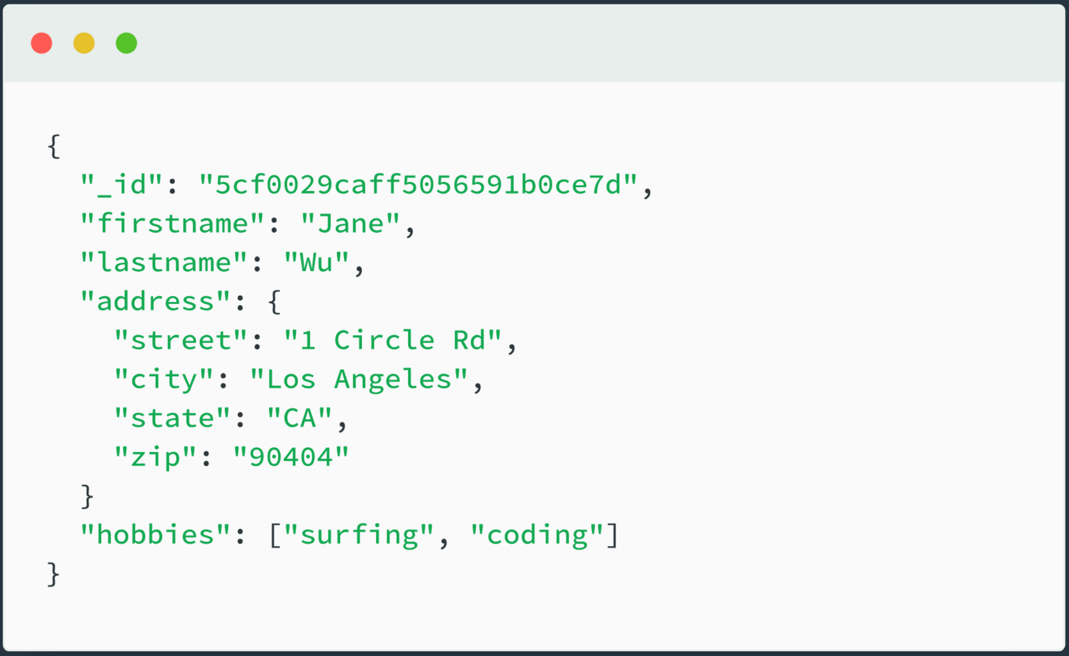


```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  }
  "hobbies": ["surfing", "coding"]
}
```

What is a Document Database?

A document database (also known as a document-oriented database or a document store) is a database that stores information in documents.

Document databases offer a variety of advantages, including:

- An intuitive data model that is fast and easy for developers to work with.
- A flexible schema that allows for the data model to evolve as application needs change.
- The ability to horizontally scale out.

Because of these advantages, document databases are general-purpose databases that can be used in a variety of use cases and industries.

Document databases are considered to be non-relational (or NoSQL) databases. Instead of storing data in fixed rows and columns, document databases use flexible documents.

Document databases are the most popular alternative to tabular, relational databases. Learn more about NoSQL databases.

In this article, we'll explore answers to the following questions:

What are documents?

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

Documents store data in field-value pairs. The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects. Documents can be stored in formats like 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"
           }
        }
     ]
  }
```

Collections

A collection is a group of documents. Collections typically store documents that have similar contents.

Not all documents in a collection are required to have the same fields, because document databases have a flexible schema. Note that some document databases provide schema validation, so the schema can optionally be locked down when needed.

Continuing with the example above, the document with information about Tom could be stored in a collection named `users`. More documents could be added to the `users` collection in order to store information about other users. For example, the document below that stores information about Donna could 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 the document for Donna does not contain the same fields as the document for Tom. The `users` collection is leveraging a flexible schema to store the information that exists for each user.

CRUD operations

Document databases typically have an API or query language that allows developers to execute the CRUD (create, read, update, and delete) operations.

- **Create:** Documents can be created in the database. Each document has a unique identifier.
- **Read:** Documents can be read from the database. The API or query language allows developers to query for documents using their unique identifiers or field values. Indexes can be added to the database in order to increase read performance.
- **Update:** Existing documents can be updated – either in whole or in part.
- **Delete:** Documents can be deleted from the database.

What are the key features of document databases?

Document databases have the following key features:

- **Document model:** Data is stored in documents (unlike other databases that store data in structures like tables or graphs). Documents map to objects in most popular programming languages, which allows developers to rapidly develop their applications.
- **Flexible schema:** Document databases have a flexible schema, meaning that not all documents in a collection need to have the same fields. Note that some document databases support schema validation, so the schema can be optionally locked down.
- **Distributed and resilient:** Document databases are distributed, which allows for horizontal scaling (typically cheaper than vertical scaling) and data distribution. Document databases provide resiliency through replication.
- **Querying through an API or query language:** Document databases have an API or query language that allows developers to execute the CRUD operations on the database. Developers have the ability to query for documents based on unique identifiers or field values.

What makes document databases different from relational databases?

Three key factors differentiate document databases from relational databases:

1. **The intuitiveness of the data model:** Documents map to the objects in code, so they are

much more natural to work with. There is no need to decompose data across tables, run expensive joins, or integrate a separate Object Relational Mapping (ORM) layer. Data that is accessed together is stored together, so developers have less code to write and end users get higher performance.
2. **The ubiquity of JSON documents:** JSON has become an established standard for data interchange and storage. JSON documents are lightweight, language-independent, and human-readable. Documents are a superset of all other data models so developers can structure data in the way their applications need – rich objects, key-value pairs, tables, geospatial and time-series data, or the nodes and edges of a graph.
3. **The flexibility of the schema:** A document's schema is dynamic and self-describing, so developers don't need to first pre-define it in the database. Fields can vary from document to document. Developers can modify the structure at any time, avoiding disruptive schema migrations. Some document databases offer schema validation so you can optionally enforce rules governing document structures.

Learn more about NoSQL vs. relational databases.

How much easier are documents to work with than tables?

Developers commonly find working with data in documents to be easier and more intuitive than working with data in tables. Documents map to data structures in most popular programming languages. Developers don't have to worry about manually splitting related data across multiple tables when storing it or joining it back together when retrieving it. They also don't need to use an ORM to handle manipulating the data for them. Instead, they can easily work with the data directly in their applications.

Let's take another look at a document for a user named Tom.

Users

```
{
     "_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 of the information about Tom is stored in a single document.

Now let's consider how we can store that same information in a relational database. We'll begin by creating a table that stores the basic information about the user.

Users

ID	first_name	email	cell
1	Tom	tom@example.com	765-555-5555

A user can like many things (meaning there is a one-to-many relationship between a user and likes), so we will create a new table named "Likes" to store a user’s likes. The Likes table will have a foreign key that references the ID column in the Users table.

Likes

ID	user_id	like
10	1	fashion
11	1	spas
12	1	shopping

Similarly, a user can run many businesses, so we will create a new table named "Businesses" to store business information. The Businesses table will have a foreign key that references the `ID` column in the `Users` table.

Businesses

ID	user_id	name	partner	status	date_founded
20	1	Entertainment 1080	Jean	Bankrupt	2011-05-19
21	1	Swag for Tweens	NULL	NULL	2012-11-01

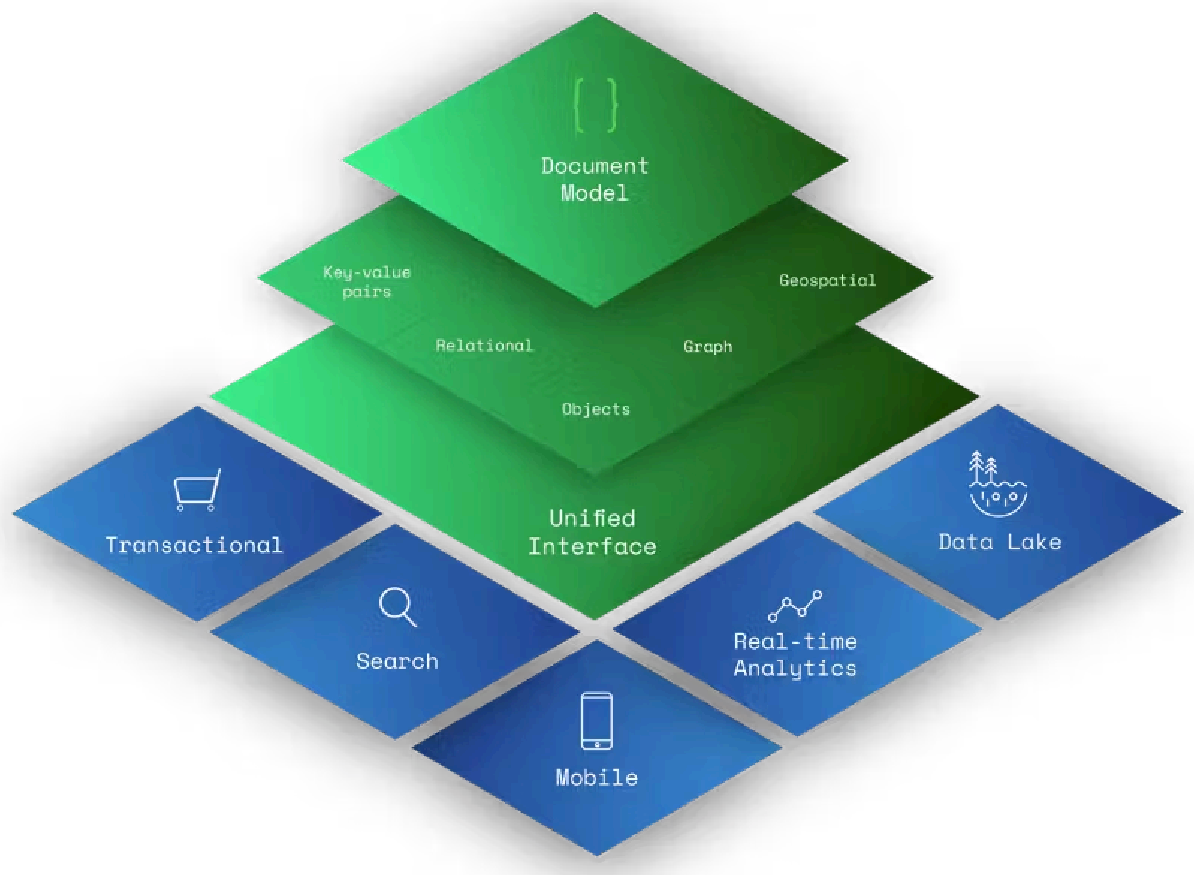
In this simple example, we saw that data about a user could be stored in a single document in a document database or three tables in a relational database. When a developer wants to retrieve or update information about a user in the document database, they can write one query with zero joins. Interacting with the database is straightforward, and modeling the data in the database is intuitive.

Visit Mapping Terms and Concepts from SQL to MongoDB to learn more.

What are the relationships between document databases and other databases?

The document model is a superset of other data models, including key-value pairs, relational, objects, graph, and geospatial.

- **Key-value pairs** can be modeled with fields and values in a document. Any field in a document can be indexed, providing developers with additional flexibility in how to query the data.
- **Relational data** can be modeled differently (and some would argue more intuitively) by keeping related data together in a single document using embedded documents and arrays. Related data can also be stored in separate documents, and database references can be used to connect the related data.
- Documents map to **objects** in most popular programming languages.
- **Graph** nodes and/or edges can be modeled as documents. Edges can also be modeled through database references. Graph queries can be run using operations like $graphLookup.
- **Geospatial** data can be modeled as arrays in documents.



The document model is a superset of other data models

Due to their rich data modeling capabilities, document databases are general-purpose databases that can store data for a variety of use cases.

Why not just use JSON in a relational database?

With document databases empowering developers to build faster, most relational databases have added support for JSON. However, simply adding a JSON data type does not bring the benefits of a native document database. Why? Because the relational approach detracts from developer productivity, rather than improve it. These are some of the things developers have to deal with.

Proprietary Extensions

Working with documents means using custom, vendor-specific SQL functions which will not be familiar to most developers, and which don’t work with your favorite SQL tools. Add low-level JDBC/ODBC drivers and ORMs and you face complex development processes resulting in low productivity.

Primitive Data Handling

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

Poor Data Quality & Rigid Tables

Relational databases offer little to validate the schema of documents, so you have no way to apply quality controls against your JSON data. And you still need to define a schema for your regular tabular data, with all the overhead that comes when you need to alter your tables as your application's features evolve.

Low Performance

Most relational databases do not maintain statistics on JSON data, preventing the query planner from optimizing queries against documents, and you from tuning your queries.

No native scale-out

Traditional relational databases offer no way for you to partition ("shard") the database across multiple instances to scale as workloads grow. Instead you have to implement sharding yourself in the application layer, or rely on expensive scale-up systems.

What are the strengths and weaknesses of document databases?

Document databases have many strengths:

- The **document model** is ubiquitous, intuitive, and enables rapid software development.
- The **flexible schema** allows for the data model to change as an application's requirements change.
- Document databases have **rich APIs and query languages** that allow developers to easily interact with their data.
- Document databases are **distributed** (allowing for horizontal scaling as well as global data distribution) and **resilient**.

These strengths make document databases an excellent choice for a general-purpose database.

A common weakness that people cite about document databases is that many do not support multi-document ACID transactions. We estimate that 80%-90% of applications that leverage the document model will **not** need to use multi-document transactions.

Note that some document databases like MongoDB support multi-document ACID transactions.

Visit What are ACID Transactions? to learn more about how the document model mostly eliminates the need for multi-document transactions and how MongoDB supports transactions in the rare cases where they are needed.

What are the use cases for document databases?

Document databases are general-purpose databases that serve a variety of use cases for both transactional and analytical applications:

- Single view or data hub
- Customer data management and personalization
- Internet of Things (IoT) and time-series data
- Product catalogs and content management
- Payment processing
- Mobile apps
- Mainframe offload
- Operational analytics
- Real-time analytics

Visit Use Case Guidance: Where to Use MongoDB to learn more about each of the applications listed above.

Summary

Document databases utilize the intuitive, flexible document data model to store data. Document databases are general-purpose databases that can be used for a variety of use cases across industries.

Get started with document databases by creating a database in MongoDB Atlas, MongoDB's database as a service. Atlas has a generous forever-free tier you can use to kick the tires and explore the document model.

Try out the power of documents for free with our sample data in MongoDB Atlas

Try it now

FAQ

What are document databases good for?

Document databases are general-purpose databases that can be used in a variety of use cases across industries. Visit When to Use NoSQL to learn more.

Is MongoDB a document database?

Yes, MongoDB is a general-purpose document database.

What is an example of a document database?

MongoDB is the world's most popular document database. Other examples of document databases include CouchDB and Firebase.

How do document databases work?

Document databases store information in documents. Document databases have rich APIs and query languages that can be used to execute the CRUD (create, read, update, and delete) operations. They have flexible schemas, allowing developers to easily evolve their data models as their application requirements change.

How are documents stored in a database?

Document databases store information in documents. Each document typically contains information about one object and any related metadata. Documents with similar contents are grouped together in collections. Groups of collections are stored in databases.

Which field is always the first field in a document?

In MongoDB, the first field in every document is named `_id`. The `_id` field serves as a unique identifier for the document. See the official MongoDB documentation for more information.

Note that each document database management system has its own field requirements.

How is MongoDB data stored?

MongoDB stores data in BSON (Binary JSON) documents.

Is MongoDB free to use?

Yes, MongoDB has two free options:

- MongoDB Atlas, MongoDB's database as a service, has a generous, forever-free option that is great for kicking the tires and learning to use MongoDB.
- If you prefer to self-host MongoDB, you can use MongoDB Community Server in accordance with the Server Side Public License (SSPL).

Document database vs. relational database

The most obvious difference between a document database and a relational database is the way data is modeled. Document databases typically model data using flexible JSON-like documents with field-value pairs. Relational databases typically model data using rigid tables with fixed rows and columns.

Contact Us

Customer Portal

Atlas Status

Customer Support

Manage Cookies

Social

GitHub

Stack Overflow

LinkedIn

YouTube

X

Twitch

Facebook