Limitations to SQL and Current Solutions

Eland Anthony

The SQL language and a relational database management system are reliant upon each other to provide comprehensive data management, and many organizations utilize this structure to store and access important information. When considering database utilization throughout its lifetime, it can be seen that relational database implementations have been available for a considerable amount of time. One such example is the credit card and banking services industry. These resources have persisted for decades and require proper data management of large datasets. Relations must be established between the card holder via a unique identifier, its associated account(s), and personal information such as a billing address among other identifiers. In order to establish and utilize such relations, relational database management systems were required as well as SQL, the querying standard that is inherent within the relational database model to enable proper management of data. It may be assumed, however, that this technology is now outdated and has been phased out for more modern database models and standards. This assumption is derived from the well known fact that programming implementations are ever changing and continually progressing. Surely a new system has been developed since the relational database and SQL standard were implemented decades back. In order to address such an inclination, one must understand the evolution of database implementations throughout history. In this paper, limitations and solutions apparent within the relational database management system and the SQL standard comparative to its successors, namely NewSQL, will be discussed utilizing scholarly sources for each topic.

Now that we are familiar with a relational database management system and its use cases in finance we are able to gain a better understanding of the advantages and disadvantages of such an implementation. In order to understand the advantages of the relational database, it is helpful to first look at its predecessor. That is, the first database implementations. Before computers came to be, databases were thought of a collection of written records. For example, libraries utilized a database of written records to keep track of book rentals and other relevant data. Of course, this implementation was inefficient as records had to be physically queuried. Along with the invention of the mainframe computer, database management systems were quickly implemented thereafter and initially consisted of two types: “By the early 1970s, two major models of DBMS were competing for dominance. The network model was formalized by the CODASYL standard and implemented in databases such as IDMS, while the hierarchical model provided a somewhat simpler approach as was most notably found in IBM’s IMS (Information

Management System)" [1]. However, both structures shared similar drawbacks. Firstly, queries were dependent upon the CODASYL standard. This standard required anticipated queries to be implemented during the initial design phase and could not be added once the database management system was implemented. In addition, queries required complex programming and thus the bar of entry was high to use such systems [1]. The relational model and SQL standard brought about solutions to such issues associated with the CODASYL standard and its two models. We will now look into the catalyst, implementation, and effects associated with relational database management systems and SQL.

The relational model of database management systems was formally introduced in 1970 as IBM employee Edgar Codd published "A Relational Model of Data for Large Shared Data Banks" [2]. The concepts described in Codd's paper led to IBM's initial database management system R, which pioneered the SQL language. This new model allowed for concepts such as constraints, tuples, functions, and relationships to be used in the database, as well as the user friendly SQL language. [2]. The simplified syntax apparent in SQL allowed for less complex querying allowing for increased utilization. Thus, the CODASYL standard was mostly phased out for the new relational database model and the SQL standard. The relational model and SQL standard dominated for decades, leading many to believe no further evolution would occur. That is, until 2007, thirty seven years after Codd's relational model. Michael Stonebraker led a research team which published "The End of an Architectural Era (It's Time for a Complete Rewrite)" [1]. Within this paper, Stonebraker outlined the issues inherent in the relational database model. He and his team realized that hardware assumptions within relational architecture were no longer relevant due to improvements in memory costs and specifications, suggesting a single architecture may not be optimal across all workloads [4]. This apparent disadvantage of the relational model led to the development of NewSQL, which retained characteristics from the relational model but disregarded a common architecture. [1] To better understand the adoption of NewSQL and its advantages, an implementation of a NewSQL database will be examined.

NewSQL contains the unique characteristic of an absence of reliance upon a common architecture, yet inherits many aspects of the relational database as well as the SQL standard. The improvement sought from this model involves the underlying physical storage of data to increase scalability and throughput. Using on board memory, NewSQL enables the capability to,

"send the query to the data rather than bring the data to the query, which results in significantly less network traffic since transmitting the queries is typically less network traffic than having to transmit data" [3]. Via this method, NewSQL models aim to increase transaction speed and scalability whilst maintaining features innate in the relational model, such as atomicity, consistency, isolation, and durability, also known as the ACID properties of database management systems [1]. One such example of this model includes the NewSQL implementation of VoltDB. In line with the advantages of NewSQL aforementioned, VoltDB was motivated by the plummeting cost of main memory. In a paper published by Mark Stonbraker, the aforementioned pioneer of NewSQL and contributor to VoltDB, the advantages of scalability are described: "A five node dual quad-core SGI system (total of 40 cores) performs 640,000 Voter transactions/second. This is about one SQL command per core every 10 microseconds. Adding nodes to the configuration resulted in linear scalability. In all a 30 node system executed 3.4 million transactions/sec, and linear scalability was observed with a slope of 0.88" [4]. Such results illustrate the linear scalability that may be achieved by VoltDB, as well as the incredible transaction speeds that may be accomplished by this NewSQL implementation. Further, the paper describes various additional features inherent within VoltDB, including multi-threaded operation and deterministic operations to reduce concurrency [4]. In addition, deterministic operations allow for high availability through replication, as transactions will either fail or succeed without need for additional synchronization or a protocol to ensure such synchronization [4]. Such features enable the retention of the atomicity, consistency and isolation evident in the ACID properties present in relational database management systems [1]. Finally, to prevent data loss from power failure, VoltDB asynchronously saves a memory image to disk storage at certain intervals determined by the operator [4]. This feature enables the durability property within ACID [1]. These notable improvements and advantages of the NewSQL database VoltDB, however, depend on whether the on board memory is sufficient to handle the size and throughput of said database as VoltDB relies on memory availability for performance.

In conclusion, database management systems have evolved slowly since their inception. The most notable transitions discussed in this paper became apparent as a result of published research from innovators such as Codd and Stonebraker leading to implementations such as the relational model and NewSQL respectively. NewSQL improved upon issues inherent in the relational model, such as improving throughput and scalability, although it brought about the

issue of reliance on memory availability. As to whether NewSQL will ultimately prevail as a database management system, however, I am doubtful. Although the relational database management system and SQL standard remained a favorite for decades and NewSQL is generally noted as an improvement upon it, I do not believe that NewSQL will follow as the future of databases for the foreseeable future. Due to the requirement of sufficient memory for the NewSQL data and its limited performance enhancements from its predecessor, I am confident that other solutions will soon prevail. Enterprises now collect massive amounts of data and therefore are not in a position to depend on NewSQL alone as it is not limitlessly scalable due to its inherent memory constraints. True scalability cannot be achieved solely by SQL or MySQL as their solutions are not foolproof. Ultimately, I believe that NoSQL web-based solutions will prevail due to their ability to handle frequent web-based requests from users increasing overall scalability utilizing a network of servers. One such example is the Dynamo service from Amazon Web Services, among several other NoSQL services [1]. I believe and suggest that further development of such NoSQL platforms will solve many of the issues apparent in legacy solutions.

References:

[1] G. Harrison, *Next generation databases: NoSQL, NewSQL, and Big Data*. New York, NY: Apress, 2015.

[2] S. Sumathi and S. Esakkirajan, *Fundamentals of relational database management systems*. Berlin, NY: Springer, 2010.

[3] A. Pavlo and M. Aslett, "What's Really New with NewSQL?," *ACM SIGMOD Record*, vol. 45, no. 2, pp. 45–55, 2016.

[4] M. Stonebraker and A. Weisberg, "The VoltDB Main Memory DBMS," *IEEE Data Eng. Bull.*, vol. 36 no. 2, pp. 21-27, 2013.