Graph Database in Large Scale Healthcare System
A Proposal for Efficient Data Management and Utilization

Yubin Park
University of Texas at Austin
TX, USA
Email: yubin.park@utexas.edu

Mallikarjun Shankar
Oak Ridge National Laboratory
TN, USA
Email: shankarm@ornl.gov

Joydeep Ghosh
University of Texas at Austin
TX, USA
Email: ghosh@ece.utexas.edu

Abstract—**Healthcare data management and utilization have been traditionally viewed as two orthogonal tasks. This structural gap between management and utilization has resulted in unnecessarily complex database systems, as well as poor data utilization. In this paper, we uncover rich graph structures from the current US healthcare systems, which potentially lead to fill this operational gap. A healthcare graph can be automatically constructed from a normalized relational database, using our "3NF Equivalent Graph" (3EG) transformation. We collect real world graph queries such as finding self-referrals, shared providers, and collaborative filtering. These queries are evaluated over a relational database and its 3EG-transformed graph. Our experimental results show that the resultant graph actually can serve as multiple de-normalized tables, thus reducing complexity in a database and enhancing data accessibility of users. Based on this finding, we propose a cost efficient ensemble framework with the current RDBMS healthcare architecture.**

I. Introduction

Data-driven approaches can substantially reduce the current healthcare expenditure in the United States [1]. The total healthcare spending of the United States recorded the highest in the world in 2011 [2], and the Health and Human Services Department expects the spending will continue to increase [3]. Both private and public healthcare sectors are experiencing drastic changes in their policies and data management systems [4]. Solutions to provide "Best Care at Lower Cost" [5], [6] need to be addressed from multiple angles. For example, legislative health reforms are expected to slow down the current increasing speed of healthcare spending [7]. Effective data utilization, on the other hand, has been widely suggested as an alternative solution to prevent the cost leakage and waste in the healthcare systems [8]. Data-driven solutions would provide better personalized, objective, and cost-effective healthcare, as well as early-detection of fraud and abuse. However, complex and dynamic characteristics of healthcare data preclude such promising data utilization.

Data management and utilization have been traditionally viewed as two independent tasks. Data schemas for healthcare systems are very volatile by their nature, and usually involve a multitude number of atomic entities and their relationships. To minimize redundancy and dependency, healthcare data are typically stored and managed using their "normalized" forms [9], [10]. Those normalized tables are later either restructured or "de-normalized" to be actually meaningful in data analytics [11], [12]. Due to this structural gap between management and utilization, poorly designed databases are prone to produce extraneous dummy schemas. For example, a normalized healthcare database normally consists of hundreds of atomic tables. If we include service-oriented de-normalized tables, then the number of tables in a system grows uncountably many. Furthermore, understanding such complex schema often serves as a bottleneck to many other analytic procedures. Efficient de-normalization techniques are still open to discussion, and those dummy tables are mostly disposable.

In this paper, we unearth rich graph structures from the current US healthcare systems. We adopt a "graph database" to extensively utilize such graph representation. Unlike traditional relational databases, a graph database directly stores and represents nodes, edges, and properties [13]. A graph database can handle a wide range of queries, especially graph queries, which would require deep join operations in normalized relational tables [14]. Several examples of such deep join queries are presented in Section V. These queries involve relationships between healthcare entities. Even these simple queries can be utilized in collaborative filtering as well as other personalization services. We evaluate the performance of such queries over a relational database and its equivalent graph representation. Our experimental results suggest that a graph database clearly exhibits its excellence on its performance scaling and intuitive query representation. We observe that a single graph database representation can serve many different types of queries that would require many analytic dummy tables. A graph database as a form of "de-normalized table" can discard generating such redundant dummy tables. This would potentially minimize the gap between management and utilization in healthcare systems.

Our ultimate goal is to propose a cost efficient data management framework for healthcare systems. A graph database is a crucial "component" of such framework, rather than a whole replacement to the existing architecture. As can be seen later in our experiments, traditional relational databases still have many advantages over graph databases. Even further, each graph database has its own specialty and application needs [15]. For example, "Neo4J" [16] is suited for online transaction processing (OLTP), while "Pregel" [17] is designed for high latency, high throughput platforms. Efficient "ensemble" of these components will lead to a better system design, capturing a wide range of services efficiently. To summarize our contributions in this paper:

- We propose a novel graph database design rationale, “3NF Equivalent Graph” (3EG) transform. This technique can automatically construct a graph database from an existing normalized relational database.
- We generate realistic and large scale synthetic healthcare data to simulate the current US healthcare systems.
- We collect eight useful graph queries, and evaluate the performance of these queries over two representative databases: MySQL for a relational database and Neo4J for a graph database.
- We propose a blueprint for a cost efficient database architecture in healthcare systems. This proposal requires minimal changes to an existing architecture.

In Section II, we briefly discuss US healthcare systems in general. We then introduce several forms of database normalization, and various graph databases. In Section III, we propose a novel graphs database design rationale, 3NF Equivalent Graph (3EG) transformation. This technique converts an existing normalized relational database to its equivalent graph representation. In Section IV, we illustrate how we generate “realistic” synthetic healthcare data. These data are firstly stored in a relational database format, then converted to a graph format using the 3EG transformation. In Section V, we collect several real world use cases in healthcare databases, and describe their actual meanings in actual systems. Empirical evaluation results using MySQL and Neo4J are provided in Section VI. Discussion and Conclusion appear in Section VII.

II. Background

In this section, we describe US healthcare systems in general. We also visit basic concepts in database normalization and graph databases.

A. US Healthcare System

We describe US healthcare systems by an illustrative example. Let’s consider a situation that a patient (or a beneficiary) visits a hospital for his illness. A doctor (or a provider) diagnoses the patient, and performs a certain procedure. Using the patient’s insurance information, a claim is later filed to a corresponding organization. For example, if the patient is a Medicare enrollee, the claim is submitted to Centers for Medicare and Medicaid Services (CMS). If the patient is with a private sector health insurance, the claim will be delivered to the corresponding institution. This filed claim usually contains a list of providers involved, referral information (occasionally), information of the patient, diagnosis and procedures performed, and cost.

Figure 1 shows the relational representation of this healthcare system. Bene and Prov tables contain the information of patients and providers, respectively. Claim table contains diagnosis (ICD-9) and procedure information. The relationships between beneficiaries and claims are captured in C2B table, and those between providers and claims are in C2P table. Referral information is recorded on C2R table, and the referral ID is a foreign key to Provider ID.

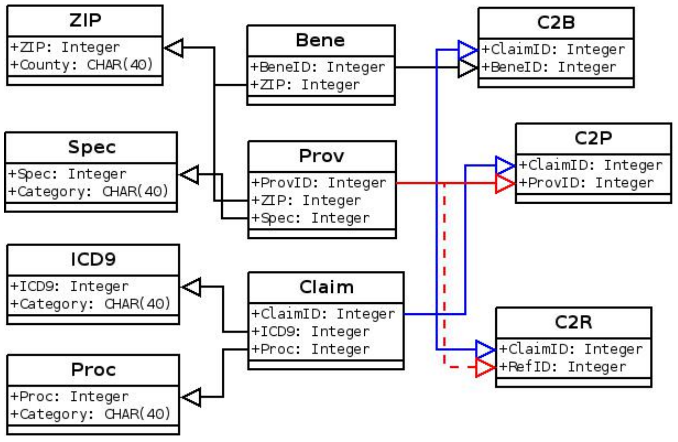

Figure 1: Third Normal Form of US Healthcare System.

B. Database Normalization

Figure 1 actually follows Third Normal Form, and this level of normalization is typically performed in practice. Database normalization, firstly proposed by E. F. Codd, is a fundamental relational database design rationale to minimize redundancy and dependency [18], [19]. By reducing redundancy and dependency of a system, transactional operations, such as additions, deletions, and updates, can be made in one table. Normalized tables, in theory, are better protected from transactional anomalies and inconsistencies from dynamic schema changes.

Each Normal Form imposes a set of rules. A specific normal form is achieved when its requirements are fulfilled. First Normal Form (1NF) pressumes the existence of the “key”, and requires domains of a table to be “atomic”. In other words, every attribute in a table should not be divided further. For example, a name attribute containing both first and last names is not atomic, since it actually holds two fields. To understand Second Normal Form (2NF), we should introduce the concept of “functional dependency”. A field X is said to have a functional dependency on a field Y if and only if values of X are precisely mapped to values of Y. 2NF is accomplished when i) 1NF is achieved and ii) no non-key element is functionally dependent on a proper subset of key elements. Third Normal Form (3NF), the most popular normal form, adds a further specification to 2NF requirements. In 3NF, no transitive dependency is allowed between every non-key elements. Specifically, if a table has ZIP code and County code as non-key elements, County code should be removed from the table since County code is transitively dependent on ZIP code.

As an illustrative example, Figure 2 shows non-3NF representation of the presented healthcare system. The design in the figure actually violates the requirement for 2NF. As discussed, each claim can have multiple providers in healthcare systems. This design may allow repetitive rows for the same claim. ICD-9 code and Procedure code for each claim are functionally dependent on the combination of ClaimID and BeneID (a set

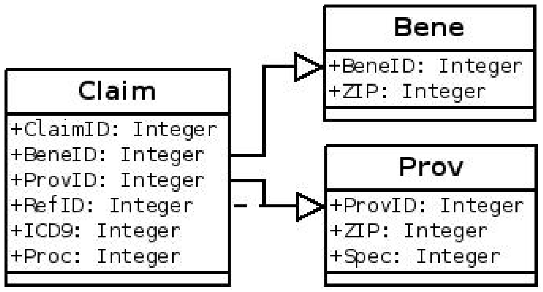

Figure 2: Non-3NF Representation of the Healthcare System.

of key elements), violating the 2NF requirement. Since this design fails to meet the 2NF specifications, this cannot be 3NF automatically. In this paper, we do not consider other higher forms of normalization such as 4NF [20] and 5NF [21], and mainly focus on 3NF. In most practical situations, higher normal forms than 3NF is not preferred due to their unnecessarily specific constraints.

C. Graph Database

A graph database can be characterized by its distinct "data model" from traditional relational databases [13]. A data model, a set of conceptual tools to represent and manage data, consists of three components [22]: i) data structure types, ii) query operators, and iii) integrity rules. Data in a graph database are stored and represented as graphs, or data structures generalizing the notion of graph. Each graph database has its own specialized graph query language, since Structured Query Language (SQL) is inadequate for this type of data. For example, Neo4J uses Cypher language, and many RDF (Resource Description Framework) databases use SPARQL (SPARQL Protocol and RDF Query Language). Finally, integrity rules in a graph database are based on its graph constraints, rather than from an imposed relational schema.

With increasing complexity of real-world data and growing needs for graph queries, numerous graph databases have been proposed and developed in recent years. There exists a clear trade-off between data size and data complexity in graph databases. Moreover, each graph database is designed for special application needs. For example, Neo4J is suited for online transaction processing (OLTP), while Pregel is designed for high latency, high throughput platforms. In this paper, we primarily use Neo4J to explore the graph structure in healthcare systems. Neo4J is a widely used open-source graph database, implemented in Java. Neo4J is characterized as an "embedded, disk-based, and fully transactional graph database engine". We decided to use Neo4J because of its popularity and well-documented APIs. We mainly focus on general graph processing capabilities of Neo4J, rather than its specialized functions compared to other graph engines. Experimental evaluations of other graph engines are also practically important, and we leave this as future work.

III. 3EG: 3NF Equivalent Graph Transform

Compared to well established RDBMS design principles, a design rationale for a graph database is not well grounded and often *ad hoc*. In this paper, we do not plan to design a graph database from scratch, and focus rather on the fact that most database systems are already in 3NF. We presume the existence of a 3NF relational database, and propose simple conversion rules from 3NF to a graph database.

Converting a relational model to another data model has been an important research topic in both software engineering and database system communities. In software engineering, a semantic structure of a relational database is used to redesign its schema and reduce its maintenance cost. Entity-Relationship (ER) model [23] has been a popular choice to represent such semantic structure. Several automatic algorithms to convert a relational model to an ER model, also known as "database reverse engineering"[1], have been introduced in [24], [25], [26]. Migrating a relational database to a completely different class of database became a natural extension of this database reverse engineering. Various database migration approaches can be found in [27], [28], [29]. These approaches include migrations between 1) ER model and relational model, 2) Object-Oriented (OO) model and Relational model, and 3) OO model and ER model. However, converting a relational database to a graph database has not been explored so far. Researchers mostly need to manually examine the semantic structure of the original relational database, and carefully port the data to a graph database [30].

Transforming a relational model to a graph model is similar to converting a relational model to an ER model. To construct rules as simple as possible, we focus on the representation in a graph database rather than the underlying semantic structure (ER model). Moreover, we mainly focus on 2-way relationships between nodes (entities), and all the relationships are bi-directional in further discussions. The existence of keys in 1NF intuitively maps to the existence of nodes in a graph database. Relationships in a graph database can be analogous to those in a relational database. Properties of nodes can be specified by the requirements in 3NF. The requirements of 3NF table are often concisely described as "Every non-key attribute must provide a fact about the key, the whole key, and nothing but the key" [31]. In other words, non-key elements describes about the key (the node in a graph), and this description is only to the key and nothing but the key (a unique connection to the node).

We formalize our idea of transforming relational tables to a graph database. Consider a table $\mathcal{T} = \{(x, y)\}$, and a graph $\mathcal{G} = \{(n, r) | n \in \mathcal{N}, r \in \mathcal{R}\}$ where $\mathcal{N}$ and $\mathcal{R}$ are sets of nodes and relationships, respectively. Suppose x is a primary key in $\mathcal{T}$. 3NF Equivalent Graph (3EG) transformation is a process of applying the following four rules:

1) Each tuple $t \in \mathcal{T}$ becomes a node $n \in \mathcal{N}$. Each node n is identified by its table name and primary key: $\text{id}(n) =$

[1]Conversion from an ER model to a relational schema is called as "database forward engineering".

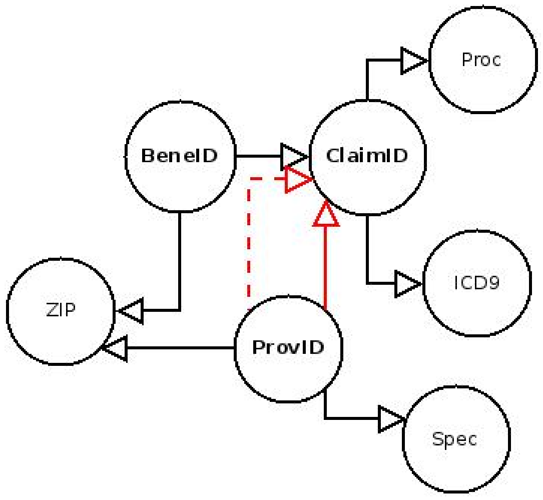

Figure 3: 3NF Equivalant Graph Design.

$\{\text{name}(\mathcal{T}), x\}$.

2) If y is a foreign key, a relationship $r \in \mathcal{R}$ is created between n and m. In this case, $\text{id}(m) = \{\text{name}(\mathcal{T}_y), y\}$ and $\mathcal{T}_y$ is a table where y is its primary key.
3) If y is a non-key element, y becomes a property of a node n where $\text{id}(n) = \{\text{name}(\mathcal{T}), x\}$.
4) For a set of keys in t, each pairwise relationship maps to an edge between the corresponding vertices.

Each rule has its origin as follows:

1) Rule 1 specifies 1NF.
2) Rule 3 is based on the 3NF requirement: about the key, the whole key, and nothing but the key.
3) Rule 2 and 4 follow from the definition of "relational databases".

Figure 3 shows a 3EG-transformed graph schema for the design in Figure 1. Each primary key in a table is converted to a node. For example, a BeneID in Bene table now maps to a Bene node. A foreign key in a table is connected to a primary key: a ProvID is connected to a ClaimID based on C2P table. Non-key elements in ZIP, Spec, ICD9, and Proc tables are properties of ZIP, Spec, ICD9, and Proc nodes, respectively.

3EG-transform provides a simple set of rules. This set of rules can be instantaneously applied to any 3NF tables. By applying these rules, we obtain a mirrored graph database of the existing relational database. This transformation is lossless, and reverse transformation is also possible. We note that the generated graph may not be the optimal design for a specific service purpose. The generated graph only guarantees that the representation of the data is equivalent with the original 3NF database.

IV. Synthetic Data Generation

In this paper, we generate realistic synthetic healthcare data, and evaluate the performance of two databases using them. Our synthetic data are firstly generated to be persistent to the design in Figure 1, then 3EG-transformed to be loaded on a graph database.

Table I: Synthetic datasets.

Dataset	#. Bene	#. Prov	#. Claims	#. Nodes	#. Rel.
1	8K	800	400K	425K	2M
2	16K	1.6K	800K	824K	4M
3	31K	3.1K	1.5M	1.62M	8M
4	63K	6.3K	3.1M	3.21M	16M
5	125K	12.5K	6M	7M	33M
6	250K	25K	12M	13M	65M
7	500K	50K	25M	26M	129M
8	1M	100K	50M	51M	257M

We first generate ZIP, Spec, ICD9, and Proc tables. These tables are based on the actual codes used in practice. For example, we use the whole list of real ICD-9 codes, and create our ICD9 table. Next, we synthesize Bene and Prov tables. The number of beneficiaries and providers are given as parameters to our synthesizer, and the corresponding number of BeneID's and ProvID's are generated. ZIP codes and Specialty codes (only for providers) are randomly assigned to BeneID's and ProvID's. We assume that each beneficiary has 50 number of claims on average. Based on the number of beneficiaries, we generate 50 times larger number of claims, and assign ClaimID's. Each ClaimID is associated with random ICD-9, Procedure codes, and one beneficiary (C2B table). A beneficiary is randomly chosen from the already synthesized beneficiary pool. Choosing a beneficiary from millions of beneficiaries, however, needs a bit of attention. To avoid the computational complexity, we adopt a tree structured multinomial sampling. We first sample a group of beneficiaries, then select a beneficiary from the group (two level samplings). Every ClaimID has at least one provider, and it can have five providers at maximum, and these associations are recorded in C2P table. With 10% chance, a claim has a referral from another provider, and this information is captured in C2R table. Finally, these relational tables are 3EG-transformed, and we obtain the equivalent graph representation in Figure 3.

Table I describes eight synthetic datasets used in our experiments. The number of nodes and relationships correspond to the number of beneficiaries and providers when the relational tables are 3EG-transformed. For example, dataset 8 is generated by specifying one million beneficiaries and 100 thousand providers. After generating the synthetic relational tables, those tables are 3EG-transformed resulting in total 51 million nodes and 257 million relationships. Figure 4 shows the actual Neo4J console when 51 million nodes and 257 million relationships are loaded.

V. Test Cases

In this section, we collect and explain eight graph query use cases.

A. Queries in Healthcare Systems

In healthcare systems, mining and identifying relationships between different entities are critical in many analytic procedures. These relationships involve beneficiaries, claims, providers, and sometimes locations, diseases, and procedures. Figure 5 illustrates such relationships between beneficiaries,

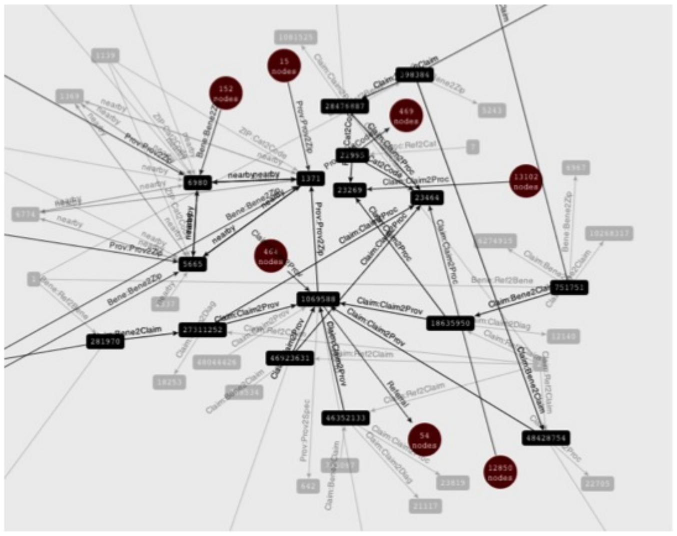

Figure 4: A screen shot of Neo4J console when 51 million nodes are loaded.

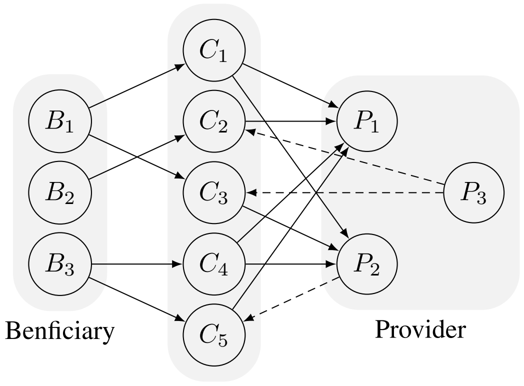

Figure 5: A Snap Shot of Graph Representation. Other types of nodes are not shown to unclutter the diagram.

claims, and providers. Careful investigation on these relationships often leads to early detection on organized frauds. Furthermore, sophisticated personalization and data mining algorithms also can be implemented on these use cases.

Queries to a healthcare database can be very diverse, ranging from web search queries to forensic queries investigating frauds and abuse. In this paper, we mainly focus on two groups of queries regarding: fraud and abuse investigations and personalized web services. Healthcare fraud cases typically involve three parties: providers, beneficiaries, and insurance providers. In [32], Li et al. shows that 69% of the literature involves mainly providers, and the rest 31% accounts for beneficiaries. However, the healthcare parties are very deeply entangled in many cases [33], and many legislative changes, such as the Patient Protection and Affordable Care Act (2010), are further increasing the complexity between the parties. Furthermore, in recent years, several organized Medicare frauds have been reported, involving more than 50 people. Personalized web services, on the other hand, have been one of the key challenges in healthcare informatics as described in [34]. Recent studies, however, indicate that the current medical vocabularies can be barrier for normal users to utilize the services [35]. Hierarchical queries on the specific medical terminologies are also found to be the most frequent queries in a clinical database [36].

Our eight queries are designed based on these recent findings. We extensively explore three themes in healthcare queries, namely 1) the relationships between beneficiaries and providers, 2) loosely specified semantic constraints, and 3) a personalized recommendation system. Note that these eight queries are formulated to justify the usefulness of graph queries in healthcare systems. Other types of queries, such as non-graph queries or more complex graph queries, can be candidates for our performance evaluation, but we only focus on these eight queries to meet our objectives.

B. Case 1: Shared Provider

Our first query example is to find out shared providers between two beneficiaries. Figure 6 illustrates the idea. Given two beneficiaries B_a and B_b, the query should return a list of providers $\{P_x\}$ shared by two beneficiaries.

Using the schema in Figure 1, the corresponding MySQL query is as follows:

```
SELECT tableA.provID FROM (
  SELECT provID FROM C2P INNER JOIN (
    SELECT claimID FROM C2B WHERE beneID=25869
  ) AS tmp ON tmp.claimID=C2P.claimID
) AS tableA INNER JOIN (
  SELECT provID FROM clpr INNER JOIN (
    SELECT claimID FROM C2B WHERE beneID=751751
  ) AS tmp ON tmp.claimID=C2P.claimID
) AS tableB ON tableA.provID=tableB.provID;
```

On the other hand, the same Neo4J Cypher query is below:

```
START beneA=node(25869), beneB=node(751751)
MATCH beneA-->()<--prov-->()<--beneB
RETURN prov;
```

We observe that two queries are quite different in their query lengths. The query constraints appear in "where" and "join" clauses in the MySQL query, and "match" clause in the Neo4J Cypher query. The match clause directly represents the diagram in Figure 6, while join clauses need more consideration to understand properly. The benefit of graph database comes from not only its direct data representation and storage, but also its intuitive queries. Graph queries are more compactly and easily represented in graph databases than in relational databases.

C. Case 2: Loosely Specified Relationship

Our second query relaxes the condition in the first query. In this case, the links between claims and providers are not limited to the C2P table. We ask shared providers between two beneficiaries either by actual visits or by referrals. Figure 7 describes this use case. To ask this query in MySQL, we separately ask three different combinations of relationships: C2P+C2P, C2P+C2R, and C2R+C2R. Each result should be

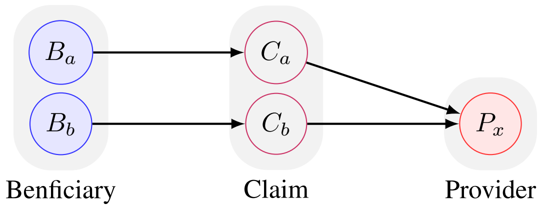

Figure 6: Case 1: A list of shared providers by Bene A and Bene B.

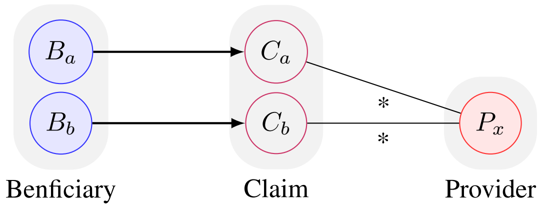

Figure 7: Case 2, 3, and 4: A list of shared providers by Bene A and Bene B.

stored and later union operation should be taken to prepare the final results. In Neo4J, on the other hand, we only need to modify the first query to include the referral relationship between claims and providers.

D. Case 3: Shared Disease

This query is a slight modification of the first query. We want to find a list of shared diseases between two beneficiaries by looking up their claim records. The provider node in Figure 7 becomes an ICD9 node in this example.

E. Case 4: Any Link between Two Entities

In this example, we find any type of links between two beneficiaries. This link can be a shared provider, shared disease code, or shared procedure code. The provider node in Figure 7 is now unspecified, and can be any of these nodes.

F. Case 5: Shared Beneficiary

This example asks the exact same type of Case 1 question from the providers' side. We want to find shared beneficiaries between two providers. Although this may look repeating Case 1, its internal data operation is not quite the same. The synthetic data in our experiments have different number of beneficiaries and providers. The number of beneficiaries are ten times larger than the number of providers in all datasets. This difference results in the difference in node degrees between beneficiaries and providers. Provider nodes typically would have ten times higher node degrees than beneficiary nodes.

G. Case 6: Self Referral

Given a provider, we find self-referred claims in this example. Figure 8 shows this self-referral in claim records. Self-referrals have been serious problems in both private and public healthcare sectors. For example, it is reported that doctors' self-referral cost Medicare more than 100 million dollars [37].

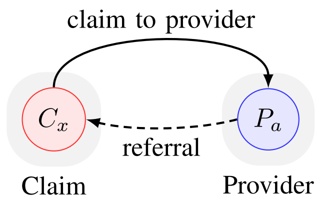

Figure 8: Case 4: Self Referrals For Provider A

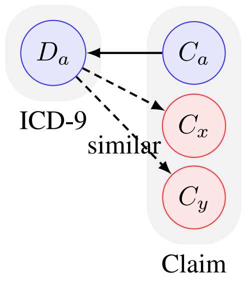

Figure 9: Similar Claims based on Diagnosis.

The self-referral diagram shown in Figure 8 would be the simplest form of such malicious activities.

H. Case 7: Similar Record

In this use case, we find similar claims based on their diagnosis codes. Even this simple looking query might consume enormous time when tables are decomposed into many and each table has huge number of records. For example, dataset 8 in our experiments has nearly 50 million claim records. Just scanning across all the claim records can be extremely hard in this case.

I. Case 8: Collaborative Filtering

In this example, we design a simple collaborative filtering application using a graph database. Suppose a situation that a patient experiences a new symptom D_a, and wants to find a new doctor. The patient, however, wants to find a trusted or referred doctor P_x by his favorite doctor P_a. Figure 10 illustrates this process. This recommendation system utilizes accumulated referral history of other claim records. This type of collaborative filtering can enhance the quality in healthcare systems, and also can provide personalized services.

VI. Experimental Evaluation

In this section, we provide experimental results based on our eight use cases. Our experiment platform is a 2.7 GHz Intel

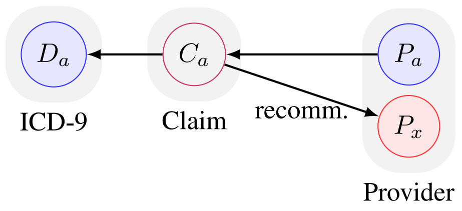

Figure 10: Collaborative Filtering using Referral History.

Core i5 machine with 4 GB 1333 MHz DDR2 Random Access Memory. We use 5.5.28 MySQL Community Server for a relational database, and 1.9.M02 Neo4J Community Server. To access MySQL, we use MySQLdb Python interface [2]. For Neo4J, we use its REST API to access its data. Specifically, we use py2neo Python interface[3], and Cypher query language is used. All the results are measured after enough cache warm up for both databases. We generate 50 random queries for each case, and measure their query execution times. Test cases are sequentially executed from Case 1 to Case 8.

Figure 11 and 12 show the query performance measured when 50 million nodes and 250 million relationships are loaded (dataset 8). To see whether random queries are sequentially dependent or not, we plot their execution time over the sequence of queries in Figure 11. Except MySQL's case 2 performance, the query performance is stable across the query sequence. For Case 2, we observe that MySQL internally optimizes the Case 2 query as it gets the same type of queries. Except the first three use cases, Neo4J outperforms MySQL in its execution time. Figure 12 shows the same results from a different angle. At this time, we try to see relative query execution times between cases. As can be seen, Case 7 and 8 are especially slow in MySQL, while Neo4J shows a stable performance over different queries.

Figure 13 and 14 illustrate the changes in query performance across a set of different datasets. All the datasets in Table I appear in both figures. Figure 13 shows the query scalability by case. MySQL outperforms for the first three queries in all datasets. For the rest of queries, MySQL fails to scale to large data. As can be seen, the performance of MySQL resembles a quadratic curve. Figure 14 shows the results from a different grouping. We compare the query performance within each database. We can observe that Neo4J also shows a quadratic performance degradation as the size of data becomes large. However, Neo4J's speed of degradation is much smaller than MySQL's speed.

The query performance of Neo4J degrades with respect to the degree of a node, while the performance of MySQL is related to the size of joining tables. This finding becomes apparent in Figure 14. In our synthetic data generation scheme, the beneficiaries and providers have the same average degree numbers across different datasets.

$$E[\text{degree}(B)] = 50 + \alpha \text{ and } E[\text{degree}(P)] = 500 + \beta$$

where α and β indicate the extra relationships such as disease codes and ZIP codes. Therefore, the queries involving only beneficiaries and providers show almost the same performance regardless of the data sizes. Unlike other cases, disease nodes are newly introduced in Case 7 and 8. The total number of diseases are fixed in the ICD-9 definition, thus the degree of a disease node becomes a function of the number of claims:

$$E[\text{degree}(D)] = \frac{|\mathcal{P}|}{|\mathcal{D}|} \approx \frac{|\mathcal{P}|}{15000}$$

[2]http://mysql-python.sourceforge.net/MySQLdb.html
[3]http://py2neo.org/

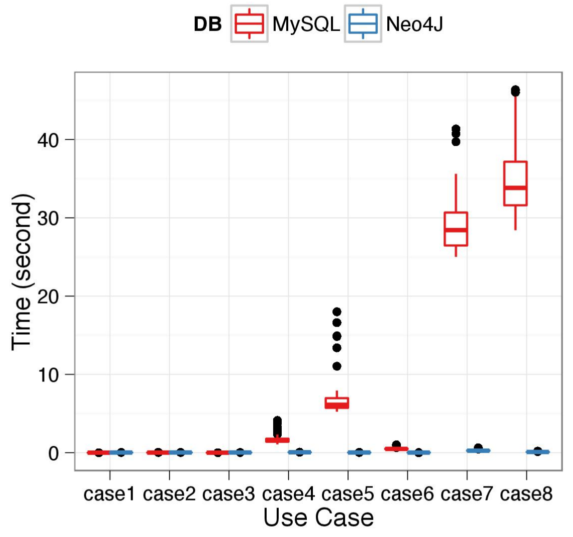

Figure 12: Performance box plot when 50 million nodes and 250 million relationships are loaded. MySQL becomes significantly slower especially in Case 7 and 8.

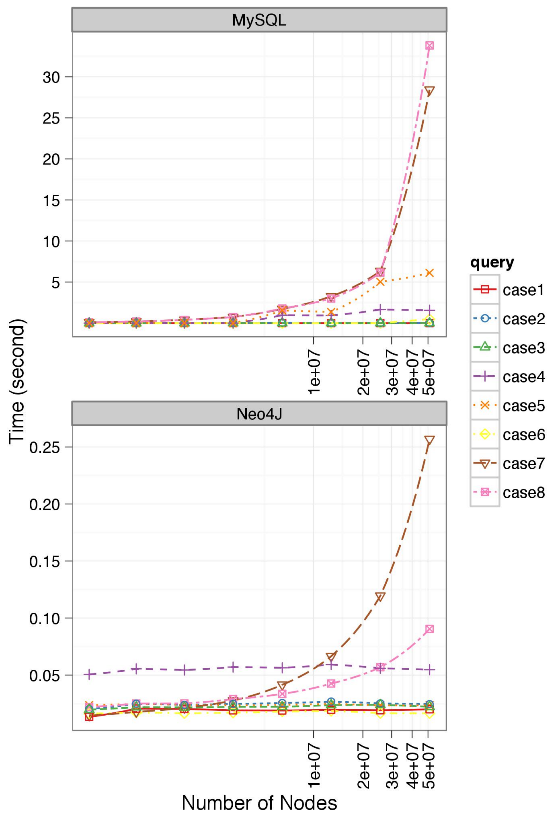

Figure 14: Data Size vs. Query Processing Time. Neo4J's performance also degrades as the size of the data grows. However, its degradation speed is much slower than MySQL's degradation speed.

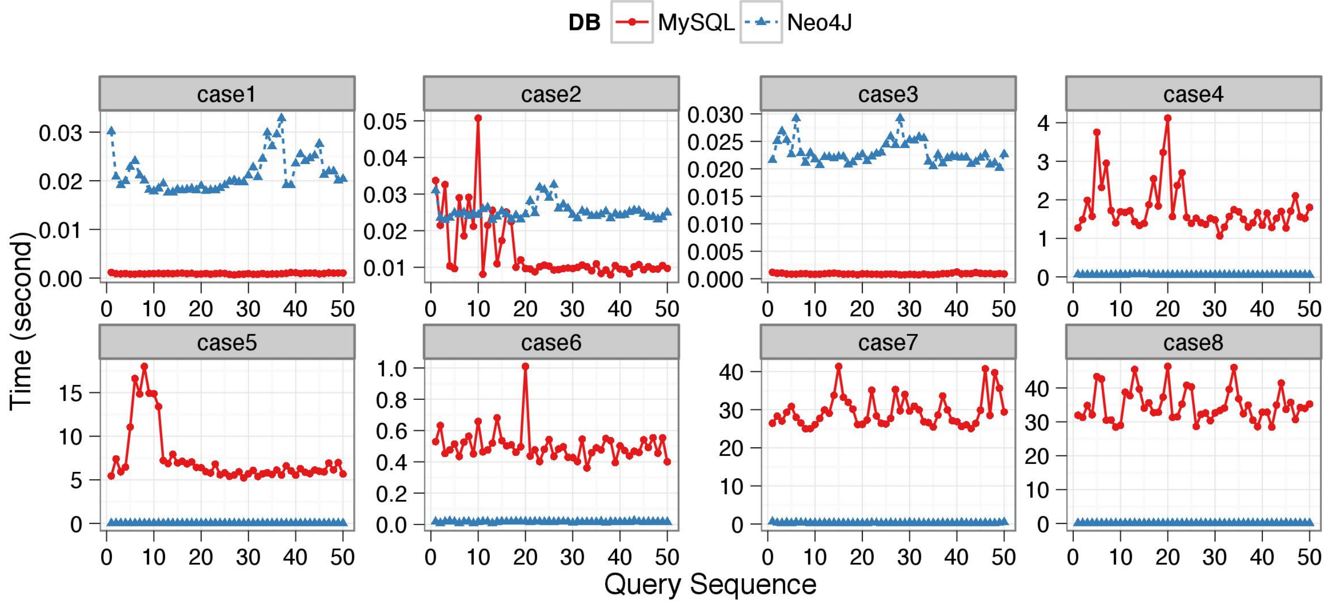

Figure 11: Sequential query performance when 50 million nodes and 250 million relationships are loaded. Except first three cases, Neo4J processes the rest of queries faster than MySQL.

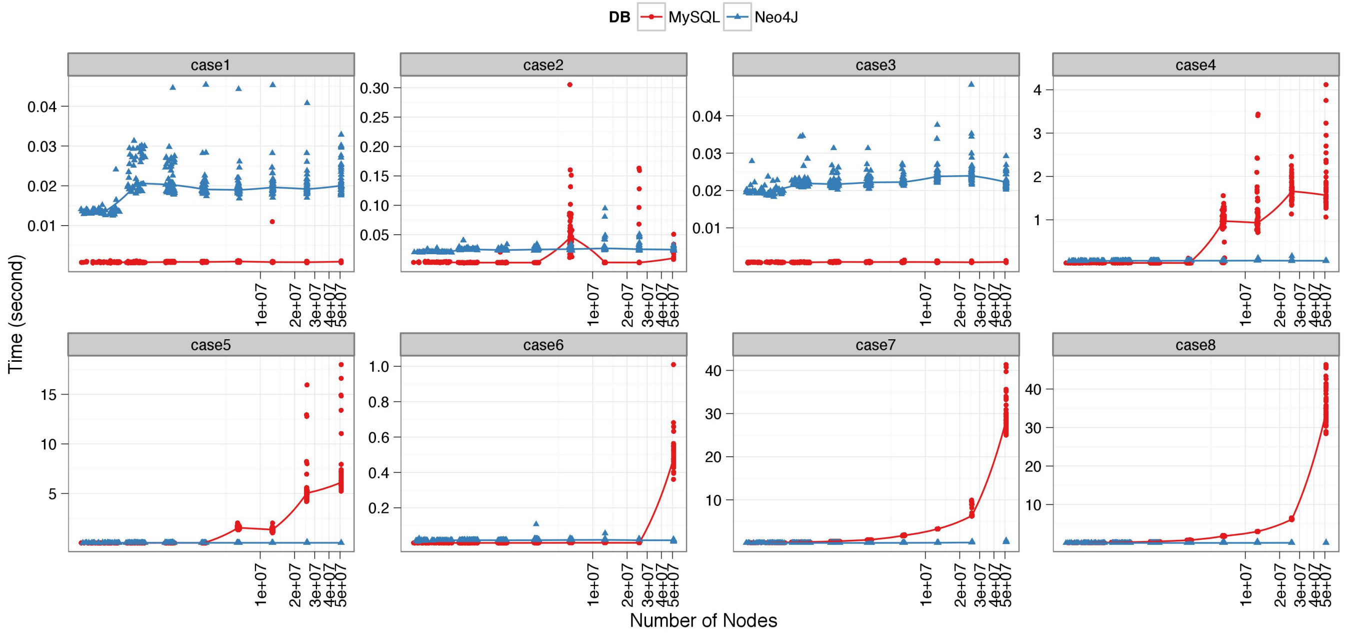

Figure 13: Data Size vs. Query Processing Time. Neo4J shows almost constant performance over a set of different sized datasets, while MySQL does not scale well in many use cases.

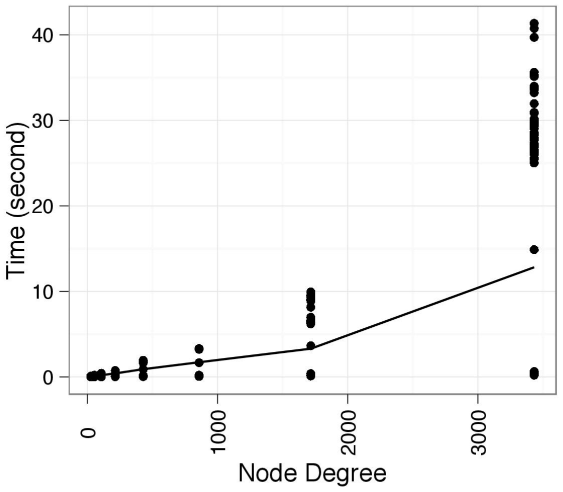

Figure 15: Node Degree vs. Query Processing Time (Case 7).

where $|\mathcal{P}|$ and $|\mathcal{D}|$ represent the total number of providers and diseases, respectively. The total number of disease codes are specified in the ICD-9 codebook, which is approximately 15K. Figure 15 illustrates the relationship between node degree and query processing time in Case 7. This result suggests that a graph database can scale to very large scale healthcare data, if their 3EG-transformed graph maintains a certain level of node degrees.

VII. Discussion and Conclusion

In this section, we propose a new method of utilizing graph databases in healthcare systems.

A. Graph Database As A De-normalized Table

The experimental results in Section VI suggests a graph database can handle a wide range of graph queries even with big data. In a relational database, these graph queries result in many dummy tables, and require heavy join operations. On the other hand, a graph database, suited for this type of online query processing, can replace such dummy table creation. Generating a graph database using the 3EG-transformation may be able to serve as hundreds of different de-normalized tables.

B. Ensemble Framework

Compared to rich history and verified stability of relational databases, the graph database technology may be relatively new and not yet fully confirmed from various applications. The use cases in this paper heavily focuses on graph queries, not the other types of queries. Basic database operations such as select, project, union, and set difference are extensively optimized in many relational databases.

Each database model has its own strength and weakness. We therefore pick the strength of each database model, and propose a cost efficient ensemble framework in healthcare systems. Figure 16 illustrates our basic idea. A graph database is mirrored using the 3EG transform. The access layer guides

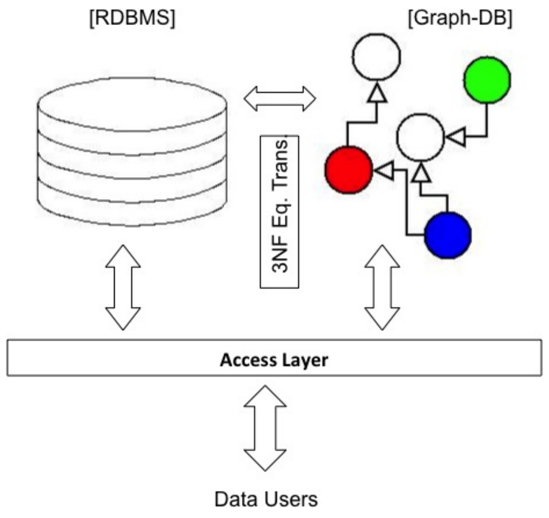

Figure 16: Cost Efficient Ensemble Framework. A graph database is mirrored using the 3EG transform. Access layer mediates data users to access both RDBMS and graph databases depending on their objectives.

data users to access both RDBMS and graph databases depending on their objectives. Many existing healthcare enterprise data trusts are built on many layers of interdependent databases [38]. This graph database component can be an additional component to such enterprise architecture. Furthermore, this graph module can be placed in a decentralized data warehousing environment as well [39]. We believe such framework substantially can help both data management and utilization in healthcare system.

In this paper, we proposed a novel graph database design rationale, "3NF Equivalent Graph" (3EG) transform, which can automatically construct a graph database from an existing normalized relational database. We generated realistic and large scale synthetic healthcare data to simulate the current US healthcare systems. Our eight graph queries are evaluated over two representative databases: MySQL for a relational database and Neo4J for a graph database. Based on the promising results from the experiments, we finally proposed a blueprint for a cost efficient database architecture in healthcare systems. Actual implementation of such system may have undetected difficulties, and may cause other systemic issues. Other types of graph databases, even further other types of NoSQL databases, also can participate in this framework. Practical evaluation of these systems would eventually lead to our ultimate end "Best Care at Lower Cost", and we leave these as future work.

Acknowledgment

References

[1] C. Meier, "A role for data: An observation on empowering stakeholders," *American Journal of Preventive Medicine*, 2013.

[2] *World Health Statistics*. World Health Organization, 2011.

[3] S. Keehan, A. Sisko, C. Truffer, S. Smith, C. Cowan, J. Poisal, M. K. Clemens, and N. H. E. A. P. Team, "Health Spending Projections Through 2017: The Baby-Boom Generation Is Coming To Medicare," *Health Affairs*, 2008.

[4] I. of Medicine, "Transformation of health system needed to improve care and reduce costs," Press Release, 9 2012.

[5] M. Smith, R. Saunders, L. Stuckhardt, and J. M. McGinnis, *Best Care at Lower Cost: The Path to Continuously Learning Health Care in America*, C. on the Learning Health Care System in America (Institute of Medicine), Ed. The National Academies Press, 2012. [Online]. Available: http://www.nap.edu/openbook.php?record_id=13444

[6] J. DeVoe, "The Unsustainable US Health Care System: A Blueprint for Change," *Annals of Family Medicine*, 2008.

[7] D. M. Cutler, K. Davis, and K. Stremikis, "The Impact of Health Reform on Health System Spending," *Commonwealth Fund Issue Brief*, 2010.

[8] D. M. Berwick and A. D. Hackbarth, "Eliminating Waste in US Health Care," *The Journal of the American Medical Association*, 2012.

[9] C. J. Date, *An Introduction to Database Systems*. Addison-Wesley, 2000.

[10] A. Silberschatz, H. Korth, and S. Sudarshan, *Database System Concepts*. McGraw-Hill, 2002.

[11] Z. Wei, J. Dejun, G. Pierre, C.-H. Chi, and M. van Steen, "Service-Oriented Data Denormalization for Scalable Web Applications," in *Proceedings of the 17th International World Wide Web Conference*, 2008.

[12] G. L. Sanders and S. Shin, "Denormalization Effects on Performance of RDBMS," in *Proceedings of the 34th Hawaii International Conference on System Sciences*, 2001.

[13] R. Angles and C. Gutierrez, "Survey of Graph Database Models," *ACM Computing Surveys*, vol. 40, no. 1, 2008.

[14] A. Khan, Y. Wu, and X. Yan, "Emerging graph queries in linked data," in *Proceedings of IEEE 28th International Conference on Data Engineering*, 2012.

[15] B. Shao, H. Wang, and Y. Xia, "Managing and mining large graphs: Systems and implementations," in *Proceedings of ACM SIGMOD 2012*, 2012.

[16] "Neo4J." [Online]. Available: http://www.neo4j.org/

[17] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A System for Large-Scale Graph Processing," in *Proceedings of ACM SIGMOD 2010*, 2010.

[18] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, 1970.

[19] ——, "Further normalization of the data base relational model," *IBM Research Report RJ 909*, 1971.

[20] R. Fagin, "Multivalued dependencies and a new normal form for relational databases," *ACM Transactions on Database Systems*, vol. 2, no. 1, 1977.

[21] ——, "Normal forms and relational database operators," in *Proceedings of ACM SIGMOD 1979*, 1979.

[22] E. F. Codd, "Data Models in Database Management," in *Proceedings of the 1980 workshop on Data abstraction, databases and conceptual modeling*, 1980.

[23] P. P. shan Chen, "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems*, vol. 1, pp. 9–36, 1976.

[24] R. H. L. Chiang, T. M. Barron, and V. C. Storey, "Reverse engineering of relational databases: Extraction of an EER model from a relational database," *Data and Knowledge Engineering*, vol. 12, pp. 107–142, 1994.

[25] V. M. Markowitz and J. A. Makowsky, "Identifying extended entity-relationship object structures in relational schemas," *Transactions on Software Engineering*, vol. 16, no. 8, 1990.

[26] W. J. Premerlani and M. R. Blaha, "An Approach for Reverse Engineeing of Relational Databases," *Communications of the ACM*, vol. 37, 1994.

[27] C. Fahrner and G. Vossen, "A survey of database design tranformations based on the entity-relationship model," *Data and Knowledge Engineering*, vol. 15, 1995.

[28] A. Maatuk, A. Ali, and N. Rossiter, "Relational database migration: A perspective," *Database and Expert Systems Applications*, vol. 5181, 2008.

[29] D. Lee, M. Mani, and W. W. Chu, "Effective Schema Conversions between XML and Relational Models," in *Proceedings of European Conference on Artificial Intelligence (ECAI)*, 2002.

[30] [Online]. Available: http://blog.neo4j.org/2012/02/webinar-follow-up-how-to-get-started.html

[31] W. Kent, "A Simple Guide to Five Normal Forms in Relational Database Theory," *Communications of the ACM*, vol. 26, no. 2, 1983.

[32] J. Li, K.-Y. Huang, J. Jin, and J. Shi, "A survey on statistical methods for health care fraud detection," *Health Care Management Science*, 2008.

[33] C. B. Garner and D. McCabe, "The Evolving Relationships Between Hospital, Physician and Patient in Modern American Healthcare," *Health, Culture, and Society*, vol. 3, no. 1, 2012.

[34] H. U. Prokosch and T. Ganslandt, "Perspectives for Medical Informatics," *Methods of Information in Medicine*, 2009.

[35] D. P. Lorence and A. Spink, "Semantics and the medical web: a review of barriers and breakthrough s in effective healthcare query," *Health Information and Libraries Journal*, vol. 21, pp. 109–116, 2004.

[36] S. N. Murphy, M. M. Morgan, and H. C. Chueh, "Optimizing Healthcare Research Data Warehouse Design through Past COSTAR Query Analysis," *AMIA*, 1999.

[37] United States Government Accountability Office, "Higher Use of Advanced Imaging Services by Providers Who Self-Refer Costing Medicare Millions." [Online]. Available: http://www.gao.gov/assets/650/648989.pdf

[38] C. G. Chute, S. A. Beck, T. B. Fisk, and D. N. Mohr, "The enterprise data trust at mayo clinic: a semantically integrated warehouse of biomedical data," *JAMIA*, 2013.

[39] S. Hanß, T. Schaaf, T. Wetzel, C. Hahn, T. Schrader, and T. Tolxdorff, "Integration of Decentralized Clinical Data in a Data Warehouse," *Methods of Information in Medicine*, 2009.