# CS992 Database Development Coursework Assignment

Lewis Britton

Academic Year 2021/2022

# Table of Contents

# 1 Background

## 1.1 The Relational Golden Era

The global database life-cycle has historically behaved much like the mergers and acquisitions market. Much like the latter, data management reached its peak in-line with the golden era of technology. Seeing the release of historical idols such as the Mk. I IBM Model M, the 5170 terminal, Alchemy Live, and the 365 GTB4 taking to Miami's streets, 1984 also saw data management evolve from basic hierarchical systems in the 60's, through IBM's relational development through the 70's, to the standardization of Structured Query Language (SQL) in the mid-80's (Quickbase, 2022). SQL was produced from interpretation of, internal IBM researcher, Edgar Codd's relational concepts leading to Structured English Query Language (SEQUEL), which was commercially developed into SQL. This process led to IBM's adoption of the famous Db2 Relational Database Management System (RDBMS) throughout the use of their subsequent systems from the 5170. Although this is and was proprietary software, the conditions if it's use and Unix-based implementation delivered a high pragmatic standard.

## 1.2 Non-Relational 'Advancements'

As we learned in 1989 however, great things don't last forever. The 80's for relational databases were much like the three-year reign of HTML before 1996's JavaScript invasion; where more bloated tools such as Visual Basic, Oracle Developer and Microsoft Excel and Access where introduced to the market. These of course 'required' what became known as Object Database Management Systems (ODBMS) to allow 'normie' home system users to interact with and understand basic data structures and interaction. A positive development in this regard however, is that there were acceptable open-source solutions such as MySQL and Apache which remain true to roots, to some degree.

At this stage, a gap for non-relational data management was found by Carlo Strozzi just before the turn-of-the-century. And thus, we saw the development of Not Only SQL (NoSQL) systems and their implementation in products developed by IBM, Microsoft and Oracle for growing interest in commercial and personal data transacting. Not only did this culture change see growth of non-relational systems, it even saw a shift in the big relational providers towards Graphical User Interfaces (GUIs).

## 1.3 Distinguishing & Addressing Relational Limitations

On a basic level, the terms 'SQL' and 'NoSQL' are generally used when respectively referring to relational and non-relational databases, based on their relevant query languages. Relational systems use tried and tested table-data structure which is based on reference to various arrays with each respective value being indexed uniquely. This of course represents the database *schema*, i.e. the tabular data structure of a database, based on *integrity constraint* formulae which associate the data. Non-relational systems do not rely on this 'row-and-column' tabular schema and follow a document structure, graph structure, involve key-value pairs, or wide-column stores (MongoDB, 2022).

It's through this wide scope which arguably make non-relational databases far more versatile in situations where they can interact with various data formats simultaneously. For example, each unique member of a database may have their own document containing all of their attributes which often vary in format. XML-document-based systems are a good example of this. Graph-based systems work in a related manner, often much like a tree structure, with *nodes*, *edges* and *properties*, based on graph theory. Unlike in tables, relationships are direct and prioritized, meaning querying is fast. Nodes act like a row would in a relational system in that they represent an entity. The edges are 'lines' which represent their relationships to other nodes. Properties account for the purely objectively relevant attributes of the nodes. This often makes use of Resource Description Framework (RDF) graphs where instead of adding properties, new nodes are added to represent properties; known as subject nodes, property subject nodes, joined by arcs.

Both document and graph structure make use of key-value pairs which is an extremely versatile method of data representation. It is known across many languages and platforms to increase extensibility and is majorly seen in modern object-oriented programming. For example, in HTML, JavaScript and XML element attributes; and, in more simple cases such as when constructing your references in Donald E. Knuth's BibTeX. In this context, arrays contain collections of records which contain fields which contain data. These are accessed via a unique identifier key with access to records. This method makes good use of non-tabular dictionary-structured data in that it recognises data collections uniquely, allowing field data to vary freely. This therefore, means less data is required on a simultaneously and records can be referenced far quicker, in comparison to traditional relational tabular systems. In an extension of this structure, wide-column stores bring a traditional tabular system into a non-relational context. They are based on tables, columns and rows however, unlike relational systems, allow variance in column data structure, across rows. This is known as a 'two-dimensional' key-value-store (MongoDB, 2022), where keys are used to access rows, otherwise referred to as 'families' in this context which contain an array of varying columns or 'super columns' $(c)$ $\forall c\{1, ..., N\}$. This acts as a suitable segue to suitability for large-scale data management.

## 2 Relational Dominance & Importance

### 2.1 Structure, Application & Querying

Relational database structures were traditionally respected for their ability to allow humans to perform, arguably revolutionary, advanced multi-layered queries and mathematical operations upon data (Jatanal, et al., 2012). This approach to data management introduced the idea of turning data into 'information', in the process of joining and querying tables to deliver meaningful output which can then by used and applied by human interpretation. SQL's mathematical abilities also allows the semantics of various data types to be manipulated by human interpreters, delivering more useful information. For example, SQL's group, combine, order, sum, count, etc. functions. SQL's integrated syntax for creating and interacting with schema elements is also extremely applicable to a wide scope of scenarios (Fu, 2002). SQL's Data Definition Language (DDL) is regarded one of the most accurate tools in executing commands upon very specific data locations, as it operates on the row-column basis and its syntax can be easily interpreted by humans. It's logical. Operations can also be executed on active databases during querying.

In data management, data 'redundancy' refers to instances in which data appears unnecessarily. Relational systems require all data to be 'normalized' upon creation, meaning that there should be a minimal number of tables with a minimal number of columns and minimal field population used to represent data. Hence, fields become more specific, making data more versatile on a large-scale and open to the formation of many relationships between tables. It goes without saying that adherence to this protocol opens more opportunity for less complex extensibility in the future.

## 2.2 Relevance

Data management systems were originally intended to account a great volume of civilian data under the Social Security Act of 1935, which made IBM's relational schema extremely applicable in the 70's/80's (NASEM, 1999). This of course, made the extensibility of fully normalized tables very useful in the ever-growing scale of records and scope of relationships. The general needs associated with these practices extended into the use of relational databases in engineering manufacturing industries, in the 80's/90's. SQL syntax is high level and intuitive (Codd, 1970), meaning that alongside the increasing commercial demand for large scale data storage and fast access, Codd's relational model was becoming more relevant and more importantly, easy and convenient. Aiding this was also the fact that there was less reliance on internal corporate hardware and storage.

# 3 New Era of Data

## 3.1 Need for Change

Sadly, 1989 has turned into 2022. The IBM Model F AT was replaced by the normie membrane post-'87 Model M, Tim Truman has played us out of the golden era with *Freefall*, the Testarossa has escaped into the sunset, and the pace of and demand for digital data transactions have significantly increased. We've come across the terms 'volume', 'variety' and 'velocity' which perfectly summarise the requirements for recent developments in data management. The volume of data requested by transactions in recent times is what initially drove the development of methods less restrictive than relational schema and since, users have optimised the space in these methods to increase the variety and flexibility offered by interaction with their systems.

Although the optimal manner in which users interact with relational systems is intended to reduce data redundancy, Maowa, et al. (2017) make the observation that non-relational formats void the need for null values completely as they often work on a purely vertical basis. For example, when making particular attributes only relevant to specific items. They also discuss the vertical scalability of non-relational models, in that an unlimited $i$ fields can be added, varying dependently across $N$ items. Thus, meaning data can be more relevant and varied on an individual basis. This is especially relevant in increasing data volumes as generally, $\uparrow$ volume $i \in \{1, ..., I\}$, $\uparrow$ variety$_i$ $n \forall \{1, ..., N\}$. There is also the argument that relational systems such as MySQL and Oracle simply have a field limit however, non-relational systems face no such restrictions; again accounting for scale and scope. Non-relational systems are also more dynamic and do not rely on a thousand pieces being in the correct place, i.e. in table dependencies through keys which prolong processes such as updates, insertions, deletions and searches. So forth, these changes implemented to aid volume and variety (scale and scope), in effect allow greater data access accuracy and therefore, velocity. Hence, a more agile and responsive answer to modern data requests is formed.

## 3.2 Hybrid Systems

Golosova, et al. (2015) discuss the relevance of hybrid relational/non-relational (RDBMS/NoSQL) systems in the context of Big Data metadata production, for statistical analysis. They explore how these hybrid systems optimally manage metadata scalability.

In this scenario, elements of High Energy Physics (HEP) experimentation use data systems which are designed to ensure transparency and integrity across their field. The example of PanDA generates metadata based on aspects of hundreds-of-millions of experiment submissions which must be globally transparent to the relevant bodies. The central PanDA database is faced with approximately two-million experiments per day, as of Golosova, et al.'s (2015) paper. This operation calls for a dynamic system which can cope with the claimed 160PB of data across 120 locations.

### 3.2.1 Development

In this case, metadata is produced upon active tasks and not after they are complete. As data is retained post-task completion, its assigned to its own set of tables when associated with an 'inactive' task. In a purely relational setting, data would simply be moved from one ('active') location to another ('archive') once its parent task is 'complete'. This volumetric demand from historic read-only data puts great pressure on the capabilities of PanDA's analytical tools. The solution to this issue is to implement a non-relational (NoSQL) system to manage the archived data of historic tasks, which better-accounts for volumetric scalability. As this follows the Basic Availability, Soft-State, Eventual Consistency (BASE) concept; as opposed to the Atomicity, Consistency, Isolation, Durability (ACID), the consistency-reliance of the active data determines that this is not a suitable comprehensive solution. This approach poses some system-logical issues however, significantly increases scalability and reduces hardware costs.

Subsequently, the Hybrid Metadata Storage (HMS) system was introduced, consisting of an element which transitions active data to archived ('HMS/sync') and an element which introduces the required NoSQL logic ('HMS/access') (Golosova, et al., 2015). Development of this structure involved use of wide-column store Apache Cassandra, with useful multi-center spanning. The system uses a series of primary keys (including a mandatory partition key), and cluster keys which form a partition index that can be used to reference records within the partition. Cassandra achieves high-speed querying due to its strict requirement for the correct hierarchy and order of partition and clustering keys.

### 3.2.2 Implementation

Golosova, et al.'s (2015) approach involves the transition from the PanDA system to a Django-nonrel one as the appropriate support for NoSQL and Cassandra plug-ins are offered. So forth, a primary column family in Cassandra exists to receive active tasks, duplicated from queued SQL tables of tasks. This is further enhanced by particular data aggregations where simulated data is generated and implemented into the NoSQL portion on a time-period basis.

Furthermore, with respect to the synchronization, archive data cannot just be ‘copied’ as it would be in a fully-relational system; it must be synchronized and maintained in the NoSQL portion. Golosova, et al. (2015) reference a Python library (‘Storage’) which was developed to coordinate this process. This is accomplished using a series of queries on the primary SQL database which fetch relevant data which is then transformed to NoSQL compatible form. From here data undergoes the discussed ‘pre-calculations’ and this cycle runs until the there is no more time-period-relevant matching SQL and NoSQL data.

## 4 Conclusions

This brief insight has shown us that although pure relational SQL will always be the pragmatic man’s favourite, there is room to expand industrial scale and scope in modern fast-transacting data through relational/non-relational hybrid systems. Traditionalist functionalism will always favour the ease of pure SQL systems however, if the inclusion of NoSQL features as discussed in the case above can help soyentists ‘solve’ some of our ‘critical’ issues and expand their scope and efficiency in the modern day, then we must deal with these commercialized consequences. As the relational querying language SQL was created by and for more functional application such as relevant mathematics and information delivery, the language’s syntax will always prevail from the point of view of people who are searching for easily human-readable and understandable input and output as it is more intuitive and generally useful, although minimally expansible.

In ultra-large-scale modern transactions, the reliance on a million components not having to be perfectly aligned at any given moment makes non-relation adaptations very relevant, especially due to the extremely high data volatility implied by the scale. Thus, a dynamic system which manages the intuitiveness of relational systems and the flexibility of non-relational systems has so far proven to work successfully. This has been reflected in the preliminary successes of Golosova, et al. (2015)’s HEP case in which they report major querying efficiency gains, delivered through their NoSQL data realignment and aggregation. They claim “significant results” with minimal “additional complexity”.

Thus in result, we see that hybrid relational/non-relational systems do in fact positively alter data system management of and access to large volumes of, possibly stagnant but still occasionally required, historical data (a.k.a. large linear data stores). It therefore helps better-align these records with the possibly time-altering and more currently relevant, different, structure of current records using dynamic methods. Thus, allowing these two sets of data to interact and to maintain use of historical records. This would not be as easily accomplished using relational systems.

# Bibliography

[1] Codd, E.F. (1970). *A Relational Model of Data for Large Shared Data Banks.* IBM Research Laboratory, San Jose, California.

[2] Fu, T. (2002). *Hierarchical Modeling of Large-Scale Systems Using Relational Databases.* The University of Arizona.

[3] Golosova, M.V., Grigorieva, M.A., Klimentov, A.A., Ryabinkin, E.A., Dimitrov, G., Potekhin, M. (2015). *Studies of Big Data Metadata Segmentation Between Relational and Non-Relational Databases.* Journal of Physics. Conference Series 664.

[4] Jatanal, N., Puri, S., Ahuja, M., Kathuria, I., Gosain, D. (2012). *A Survey and Comparison of Relational and Non-Relational Database.* International Journal of Engineering Research & Technology, Volume 1, Issue 6.

[5] Maowa, J., Hoque, S., Mustafa, R., Rahman, M.O. (2017). *A Comparative Study on Big Data Handling Using Relational and Non-Relational Data Model.* International Journal of Data Mining & Knowledge Management Process. Volume 7. Number 3.

[6] MongoDB. (2022). *NoSQL vs Relational Databases.* Available At: `https://www.mongodb.com/scale/nosql-vs-relational-databases.` (Accessed: 23/03/2022).

[7] National Academies of Sciences, Engineering, and Medicine. (1999). *Funding a Revolution: Government Support for Computing Research.* The National Academies Press.

[8] Quickbase. (2022). *A Timeline of Database History & Database Management.* Available At: `https://www.quickbase.com/articles/timeline-of-database-history.` (Accessed: 23/03/2022).