

Comparison between SQL and NoSQL Databases

Points	SQL Database	NoSQL Database
Data Model	SQL databases are based on a rigid, predefined schema that enforces a tabular structure with rows and columns. Each table represents an entity, and relationships are established using foreign keys.	NoSQL databases, on the other hand, use flexible, schema-less data models. They can be document-based (e.g., MongoDB), key-value pairs (e.g., Redis), columnar (e.g., Cassandra), or graph-based (e.g., Neo4j).
Scalability	SQL databases traditionally scale vertically by adding more powerful hardware to a single server, which can be limited in terms of scalability.	NoSQL databases are designed to scale horizontally, allowing for easy distribution of data across multiple servers or nodes. This enables them to handle large amounts of data and high read/write loads effectively.
Querying Language	SQL databases use the structured query language (SQL) for querying and manipulating data. SQL provides a standardized syntax for performing complex queries, joins, aggregations, and data modifications.	NoSQL databases, however, have varied query languages. Some use specialized query languages (e.g., MongoDB's query language), while others use APIs or even simple key-value lookups.
Data Consistency	SQL databases typically adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties, ensuring strong consistency and transactional integrity.	NoSQL databases often sacrifice some level of consistency in favour of high availability and partition tolerance (the CAP theorem). They offer eventual consistency or other consistency models, allowing for faster performance and scalability.
Schema Flexibility	SQL databases have a predefined schema that must be defined before data insertion. Modifying the schema can be challenging and may require altering existing data.	In contrast, NoSQL databases offer schema flexibility, allowing documents or records to have varying structures within the same collection or table. This flexibility enables agile development and the ability to evolve the data model over time.
Use Cases	SQL databases are well-suited for applications that require complex transactions, strict data integrity, and structured data with predefined relationships. They are commonly used for financial systems, e-commerce platforms, and applications with well-defined schemas.	NoSQL databases excel in handling unstructured or semi-structured data, handling large-scale distributed systems, and enabling rapid development in agile environments. They are often used for content management, real-time analytics, IoT applications, and handling high-velocity data.

It's important to note that the decision to choose between a SQL or NoSQL database depends on factors such as the nature of the data, scalability requirements, performance needs, development flexibility, and specific use case requirements. Each type has its strengths and weaknesses, and the choice should align with the project's objectives and characteristics.

Different types of databases are

Relational

- a. Data stored must be structured with specific schema

Document Oriented

- a. Data stored must not follow a specific schema
- b. It has a collection of documents. Collections can be referred to python list and document as a python dictionary with a key-value pair, which can hold any datatype of value

Key-Value oriented

- a. Databases like Redis follow a python dictionary like structure with key-value pair
- b. Key points to a value.
- c. These type of database are fast as data is stored in memory rather than disk
- d. That means such databases strictly depend on the capacity of memory used by the system
- e. These databases are NOT good in maintaining relational data

Wide column

- a. Database like Cassandra have rows-columns like relational database but are also schema-less

Graph database

- a. Database like ArangoDB store relational data through the use of individual nodes & their edge relations

Database like Elastic Search Database

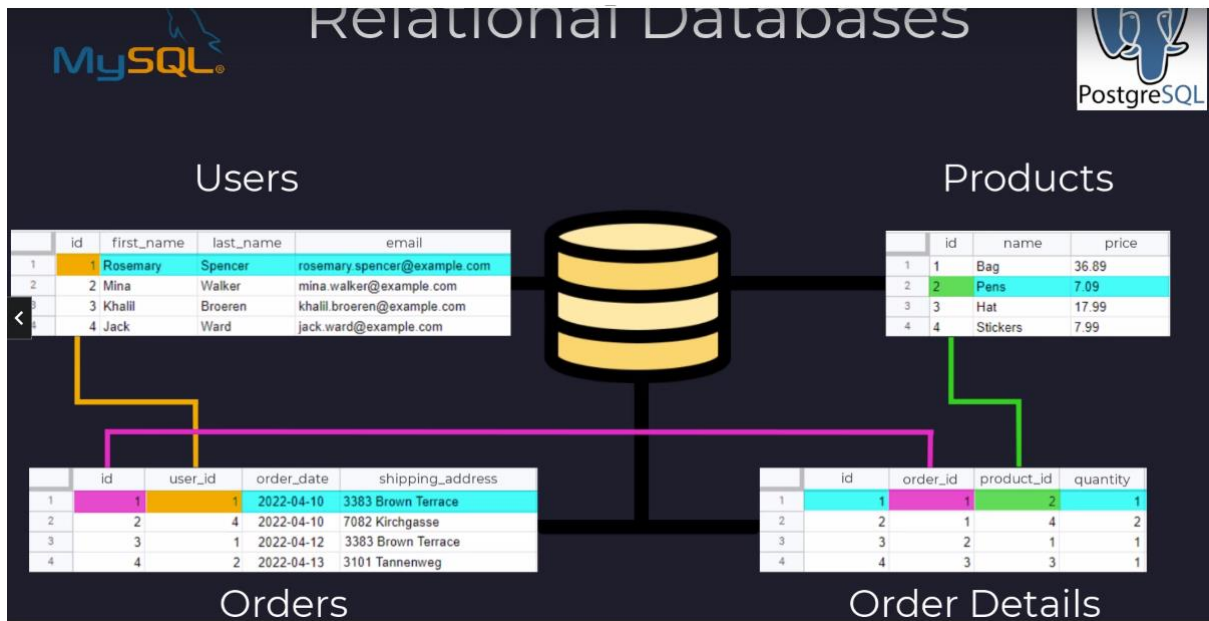
Elasticsearch falls into the category of NoSQL databases, specifically within the document-oriented database type. NoSQL databases differ from traditional relational databases by offering a more flexible schema design and scalability for handling large amounts of data.

As a document-oriented database, Elasticsearch stores and retrieves data in the form of JSON documents. Each document is self-contained and can have a different structure, allowing for dynamic and schema-less data modeling. This flexibility is particularly useful when dealing with unstructured or semi-structured data, as well as in scenarios where the data schema may evolve over time.

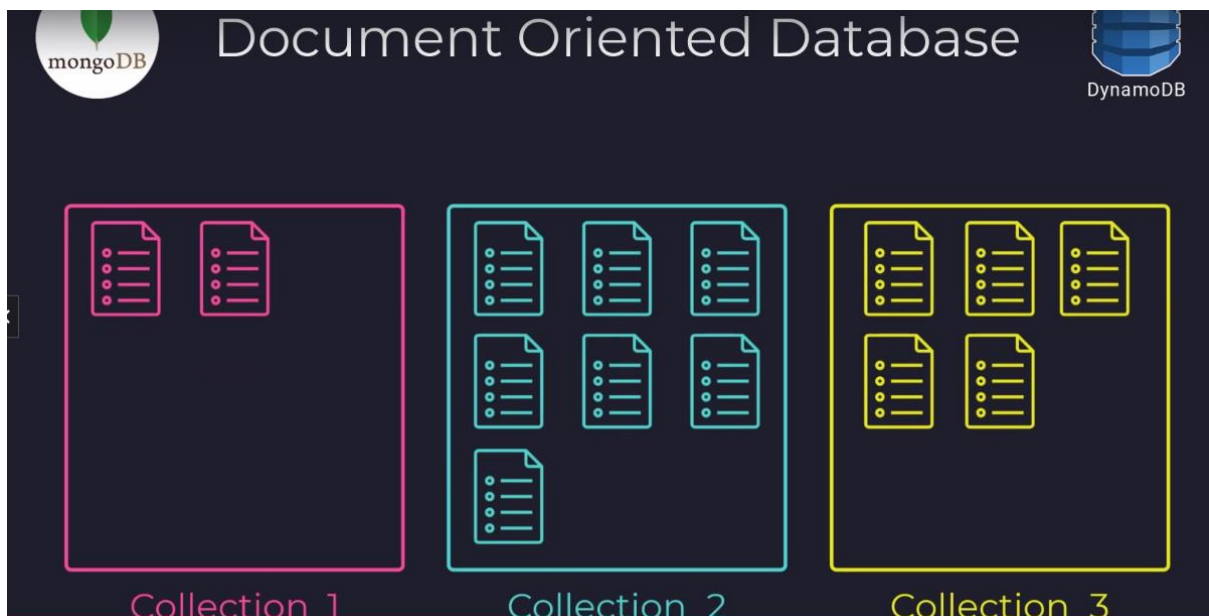
However, it's important to note that Elasticsearch is not limited to being just a database. It is often used as a distributed search and analytics engine as well.

By combining full-text search capabilities with the document storage and retrieval features, Elasticsearch provides a powerful solution for building search applications and performing real-time analytics on large datasets.

Relational DB



Document Oriented DB



Similar to Python dictionary with key-value pair

```
[ { "key": "value",  
  "name": "Jake",  
  "num": 42  
}, { "name": "Ann",  
  "num": 23.5  
  "hello": "there"  
}
```

Example :

1. Users can have multiple emails


Users	Products	Orders
<pre>{ "_id": ObjectId("62537c4fb08fa266d5f494eb"), "first_name": "Rosemary", "last_name": "Spencer", "email": "rosemary.spencer@example.com" }</pre>	<pre>{ "_id": ObjectId("62537ddcb08fa266d5f494fb"), "name": "Bag", "price": 36.89 }</pre>	<pre>{ "_id": ObjectId("62537ec6b08fa266d5f4950d"), "user_id": ObjectId("62537c4fb08fa266d5f494eb"), "order_date": 2021-04-10T23:11:58.318+00:00, "shipping_address": "3303 Brown Terrace" "items": Array ~ 0: Object product_id: ObjectId("62537e0cb08fa266d5f494fc") quantity: 1 ~ 1: Object product_id: ObjectId("62537e3cb08fa266d5f494fe") quantity: 2 }</pre>
<pre>{ "_id": ObjectId("62537d04b08fa266d5f494ed"), "first_name": "Mina", "last_name": "Walker", "email": Array 0: "mina.walker@example.com" 1: "mina.w@example2.com" }</pre>	<pre>{ "_id": ObjectId("62537e0cb08fa266d5f494fc"), "name": "Pens", "price": 7.09 }</pre>	
<pre>{ "_id": ObjectId("62537d51b08fa266d5f494ee"), "first_name": "Khalil", "last_name": "Broeren", "email": "khalil.broeren@example.com" }</pre>	<pre>{ "_id": ObjectId("62537e24b08fa266d5f494fd"), "name": "Hat", "price": 17.99 }</pre>	<pre>{ "_id": ObjectId("62538364b08fa266d5f4950f"), "user_id": ObjectId("62537d72b08fa266d5f494ef"), "order_date": 2021-04-10T23:15:58.318+00:00, "shipping_address": "7082 Kirchgasse" "items": Array ~ 0: Object product_id: ObjectId("62537dd6b08fa266d5f494fb") quantity: 1 }</pre>
<pre>{ "_id": ObjectId("62537d72b08fa266d5f494ef"), "first_name": "Jack", "last_name": "Ward", "email": "jack.ward@example.com" }</pre>	<pre>{ "_id": ObjectId("62537e3cb08fa266d5f494fe"), "name": "Stickers", "price": 7.99 }</pre>	

We can still utilize relational ability by referencing the id's from other collections

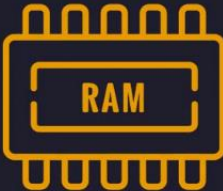

Users	Products	Orders
<pre> _id: ObjectId("62537c4fb08fa266d5f494eb") first_name: "Rosemary" last_name: "Spencer" email: "rosemary.spencer@example.com" _id: ObjectId("62537d04b08fa266d5f494ed") first_name: "Mina" last_name: "Walker" email: Array 0: "mina.walker@example.com" 1: "mina.w@example2.com" _id: ObjectId("62537d51b08fa266d5f494ee") first_name: "Khalil" last_name: "Broeren" email: "khalil.broeren@example.com" _id: ObjectId("62537d72b08fa266d5f494ef") first_name: "Jack" last_name: "Ward" email: "jack.ward@example.com" </pre>	<pre> _id: ObjectId("62537ddcb08fa266d5f494fb") name: "Bag" price: 36.89 _id: ObjectId("62537e0cb08fa266d5f494fc") name: "Pens" price: 7.09 _id: ObjectId("62537e24b08fa266d5f494fd") name: "Hat" price: 17.99 _id: ObjectId("62537e3cb08fa266d5f494fe") name: "Stickers" price: 7.99 </pre>	<pre> id: ObjectId("62537ecb08fa266d5f4950d") user_id: ObjectId("62537c4fb08fa266d5f494eb") order_date: 2021-04-10T23:11:58.318+00:00 shipping_address: "3383 Brown Terrace" items: Array 0: Object product_id: ObjectId("62537e0cb08fa266d5f494fc") quantity: 1 1: Object product_id: ObjectId("62537e3cb08fa266d5f494fe") quantity: 2 _id: ObjectId("62538364b08fa266d5f4950f") user_id: ObjectId("62537d72b08fa266d5f494ef") order_date: 2021-04-10T23:15:58.318+00:00 shipping_address: "7082 Kirchgasse" items: Array 0: Object product_id: ObjectId("62537ddcb08fa266d5f494fb") quantity: 1 </pre>

Key-Value Databases

Key-Value



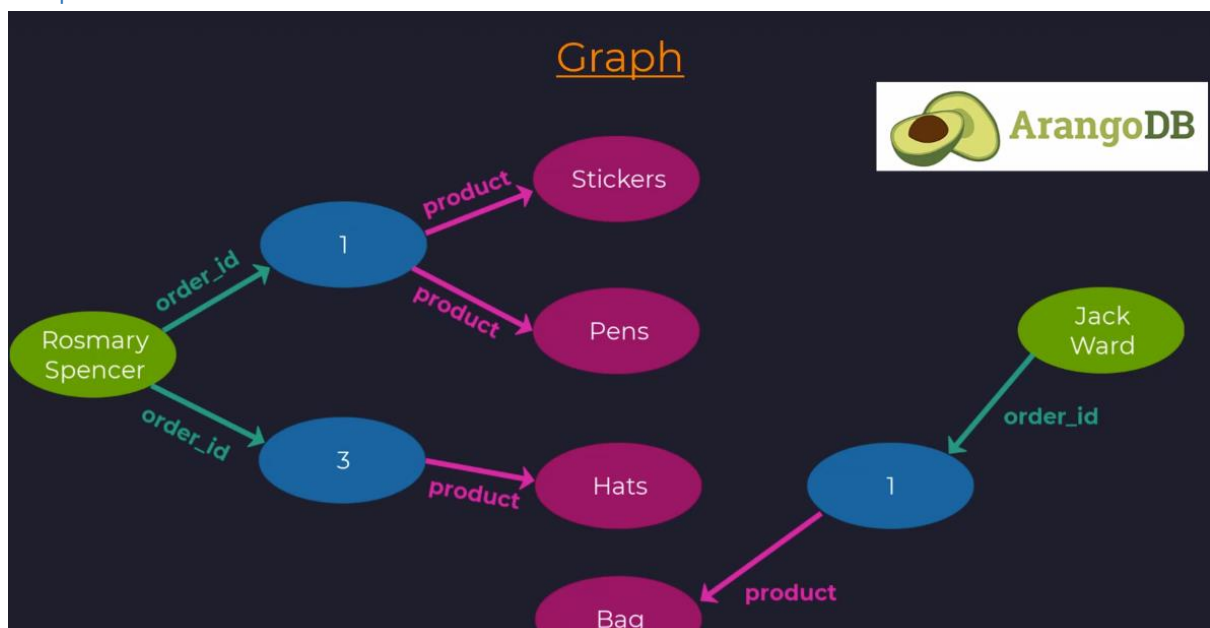
user:1:first_name	→	Rosemary
user:1:last_name	→	Spencer
user:2:first_name	→	Mina
user:2:last_name	→	Walker
product:1:name	→	Bag
product:1:price	→	36.89

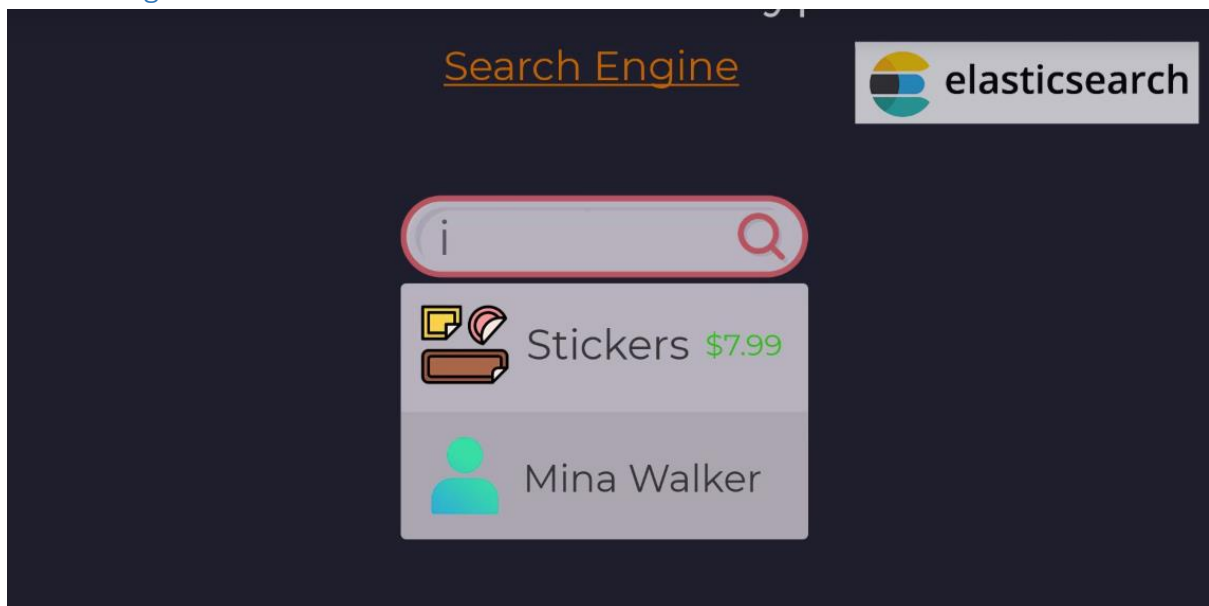



Wide Column Database



Graph Database



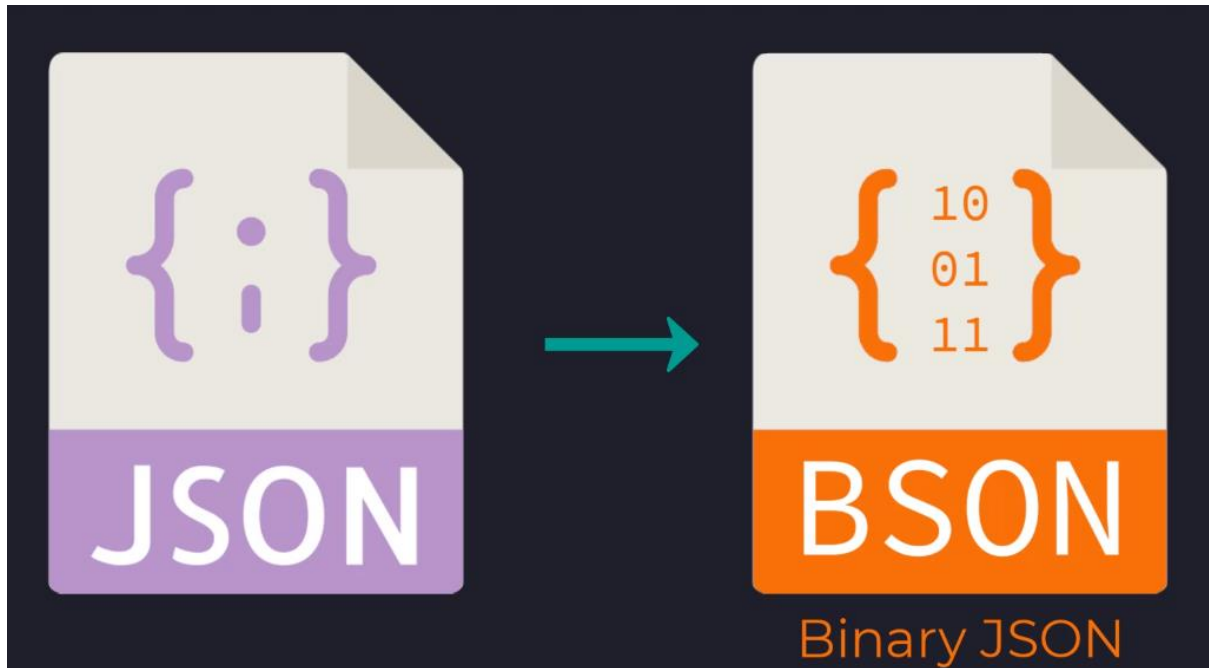


MongoDB Data Structure Overview

- Documentation - <https://www.mongodb.com/docs/manual/reference/limits/>
- MongoDB follows a document oriented paradigm. Documents are organized within collection.
- Documents are Json based that means they follow a key-value schema. Hence we can store data in document irrespective of their datatypes



The way MongoDB is able to efficiently store and retrieve the JSON document data is by encoding it to BSON (Binary JSON)



- This conversion happens behind the scenes so we do not have to explicitly assign the data types of every value within document
- BSON has some limitations

MongoDB BSON Limits

- 16 MB max per document

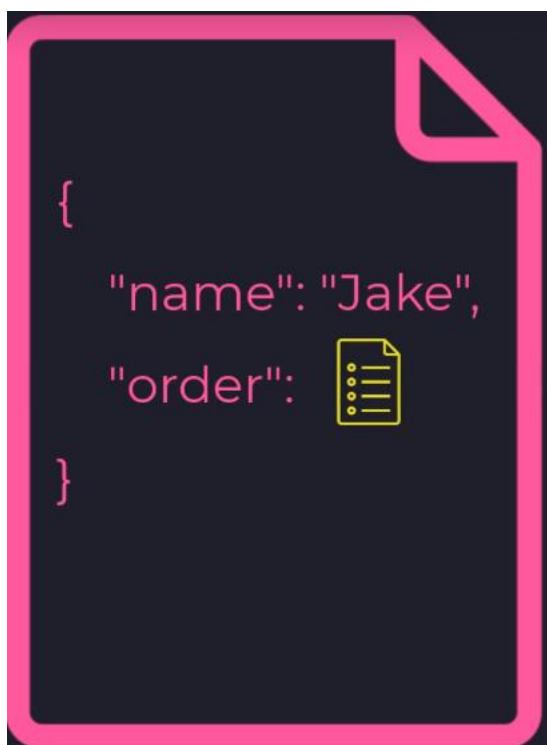


- Images, videos, large format files are not recommended to be stored in MongoDB. Instead they should be hosted on a server or cloud service like Amazon S3 storage

- The reason to it is that the maximum BSON document size can be 16 MB.
To store documents larger than the maximum size, MongoDB provides the GridFS API



- Document can also contain their own embedded sub documents. There is a limit of 100 levels of embedded documents. This does not mean you can have 100 embedded documents in a single document. Instead assume you have a document that has an embedded document that in turn has 98 levels of embedded documents




- MongoDB doesn't support the declaration of duplicate field names within the same document. Instead you can use arrays to hold duplicate values. These array values does not need to be of same datatype and can store sub documents.



To summarize

MongoDB BSON Limits

- 16 MB max per document
- 100 levels of embedded subdocuments 
- Can't have duplicate field names
- Can find more information at:

<https://www.mongodb.com/docs/manual/reference/limits/>

