Chessica Nation

CS-300-ON

Database Management Systems

5 Sept 2022

Assignment 1

While relational database management systems are still the foundation of many applications, they are not a perfect solution for every situation. In these cases, we look to the "NoSQL" databases that provide mechanisms for storing and retrieving data. In the 2000s, the term "NoSQL" was applied to these databases, which were first created in the late 1960s, when they experienced a wave of popularity leading to their widespread implementation. The moniker was used to separate the databases from the standard relational database management systems and is used interchangeably with the term "nonrelational" ("What is NoSQL?"). Although it may seem like the term would signify "no SQL" is utilized, "NoSQL Databases" clarifies that it actually stands for "Not only SQL" to emphasize that both SQL and non-SQL databases can be utilized.

E.F. Codd published his research paper, "A Relational Model of Data for Large Shared Data Banks," in June 1970, which included a proposal of the relational model of data that would eventually be developed into the relational database. He suggested certain properties for arrays that would represent relational data, including: (1) each row represents an n-tuple of R, (2) the order of rows does not matter, (3) however, all rows must be distinct, (4) the order of columns is important, and (5) each column's significance is partially expressed by the domain name it is given (Codd 379). Leavitt explains that relational databases use sets of tables, arrays essentially, to contain data that can fit into specific, predefined categories. These tables contain data categories that are placed into columns, and within each row is an instance of the data for the columnized category. This configuration allows users to access the data without requiring them to reorganize the tables of the database (12).

Such a rigid arrangement of rows and columns works well for reasonably small amounts of structured data that can be easily placed into tables; however, relational databases may not offer the best solution when it comes to unstructured data and large amounts of it. Relational databases were created and rose to popularity long before the internet changed the landscape of data that needs to be stored. The size of data banks that Codd referred to as "large" in his research paper would be laughable today. Technologies such as the cloud, mobile technology, and Big Data were unimaginable at the time. In comparison to their predecessors, relational databases were considered to be much

more efficient and to have higher economies of scale. This meant that more users could be supported by the databases than before. Relational databases were designed to run on one server, so when capacity needed to be increased, the only way to do so was by upgrading the processors, memory, and storage of the single server ("NoSQL Databases"). This was a manageable task for a few decades, until the overwhelming increase in online activities, and therefore demands on databases, in the 2000s.

It is no question that the amount of data that needs to be stored and retrieved has grown exponentially in the past two decades. With each new technology that is developed, from social media platforms to virtual educational programs to electronic medical records, more and more users are engaging in online activities, meaning more and more data is being generated. While the increased online activities provide society with more beneficial access, they do not come without their fair share of problems for the developers responsible for creating and maintaining them.

As a result of the exponential growth of the internet and surge of web applications, relational databases struggle to scale efficiently. Businesses must purchase increasingly bigger servers in order to accommodate the additional users and data – up to millions of users and terabytes of operational data, which of course, has become an increasingly expensive requirement ("NoSQL Databases"). Beyond a certain point, however, a single server cannot fulfill the needs and relational databases will have to be distributed amongst multiple servers (Leavitt 13). This is not a quick solution, though, because relational databases do not function well when distributed because it is difficult to join their tables and distribute their functionality because they are not designed for data partitioning (Leavitt 13). Relational databases also face periods of downtime if the database has to be taken offline in order to perform upgrades to the hardware or if the server fails and the database becomes unavailable ("NoSQL Databases").

In relational databases, the structure of the data is predefined by the arrangement of the tables and the fixed names and types of the columns (Leavitt 12). Due to their fixed model structure, relational databases do not support agile development very well ("NoSQL Databases"). When the requirements of an application change, the data model also changes, but relational databases struggle with this because their data model is fixed and it is defined by a static schema. If and when changes do arise, developers either have to modify the schema themselves or request a "schema change" from database administrators ("NoSQL Databases"). This process is not a simple one – rather, it must be done manually and is time-consuming. Development must then slow down or halt while the modifications are made, which ultimately impacts other applications, as well ("NoSQL Databases").

Also, relational databases are unable to handle data as it is presented. They must disassemble, or "shred," and reassemble objects in order to read and write data. The workaround for this, which is inefficient and sometimes problematic, is to transform data through object-relational mapping frameworks ("NoSQL Databases").

NoSQL databases began to emerge in response to these problems with relational databases. They were engineered to meet the needs of the new era in which we are living where almost everything we do is online. The applications of this era need to be able to support large numbers of concurrent users, be available 24/7 with no downtime, and handle semi- and unstructured data ("NoSQL Databases"). These requirements are a struggle for relational databases; therefore, NoSQL databases are rising in popularity. They can accommodate unprecedented levels of scale and store new types of data, making it readily available to search, consolidate, and analyze ("NoSQL Database – What is NoSQL?"). NoSQL databases are built to be flexible, scalable, and able to respond quickly to the ever-increasing data management demands of today's businesses ("NoSQL Databases").

There are three popular categories of NoSQL databases (aside from graph databases, which will be discussed later) that are optimized for performance and scale ("What is NoSQL?"). The first category is document databases and MongoDB is one such product. Document databases, as the name implies, are generally built for storing information as documents. These documents can include JSON documents or others, such as XML documents ("NoSQL Databases"). Entire documents are organized into groups called collections, which allow queries on any attribute within a document ("NoSQL Database – What is NoSQL?"). Rather than storing and organizing data into rigid tables with fixed fields, as relational databases do, document databases allow users to customize the data by adding any number of fields to a document in a collection (Leavitt 13). Document databases are able to evolve along with the needs of the applications due to their "flexible, semistructured, and hierarchical nature" and the model works well with content management systems where each document is unique and changes as time progresses ("What is NoSQL?").

The second category of NoSQL databases is key-value stores and Amazon DynamoDB is an example of this category. Essentially, a key-value store is a large hash table where a unique key, a simple string, is stored along with pointers to the corresponding data values ("What is a Key-Value Database?"). Many different data types can be stored as values in key-value databases, including integers, strings, JSON, arrays, and more ("What is a Key-Value Database?"). Key-value stores incorporate structure in the method of tables in order to maintain that advantage of relational databases, but they still preserve the benefits of a NoSQL database ("NoSQL Databases"). Key-value stores

support simple query, adding, and removing operations, but they do not have a query language. Due to this, key-value stores are better suited (and highly optimized) for applications that only need to perform simple queries using a key, but are not well suited for applications that need to search data across multiple tables (Tejada).

The third category of NoSQL databases is wide-column databases and an example product that fits into this category is Cassandra. Conceptually, wide-column databases can appear to be very similar to relational databases: they organize data into columns and rows (Tejada). The difference, however, is their increased flexibility. Wide-column databases divide their columns into "families" and each column family holds a set of columns that are related and typically accessed or modified as a unit. New columns can be added dynamically to column families and rows can be "sparse," meaning values are not required in every column of the row (Tejada). Like key-value stores, the organization of data in wide-column databases retains the structure of relational databases, but also provides a great amount of flexibility for applications that need to store data that might require a variety of number of rows from entry to entry ("NoSQL Databases").

NoSQL databases have many advantages over relational databases. One of the biggest benefits of NoSQL databases is their high scalability. The distinguishing factor is that NoSQL databases can handle large volumes of data in ways that relational databases cannot because NoSQL databases are not limited only to rows and tables ("NoSQL Database – What is NoSQL?"). As discussed previously, relational databases are not able to easily scale out because they typically scale up, or add more resources to the existing servers. NoSQL databases are specifically designed to scale out, meaning they are designed to be able to function well using distributed clusters of hardware ("What is NoSQL?"). This ability to scale out fosters efficiency in businesses because they are able to have the appropriate number of servers and hardware that are currently required; if more are needed, they can be easily added. This prevents businesses from deploying more servers than are needed, meaning some sit unused – which is an added expense – and it allows them to scale on demand easily with no downtime ("NoSQL Databases").

Another advantage of NoSQL databases is the increased flexibility they provide. While relational databases are limited to structured data, NoSQL databases can handle semi- and unstructured data. Their flexible schemas enable faster iterative developments ("What is NoSQL?"). This gives developers more freedom to change the applications to adapt to new and changing data requirements. NoSQL databases can also provide more speed and agility compared to relational databases since they do not have a fixed data model and instead allow the developers of the specific application to decide how the data should be modeled ("NoSQL Databases").

NoSQL databases do also have their fair share of disadvantages, however. One of the main challenges of NoSQL databases is that they often trade reliability and consistency for more flexible data models that lead to increased scalability ("What is NoSQL?"). NoSQL databases do not natively support ACID, which stands for Atomicity, Consistency, Isolation, and Durability (Leavitt 13). ACID principles are a set of database transaction properties that ensure that a database remains consistent if a transaction were to fail while processing (Davardoost 1). Relational databases strictly provide ACID properties, which may make them a better fit for applications that need high levels of consistency and reliability, such as those used in the accounting, finance, and banking fields (Tejada).

Additionally, like any new technology, there is the financial and time cost of updating to NoSQL databases. For legacy systems that were built specifically for a relational structure, it may not be worth a business' effort to convert to NoSQL databases. NoSQL's ability to handle extremely large amounts of data can be a double-edged sword. With additional data comes larger databases and the need for additional storage. However, many may consider this a minor downside because of the increased availability and affordability of storage these days (Schaefer).

A fourth category of NoSQL databases is the graph database. Graph databases are built specifically to store and navigate relationships and a lot of their value comes from these relationships ("What is a Graph Database?"). Graph databases manage information as nodes, which represent entities, and edges, which specify the relationships between the entities (Tejada). Rather than a table, like relational databases and a few other NoSQL databases, diagrams of graph databases resemble a flow chart with the various entities and arrows showing their edges, or relationships.

As discussed previously, relational databases have rigid schemas that make it difficult to add connections or adapt to changing requirements. Because of this structure, it is difficult to glean any information about the relationships between the data, which in today's world of Big Data, is invaluable information to have. For applications such as product recommendation engines or fraud detection solutions, insight about data relationships is integral. In order to find this information with relational databases, multiple database tables must be joined, which is a tedious and cost-intensive process ("Why Graph Databases?").

One example of a graph database product is Amazon Neptune, which is part of Amazon Web Services (AWS). Neptune is touted by Amazon as a "high-performance graph database engine optimized for storing billions

of relationships and querying the graph with milliseconds of latency." The product works by allowing users to load bulk amounts of data into Amazon Neptune where the data and relationships are then stored and available for query. Amazon Neptune supports two popular graph models, property graphs and W3C's Resource Description Framework (RDF). Their respective query languages, Apache TinkerPop Gremlin and SPARQL, can be used to execute queries that navigate highly-connected datasets in an efficient manner ("What is a Graph Database?"). This feature sets Amazon Neptune apart from its competitors as other graph database platforms require users to choose either property graphs or RDFs (Baer).

Another example of a graph database product on the market is the Neo4j Graph Database. This graph database handles both transactional and analytics workloads and performance across billions of nodes and trillions of relationships within milliseconds. Neo4j claims that its graph database stands apart from other similar products because it connects data as it is stored, instead of being computed at query time. Neo4j has its own query language, Cypher, that can be used with its graph database. Since the data connections have already been stored, Cypher can traverse groundbreaking deep connections at speeds previously unimaginable ("The Neo4j Graph Data Platform").

Graph databases have several advantages over other types of databases. The biggest benefit, as has already been mentioned, is their ability to harness the power of data connections and relationships. This advantage of graph databases is critical to social media, fraud detection, and recommendation engine applications. Without graph databases, businesses would have to spend a lot of money and effort on joining multiple tables of data in order to access these relationships. Additionally, graph databases are able to perform consistently, even with intensive and increasing amounts of data that must be handled. Relational databases are heavily limited when it comes to large amounts of data. Graph databases are also extremely flexible compared to other databases since developers can easily add to the existing graph structure, rather than having to model the domain ahead of time ("Why Graph Databases?").

As beneficial as graph databases may seem, they are not a silver bullet in the world of databases. If the data in question is not highly-connected and/or developers are not interested in the relationships between the data, then graph databases are not going to be the best fit since their main focus is specifically data connections ("Why Graph Databases?"). While excellent at analyzing large quantities of data and their web of relationships, graph databases are not ideal for everyday retrieval of data, such as range queries (Schaefer). Another drawback of graph databases

is a lack of a standardized query language (Dancuk). As previously mentioned, the query language used will depend upon the platform. Between Amazon Neptune and Neo4j, there are three query language options.

The theme of these databases, both relational and NoSQL, is that there are many different options, each with its own sets of advantages and disadvantages. There is no end-all be-all, best form of database to choose that will meet every need of every business. While it may seem overwhelming to have so many choices, businesses having the ability to utilize the database that works best for them leads to improved performance, instead of being stuck with a generalized database that does not meet their needs. Developers can consider their project and its specific needs and weigh them against the advantages and disadvantages of the multitude of databases available in order to select the best fit.

Works Cited

Baer, Tony. “Looking under the hood at Amazon Neptune.” *ZDNET*, 5 Apr. 2018. https://www.zdnet.com/article/looking-under-the-hood-at-amazon-neptune/. Accessed 2 Sept. 2022.

Codd, E.F. “A Relational Model of Data for Large Shared Data Banks.” *Communications of the ACM*, vol. 13, no. 6, 1970, 377-387, https://dl.acm.org/doi/pdf/10.1145/362384.362685. Accessed 2 Sept. 2022.

Dancuk, Milica. “What is a Graph Database?” *PhoenixNAP*, 22 Apr. 2021. https://phoenixnap.com/kb/graph-database#ftoc-heading-12. Accessed 2 Sept. 2022.

Davardoost, Farnaz, et al. “An Innovative Model for Extracting OLAP Cubes from NOSQL Database Based on Scalable Naïve Bayes Classifier.” Mathematical Problems in Engineering, Apr. 2022, pp. 1–11. EBSCOhost, https://doi-org.libproxy.bellarmine.edu/10.1155/2022/2860735.

Leavitt, Neal. “Will NoSQL Databases Live Up to Their Promise?” *Computer*, vol. 43, no. 2, 2010, pp. 12-14, http://www.leavcom.com/pdf/NoSQL.pdf. Accessed 2 Sept. 2022.

“NoSQL Database – What is NoSQL?” *Microsoft Azure*. azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-nosql-database/. Accessed 2 Sept. 2022.

“NoSQL Databases.” *Couchbase*. www.couchbase.com/resources/why-nosql. Accessed 2 Sept. 2022.

Schaefer, Lauren. “NoSQL vs. SQL Databases.” *MongoDB*. https://www.mongodb.com/nosql-explained/nosql-vs-sql. Accessed 2 Sept. 2022.

Tejada, Zoiner. “Non-relational data and NoSQL.” *Microsoft Azure*. docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data. Accessed 2 Sept. 2022.

“The Neo4j Graph Data Platform.” *Neo4j*. https://neo4j.com/product/. Accessed 2 Sept. 2022.

“What is a Graph Database?” *Amazon AWS*. https://aws.amazon.com/nosql/graph/. Accessed 2 Sept. 2022.

“What is a Key-Value Database?” *Redis*. https://redis.com/nosql/key-value-databases/. Accessed 2 Sept. 2022.

“What is NoSQL?” *Amazon AWS*. aws.amazon.com/nosql/. Accessed 2 Sept. 2022.

“Why Graph Databases?” *Neo4j*. https://neo4j.com/why-graph-databases/. Accessed 2 Sept. 2022.