility for processing structured and unstructured data sets, or the performance and reliability of mission-critical cloud applications.

MongoDB use cases

* mobile application

MongoDB's JSON document model allows you to store your backend application data wherever you need it, including Apple iOS and Android devices and cloud-based storage solutions. This flexibility allows developers to seamlessly scale mobile applications by using secondary and geospatial indexes to aggregate data across multiple environments.

* Real-time analysis

As companies expand their operations, it's critical to access key metrics and business insights from large data pools. MongoDB easily handles the conversion of JSON such as BSON and JSON-like documents to Java objects, making reading and writing data in MongoDB fast and very efficient when analyzing real-time information across multiple development environments. to This has proven beneficial for several business sectors including government, financial services and retail.

* content management system

A content management system (CMS) is a powerful tool that plays a key role in ensuring a positive user experience when accessing e-commerce sites, online publications, document management platforms, and other applications and services. MongoDB makes it easy to add new features and attributes to your online applications and websites using a single database and high availability.

* Enterprise data warehouse

The Apache Hadoop framework is a collection of open source modules, including Hadoop Distributed File System and Hadoop MapReduce, that work with MongoDB to store, process, and analyze large amounts of data. Organizations can use MongoDB and Hadoop to perform risk modeling, predictive analytics, and real-time data processing.

Mongo DB benefits

Over the years, MongoDB has become a go-to solution for many businesses looking for a powerful, highly scalable NoSQL database. But MongoDB is more than just a traditional document-based database, it boasts some great features that make it stand out from other DBMS.

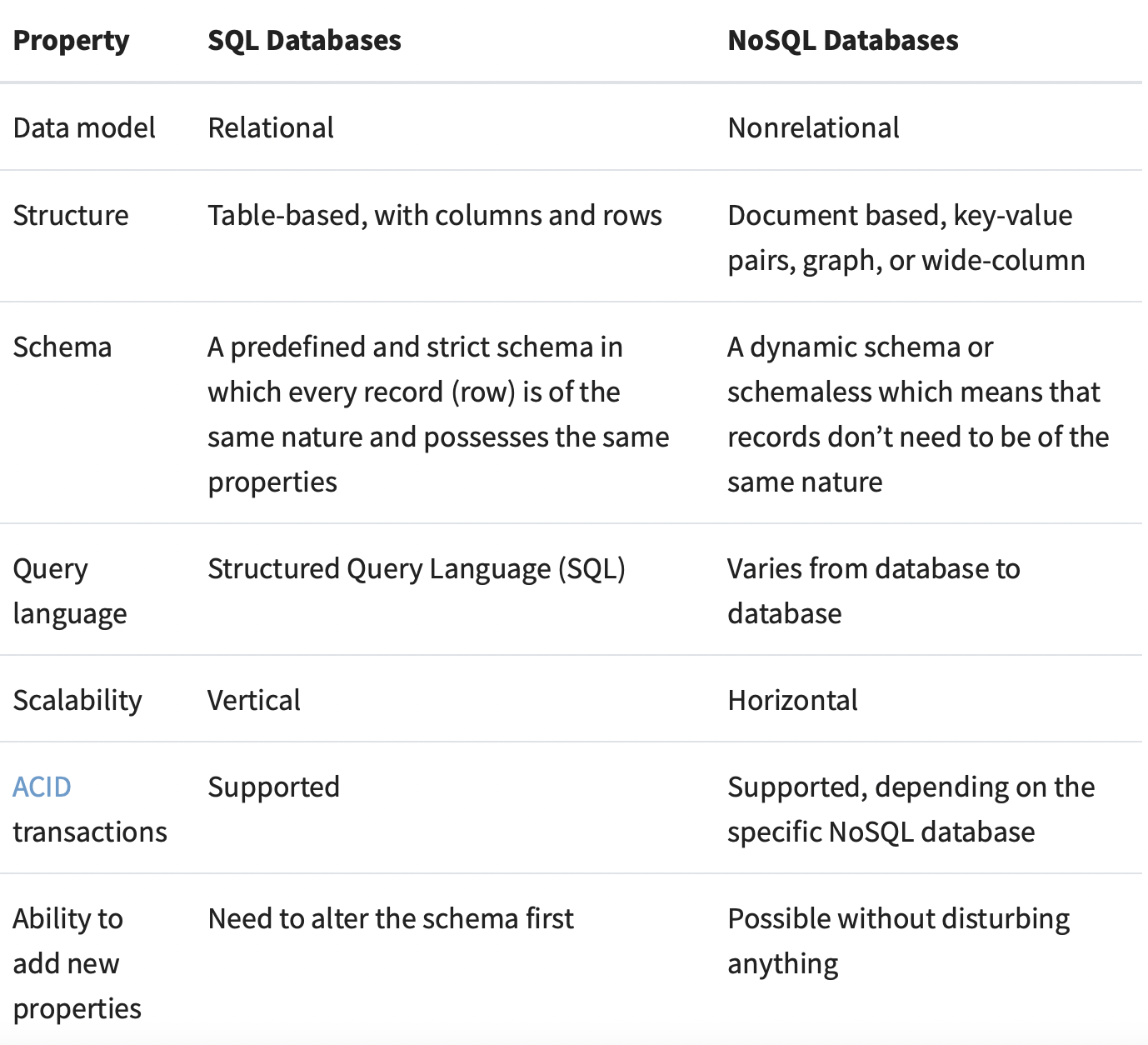
load balancing

As enterprise cloud applications expand and resource demands increase, problems can arise in ensuring service availability and reliability. MongoDB's load balancing shared process distributes large datasets to multiple virtual machines at once while maintaining acceptable read and write throughput. This horizontal scaling, called sharding, helps organizations increase the capacity of cloud-based deployments while avoiding the costs of vertical scaling of hardware.

ad hoc database query

One of MongoDB's biggest advantages over other databases is its ability to handle ad-hoc queries that don't require a predefined schema. MongoDB databases use a query language similar to SQL databases, making them very familiar to both beginners and advanced developers. This accessibility makes it easy to push, query, sort, update, and export data using common help methods and simple shell commands.

Multilingual support

One of the great things about MongoDB is its multilingual support. Several versions of MongoDB have been released, Python, PHP, Ruby, Node.js, C++, Scala, JavaScrip

<https://realpython.com/introduction-to-mongodb-and-python/>

There are many other differences between the two types of databases, but the databases above are some of the more important ones to know.

When choosing a database, you should carefully consider its advantages and disadvantages. You should also consider how your database fits into your particular scenario and application requirements. A good solution is to use a combination of SQL and NoSQL databases to handle different aspects of the wider system.

A review of MongoDB features

So far, we have learned what MongoDB is and what its main goals are. In this section, you will learn about some of MongoDB's more important features. On the database management side, MongoDB offers the following features:

* + Query support: Many standard query types are available, including matching (==), comparison (<, >), and regular expressions.
  + Data Coordination: Store virtually any kind of data, including structured, substructured, and even polymorphic.
  + Scalability: Handle more queries by simply adding more machines to your server cluster.
  + Flexibility and Agility: Rapid application development.
  + Document Orientation and Schemaless: All information about your data model can be stored in a single document.
  + Tunable Schema: Ability to change the database schema on-the-fly, reducing the time required to deliver new features or fix existing issues.
  + Relational database functions: Perform common actions on relational databases, such as indexing.

On the operational side, MongoDB offers several tools and features not found in other database systems.

* + Scalability: Whether you need a standalone server or a complete cluster of independent servers, MongoDB can grow to any size you need.
  + Support for load balancing: MongoDB automatically moves data between different shards.
  + Automatic Failover Support: If the primary server goes down, the new primary will automatically be up and running.
  + Management Tools: Machines can be tracked using the cloud-based MongoDB Management Service (MMS).
  + Memory efficiency: Thanks to memory-mapped files, MongoDB is often more efficient than relational databases.

All these features are very useful. For example, with the indexing feature, much of the data is kept in memory for quick retrieval. Even without indexing specific document keys, MongoDB caches quite a bit of data using techniques that are not used these days.



<https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mongodb>

MongoDB example

MongoDB has its fans, and here are some examples of organizations and companies that use databases.

* + Aadhar. India's unique identification project boasts the world's largest biometric database. The Aadhar project uses MongoDB to store vast demographic and biometric data of over 1.2 billion Indians. Aadhar also uses MongoDB to store project images.
  + eBay. A popular online auction site uses MongoDB for projects such as cloud management, metadata storage, search suggestions, and product classification.
  + Shutterfly. This well-known internet-based photo sharing company has over 6 billion images and a transaction rate of up to 10,000 operations/second. Shutterfly migrated from Oracle to MongoDB because they found a non-relational database better suited to their needs.
  + Electronic Arts. EA uses MongoDB for FIFA Online 3, the popular online multiplayer game.

Limits of MongoDB

1. MongoDB uses high memory for data storage.

2. BSON document size cannot exceed 16MB.

3. Windows database naming restrictions.

1. Document oriented database

A document database offers many advantages, including:

* + A fast, easy and intuitive data model for developers to work with.
  + A flexible schema that allows the data model to evolve as application needs change.
  + Ability to scale out horizontally.
* These advantages make the document database a versatile database that can be used across a wide variety of use cases and industries.

Document databases are considered non-relational (or NoSQL) databases. Document databases use flexible documents instead of storing data in fixed rows and columns. Document databases are the most popular alternative to tabular relational databases.

What are documents?

A document is a record in a document database. A document typically stores information about one object and its associated metadata.

Documents store data in field-value pairs. Values can be of various types and structures such as strings, numbers, dates, arrays, or objects. Documents can be saved in formats such as JSON, BSON, and XML.

Below is a JSON document that stores information about a user named Tom.

{

"\_id": 1,

"first\_name": "Tom",

"email": "tom@example.com",

"cell": "765-555-5555",

"likes": [

"fashion",

"spas",

"shopping"

],

"businesses": [

{

"name": "Entertainment 1080",

"partner": "Jean",

"status": "Bankrupt",

"date\_founded": {

"$date": "2012-05-19T04:00:00Z"

}

},

{

"name": "Swag for Tweens",

"date\_founded": {

"$date": "2012-11-01T04:00:00Z"

}

}

]

}

Collection

A collection is a group of documents. Collections typically store documents with similar content.

Document databases have flexible schemas, so not all documents in a collection need to have the same fields. Note that some document databases offer schema validation, so you can optionally lock down your schema if you want.

Continuing the example above, documents containing information about Tom can be stored in a collection named users. More documents can be added to the users collection to store information about other users. For example, the following document that stores information about Donna can be added to the users collection.

{

"\_id": 2,

"first\_name": "Donna",

"email": "donna@example.com",

"spouse": "Joe",

"likes": [

"spas",

"shopping",

"live tweeting"

],

"businesses": [

{

"name": "Castle Realty",

"status": "Thriving",

"date\_founded": {

"$date": "2013-11-21T04:00:00Z"

}

}

]

}

Note that Donna's document does not contain the same fields as Tom's document. The users collection leverages a flexible schema to store information that exists for each user.

CRUD operations

Document databases typically have an API or query language that allows developers to perform CRUD (Create, Read, Update, Delete) operations.

* + Creation: Documents can be created in the database. Each document has a unique identifier.
  + Read: Documents can be read from the database. An API or query language allows developers to query documents using unique identifiers or field values. Indexes can be added to the database to improve read performance.
  + Update: Existing documents can be updated in whole or in part.
  + Delete: Documents can be deleted from the database.

What are the main features of the document database?

A document database has the following main features:

* + Document model: Data is stored in documents (unlike other databases that store data in structures such as tables and graphs). Documents are mapped to objects in the most common programming languages, allowing developers to rapidly develop applications.
  + Flexible Schema: The document database has a flexible schema. That is, not all documents in the collection have to have the same fields. Note that some document databases support schema validation, so the schema can optionally be locked down.
  + Distributed and resilient: Document databases are distributed, which allows for horizontal scaling (which is typically cheaper than vertical scaling) and data distribution. Document databases provide resilience through replication.
  + Query via API or Query Language: Document databases have an API or query language that allows developers to perform CRUD operations on the database. Developers can query documents based on unique identifiers or field values.

How is a document database different from a relational database?

Three key elements that distinguish document databases from relational databases.

1. Intuitiveness of the data model: Documents are mapped to objects in code, making it much more natural to work with. There is no need to decompose data between tables, perform expensive joins, or integrate another object-relational mapping (ORM) layer. Data that is accessed together is stored together, so developers write less code and end users get better performance.
2. The ubiquity of JSON documents: JSON has become an established standard for data exchange and storage. JSON documents are lightweight, language-neutral, and human-readable. Documents are a superset of all other data models, allowing developers to manipulate data in any way their application requires: rich objects, key-value pairs, tables, geospatial and time-series data, or graph nodes and edges. can be structured.
3. Schema flexibility: The document schema is dynamic and self-describing, so the developer does not need to predefine it in the database first. Fields may vary from document to document. Developers can change the structure at any time and avoid destructive schema migrations. Some document databases offer schema validation, so you can optionally apply rules governing document structure.

How much easier are documents to work with than tables?

Developers typically find working with data in documents easier and more intuitive than working with data in tables. Documents are mapped to data structures in most common programming languages. Developers don't have to worry about manually splitting related data across multiple tables on save or joining on retrieval. Also, you don't need to use an ORM to handle manipulation of data. Instead, you can easily manipulate the data directly in your application.

Let's look again at the documentation for the user named Tom.

user

{

"\_id": 1,

"first\_name": "Tom",

"email": "tom@example.com",

"cell": "765-555-5555",

"likes": [

"fashion",

"spas",

"shopping"

],

"businesses": [

{

"name": "Entertainment 1080",

"partner": "Jean",

"status": "Bankrupt",

"date\_founded": {

"$date": "2012-05-19T04:00:00Z"

}

},

{

"name": "Swag for Tweens",

"date\_founded": {

"$date": "2012-11-01T04:00:00Z"

}

}

]

}

All information about Tom is stored in one document.

Why not use JSON in relational databases?

Document databases allow developers to build more quickly, so most relational databases have added support for JSON. However, just adding the JSON data type doesn't give you the benefits of a native document database. Why? Because the relational approach hurts developer productivity instead of making it more productive. These are some of the things developers have to deal with.

Proprietary extension

Working with documents means using custom, vendor-specific SQL functions that are unfamiliar to most developers and won't work with your favorite SQL tools. Add low-level JDBC/ODBC drivers and ORMs and you're faced with a complicated development process and lost productivity.

primitive data processing

Viewing JSON data as simple strings and numbers rather than the rich data types supported by native document databases such as MongoDB makes calculating, comparing, and sorting the data complex and error-prone.

Poor data quality and rigid tablest

Relational databases are used to validate the schema of documents.There is no way to apply quality controls to JSON data, as it provides little for You also need to define a schema for regular tabular data, with all the overhead that comes when you need to modify your tables as your application's functionality evolves.

low performance

Most relational databases do not keep statistics on JSON data, which hinders query planners from optimizing queries against documents and tuning queries.

No native scale out

Traditional relational databases do not provide a way to divide (“shard”) the database across multiple instances to scale as your workload grows. Instead, you have to implement sharding in your application layer or resort to expensive scale-up systems.

What are the pros and cons of document databases?

Document databases have many strengths.

* + The documentation model is ubiquitous and intuitive, enabling rapid software development.
  + Flexible schema allows you to change your data model as your application requirements change.
  + The document database has a rich API and query language that allows developers to easily interact with the data.
  + The document database is distributed (allowing for horizontal scaling and global data distribution) and elastic.

These strengths make document databases ideal for general-purpose databases.

A common weakness people cite about document databases is that many do not support multi-document ACID transactions.

What are the document database use cases?

A document database is a general-purpose database that serves a variety of use cases in both transactional and analytical applications.

* + Single View or Data Hub
  + Customer data management and personalization
  + Internet of Things (IoT) and time series data
  + Product catalog and content management
  + Payment processing
  + Mobile app
  + Mainframe offload
  + Operational analysis
  + Real-time analytics

1. Reference list

* Install MongoDB (Mac)

<https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/#install-mongodb-community-edition-on-macos>

[**https://linuxhint.com/install-mongodb-mac-m1/**](https://linuxhint.com/install-mongodb-mac-m1/)

[**https://www.positronx.io/install-mongodb-on-macos/?amp**](https://www.positronx.io/install-mongodb-on-macos/?amp)

[**https://dev.to/honorezemagho/ultimate-guide-to-install-and-launch-mongodb-on-macos-12kp**](https://dev.to/honorezemagho/ultimate-guide-to-install-and-launch-mongodb-on-macos-12kp)

[**https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/**](https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/)

* Basic operations of MongoDB

[**https://www.mongodb.com/docs/manual/tutorial/getting-started/**](https://www.mongodb.com/docs/manual/tutorial/getting-started/)

[**https://www.bmc.com/blogs/mongo-shell-basic-commands/**](https://www.bmc.com/blogs/mongo-shell-basic-commands/)

[**https://www.freecodecamp.org/news/learn-mongodb-a4ce205e7739/amp/**](https://www.freecodecamp.org/news/learn-mongodb-a4ce205e7739/amp/)

[**https://www.digitalocean.com/community/tutorials/how-to-use-the-mongodb-shell**](https://www.digitalocean.com/community/tutorials/how-to-use-the-mongodb-shell)

[**https://www.bmc.com/blogs/mongo-shell-basic-commands/**](https://www.bmc.com/blogs/mongo-shell-basic-commands/)

[**https://www.digitalocean.com/community/tutorials/how-to-use-the-mongodb-shell**](https://www.digitalocean.com/community/tutorials/how-to-use-the-mongodb-shell)

[**https://www.tutorialspoint.com/mongodb/mongodb\_create\_database.htm**](https://www.tutorialspoint.com/mongodb/mongodb_create_database.htm)

[**https://www.freecodecamp.org/news/learn-mongodb-a4ce205e7739/amp/**](https://www.freecodecamp.org/news/learn-mongodb-a4ce205e7739/amp/)

[**https://kb.objectrocket.com/mongo-db/start-mongodb-mac-how-to-start-mongodb-on-a-mac-438**](https://kb.objectrocket.com/mongo-db/start-mongodb-mac-how-to-start-mongodb-on-a-mac-438)

[**https://www.digitalocean.com/community/tutorials/how-to-use-the-mongodb-shell**](https://www.digitalocean.com/community/tutorials/how-to-use-the-mongodb-shell)

[**https://www.codeproject.com/Articles/828392/Basics-of-MongoDB**](https://www.codeproject.com/Articles/828392/Basics-of-MongoDB)

[**https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/**](https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/)

* What is MongoDB

[**https://www.knowledgehut.com/blog/web-development/install-mongodb-mac**](https://www.knowledgehut.com/blog/web-development/install-mongodb-mac)

[**https://www.ibm.com/topics/mongodb**](https://www.ibm.com/topics/mongodb)

[**https://realpython.com/introduction-to-mongodb-and-python/**](https://realpython.com/introduction-to-mongodb-and-python/)

[**https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mongodb**](https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mongodb)

* Document oriented database

[**https://www.mongodb.com/document-databases**](https://www.mongodb.com/document-databases)

[**https://www.techtarget.com/searchdatamanagement/tip/NoSQL-database-types-explained-Document-based-databases**](https://www.techtarget.com/searchdatamanagement/tip/NoSQL-database-types-explained-Document-based-databases)

[**https://en.m.wikipedia.org/wiki/Document-oriented\_database**](https://en.m.wikipedia.org/wiki/Document-oriented_database)

[**https://www.digitalocean.com/community/conceptual-articles/an-introduction-to-document-oriented-databases**](https://www.digitalocean.com/community/conceptual-articles/an-introduction-to-document-oriented-databases)

[**https://www.geeksforgeeks.org/document-databases-in-nosql/amp/**](https://www.geeksforgeeks.org/document-databases-in-nosql/amp/)