Healthcare Big Data Exploration in Real-Time

Muaz A Mian

A Project
Submitted in partial fulfillment of
the requirements for degree of

Masters of Science in Computer Science and Systems

University of Washington

2014

Committee:
Dr. Ankur Teredesai
Dr. Senjuti Basu Roy
David K Hazel

Program Authorized to Offer Degree:
Institute of Technology

University of Washington

Abstract

Healthcare Big Data Exploration in Rea-Time

Muaz A Mian

Chair of Supervisory Committee:
Associate Professor Dr. Ankur M Teredesai
Institute of Technology

Advances in healthcare data management and analytics have opened new horizons for healthcare providers such as cost effective treatments, ability to detect medical fraud, and diagnose diseases at an early stage. Central to these abilities is the need for fast ad-hoc query processing of large volumes of complex healthcare datasets. End users who work with healthcare databases spend enormous effort in data exploration since exploration is the first step to any subsequent predictive modeling to generate actionable insights for patients, providers and physicians. Unfortunately, unlike other domains the complexity and volumes of claims (ICD9 or 10) as well as clinical (HL7) healthcare datasets results in data exploration solutions being extremely slow and cumbersome when attempted using traditional disk resident data warehousing approaches. In this project we perform the first ever attempt of real-time data exploration for healthcare datasets using in-memory databases. We benchmark and compare two such in-memory database systems to study responsiveness and ability to handle complexity of typical health data exploration tasks. We developed a browser based tool that can act as an extremely responsive real-time visual query interface using an in-memory database.

TABLE OF CONTENTS

1 INTRODUCTION

This project explores motivating scenarios for exploring very complex medical data sets in real time, as well as exploring the implications that emerging technologies can have on existing analysis efforts that have risen out of traditional RDBMS enterprise data warehouses. Further we explore emerging capabilities in being able to perform ad hoc query on healthcare data in real-time. We compare and contrast two commercially available In-memory database technologies MemSQL [4] and VoltDB [9] for responsive healthcare data exploration. Both of these databases are NewSQL [5] relational databases. For comparison we used Medicare Claims Synthetic data from Center for Medicare and Medicaid Services (CMS). Medicare Claims Synthetic Public Use Files (SynPUFs) are created to allow interested parties to gain familiarity using Medicare claims data while protecting beneficiary privacy [1].

The main contributions of this work are:

- Faster data exploration
- Enabling real-time predictions
- Database benchmark on healthcare data

1.1 HEALTHCARE DATA

Healthcare data has been digitized for more than 20 years resulting in about 50 Petabytes today. It is estimated to grow significantly by a factor of 50, to around 25,000 Petabytes by 2020 [6], this data is not just big in volume but big in value too. Thus it is natural to aspire that this huge influx of healthcare data has enabled data scientists to make predictions more accurately for various quality of life improvement and cost saving tasks [3, 8]. Finding value in the data requires a thorough examination by data architects to prepare the data for prediction. There are many challenges in exploring large amounts of healthcare data due to lack of adequate data management technologies that enable responsive data exploration and analysis. For example, due to the significant number of attributes, identifying the optimal subset of patients and generating a dataset of a particular cohort, even over a relatively modestly sized dataset, still takes significant time.

Many concerns arise when dealing with healthcare data. Healthcare data is very high in dimensionality, extremely complex and skewed, a single patient's record contains hundreds of attributes. The data is both sparse, and noisy. Many relevant attributes that contribute to effective analytics, such as predicting risk of readmission [11] still exist in freeform fields such as doctor's notes. As a result cleaning and transforming must be done before prediction.

1.2 IN-MEMORY DATABASES

The idea of in-memory database has been floating around for quite some time now, but size of data increased and increasing memory size was not feasible, so in-memory databases were not well-known. Recent trends in big data and demand for faster analytics revived in-memory databases. Is it reasonable to assume that the entire database fits in main memory? Yes, for some applications. In some cases, the database is of limited size or is growing at a slower rate than memory capacities are growing [2]. Modern database have the potential to work on cluster and support a larger memory size.

An in-memory databases uses main memory as main source of data storage, keeping data in main memory enables faster read and write performance since there is no disk overhead. Durability of an in-memory database can be maintained by mechanisms such as snapshot, checkpoint or data replication. Non-Volatile DIMMS, currently being developed, can maintain data even in power failure and will be commercially available in the near future.

1.3 HEALTHCARE DATA EXPLORATION

Data exploration is an integral part of developing any prediction model. Scientist needs to closely study and prepare the data for prediction. Typically in a healthcare system, data would initially reside in an Electronic Medical Record (EMR) system and after 24 hours moved to an Enterprise Data Warehouse (EDW). A Data Architect would preprocess, clean, and transform the data, a subset of the resultant data is analyzed to identify useful attributes

Currently this process is time consuming mainly because of slower disk accesses. A single query can take several minutes and numerous queries have to be executed to clearly understand the data. Keeping all or most of the data in-memory removes the I/O cost and provides quick access.

Normally data is moved into an Enterprise Data Warehouse after 24 hours, so prediction models rely on a day old data. Both database systems allow data to be loaded as it is being generated and immediately able to query it. Such feature extends the capability to provide real time risk predictions, allowing doctors to further understand the results of the model by allowing them to enter additional queries on existing data, or perform hypothetical scenarios, changing some attributes and seeing how these changes impact the predictive models, providing further insights to help him diagnose his patient.

2 RELATED WORK

Recognizing that real-time analytical systems offer many advantages to organizations, startups continually emerge, each having developed their own in house storage engine. SAS High-Performance-Analytic (HPA) [7] is capable of quickly analyzing terabytes of data and lets the user generate statistics in just a few seconds with a simple drag and drop functionality. Tableau [10] is also a visual analytic tool created in 2003 that allows users who are not familiar with databases or SQL language to explore their database. Over time they have developed and released support for all major RDBMS, column-oriented databases and big data Hadoop applications. Analytic packages like SAS and Tableau require to be installed as a separate application.

While the argument still continues whether to use a SQL or a NoSQL approach, we see an emergence of NewSQL database systems. NewSQL are relational databases with faster read-write performance similar to NoSQL databases and guarantee ACID property similar to SQL databases. With the power of NewSQL databases enterprises can develop faster and responsive application along with their own in-house analytical engine.

3 METHODOLOGY

Experiments were carried on machines with configuration that you would normally find in a server machine commercially available. The two machines used had an AMD Opteron(tm) Processor 6128 (2 X 8 Cores 2GHz) and 128GB RAM with 64-Bit Linux Operating System. Experiments were conducted in a distributed/multi-clustered environment. Greater the number of nodes, greater is the in-memory support. Both systems support data backup and recovery in case a node goes down. Below are the setup explanations for both MemSQL and VoltDB.

3.1 MEMSQL SETUP

These experiments were conducted on MemSQL version 2.5. In MemSQL there are two type of nodes, aggregator and leaf. An aggregator node stores only the Meta data of the database and all the actual data is stored in leaf nodes. Since we had only two nodes, one was being used as aggregator and as a leaf by running two separate instances of MemSQL. Figure 1 shows the setup for MemSQL.

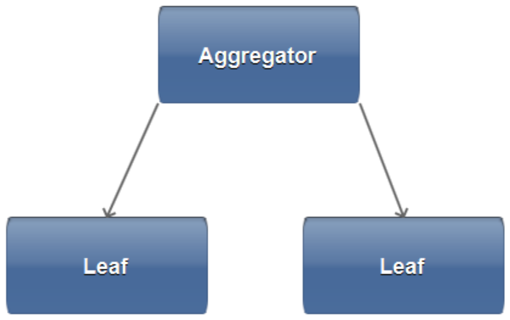


Figure 1. Setup for MemSQL experiment

3.2 VOLTDB SETUP

These experiments were carried on VoltDB Community Edition version 3.7. In a cluster of nodes for VoltDB one node has to be specified as the leader, the leader will be the master node communicating with slaves. Unlike an aggregator in MemSQL it will also store records. In our two node cluster one was selected as the leader randomly.

4 EXPERIMENTATION

CMS [2] data was loaded into two tables. First table is Beneficiary table with 32 attributes which contains anonymous information about beneficiaries e.g. date of birth, gender, state or any diagnoses. Second table is Carrier Claims table with 142 attributes which contains information about beneficiary's medical provider, claims, expense and diagnosis. The amount of data loaded in tables varied between experiments to observe performance on different size of datasets. CMS data can be downloaded in total 20 folders but for the three experiments data was loaded of 4, 8 and 12 folders respectively. Common questions that can be asked on these tables are same as mentioned in the introduction i.e. how many Washington State Resident females in the dataset have diabetes or select all males under 40 currently under treatment for depression

4.1 BENCHMARK QUERIES

Queries designed for the experiments can be categorized into two types. First category consists of simple queries to measure how fast does a database processes information, mainly consisting of aggregates, group by and join queries without any conditions. The goal was to make sure all data is processed which might not always be the case. Second type of queries were normal exploration queries that necessarily don't process all of data. These are normal exploration queries e.g. get me the number of females in Washington who have diabetes or, get me number of people who have congestive heart failure and have claimed for a diabetes treatment. Given our focus was to analyze performance on healthcare data, we designed 12 queries that were optimized to investigate performance. The queries varied from simple aggregate scan queries designed to process entire data to your typical exploration scan queries. Table 1 shows details of the queries we selected to execute on both databases systems.

Name	Description
Q1	SELECT COUNT(DISTINCT desynpuf_id) FROM carrier_claims ;
Q2	SELECT COUNT(DISTINCT(desynpuf_id)) FROM carrier_claims WHERE clm_from_dt >= 20090101 AND clm_thur_dt <= 20091231;
Q3	SELECT COUNT(DISTINCT desynpuf_id) FROM carrier_claims WHERE icd9_dgns_cd_1 = '29606';
Q4	SELECT COUNT(DISTINCT desynpuf_id) FROM carrier_claims WHERE prf_physn_npl_1 = 9249792168;
Q5	SELECT COUNT(clm_id) FROM carrier_claims WHERE clm_from_dt >= 20090101 AND clm_thur_dt <= 20091231;
Q6	SELECT COUNT(bene_sex_ident_cd) FROM beneficiary WHERE sp_chf = 1 AND sp_state_code = 50 AND bene_sex_ident_cd = 2;
Q7	SELECT COUNT(bene_sex_ident_cd) FROM beneficiary WHERE sp_diabetes = 1 AND bene_sex_ident_cd = 2;
Q8	SELECT COUNT(0),bene_sex_ident_cd FROM beneficiary INNER JOIN carrier_claims USING (desynpuf_id) WHERE sp_diabetes = 1 AND icd9_dgns_cd_1 = '24981' GROUP BY bene_sex_ident_cd;
Q9	SELECT COUNT(0) FROM beneficiary INNER JOIN carrier_claims USING (desynpuf_id);
Q10	SELECT COUNT(0) FROM carrier_claims INNER JOIN beneficiary USING (desynpuf_id);

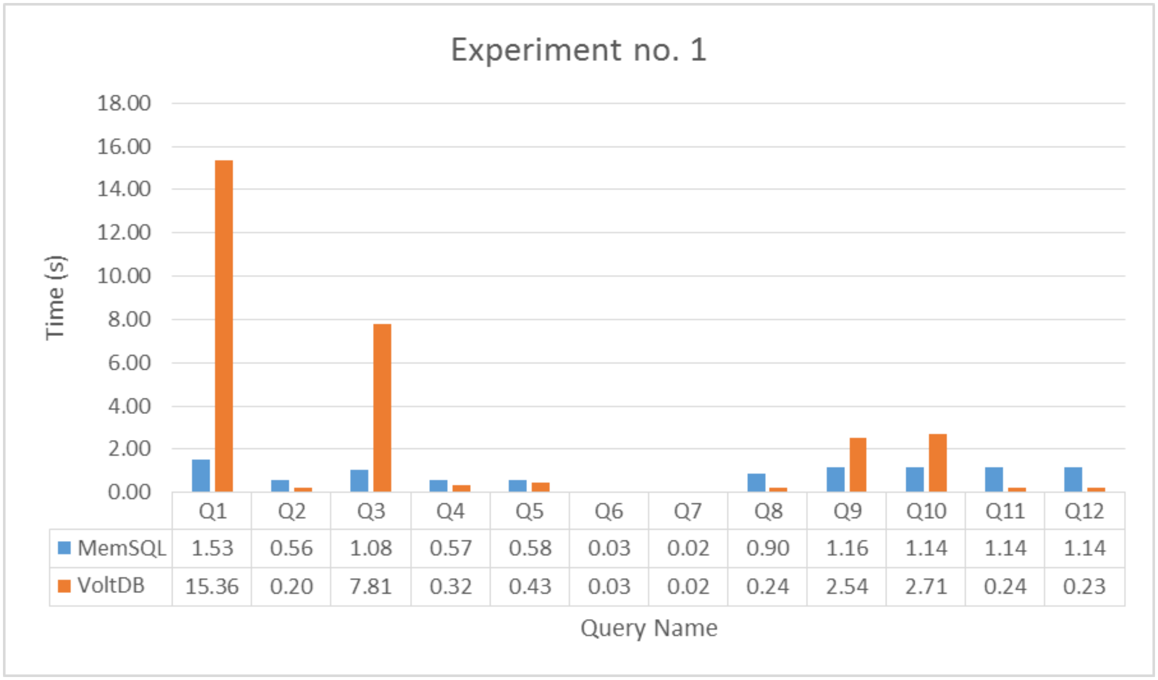
Q11	SELECT COUNT(0) FROM beneficiary INNER JOIN carrier_claims USING (desynpuf_id) WHERE bene_race_cd = 1 AND icd9_dgns_cd_1 = '29606';
Q12	SELECT COUNT(0) FROM carrier_claims INNER JOIN USING beneficiary (desynpuf_id) WHERE bene_race_cd = 1 AND icd9_dgns_cd_1 = '29606';

Table 1. List of Queries

Q1 – Q3 are queries to get all unique values, these were designed to see how quickly data is aggregated. Q4 – Q8 are normal exploration queries to find a specific group of beneficiaries. Finally Q9 – Q12 are simple joins between the two tables and joins with condition. Notice that Q9 and Q10 are same queries but with reverse table order, meaning in Q9 beneficiary table joins on carrier_claims where as in Q10 carrier_claims joins on beneficiary. This was to see if order of tables effects the overall performance. Same is the case with Q11 and Q12.

4.2 RESULTS

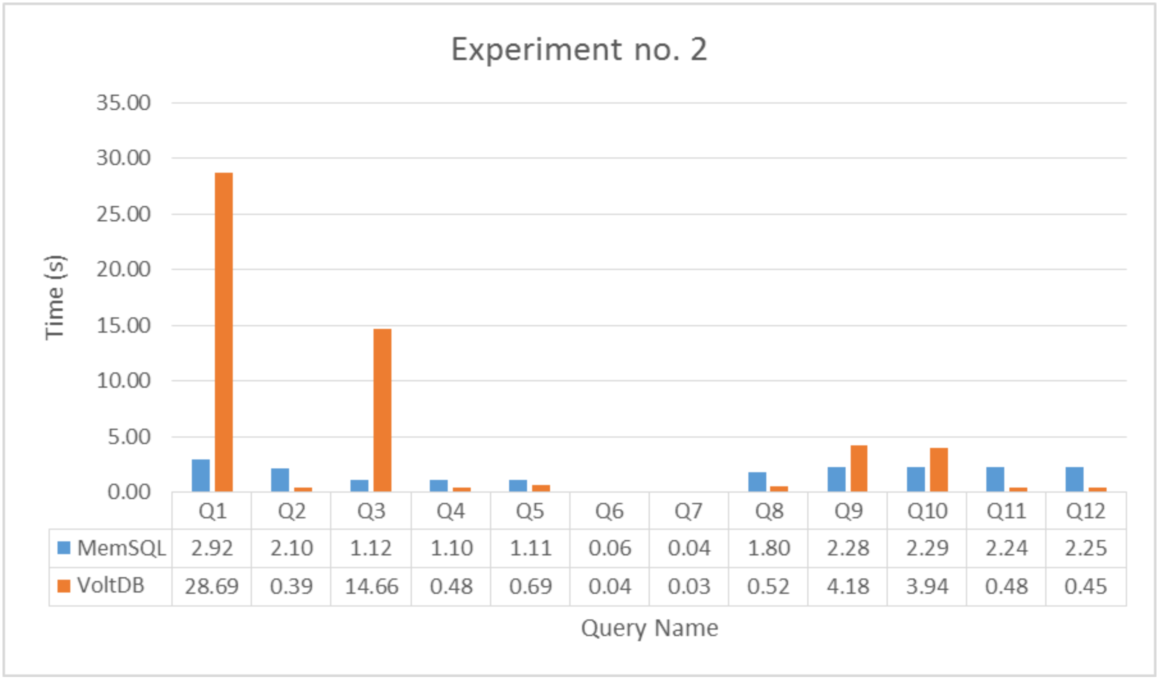
Figure 2 contains time comparisons after first experiment. For this experiment Beneficiary table had 458,328 records and Carrier Claims had 18,975,932 records.



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
■ MemSQL	1.53	0.56	1.08	0.57	0.58	0.03	0.02	0.90	1.16	1.14	1.14	1.14
■ VoltDB	15.36	0.20	7.81	0.32	0.43	0.03	0.02	0.24	2.54	2.71	0.24	0.23

Figure 2. Time comparison for first experiment

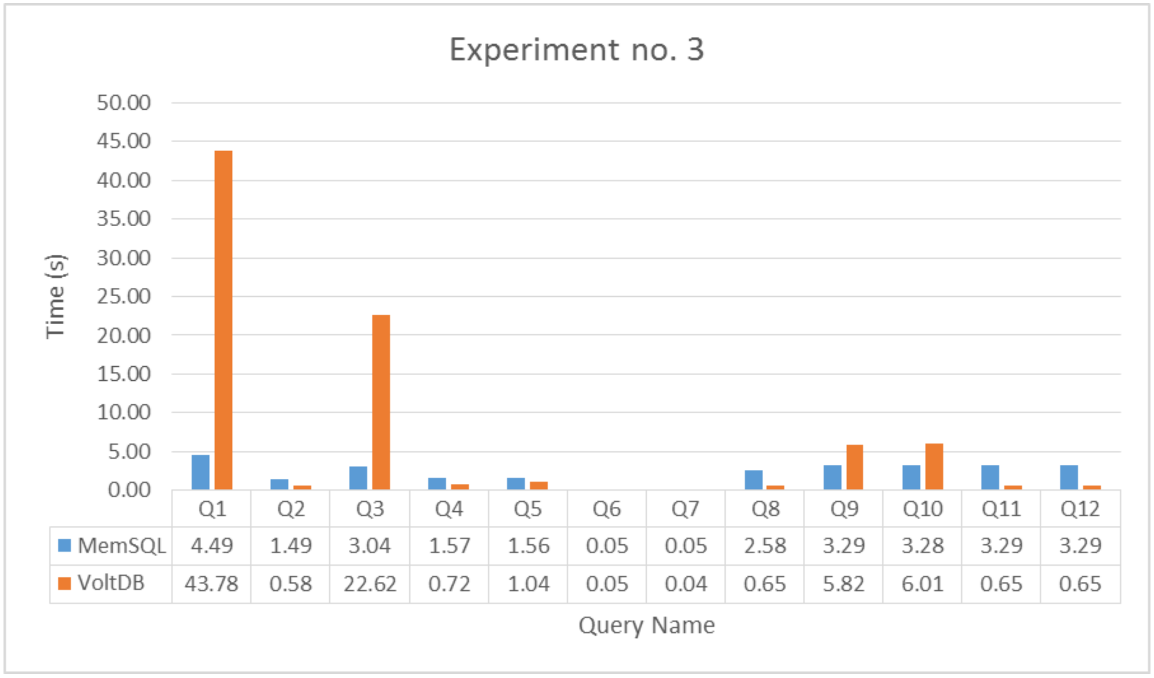
Figure 3 contains time comparisons after second experiment. For this experiment Beneficiary table had 916,557 records and Carrier Claims had 37,936,964 records.



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
■ MemSQL	2.92	2.10	1.12	1.10	1.11	0.06	0.04	1.80	2.28	2.29	2.24	2.25
■ VoltDB	28.69	0.39	14.66	0.48	0.69	0.04	0.03	0.52	4.18	3.94	0.48	0.45

Figure 3. Time comparison for second experiment

Figure 4 contains time comparisons after third experiment. For this experiment Beneficiary table had 1,374,745 records and Carrier Claims had 54,536,284 records.



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
■ MemSQL	4.49	1.49	3.04	1.57	1.56	0.05	0.05	2.58	3.29	3.28	3.29	3.29
■ VoltDB	43.78	0.58	22.62	0.72	1.04	0.05	0.04	0.65	5.82	6.01	0.65	0.65

Figure 4. Time comparison for third experiment

4.3 OBSERVATION

Looking at the charts above, we can observe how VoltDB consistently performs slower when asked to return number of unique beneficiaries in carrier_claims table, although Q2 looks faster but result set was very small.

Normal exploration queries were fast for both database systems, there is not much time difference. But an interesting observation is when join queries are executed. MemSQL gives a consistent performance with join queries in all three experiments, it does not matter whether the order of tables is reversed or a condition is added at the end, all five join queries take almost the same time in each test. For VoltDB, join queries gave slower performance when there is no condition but a faster performance with the condition.

5 APPLICATION

After establishing that in-memory databases will respond back in real-time we developed a web based query builder that can allow non-technical users to be able to query data in real-time. Main feature of our application is provide an effective data exploration with an easy-to-use interface.

5.1 TABLE LIST

The application will read the Meta data of the selected database and generate a list of tables. The user can select any one of the table which will expand and display all fields from selected table. Figure 5 and 6 show list and tables and expanded list containing field values of one table.

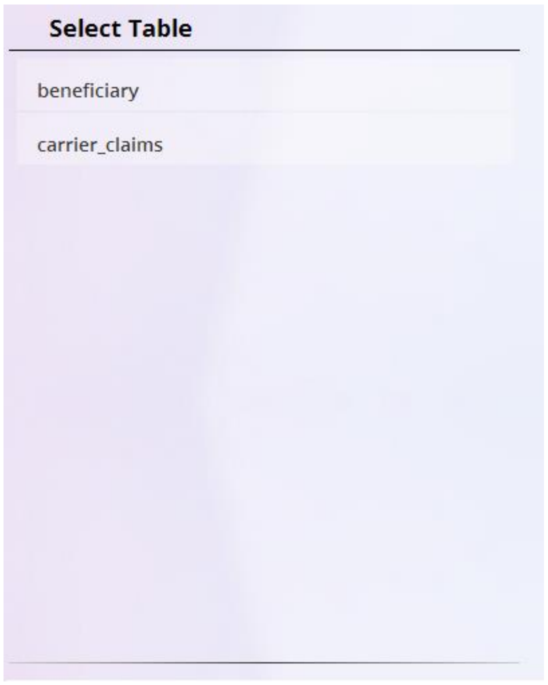


Figure 5. List of Tables

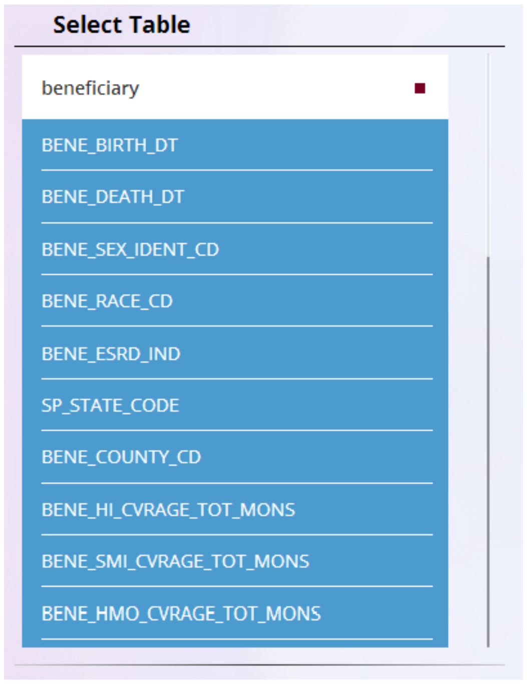


Figure 6. List of Fields for selected table

5.2 VALUE SELECTION

All the field in figure 6 can be dragged and dropped into drop area, dropping a field will generate a list of possible values that currently exist in database. User can select the values, selecting a value will generate charts in our analytics box. Figure 7 shows drop area after a field was dropped into it.

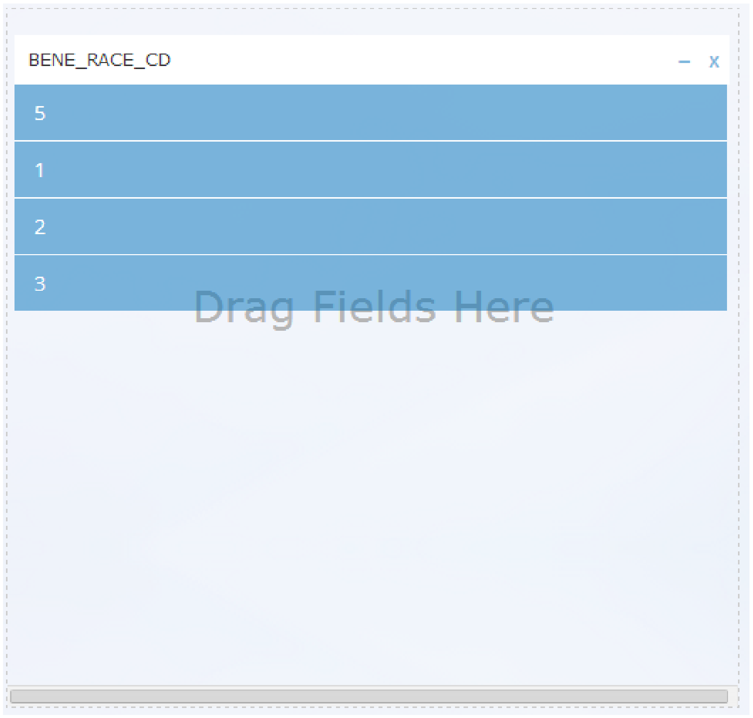


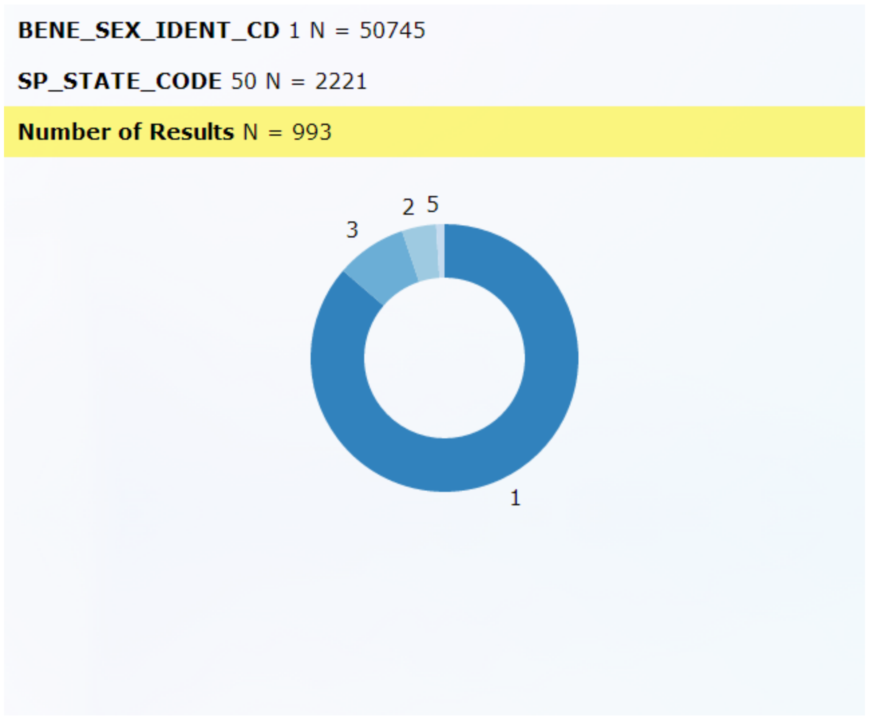
Figure 7. List of possible values for BENE_RACE_CD field

5.3 DATA VISUALIZATION

As application's main purpose is for data exploration, it provides multiple visualizations to help you understand the data. By default visualization is auto detect, based on selected data set the application will infer a visualization. User can also select appropriate visualization based on his knowledge of data. Currently application supports three types of visualizations pie chart, bar chart and map. Figure 8 shows race distribution of all male beneficiaries from Washington State. Table 2 shows race mapping.

Code	**Race**
1	White
2	African American
3	Other
5	Hispanic

Table 2 Race Code Mapping


BENE_SEX_IDENT_CD 1 N = 50745

SP_STATE_CODE 50 N = 2221

Number of Results N = 993

Figure 8. Race distribution for all males in Washington

6 CONCLUSION

As we have shown, there are massive challenges in healthcare data exploration and prediction, which due to its complex nature is time consuming. We propose to replace traditional SQL databases with in-memory SQL databases in the ecosystem, which could result in quicker data exploration and real-time prediction with latest data. We compared two in-memory relational systems using synthetic medical data measuring their performance on various type of queries. Our study shows that most of queries will respond back within first 10 seconds. We presented our application called Query Builder designed to explore data in real-time and provide analytic.

7 ACKNOWLEDGEMENT

The author of this report would like to thank MemSQL for granting an evaluation copy for our research and VoltDB for their prompt technical support. The author is thankful to LigaData for providing hardware for experiments, especially Sreenivasulu Pokuri for constant guidance in developing test queries. This project is an ongoing research project for Multicare Health System in collaboration with Center for Web and Data science at University of Washington.

8 REFERENCES

[1] (2/15/2014). *Center for Medicare and Medicaid Services*. Available: http://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/SynPUFs/.

[2] H. Garcia-Molina and K. Salem. Main memory database systems: An overview. *Knowledge and Data Engineering, IEEE Transactions on 4(6),* pp. 509-516. 1992. . DOI: 10.1109/69.180602.

[3] Koh, Hian Chye and Tan, Gerald and others. Data mining applications in healthcare. *Journal of Healthcare Information Management—Vol 19(2),* pp. 65. 2011.

[4] (6/1/2013). *MemSQL*. Available: http://www.memsql.com/.

[5] (2/16/2014). *NewSQL*. Available: http://en.wikipedia.org/wiki/NewSQL.

[6] (2/22/2014). *How Big Data is Improving Healthcare*. Available: http://readwrite.com/2012/10/02/how-big-data-is-improving-healthcare.

[7] (5/31/2013). *SAS High-Performance-Analytic*. Available: http://www.sas.com/high-performance-analytics/index.html;.

[8] J. Sun and C. K. Reddy. Big data analytics for healthcare. Presented at Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2013, Available: http://doi.acm.org.offcampus.lib.washington.edu/10.1145/2487575.2506178. DOI: 10.1145/2487575.2506178.

[9] (5/30/2013). *VoltDB*. Available: http://voltdb.com/.

[10] (6/1/2013). Tableau Software. Available: http://www.tableausoftware.com/.

[11] K. Zolfaghar, J. Agarwal, D. Sistla, S. Chin, S. Basu Roy and N. Verbiest. Risk-O-meter: An intelligent clinical risk calculator. Presented at Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2013, Available: http://doi.acm.org/10.1145/2487575.2487717. DOI: 10.1145/2487575.2487717.