# Introduction to NoSQL

**Learn about NoSQL and the different types of NoSQL databases.**

## Introduction to NoSQL

In the world of databases, there are many different ways to organize and store data. At this point, we are familiar with the concept of relational databases that store data in rows, form relationships between the tables, and query the data using SQL. However, a new type of database, **NoSQL**, started to rise in popularity in the early 21st century.

NoSQL is short for “not-only SQL”, but is also commonly called “non-relational” or “non-SQL”. Any database technology that stores data differently from relational databases can be categorized as a NoSQL database. To get a good grasp on NoSQL, in this article, we will:

- Cover a brief overview of how we arrived at NoSQL technology.
- Examine some of the distinct reasons to choose or not choose a NoSQL database.
- Explore common types of NoSQL databases and how each type structures data.

Let’s dive in!

## Arriving at NoSQL

The need to store and organize data records dates back to way before the term “database” was coined. It wasn’t until around the late 1960s (although there were methods of data storage long before then) that the first implementation of a computerized database came into existence. Relational databases gained popularity in the 1970s and have remained a staple in the database world ever since. However, as datasets became exponentially larger and more complex, developers began to seek a flexible and more scalable database solution. This is where NoSQL came in. Let’s examine some of the notable reasons developers may choose a NoSQL database.

## Is NoSQL the Right Option?

When considering what database suits an application’s needs, it’s important to note that relational and non-relational (NoSQL) databases each offer distinct advantages and disadvantages. While not an exhaustive list, here are some notable benefits that a NoSQL database may provide:

- **Scalability**: NoSQL was designed with scalability as a priority. NoSQL can be an excellent choice for massive datasets that need to be distributed across multiple servers and locations.
- **Flexibility**: Unlike a relational database, NoSQL databases don't require a schema. This means that NoSQL can handle unstructured or semi-structured data in different formats.
- **Developer Experience**: NoSQL requires less organization and thus lets developers focus more on using the data than on figuring out how to store it.

While these are important benefits, NoSQL databases do have some drawbacks:

- **Data Integrity**: Relational databases are typically ACID compliant, ensuring high data integrity. NoSQL databases follow BASE principles (basic availability, soft state, and eventual consistency) and can often sacrifice integrity for increased data distribution and availability. However, some NoSQL databases do offer ACID compliance.
- **Language Standardization**: While some NoSQL databases do use the Structured Query Language (SQL), typically, each database uses its unique language to set up, manage, and query data.

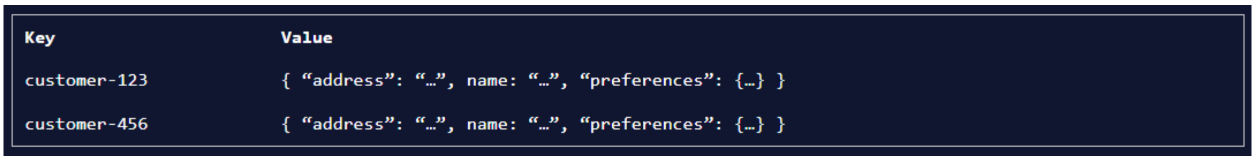
Now that we have a better idea about why we may choose or not choose a NoSQL solution, let's explore our choices for data organization by exploring a few different types of NoSQL databases.

## Types of NoSQL Databases

There are four common types of NoSQL databases that store data in slightly different ways. Each type will provide distinct advantages and disadvantages depending on the dataset. In the examples below, we will be using an e-commerce website to illustrate how each database may store the data.

### Key-Value

A key-value database consists of individual records organized via key-value pairs. In this model, keys and values can be any type of data, ranging from numbers to complex objects. However, keys must be unique. This means this type of database is best when data is attributed to a unique key, like an ID number. Ideally, the data is also simple, and we are looking to prioritize fast queries over fancy features. For example, let's say we wanted to store shopping cart information for customers who shop in an e-commerce store. Our key-value database might look like this:



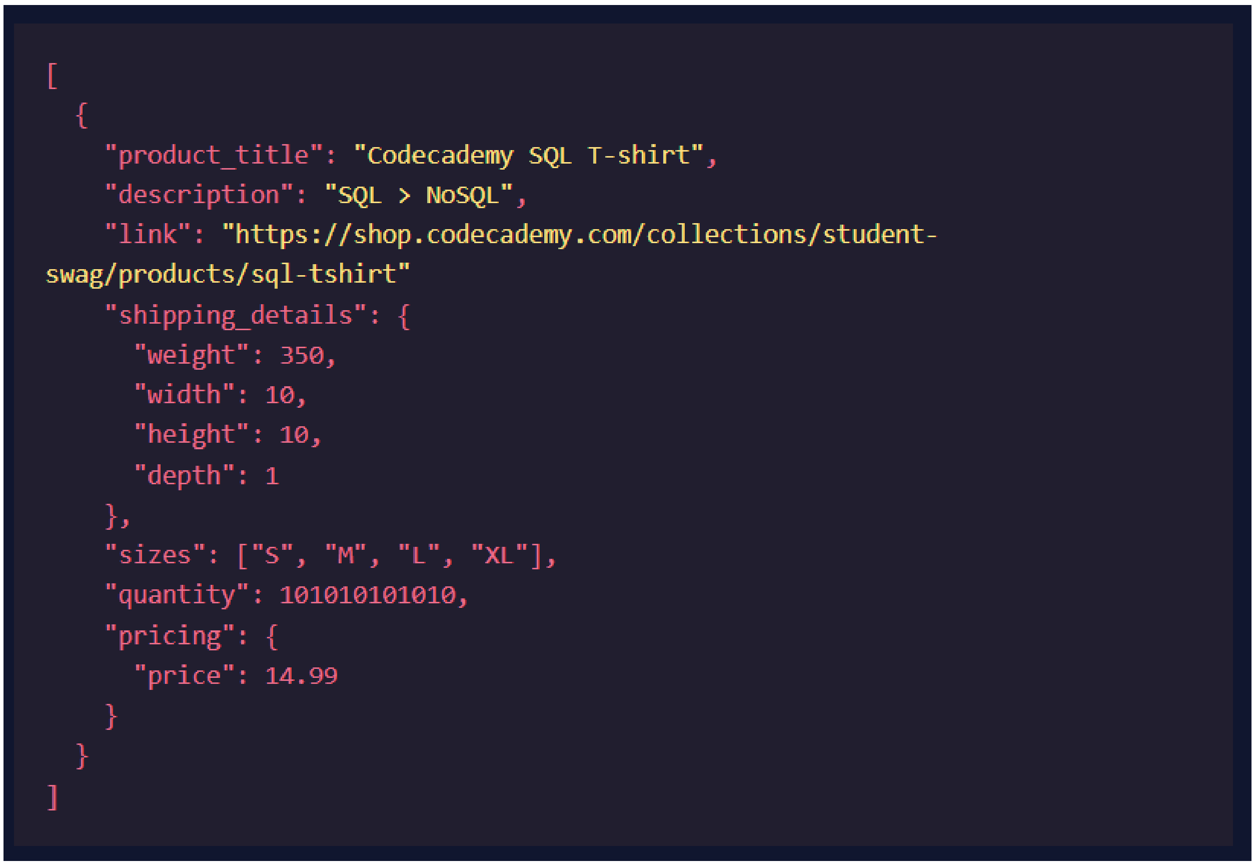
| Key | Value |
|---|---|
| customer-123 | { “address”: “…”, name: “…”, “preferences”: {…} } |
| customer-456 | { “address”: “…”, name: “…”, “preferences”: {…} } |

Amazon DynamoDB and Redis are popular options for developers looking to work with key-value databases.

## Document

A document-based (also called document-oriented) database consists of data stored in hierarchical structures. Some supported document formats include JSON, BSON, XML, and YAML. The document-based model is considered an extension of the key-value database and provides querying capabilities not solely based on unique keys. Documents are considered very flexible and can evolve to fit an application’s needs. They can even model relationships!

For example, let’s say we wanted to store product information for customers who shop in our e-commerce store. A `products` document might look like this:

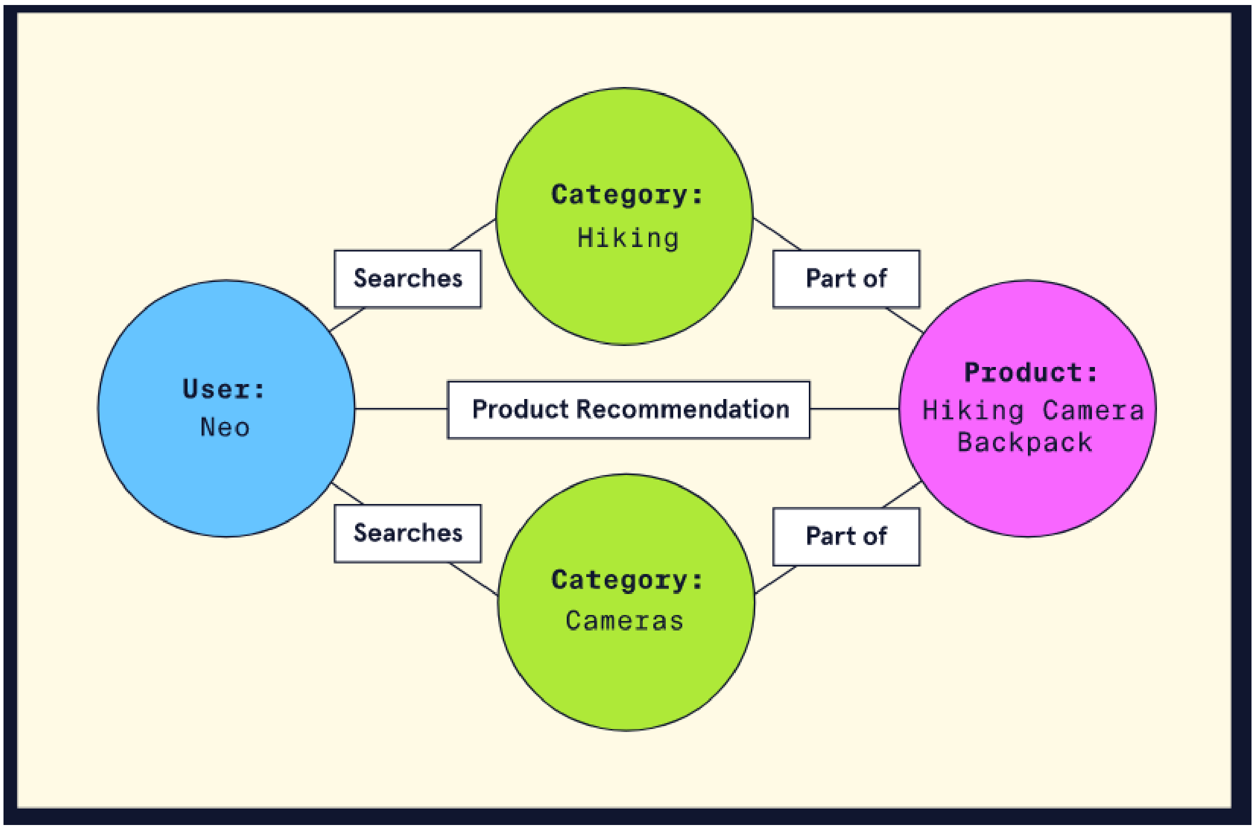


```
[
  {
    "product_title": "Codecademy SQL T-shirt",
    "description": "SQL > NoSQL",
    "link": "https://shop.codecademy.com/collections/student-
swag/products/sql-tshirt"
    "shipping_details": {
      "weight": 350,
      "width": 10,
      "height": 10,
      "depth": 1
    },
    "sizes": ["S", "M", "L", "XL"],
    "quantity": 101010101010,
    "pricing": {
      "price": 14.99
    }
  }
]
```

MongoDB is a popular option for developers looking to work with a document database.

## Graph

A graph database stores data using a graph structure. In a graph structure, data is stored in individual nodes (also called vertices) and establishes relationships via edges (also called links or lines). The advantage of the relationships built using a graph database as opposed to a relational database is that they are much simpler to set up, manage, and query. For example, let's say we wanted to build a recommendation engine for our e-commerce store. We could establish relationships between similar items our customers searched for to create recommendations.

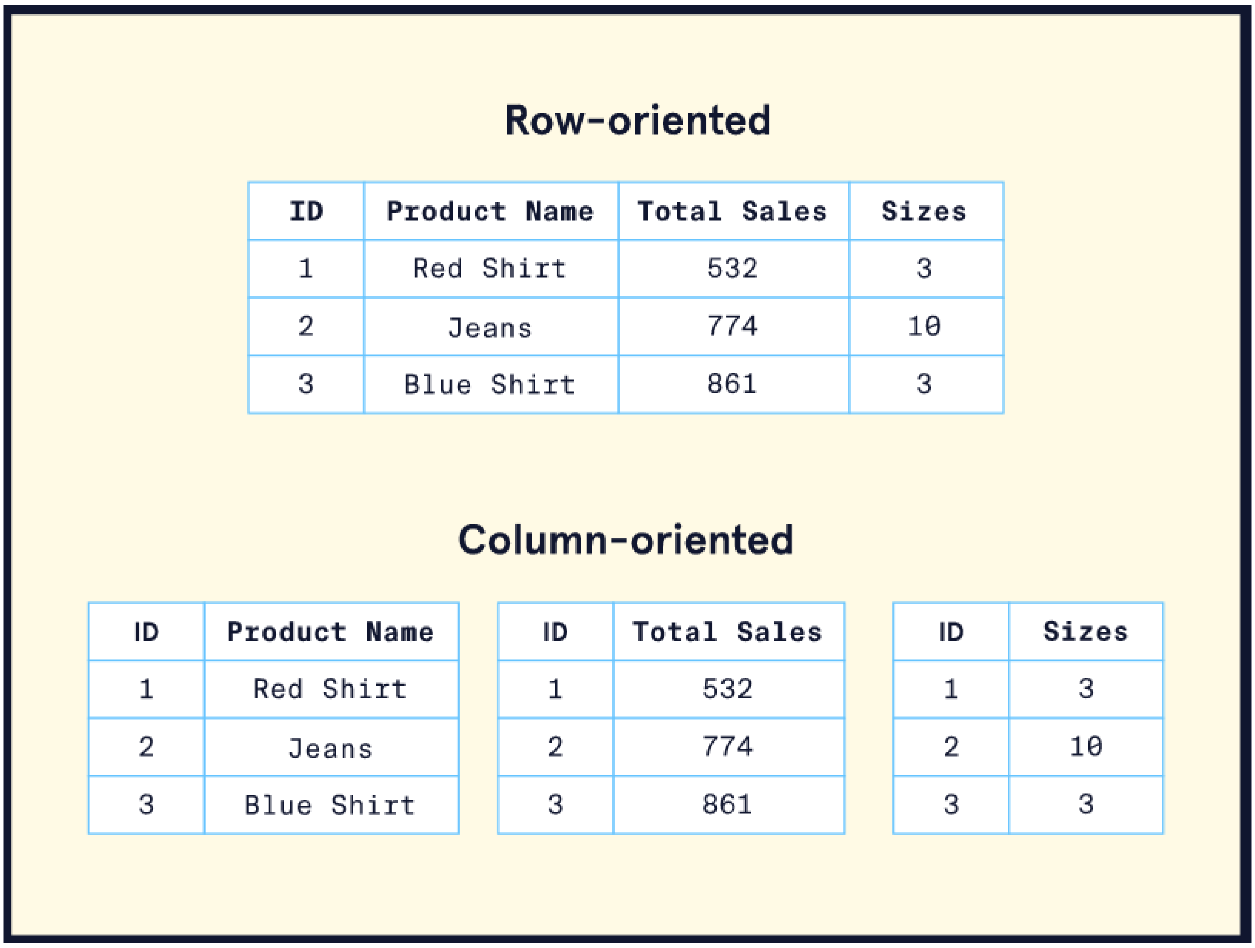


In the graph above, we can see that there are four nodes: "Neo", "Hiking", "Cameras", and "Hiking Camera Backpack". Because the user, "Neo", searched for "Hiking" and "Cameras", there are edges connecting all 3 nodes. More edges are created after the search, linking a new node, "Hiking Camera Backpack".

Neo4j is a popular option for developers looking to work with a graph database.

## Column Oriented

A column-oriented NoSQL database stores data similar to a relational database. However, instead of storing data as rows, it is stored as columns. Column-oriented databases aim to provide faster read speeds by being able to quickly aggregate data for a specific column. For example, take a look at the following e-commerce database of products:



Row-oriented

| ID | Product Name | Total Sales | Sizes |
|---|---|---|---|
| 1 | Red Shirt | 532 | 3 |
| 2 | Jeans | 774 | 10 |
| 3 | Blue Shirt | 861 | 3 |

Column-oriented

| ID | Product Name |
|---|---|
| 1 | Red Shirt |
| 2 | Jeans |
| 3 | Blue Shirt |

| ID | Total Sales |
|---|---|
| 1 | 532 |
| 2 | 774 |
| 3 | 861 |

| ID | Sizes |
|---|---|
| 1 | 3 |
| 2 | 10 |
| 3 | 3 |

If we wanted to analyze the total sales for all the products, all we would need to do is aggregate data from the sales column. This is in contrast to a relational model that would have to pull data from each row. We would also be pulling adjacent data (like size information in the above example) that isn't relevant to our query.

Amazon's Redshift is a popular option for developers looking to work with a column-oriented database.

## Wrap Up

We've now learned some of the fundamentals of NoSQL database technology. Let's take a moment to review what we've learned:

- NoSQL stands for "not-only SQL" (also called "non-relational", or "non-SQL") and refers to any database that stores data in any format other than relational tables.
- NoSQL database technology grew in popularity due to datasets growing in size and complexity.
- NoSQL databases may provide flexibility, scalability, and speed advantages.
- NoSQL databases experience disadvantages such as lack of data integrity and lack of language standardization across different NoSQL databases.
- Common types of NoSQL databases include key-value, document, graph, and column-oriented.

Next time a project requires a database, consider the tradeoffs between a relational and a NoSQL database and pick one that best suits your use case!