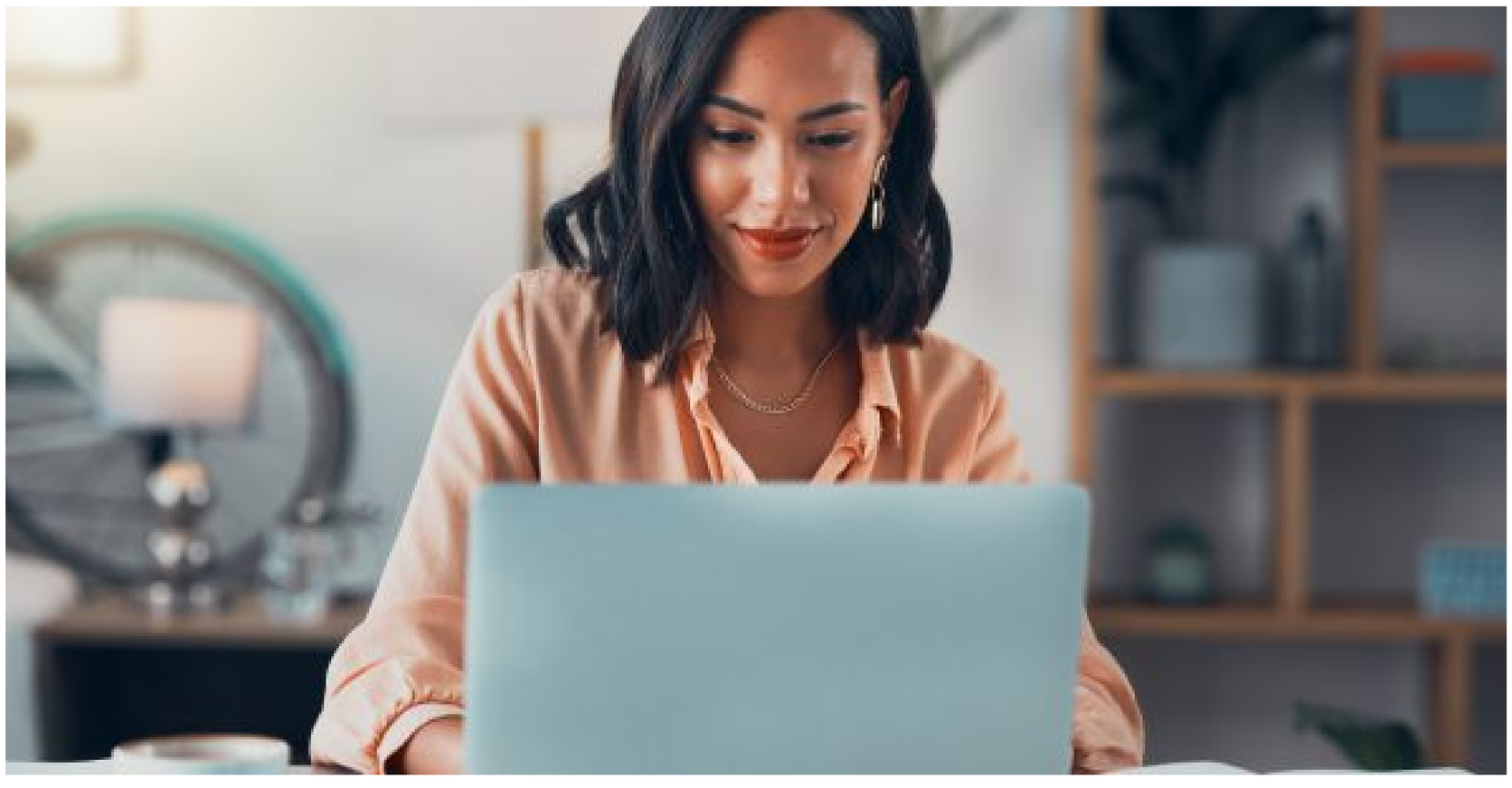
What is a NoSQL database?



What is a NoSQL database?

NoSQL, also referred to as “not only SQL” or “non-SQL”, is an approach to database design that enables the storage and querying of data outside the traditional structures found in relational databases.

While NoSQL can still store data found within relational database management systems (RDBMS), it just stores it differently compared to an RDBMS. The decision to use a relational database versus a non-relational database is largely contextual, and it varies depending on the use case.

Instead of the typical tabular structure of a relational database, NoSQL databases, house data within one data structure, such as JSON document. Since this non-relational database design does not require a schema, it offers rapid scalability to manage large and typically unstructured data sets.

NoSQL is also type of distributed database, which means that information is copied and stored on various servers, which can be remote or local. This ensures availability and reliability of data. If some of the data goes offline, the rest of the database can continue to run.

Today, companies need to manage large data volumes at high speeds with the ability to scale up quickly to run modern web applications in nearly every industry. In this era of growth within cloud, big data, and mobile and web applications, NoSQL databases provide that speed and scalability, making it a popular choice for their performance and ease of use.

NoSQL versus SQL

Structured query language (SQL) is commonly referenced in relation to NoSQL. To better understand the difference between NoSQL and SQL, it may help to understand the history of SQL, a programming language used for retrieving specific information from a database.

Before relational databases, companies used a hierarchical database system with a tree-like structure for the data tables. These early database management systems (DBMS) enabled users to organize large quantities of data. However, they were complex, often proprietary to a particular application, and limited in the ways in which they could uncover within the data. These limitations eventually led to the development of relational database management systems, which arranged data in tables. SQL provided an interface to interact with relational data, allowing analysts to connect tables by merging on common fields.

As time passed, the demands for faster and more disparate use of large data sets became increasingly more important for emerging technology, such as e-commerce applications. Programmers needed something more flexible than SQL databases (i.e. relational databases). NoSQL became that alternative.

While NoSQL provided an alternative to SQL, this advancement by no means replaced SQL databases. For example, let's say that you are managing retail orders at a company. In a relational model, individual tables would manage customer data, order data and product data separately, and they would be joined together through a unique, common key, such as a Customer ID or an Order ID. While this is great for storing and retrieving data quickly, it requires significant memory. When you want to add more memory, SQL databases can only scale vertically, not horizontally, which means your ability to add more memory is limited to the hardware you have. The result is that vertical scaling ultimately limits your company’s data storage and retrieval.

In comparison, NoSQL databases are non-relational, which eliminates the need for connecting tables.Their built-in sharding and high availability capabilities ease horizontal scaling. If a single database server is not enough to store all your data or handle all the queries, the workload can be divided across two or more servers, allowing companies to scale their data horizontally.

While each type of database has its own advantages, companies commonly utilize both NoSQL and relational databases in a single application. Today’s cloud providers can support SQL or NoSQL databases. Which database you choose depends on your goals.

For a deeper dive into the differences between the two options, see "SQL vs. NoSQL Databases: What's the Difference?"

Types of NoSQL databases

NoSQL provides other options for organizing data in many ways. By offering diverse data structures, NoSQL can be applied to data analytics, managing big data, social networks, and mobile app development.

A NoSQL database manages information using any of these primary data models:

Key-value store

This is typically considered the simplest form of NoSQL databases. This schema-less data model is organized into a dictionary of key-value pairs, where each item has a key and a value. The key could be like something similar found in a SQL database, like a shopping cart ID, while the value is an array of data, like each individual item in that user’s shopping cart. It’s commonly used for caching and storing user session information, such as shopping carts. However, it's not ideal when you need to pull multiple records at a time. Redis and Memcached are examples of an open-source key-value databases.

Document store

As suggested by the name, document databases store data as documents. They can be helpful in managing semi-structured data, and data are typically stored in JSON, XML, or BSON formats. This keeps the data together when it is used in applications, reducing the amount of translation needed to use the data. Developers also gain more flexibility since data schemas do not need to match across documents (e.g. name vs. first_name). However, this can be problematic for complex transactions, leading to data corruption. Popular use cases of document databases include content management systems and user profiles. An example of a document-oriented database is MongoDB, the database component of the MEAN stack.

Want to know more about MongoBD? Check out the IBM tutorial on getting started with using IBM Cloud Databases for MongoDB.

Wide-column store

These databases store information in columns, enabling users to access only the specific columns they need without allocating additional memory on irrelevant data. This database tries to solve for the shortcomings of key-value and document stores, but since it can be a more complex system to manage, it is not recommended for use for newer teams and projects. Apache HBase and Apache Cassandra are examples of open-source, wide-column databases. Apache HBase is built on top of Hadoop Distributed Files System that provides a way of storing sparse data sets, which is commonly used in many big data applications. Apache Cassandra, on the other hand, has been designed to manage large amounts of data across multiple servers and clustering that spans multiple data centers. It's been used for a variety of use cases, such as social networking websites and real-time data analytics.

Graph store

This type of database typically houses data from a knowledge graph. Data elements are stored as nodes, edges and properties. Any object, place, or person can be a node. An edge defines the relationship between the nodes. For example, a node could be a client, like IBM, and an agency like, Ogilvy. An edge would be categorize the relationship as a customer relationship between IBM and Ogilvy.

Graph databases are used for storing and managing a network of connections between elements within the graph. Neo4J , a graph-based database service based on Java with an open-source community edition where users can purchase licenses for online backup and high availability extensions, or pre-package licensed version with backup and extensions included.

In-memory store

With this type of database, like IBM solidDB, data resides in the main memory rather than on disk, making data access faster than with conventional, disk-based databases.

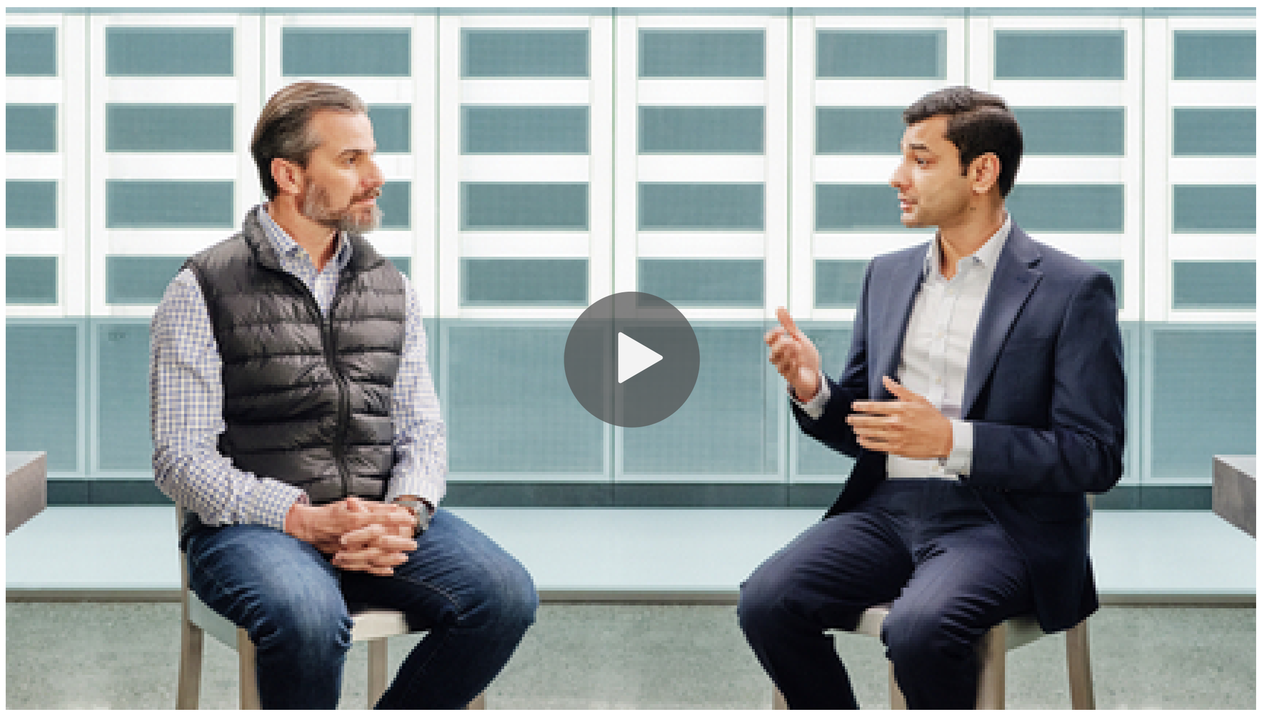
Examples of NoSQL databases

Many companies have entered the NoSQL landscape. In addition to those mentioned above, here are some popular NoSQL databases:

- Apache CouchDB, an open source, JSON document-based database that uses JavaScript as its query language.
- Elasticsearch, a document-based database that includes a full-text search engine.
- Couchbase, a key-value and document database that empowers developers to build responsive and flexible applications for cloud, mobile, and edge computing.

To learn more about the state of databases, see "A Brief Overview of the Database Landscape."

AI Academy



Is data management the secret to generative AI?

Explore why high-quality data is essential for the successful use of generative AI.

Go to episode →

Advantages of NoSQL

Each type of NoSQL database has strengths that make it better for specific use cases. However, they all share the following advantages for developers and create the framework to provide better service customers, including:

- **Cost-effectiveness:** It is expensive to maintain high-end, commercial RDBMS. They require the purchase of licenses, trained database managers, and powerful hardware to scale vertically. NoSQL databases allow you to quickly scale horizontally, better allocating resources to minimize costs.

- **Flexibility:** Horizontal scaling and a flexible data model also mean NoSQL databases can address large volumes of rapidly changing data, making them great for agile development, quick iterations, and frequent code pushes.

- **Replication:** NoSQL replication functionality copies and stores data across multiple servers. This replication provides data reliability, ensuring access during down time and protecting against data loss if servers go offline.

- **Speed:** NoSQL enables faster, more agile storage and processing for all users, from developers to sales teams to customers. Speed also makes NoSQL databases generally a better fit for modern, complex web applications, e-commerce sites, or mobile applications.

In a nutshell, NoSQL databases provide high performance, availability, and scalability.

NoSQL use cases

The structure and type of NoSQL database you choose will depend on how your organization plans to use it. Here are some specific uses for various types of NoSQL databases.

- **Managing data relationships:** Managing the complex aggregation of data and the relationships between these points is typically handled with a graph-based NoSQL database. This includes recommendation engines, knowledge graphs, fraud detection applications, and social networks, where connections are made between people using various data types.

- **Low-latency performance:** Gaming, home fitness applications, and ad technology all require high throughput for real-time data management. This infrastructure provides the greatest value to the consumer, whether that's market bidding updates or returning the most relevant ads. Web applications require in-memory NoSQL databases to provide rapid response time and manage spikes in usage without the lag that can comes with disk storage.

- **Scaling and large data volumes:** E-commerce requires the ability to manage huge spikes in usage, whether it's for a one-day sale or the holiday shopping season. Key-value databases are frequently

used in e-commerce applications because its simple structure is easily scaled up during times of heavy traffic. This agility is valuable to gaming, adtech, and Internet of Things (IoT) applications.

Microservices and NoSQL databases

The need for large companies to provide services without latency and to scale more quickly has spurred growth for microservices, which has led companies to examine what type of database to use for different applications.

Companies have found that using a single, relational database for every component of an application has its limitations, especially when better alternatives exist for specific components. Microservices are an attractive option, in part, because they eliminate the need for a single, shared data store for an entire application. Instead, the application has many, loosely coupled and independently deployable services, each with their own data model and database, and integrated via API gateways or an iPaaS.

The pattern of using multiple databases within a single application, also known as polyglot persistence, has helped to create space in the market for NoSQL databases to thrive. Today, developers can leverage the right database for the right microservice without trying to make everything work in the context of a single, relational database.