Ethan Craycroft

CS-300-ON

Dr. Johnson

March 10, 2022

# NoSQL Databases

NoSQL databases, also referred to as “not only SQL” databases, are databases that allow storage and retrieval of data in a non-relational way. Four of the most popular NoSQL database systems are document databases, graph databases, key-value stores, and wide-column stores. NoSQL databases were primarily created to support systems that target cloud applications. They can store and process huge quantities of data at scale, mainly storing them in ways other than using relational tables. NoSQL databases are unique because they are able to process, store, and retrieve data using no SQL at all, or in cooperation with SQL. NoSQL databases are extremely flexible and scalable. Relational databases have their merit, but NoSQL databases manage to cover a lot of the shortcomings of SQL databases. NoSQL databases are primarily used for applications with large amounts of data, low latency, and flexible data models. NoSQL allows duplicates of data and has no problem accommodating extra-large data sizes that result from these duplications. SQL databases scale vertically – their capacity can only be expanded by increasing memory such as RAM. NoSQL databases scale horizontally – their capacity can be expanded by simply adding more servers to hold the data.

NoSQL has exploded since the cost of storage has gone down drastically. NoSQL databases were designed to maximize productivity. Another reason for the surge of NoSQL was the massive increase in data volume and variety. With so much data flowing constantly, relational databases weren’t cutting it anymore. NoSQL databases have been around since the 1960s, but with increasingly modern technological innovations such as the internet, cloud, and mobile services, the need for them has boomed since the late 2000s. The flexibility of NoSQL databases lies within the data it can store. NoSQL databases allow developers to store a multitude of data types, including structured, semi-structured, and polymorphic data.

Three examples of NoSQL databases are Cassandra, Redis, and MangoDB. Each of these 3 NoSQL databases use different approaches. Cassandra is a NoSQL database that uses keys and columns to store data. This is a wide-column, or tabular, database. With Cassandra and other similar databases, data is organized into related rows that are stored together for quick queries. These databases use hashing to retrieve rows from the table. A key word is

provided, and the corresponding rows are accessed and retrieved. Simple queries for a value in a column return quickly in wide-column databases. The database can find the particular column given a key word, and the entire column can be searcehd very rapidly. Redis is a very popular NoSQL database that uses a hash table to store data. With hash tables, as we learned, there are two datapoints involved: a key and a value. This makes Redis a key-value store. Key-value stores are great for large databases, providing a constant access time. The method of retrieving a value that is associated with a given key is very efficient. Key-value stores are very divisible and can achieve very high levels of horizontal scaling. MangoDB is a widely known NoSQL database that uses documents to store data. With MangoDB and other document databases, data is stored in documents similar to JSON objects. These documents contain data and each one is assigned a specific key. The key is used by the database to retrieve the particular document containing the desired data. Document databases such as MangoDB are able to store a variety of data types.

NoSQL databases have both pros and cons, but at this point in time it seems the good outweighs the bad. One of the advantages of NoSQL databases is their performance speed and efficiency with simple queries. SQL databases can also return quick and efficient simple queries, but as the database grows larger the speed slows greatly whereas NoSQL databases can handle the increasing size with ease while maintaining its speed and efficiency. Another advantage of NoSQL databases is the ability to support vast quantities of concurrent users, or users all accessing the database at the same time. Another benefit to using NoSQL databases is the cost-effectiveness. While SQL databases scale up, requiring more expensive and upgraded hardware, NoSQL databases scale out. This means that the database can be expanded by adding more commodity servers.

NoSQL databases also exhibit stability and resiliency. Being dispersed across multiple servers allows for backups if one specific server were to crash. NoSQL databases provide continuous availability with very little down time. Flexibility is one of NoSQL's finest features, as it allows developers to use a variety of storing techniques to find which one works the best for their product. Another advantage that has been heavily discussed is the sheer volume of data NoSQL databases can handle. In today's society we generate about 1.145 *trillion* megabytes *per day*. The amount of data floating around the internet daily, databases need to be designed to be flexible and to be able to handle vast amounts of data. NoSQL databases accomplish both of these goals.

NoSQL databases are great and have so many practical uses. However, they do have some faults. NoSQL databases often require multiple types of databases. With its flexibility and numerous options, one option may not be

able to satisfy all of the different needs a developer has. NoSQL is made to have very specialized options which are great when they fit, but they aren't an end-all-be-all. Another potential negative of NoSQL databases are the massive size of the databases. While the database can handle these large amounts of data and includes duplicate data without care, this can lead to colossal databases that take up huge amounts of memory. NoSQL databases also typically need less ongoing maintenance. Since NoSQL databases can divide and replicate data across nodes.

NoSQL databases are also far behind SQL in terms of users. Since their popularity has been climbing only since the late 2000s, NoSQL hasn't grown as much in its lifetime as SQL has in its own lifetime. As time goes on this problem will solve itself, but with SQL being mainstream for so long it has gained more knowledge, experience, and investment. Support for NoSQL could potentially be limited with this surge. With the shift to NoSQL, it will eventually catch up to SQL as people begin investing in it and becoming more knowledgeable about it. The more people work with NoSQL, invest money, time, and knowledge into it, the better and more practical it will become. While NoSQL databases are great for simple queries, complex queries can prove to be challenging. Due to the varying data structures involved with NoSQL databases, complex queries are just less efficient because there isn't a standard interface to conduct them.

NoSQL databases also suffer from data retrieval inconsistency. With the scale-out approach, data is more readily available and can be retrieved much quicker than a database that scales up. However, with scaling out you run the risk of retrieving data that is not updated, or receiving different values based on which servers are accessed. Data loss is possible with some NoSQL systems ACID and BASE are consistency models used to evaluate databases. ACID stands for Atomic, Consistent, Isolated, and Durable. If a database uses an ACID consistency model, it means the database does the following: A) all operations succeed or every operation is rolled back, C) the database remains structurally sound after a transaction, I) transactions run sequentially, and D) results of any transactions are permanent. BASE stands for Basic Availability, Soft-state, Eventual consistency. If a database uses a BASE consistency model, it means the database does the following: BA) works most of the time, S) stores aren't required to be write-consistent, and replicas don't need to be mutually consistent at all times, and E) exhibits consistency at some point in time. NoSQL subscribes to the BASE consistency model. This means that its data will *eventually* be consistent. Because of their more flexible data model, NoSQL databases have to relax some of the ACID properties, thus making them BASE.

NoSQL databases are widely used and the preferred database for developers. NoSQL databases create a agile and creative environment for developers to work with. They also help store data in ways that more closely relate to the ways the data is being used in a program or application. NoSQL using unstructured storage is one of the biggest draws to using this particular type of database. This lends to the database being more flexible and agile. While NoSQL provides an alternate to SQL, it has not completely replaced SQL. SQL is still highly popular and used all over the world. SQL's use of relational tables is very useful for certain situations, but when the data set becomes too large, the efficiency takes a big hit.

A graph database is one of the major types of NoSQL databases. Graph databases store data in nodes, edges and properties. The nodes of a graph database store data, and the edges of a graph database represent connections between nodes. Graph databases provide a more in-depth representation of data because of the edges demonstrating the relationships between particular data points. Graph databases are unique in that they can evolve over time and as they are being used. Graph databases are crucial for many modern applications, such as social media, reservation systems, customer relationships, etc. Graph databases have no schema, like other NoSQL databases, and this makes them more flexible than SQL databases.

Graph databases were designed to handle very complex and connected data. They were created to compensate for the different limitations displayed by SQL databases by emphasizing the connectivity of nodes. They are very efficient at finding commonalities between data, as well as finding anomalies among data. Some graph databases such as Neo4J claim to uphold the ACID consistency model. Individual data nodes can be tagged with labels and can hold any number of key-value pairs. The edges, or relationships, provide directed connections between two nodes. Edges of a graph database always have direction, type, and start/end nodes. Nodes of a graph database can have infinite edges with other nodes without any loss of efficiency or performance. Graph databases allow users to execute traversal queries based on nodes' edges. Algorithms can be used to analyze the nodes of a graph database as well as their connections, the distance between them, their importance, etc. This allows people to find patterns and statuses of users. If queries and algorithms utilize the relationships and connectivity of nodes, they will return very quickly and efficiently. Graph databases have two widely popular models – property graphs and RDF graphs.

Property graphs are used to represent relationships among different data. Property graphs contain vertices containing information. The edges between vertices represent some sort of relationship. Vertices and edges alike can

both have attributes, or properties. RDF graphs, or "Resource Description Framework" graphs, are used to represent linked data, data integration, and knowledge graphs. An RDF statement is used to express a relationship between two data, and these statements have three specific components: a subject, a predicate, and an object. The subject and object are both nodes, whereas the predicate is an arc. The subject and the object of the statement represent the two things that are being related to one another. The predicate represents the relationship itself. The predicate is thought of as a verb that describes a particular relationship between a subject and an object. For example: <Ethan> <loves> <java>. In this particular example, "Ethan" would be the subject, "loves" would be the predicate, and "java" would be the object. This example shows a relationship between Ethan and java, with love being the relationship. RDF statements are often referred to as "RDF triples" due to their nature and their three components. The same subject/object can be used in multiple RDF triples. In an RDF graph, the addition of new information requires a separate node. Each and every vertex and edge is identified by a Unique Resource Identifier (URI). URIs can appear in any position of a RDF triple. URIs are used to identify resources. Nodes can also be left blank. Blank nodes can only appear in the subject or object position of an RDF triple. Blank nodes can be used to represent an object without naming them with a URI. Literals are basic values that can only appear in the object position of and RDF triple.

As with any other database, graph databases have their pros and cons. Graph databases are very useful for their flexibility and agility – the complexity and amount of data have no effect on the query process. Graph data base efficiency is not determined by the sheer amount of data it contains, but on the number of relationships. Graph databases give real time results. Graph databases help to find the relationships between things, allowing users to find commonalities, anomalies, trends, etc. The importance of relationships in today's society cannot be overstated. Graph databases can be very useful in many real life situations. From social media, to fraud protection, to network management, to routing, so many important things in our lives can be represented by a graph database. Another pro for graph databases is that data can be stored in a natural way – by denoting relationships between two entities. This way of storing information makes it easier to interpret and understand, because it is very human-like. Graph databases do have their drawbacks, though. Namely, they are very hard to scale. Graph databases are designed for a one tier architecture, and this makes it hard to grow. As with most NoSQL databases, graph databases don't quite have the maturity that SQL databases have. This makes it hard to find support when an issue occurs with a graph database.

Graph databases can store massive quantities of complex data and its speed and efficiency aren't sacrificed. The way graph databases store information is very useful for many real-life issues. One of the biggest examples being social media. Graph databases can be used to map every individual user as well as their relationships to other people. Graph databases are also a great way to recognize patterns in people's behavior. One big use for this is in fraud protection/detection. By mapping a person's spending patterns based on different things like location, price, frequency, etc. By mapping this data, a program could detect when a new situation arises that doesn't match that person's typical pattern. This is a great way to indicate potential fraud and protect credit card users from being stolen from.

NoSQL databases, including graph databases, have become widely popular in recent years. With storage being cheaper and easier to attain, some of these databases' biggest cons were negated. Storage concerns were the biggest issue for NoSQL databases. But now that they are accessible, it makes NoSQL the smart and cost effective choice in many situations. By scaling horizontally, NoSQL databases can simply add servers. The cloud is becoming more and more prevalent as the years go by. Cloud databases can include more traditional SQL databases, but they also utilize the more modern NoSQL database. To summarize, no one database is inherently better than another. The developer of a program or application knows their needs more than anyone. The numerous options and flexibility allow NoSQL databases to be a very viable option for these developers. The best choice for a database is almost always dependent on the developer and their application. The best choice for a database is one that harmonizes with the application, and stores data in a similar way that the application does. Graph databases are a type of NoSQL databases that are often the best choice for modern programs and applications. With so much emphasis on relationships, they can be a useful tool.

Sources


Editor. (2021, January 11). *NoSQL beginner guide: Pros, cons, types, and philosophy*. AltexSoft. Retrieved March 10, 2022, from https://www.altexsoft.com/blog/nosql-pros-cons/

MENEGASSO, A. N. D. R. E. E. D. U. A. R. D. O. (2018). *NOSQL*. Amazon. Retrieved March 10, 2022, from https://aws.amazon.com/nosql/

Mullins, C. S., Vaughan, J., & Beal, B. (2021, April 8). *What is nosql and how do Nosql databases work?*SearchDataManagement. Retrieved March 10, 2022, from https://www.techtarget.com/searchdatamanagement/definition/NoSQL-Not-Only-SQL

*NoSQL databases*. Couchbase. (n.d.). Retrieved March 10, 2022, from https://www.couchbase.com/resources/why-nosql

RDF 1.1 Primer. (n.d.). Retrieved March 10, 2022, from https://www.w3.org/TR/rdf11-primer/

*What is a graph database? - developer guides*. Neo4j Graph Data Platform. (n.d.). Retrieved March 10, 2022, from https://neo4j.com/developer/graph-database/

*What is a graph database?* Oracle. (n.d.). Retrieved March 10, 2022, from https://www.oracle.com/autonomous-database/what-is-graph-database/

*What is a graph database? {definition, use cases & benefits}*. Knowledge Base by phoenixNAP. (2022, January 12). Retrieved March 10, 2022, from https://phoenixnap.com/kb/graph-database

*What is nosql? non-relational databases explained*. DataStax. (n.d.). Retrieved March 10, 2022, from https://www.datastax.com/nosql

*What is nosql? NoSQL databases explained*. MongoDB. (n.d.). Retrieved March 10, 2022, from https://www.mongodb.com/nosql-explained

*Why do developers prefer nosql databases?* Oracle. (n.d.). Retrieved March 10, 2022, from https://www.oracle.com/database/nosql/what-is-nosql/