Survey on Databases being used in Electronic Health Records

Enver Bahar

October 15, 2013

Abstract

This article gives a brief comparison between SQL and NoSQL databases. It discusses what kind of database specifications might be beneficial for storage of Electronic Health Records (EHR), and gives examples of EHR systems using both SQL and NoSQL solutions. Finally it gives some suggestions for internal use cases.

Storage of data and its process are the key points in most of the business cases. Healtcare industry which also deals with a lot of patient information is one of the key industries in this field. Healthcare information is not just about number of records or entities. It deals with hugely diversified data from different specialties (like cardialogy, neurology, etc..), and need to deal with data of Clinical Care, Medication, Labs, History, Demographics, Reports and more. And we should clearly understand that each and every data will have changes every now and then. For example, healthcare standards organization may ask any healthcare providers to introduce / modify / remove a procedure in any given specialty any time. Hence the platform design should incorporate all these demands without any compromise[2]. And maybe the most important point is that healtcare information is extremely conservative: people could die if the data is inconsistent[5].

Electronic Health Records are really important and getting more and more popular. Currently a lot of hospitals are trying to store EHR so that extensive queries can be run against and speed up the process of information retrieval about the storage systems. Even governments are spending billions of dollars to encourage doctors and hospitals to switch to electronic records to track patient care[6].

1 SQL vs NoSQL - which is better for EHR

There are several database systems available, and making a decision between available options is a pretty tough issue and mostly depends on the use cases. Beforehand a database selection, a lot of questions need to be answered such as: How big is the database? How will the database grow over time? What are the required access speeds? Should data be partitioned? In this section we will explain two main database branches, I will provide their pros and cons and try to compare them according to these questions.

Let's first talk about the SQL databases which are also called Relational Databases.

1.1 SQL - Relational Databases

SQL(Structured Query Language) databases are formally named as *Relational Database*. Because it is manipulated by SQL language, the term "SQL Database" became more popular. Relational database is perceived by the user as a collection of two dimensional tables where all data are stored. The concept is proposed by Dr. Codd in 1970 and since 1980 it is the predominant choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personnel data, and much more [3].

A relational database essentially is a group of entities. Tables are made of rows and columns. There are some constraints and relationships that defined between them. Relational databases use SQL for queries, and result sets are acquired from queries that access data from one or more tables. Multiple tables being accessed in a single query are joined together, typically by a criterion defined in the table relationship columns. In order to ensure data consistency and to remove data duplication, database normalization, which is the process of organizing the fields and tables of a relational database, is used [10].

Relational databases are managed by Relational Database Management System(RDBMS). RDBMS is basically a software to manage the database. According to DB-Engine [1], which is an initiative to collect and present information on database management systems, top five RDBMS by means of market sharing are as following: Oracle, MySQL, Microsoft SQL Server(MSSQL), PostreSQL and DB2. In Brainlab, all of the databases are Relational Databases and the database management system that is currently being used is MSSQL.

Let's talk about the good sides of relational databases. First of all relational databases are a known quantity: there are proven solutions for nearly every major problem a web developer would run into. They have a long history, which means that they are stable, and have long established standard. All relational databases offer SQL as a language, therefore moving from one vendor to another is trivial. SQL is a complete database language that is written like the English language - the statements are easy and understandable.

One of the bad sides of relational databases is it might be really hard to design complex databases. One needs to carefully create ER diagrams, normalize the tables, choose the primary keys and set the relationships in order to make a good design which are kind of a long and hard task. Additionally, relational databases are not good if the data size is bigger than 500GB. Scaling after that amount of data is getting more and more troublesome, and can be really expensive as well. Furthermore, relational databases have a strict table-column-row organization. When the content is changing dynamically, or the content needs to be configurable, such a relational database might not be a good solution.

	Circa 1975 Online applications	Circa 2011 Interactive web applications
Users	2000 online users=End point Static user population	2000 Online users=Start point Dynamic user population
Applications	Business process automation Highly structured data records	Business process innovation Structured, semi- and unstructured
Infrastructure	Data network in its infancy Centralized computing (Main frames and mini computing) Memory scarce and expensive	Universal high speed data networking, distributed computing (Networked servers) and virtual machines Memory plenty and cheap

Table 1: Comparison of tendencies from 1975 and today

1.2 NoSQL Databases

Currently relational databases are the predominant choice and the reasons for this are not trivial: they are constantly offered as one of the best mix of simplicity, robustness, flexibility, performance, scalability, and compatibility in managing generic data. However to offer all of this relational databases have to massively complex internally. For example, a relatively simple SELECT statement could have hundreds of potential query execution paths, which the optimizer would evaluate at run time. RDBMS determines the execution plan that best answers the requests by using cost based algorithms, and these are hidden to the users [7]. Although RDBMS have provided database users with one of the best *mix* of simplicity, robustness, flexibility, performance, scalability, and compatibility, their performance in each of these areas are not necessarily better than an alternate solution concentrating on one of these benefits specially. This has not been a problem so far because RDBMS popularity has dominated the need to push any of these boundaries, because if you really need that could be solved by a relational database, alternatives always were available to fill those gaps [8].

Today however, there is a different situation(Table 1). The new generation applications require processing of hundreds of gigabytes, terabytes, even petabytes of data. For an increasing number of applications, one of these benefits becoming more crucial and it is rapidly becoming mainstream. This requirement is beginning to eclipse others in importance. This benefit is *Scalability.* Scalability is defined as the ability of a database to continue to function well when its volume is changed[4]. Every day more and more applications with massive workloads are launched, such as web services. Their scalability requirements can change very quickly and grow very large. There are cases where the load triples in a single day, and the hardware needs to be quickly upgraded. This scenario is too difficult to manage with a relational database. When scaling needs to be done on a single server node, it can be done on relational databases well. However, when the capacity of that single node is reached, the load needs to be distributed to multiple server nodes. This is when the complexities become overpowering, and hard to implement. When this amount reaches to hundreds or thousands of nodes, the characteristics that make RDBMS so charming drastically reduce their viability as platforms for large distribud systems [7]. In response to these demands, a new type of database is created: NoSQL.

The term of *NoSQL* describes the new kind of databases that are available in response to the limitations of existing relational databases. In NoSQL databases, data is still stored in tables, but it does not have the expected normalization. Instead, one of the primary storage is key-value pairs which is similar to a hash map. The transaction speed is overwhelming, and this requires large portions of the database to be in memory for quick reads and writes rather than stored on the disk. Old fashioned databases are not built with this kind of performance in mind. NoSQL databases allow for a great deal of optimization such as horizontal partitioning, or sharding which allows logically segregated data sets to be stored seperately [9].

So NoSQL is a database that is especially build for big data, and is the ideal choice for the data which increase really fast. How about healthcare? I assume that in healthcare this increase is not going to be as fast as big web companies, but still there are good sides of NoSQL that can be used. First thing that might be is the usage of dynamic schemas. In relational databases table schemas have to be defined before you can add data. For example you might want to store information about your patients such as name, phone numbers, address - a SQL database needs to know this in advance. So if you decide in further to change or add some fields like whether this patient has cancer or not, you will need to add that column to the database, and then migrate the entire database to the new schema. If the database is really large, this is a very slow process and this will cause significant downtime. In NoSQL databases on the other hand, there is no such a defined schema. That makes it easy to make significant application changes in real-time, without worrying about service interruptions - which means development is faster, code integration is more reliable, and less database administrator time is needed. If we want to add another column to our patient table, for example the blood type or whether he has a disease or not, it is done immediately in NoSQL databases. Furthermore, all of the NoSQL databases are open-source and data manipulation is done through object oriented APIs.

However there are some cons of NoSQL databases as well. For example, NoSQL databases are not standardized and because they are kind of a new technology the support is not as easy as in relational databases. Encountered problems are quite new, and most of the NoSQL products are currently in the beta stage.

2 Market Examples

Although NoSQL databases are kind of a new technology, there are examples of Electronic Health Records using NoSQL:
VistA:
CHCS:
AHLTA:
Epic:
Cerner:

References

[1] Db engines ranking of relational dbms. `http://db-engines.com/en/ranking/relational+dbms`. Accessed: 2013-10-10.

[2] Mongo db and its application (a case study of mongodb in healthcare). `http://kousikraj.wordpress.com/2012/06/05/mongo-db-and-its-application-a-case-study-of-mongodb-in-healthcare/`. Accessed: 2013-10-10.

[3] Relational database management systems, database design, and gis. `http://msdis.missouri.edu/resources/gis_advanced/pdf/Relational.pdf`. Accessed: 2013-10-10.

[4] scalability. `http://searchdatacenter.techtarget.com/definition/scalability`. Accessed: 2013-10-15.

[5] Sql vs nosql databases: a new debate of interest for healthcare systems. `http://ranger.uta.edu/~zikos/courses/5339-4392_content_repository/presentations/WEEK3THEORY6-Relational%20vs%20NoSQL%20Databases.pdf`. Accessed: 2013-10-10.

[6] Sql vs. nosql in healthcare. `http://www.bigdatarepublic.com/author.asp?section_id=2642&doc_id=256137`. Accessed: 2013-10-15.

[7] Tony Bain. Is the relational database doomed. 2009.

[8] Uma Bhat and Shraddha Jadhav. Moving towards non-relational databases. *International Journal of Computer Applications*, 1(13):40–46, 2010.

[9] Ruxandra Burtica, Eleonora Maria Mocanu, Mugurel Ionuţ Andreica, and N Tapus. Practical application and evaluation of no-sql databases in cloud computing. In *Systems Conference (SysCon), 2012 IEEE International*, pages 1–6. IEEE, 2012.

[10] Terence Lucey. *Management information systems.* CengageBrain. com, 2004.