# A Comparison of Two Database Partitioning Approaches that Support Taxonomy-Based Query Answering

Jero Mario Schäfer
Institute of Computer Science
University of Göttingen
Göttingen, Germany
jeromario.schaefer@stud.uni-goettingen.de

Lena Wiese
Institute of Computer Science
Goethe University Frankfurt
Frankfurt, Germany
lwiese@cs.uni-frankfurt.de

## ABSTRACT

In this paper we address the topic of identification of cohorts of similar patients in a database of electronic health records. We follow the conjecture that retrieval of similar patients can be supported by an underlying distributed database design. Hence we propose a fragmentation based on partitioning the health records and present a benchmark of two implementation variants in comparison to an off-the-shelf data distribution approach provided by Apache Ignite. While our main use case in this paper is cohort identification, our approach has advantages for taxonomy-based query answering in other (non-medical) domains.

## CCS CONCEPTS

• **Information systems** → *Query optimization*; *Relational parallel and distributed DBMSs.*

## KEYWORDS

Distributed database system, relational databases, taxonomy-based query answering

## 1 INTRODUCTION

A typical application scenario that can benefit from a distributed database system (DDBS) is the management of medical "big data" collected from different data providers to be used in healthcare for example for predictions or research [11]. In particular, novel biomedical technology – for example, next-generation sequencing [27] – produces vast amounts of data. In this article, our application scenario is a *medical information system* that uses a distributed database system as a storage backend. In our example system, patients' personal information as well as the diseases they suffer from are contained in a distributed database. A cooperation of hospitals could maintain this database as a joint information system. Access to the patient data can be provided to doctors and nurses that need to work with these data from different locations. The medical information system can then act as a Clinical Decision Support System (CDSS).

*Example 1.1.* We consider a medical information system that consists of three database tables (one for the diagnosed illnesses, one for the examinations done for the anamnesis and one for administrative information):

| Ill | ID | Diagnosis |
|---|---|---|
| | 1 | Corneal Perf |
| | 2 | Corneal Perf |
| | 2 | Asthenopia |
| | 2 | Ranula |
| | 3 | Asthenopia |
| | 4 | Breast Cyst |

| Anamnesis | ID | Examination |
|---|---|---|
| | 1 | Seidel test |
| | 2 | Seidel test |
| | 2 | Ultrasound |
| | 3 | Eye Examination |
| | 4 | Biopsy |

| Info | ID | Name | Address | Age |
|---|---|---|---|---|
| | 1 | Miller | Brisbane | 41 |
| | 2 | Smith | Sydney | 48 |
| | 3 | Brown | Canberra | 22 |
| | 4 | Jones | Perth | 53 |

Our intended medical information system is supposed to support medical staff in identifying relevant data from these three tables in an efficient way.

Our main aim is to provide the advanced functionality of *intelligent query answering* for such a medical information system; this query answering functionality is defined by a notion of similarity between terms (in our case, diagnoses specified by disease descriptors) that can be obtained from an underlying taxonomy of terms. Similarity-based query answering has several purposes in the medical domain. For example, a notion of patient similarity is the major precondition to obtain reliable cohorts for clinical studies. To achieve the goal of similarity-based query answering in an efficient way, we enhance the basic distributed data management with *two* different implementations of database fragmentation. In our application, a clustering procedure is applied to the disease information of patients; we assume that the disease terms conform to the vocabulary provided by the *Medical Subject Headings* taxonomy (MeSH) of the U.S. National Library of Medicine [23].

*Example 1.2.* In our example, the clustering will be executed on disease terms such that we are able to partition the table *Ill* into two fragments: one for eye diseases (tuples containing *Corneal Perforation* and *Asthenopia*) and one for neoplasms (tuples containing *Ranula* and *Breast Cyst*). Similarity-based query answering enables us to retrieve answers that are similar to a query condition: when

L.W. is also affiliated to Research Group Bioinformatics, Fraunhofer Institute for Toxikology and Experimental Medicine (ITEM), Hannover, Germany.


*iiWAS '20, November 30-December 2, 2020, Chiang Mai, Thailand*

ACM ISBN 978-1-4503-8922-8/20/11…$15.00
https://doi.org/10.1145/3428757.3429108

for example asking for patients having *Keratitis*, the user will retrieve information of patients having eye diseases; however, the user will not retrieve information of patients having neoplasms.

Our specific contribution in this article is that we comparatively analyze performance properties of standard out-of-the-box distributed behavior with two different implementation variants of intelligent clustering-based query answering in a widely used SQL database system. The remainder of this article is organized as follows. Section 2 surveys related work of medical data integration and flexible query answering. Section 3 provides the necessary theoretical background on DDBSs. Section 4 gives an in-depth description of our proposed intelligent information system. Section 5 describes the implementation details while Section 6 comparatively evaluates the three implementation variants. Section 7 concludes the article.

## 2 RELATED WORK

### 2.1 Distributed Databases and Query Rewriting

In this article, we assume a classical relational database as the data model underlying our medical information system: a database instance consists of a set of relations (or tables) each ranging over a set of attributes (or columns) and each relation contains a set of tuples (or rows). Relational database systems are still the most widely used data model. In particular, our use case (identification of cohorts of similar patients) lies in the medical domain. In this area many data models for health records still follow the relational paradigm – for example, the i2b2 system [15] is built on a relational database system.

Query decomposition [14] denotes the process of rewriting the query to a normalized form, followed by a semantical analysis, a simplification of the query and a rewriting and restructuring to a relational algebra query. These four steps provide as their result an algebraic query that can be optimized. These techniques are dependent on the type of fragmentation, i.e. whether it is a (primary/derived) horizontal or vertical fragmentation or even hybrid fragmentation, and they are applied as rules in order to make use of the possible simplifications and improvements regarding the query. Other authors [10, 20] survey methods of distributed query processing including the requirements of different strategies that allow for transforming a query stated against the database, e.g. written in SQL, into an execution plan consisting of relational algebra instructions for the database which yield the correct result set for the query in an optimal way. In our system we apply two rewriting techniques to support the intended similarity-based query answering.

### 2.2 Flexible Query Answering and Similarity Search

A problem with exact database query answering is that the user does not always retrieve all the relevant answers that might interest him or her. Failing queries, that have an empty result set, are only desirable for a user in the fewest cases as an empty result cannot provide any information except the information that this query could not be answered exactly under the current database state. A failing query forces the user to check whether the query contains some logical error itself leading to no exact answers, or whether the current database state does actually not provide any matching data. Moreover, even if there are some records returned as the database answer, there might be more information contained in the database that is highly *similar* to the query intention. In other words, the database would probably be able to give an answer that has an informational content that is similar to an exact answer or – in case of a failing query – is non-empty. Therefore, the user could already be satisfied with a similar answer as this similarity can already enable the user to gain information and draw conclusions based on this similar answer.

Flexible query answering exists in different data models. For example, [6] analyze flexible query answering by rewriting according to a description logic language. On the other hand, the MSQL system [12] addresses similarity search in metric spaces. [22] present similarity query operators for SQL. [29] address semantic similarity search in knowledge graphs. As opposed to these approaches, we combine similarity search based on a taxonomy with an underlying distributed database design supporting derived fragmentations. An approach similar to ours is [16] using the SNOMED disease ontology, yet their focus is more on data integration in a polystore and not on data fragmentation.

### 2.3 Clinical Decision Support

Our aim is to implement an intelligent query answering system that is backed by a distributed database system – and hence can improve effectiveness of Clinical Decision Support Systems (CDSSs) by providing an efficient technology for cohort identification. In other words, it can be used to identify a cohort of patients [7] that are similar to a current "target patient". Historical data of similar patients in this cohort are valuable information in order to assess health conditions as well as decide on further treatment of the target patient. CDSSs have widely been conjectured to identify optimal treatments when considered by experienced medical staff as an extra source of information [13] – in addition to their personal professional expertise. Moreover, another possible usage scenario is that researchers make use of these patient data when investigating the co-occurrences of several groups of similar diseases.

For clinical decision making, the deployment flexible query answering that is based on some semantic guidance – which in our use case corresponds to the clustering based on the similarity defined for diseases represented by MeSH terms – can be tremendously helpful; for example, when considering the search for similar patient profiles for decision making regarding the treatment options and for predictions for the patient's future (cf. the oncology use case in [9]). The data of identified similar patients can be used to compare the current situation of the patient as well as probabilities of success or failure of treatments. The higher the similarity is to other patients, the higher are the chances for an optimal treatment of the specific disease – especially early detection and methods of prevention could be enabled. Yet, the specific definition of similarity plays a crucial role: as soon as the similarity decreases beyond some threshold, the informational content that can be derived will not be as useful anymore. Moreover, the reliable identification of similar cases may require more sophisticated similarity measures in order to identify helpful profiles and information that allows for ranking similar cases based on the similarity measures [9]. Hence,

the parameters of similarity for clinical decision making must be adaptable to the setting under investigation. Our system offers this behavior by providing a configurable threshold for the similarity.

Identifying a cohort of patients is an important task for secondary usage of Electronic Health Records (EHR) data for personalized medicine, clinical research or result quality improvement [4, 17, 19]. For example, in [28] a similarity-based modeling on patient data that matches individuals to subjects with similar conditions was proposed and implemented for the diagnosis and prognosis of Alzheimer's disease.

## 3 BACKGROUND

The relations (that is, tables) of a DDBS can be partitioned into fragments – in other words, the data from the tables are split into subsets. These fragments are then assigned ("allocated") to one or more of the database instances belonging to the DDBS. We will use the terms *partitioning* and *fragmentation* interchangably: while on the theoretical level commonly the term fragmentation is used, at the level of database system implementations often the term partitioning is preferred (like the Apache Ignite system that we use in our implementation).

*Horizontal fragmentation* divides a relation in a row-wise manner into smaller subsets of tuples (i.e. table rows) – in contrast to vertical fragmentation that splits a relation into subsets of table columns. Horizontal fragmentation splits a relation $R$ into fragments $F_1, F_2, \ldots, F_n$ by assigning each tuple $\mu$ of the relation $R$ to at least one fragment $F_i,\ i \in \{1, \ldots, n\}$. The result of this is that for all $i \in \{1, \ldots, n\}$, the subset relation $F_i \subseteq R$ holds. Additionally, in order to avoid redundancy of data, we can require that each tuple is only assigned to exactly one fragment; more formally, the fragments are pairwise disjoint: $\forall \mu \in F_i$ it holds that $\mu \notin F_j, i \neq j$ for $i, j \in \{1, \ldots, n\}$. In relational algebra such a *primary horizontal fragmentation* can be described by a selection operation $\sigma$ on the relation $R$, where the selection condition defines the desired mapping of tuples to fragments. Based on this primary horizontal fragmentation, a further fragmentation of another relation $S$ can be *derived* by computing the semi-join (denoted as $\ltimes$) of the relation $S$ with fragments $F_i,\ i \in \{1, \ldots, n\}$ of the primary relation $R$, i.e. the derived fragments $G_i$ of the relation $S$ are computed as $G_i = S \ltimes F_i$ for $i \in \{1, \ldots, n\}$. The *derived horizontal fragmentation* depends on the underlying primary horizontal fragmentation, and, to prevent tuples in $S$ from getting lost during the semi-join with fragments of $R$, it is necessary to have for each tuple $y \in S$ matching tuples in $R$ in order to let the tuples from $S$ "survive" the semi-join. An integrity constraint in form of a foreign key reference of the relation $S$ to the relation $R$ can be used to enforce this condition for the sake of completeness of the derived fragmentation. Inherently with the definition of the derived horizontal fragmentation on a semi-join, redundancy of tuples of $S$ in the derived fragments may occur if tuples in $S$ match multiple tuples that belong to different fragments of $R$. This causes the fragments $G_i$ of $S$ to be non-disjoint in general. However we will make use of this property to ensure data locality of primary and derived fragments: while we require the primary fragmentation to be disjoint (and hence non-redundant) the derived fragmentation might contain fragments that have some tuples in common. These derived fragments are however stored on different sites together with their matching primary fragment.

## 4 SYSTEM DESIGN AND ARCHITECTURE

This section presents the methods underlying the intelligent information system in terms of data storage and fragmentation.

### 4.1 Data Storage

Our setting is based on several in-memory Apache Ignite instances that are spread across a set of interconnected servers that form a so-called *cluster*. Each database instance (that is, database "node") is run in a Java Virtual Machine (JVM) hosted by a different server. Yet, technically there can be multiple database nodes hosted on the same server, too; even multiple nodes running in the same virtual machine are possible. With the underlying Client-Server architecture, a node can be either a server node, which is responsible for storing and processing data, or a client node that is able to connect remotely to the cluster of server nodes and process a query from the client side. Servers can be accessed remotely from clients by connecting to them via a native programming language API (e.g. Java), via a JDBC or ODBC connection or via a REST API. We used the advanced functionality of the native API for setting up the fragmentation and inserting data; while we used JDBC for running the benchmark queries because it is more lightweight and straightforward. The Ignite server nodes store data depending on the fragmentation (called partitioning with Ignite) and replication. The fragmentation and replication is defined per relation. In the partitioned mode a relation is partitioned and each server is then responsible for only a subset of the data. It is also possible to have partitioning and replication at the same time: the partitions of the data are stored on more than one server such that each partition of the data are stored on one server as primary partition and on the other servers as backup copies for higher availability and data resiliency in case of server node failures. In replicated mode, a balancing of the data load is achieved by partitioning the data of the relation with hash functions into subsets of nearly equal size that are dispersed equally among all server nodes in the network to exploit as much memory as possible on all nodes.

By default Apache Ignite provides automatic partitioning and distribution of the data; the DDBS strives to equalize the load on all participating nodes yielding a balanced data distribution. Yet, this equalized approach does not support similarity search. We compare this basic Ignite setting with our developed approaches that support cohort identification by clustering-induced data distribution.

### 4.2 Ignite's Partitioning Settings

One important concept of Ignite is the collocation of data – which corresponds to the concept of a derived horizontal fragmentation: data that are accessed together, e.g. because they are joined via a common attribute or a foreign key reference, are also stored together on the same server; this ensures data locality and allows for collocated distributed joins (between the primary and a derived fragment) that benefit from data locality because the costly data transfer between server nodes across the network is avoided.

Ignite offers the database administrator a method to influence and change the assignment of a tuple to a partition. This affinity

collocation can be defined in Ignite by so-called *affinity keys*: An affinity key can be identical to the primary key of a relation or an attribute of a composite primary key of a relation. Ignite ensures that all tuples where the affinity keys match are stored on the same server. The usage is restricted to a single affinity key definition per relation. With this restriction, there cannot be a collocation of three or more relations that could be joined via a chain of join conditions where the join attributes would form possible affinity keys of the different relations. If all the joined relations in the SQL query are collocated, the query can be evaluated locally by each node: all the data they need to compute a correct result set regarding their portion of the whole data in the cluster are available locally, i.e. stored by themselves. The concept of affinity collocation strongly corresponds to the primary and derived horizontal fragmentation of relations, i.e. the derived fragmentation also ensures that tuples from the derived horizontal fragments are located at the same server where also the tuples reside with which they can and will be joined in SQL queries. Joins between partitioned relations require for collocation of the data that are about to be joined. Otherwise, the result set will be incomplete as the non-collocated parts of the data cannot be joined locally by the nodes. The query execution for the collocated join case is that a query $Q$ is sent from the client to each server node which executes the query $Q$ locally and returns the result set $R_i$ to the query according to its stored data set. The final result set is the union of the result sets $R_i$ from each of the nodes. This is the default Ignite behavior.

On the other hand, non-collocated joins require for additional communication and data transfer between the nodes of the cluster as the data that are needed for a join is not locally present. Thus, the nodes have to send requests for the data to other nodes to complete their result set computations appropriately. The data request of a node can be sent as a unicast request to a certain node of the cluster (peer-to-peer communication) or as a broadcast request to all other nodes. This decision depends on whether the requesting node can identify the exact location of the data, i.e. if it is a join on a primary or affinity key, or if it has to request the missing data from all other nodes otherwise. This additional data transfer implies a bad efficiency of query execution as it depends on the transmission duration of probably bigger data sets between the nodes. Furthermore, this causes additional load on the network that can decrease the performance of other tasks or operations. Non-collocated joins must be enabled explicitly in Ignite to enforce the distributed answering with data transfer across the network if necessary. If not enabled explicitly, the query will be executed in a collocated manner which, in general, leads to incomplete result sets due to required data not being available locally.

In our implementation – in order to avoid network communication overhead – we ensure appropriate collocation of the data and make use of collocated joins for better query execution efficiency.

## 4.3 Clustering-based Fragmentation

In contrast to Ignite's hash-based horizontal partitioning, our proposed clustering-based fragmentation is computed in a semantic way: the horizontal fragments are induced by a previously calculated clustering that provides a semantic guidance for the assignment of the tuples to the horizontal fragments. More precisely, the clustering-based fragmentation is a horizontal fragmentation strategy that, on the one hand, enables fragmentation regarding a similarity metric which allows for a more semantic partitioning of the data set, and, on the other hand, supports the *similarity-based query answering*. The underlying clustering is computed with an approximation algorithm [8] on all values that occur in the *active domain* of a chosen attribute of the relation defined as follows.

*Definition 4.1.* Given an attribute $A$ of a relation instance $R$, the *active domain* of the attribute $A$ is the projection $\pi_A(R)$ of $R$ to the values of $A$.

On the active domain we aim to execute the clustering procedure. In order to do this, we need a similarity relationship $sim(a, b)$ for pairs of elements $a, b$ from the active domain $\pi_A(R)$ – more formally, $sim : \pi_A(R) \times \pi_A(R) \rightarrow \mathbb{R}$. It is customary to restrict the range of the similarity to the interval $[0, 1]$ such that a similarity of 1 denotes that the elements $a$ and $b$ have highest similarity, whereas the closer the similarity value gets to 0, the more dissimilar the two elements are. A clustering (in terms of a partitioning of the active domain into disjunctive subsets) can be obtained by evaluating the similarities and assigning the elements to the clusters. An appropriate criterion for determining how "similar" two elements have to be to belong to the same cluster is required. We rely on a *head* element (also called centroid) which is an element from the cluster that represents the cluster. In addition, we define a threshold parameter $\alpha$. This similarity threshold $\alpha$ which is used in the computation of the clustering of the active domain of the relaxation attribute (Disease); it allows to configure how big the similarity of a term to the head of a cluster has to be at least such that they will be assigned to the same cluster. By choosing an appropriate threshold, it is assured that no two diseases, represented by terms, that have no significant similarity at all belong to the same cluster and are subsequently seen as similar diseases.

Under this restriction, for any element $a$ of a cluster $c$, the corresponding head element of this cluster, $head \in c$, and a given similarity threshold $\alpha$ it holds that $sim(a, head) \geq \alpha$.

The pseudocode of the clustering procedure is described in Listing 1. The clustering starts with a single cluster (Line 1) containing the whole active domain and an arbitrarily chosen head element from the cluster and then identifies the minimal similarity inside the cluster between all elements from the active domain and the cluster head (Line 6). Subsequently, new clusters are created with new head elements based on the minimal similarity of a term to the head of a cluster as long as the similarity threshold is not exceeded; all elements are reassigned if they are more similar to the head of the newly created cluster (Line 7). The procedure iterates as long as there are still elements inside one of the clusters that have a similarity to the corresponding head element that is lower than the similarity threshold $\alpha$ (while-condition in Line 5). Hence, the iteration proceeds until each element of the active domain is clustered such that the minimal similarity according to the threshold $\alpha$ from Definition 4.4 can be ensured. The obtained clustering is finally returned and used later to induce a horizontal fragmentation (cf. Definition 4.4). The following example shows how a clustering is computed for a given set of disease terms and the pairwise similarities of the diseases.

**Listing 1** Clustering procedure

**Input:** Set $\pi_A(F)$ of values for attribute $A$, similarity threshold $\alpha$
**Output:** A set of clusters $c_1, \ldots, c_f$

Let $c_1 = \pi_A(F)$
Choose arbitrary $head_1 \in c_1$
$sim_{min} = min\{sim(a, head_1) \mid a \in c_1; a \neq head_1\}$
$i = 1$
**while** $sim_{min} < \alpha$ **do**
    Choose $head_{i+1} \in \{b \mid b \in c_j; b \neq head_j; sim(b, head_j) = sim_{min}; 1 \leq j \leq i\}$
    $c_{i+1}$=$\{head_{i+1}\} \cup \{c \mid c \in c_j; c \neq head_j; sim(c, head_j) \leq sim(c, head_{i+1}); 1 \leq j \leq i\}$
    $i = i + 1$
    $sim_{min} = min\{sim(d, head_j) \mid d \in c_j; d \neq head_j; 1 \leq j \leq i\}$
**end while**

*Example 4.2.* Consider the sample disease term set

{*Asthenopia*, *Corneal Perforation*, *Ranula*, *Breast Cyst*}

and the corresponding pairwise similarities of the diseases like shown in Table 1 in order to illustrate the functionality of the clustering procedure. All eye diseases are similar to one another; the same holds for the neoplasms – but neoplasms are less similar to eye diseases, and vice versa.

| | Asthenopia | Corneal Perf | Ranula | Breast Cyst |
|---|---|---|---|---|
| Asthenopia | 1 | 0.2 | 0.167 | 0.167 |
| Corneal Perf | 0.2 | 1 | 0.125 | 0.125 |
| Ranula | 0.167 | 0.125 | 1 | 0.333 |
| Breast Cyst | 0.167 | 0.125 | 0.333 | 1 |

**Table 1: Pairwise similarities of some diseases**

Let $\alpha = 0.2$. The clustering procedure starts by creating the first cluster $c_1$ that is represented by an arbitrarily chosen head, e.g. the first disease of the set ($head_1 = Asthenopia$), and moves all remaining terms to the cluster's subset. In the next step, the minimal similarity of any disease term to the cluster head, $sim_{min} = \min_{a \in c_1} sim(a, head_1)$, is calculated and compared to the similarity threshold. As this minimal similarity still is below the threshold, $sim_{min} = 0.167 < 0.2 = \alpha$, the identified minimizing disease term is chosen to be the head element of the next cluster, i.e. $\arg\min_{a \in c_1} sim(a, head_1) = Ranula = head_2 \in c_2$. After this step, all diseases of cluster $c_1$ that are more similar to the neoplasm than to Asthenopia will be reassigned to the cluster $c_2$. The clustering computation is complete because the minimal intracluster similarities for all clusters are above the threshold:

$$\forall c_i, i \in \{1, 2\} : \min_{a \in c_i} sim(a, head_i) \geq \alpha$$

The resulting clustering is $\{c_1, c_2\}$ where $c_2 = \{Ranula, Breast\ Cyst\}$ and $c_1 = \{Asthenopia, Corneal\ Perforation\}$.

By varying $\alpha$, the clustering can be influenced: if the similarity threshold is below the minimal pairwise similarity of any two diseases, e.g. $\alpha = 0.12$, then the computation stops after the initial step before the first iteration and the resulting clustering consists of only one cluster. If in contrast $\alpha > 0.2$ (but $< 0.333$), there would be one cluster with the two neoplasms and an own cluster for each of the eye diseases. For our further evaluations the chosen value for $\alpha$ was fixed for the different term subsets between 0.1 and 0.2.

The different clusters contained in the clustering can be used to obtain a horizontal fragmentation of a relation instance $R$. Each cluster induces one horizontal fragment: any two tuples are mapped to the same fragment if their values for clustering attribute $A$ belong to the same cluster.

*Example 4.3.* Based on the clustering, we obtain two fragments of table *Ill*.

| Ill_0 | ID | Diagnosis |
|---|---|---|
| | 1 | Corneal Perf |
| | 2 | Corneal Perf |
| | 2 | Asthenopia |
| | 3 | Asthenopia |

| Ill_1 | ID | Diagnosis |
|---|---|---|
| | 2 | Ranula |
| | 4 | Breast Cyst |

The clustering-based fragmentation [25] can be formally defined as follows:

*Definition 4.4.* Given the active domain $\pi_A(R)$ regarding an attribute $A$ of a relation instance $R$, and a complete clustering $C = \{c_i | i = 1, ..., n\}$ of $\pi_A(R)$ with threshold $\alpha$ and head elements $head_i \in c_i$ for the clustering. Then, $F = \{F_1, ..., F_n\}$ is a (horizontal) *clustering-based fragmentation* of $R$ if

- every horizontal fragment $F_i$ corresponds to one cluster $c_i \in C$ such that $c_i = \pi_A(F_i)$
- the threshold $\alpha$ is respected in every cluster such that $\forall i \in \{1, ..., n\} : \forall a \in c_i : sim(a, head_i) \geq \alpha$
- the clustering is complete and hence for every tuple $t$ in $R$ there is an $F_i$ in which $t$ is contained
- the original instance can be reconstructed as $R = F_1 \cup \ldots \cup F_n$
- fragments are non-redundant such that for any $i \neq j$, $F_i \cap F_j = \emptyset$ (or in other words $c_i \cap c_j = \emptyset$)

The clustering-based fragmentation is a primary horizontal fragmentation; as already discussed in Section 3, it is possible and efficient to derive a horizontal fragmentation for a second relation from it. Hence we achieve the usual collocation of the data by allocating tuples of the second relation to the corresponding derived horizontal fragments by considering common join attributes.

*Example 4.5.* We have two fragments of *Ill*, hence we obtain two derived fragments of *Anamnesis* and *Info* as well based on the join attribute ID:

| Ill_0 | ID | Diagnosis |
|---|---|---|
| | 1 | Corneal Perf |
| | 2 | Corneal Perf |
| | 2 | Asthenopia |
| | 3 | Asthenopia |

| Anamn_0 | ID | Examination |
|---|---|---|
| | 1 | Seidel test |
| | 2 | Seidel test |
| | 3 | Eye Examin |
| | 2 | Ultrasound |

| Info_0 | ID | Name | Address | Age |
|---|---|---|---|---|
| | 1 | Miller | Brisbane | 41 |
| | 2 | Smith | Sydney | 48 |
| | 3 | Brown | Canberra | 22 |

| Ill_1 | ID | Diagnosis |
|---|---|---|
| | 2 | Ranula |
| | 4 | Breast Cyst |

| Anamn_1 | ID | Examination |
|---|---|---|
| | 2 | Seidel test |
| | 2 | Ultrasound |
| | 4 | Biopsy |

| Info_1 | ID | Name | Address | Age |
|---|---|---|---|---|
| | 2 | Smith | Sydney | 48 |
| | 4 | Jones | Perth | 53 |

As the data are distributed according to the clustering into horizontal fragments, similarity-based queries can now be executed in a distributed manner according to the clustering. Finding the relevant data fragment is done based on the selection condition on the attribute chosen for the clustering: for each term in the selection condition the term's similarity to the head elements of the different clusters is obtained. The maximal similarity of the comparison element and all cluster heads determines the matching cluster as well as the induced horizontal fragment. More formally, we can define a similarity-based answer for each selection (sub-)query on attribute $A$ as follows.

*Definition 4.6.* For a clustering-based fragmentation of relation $R$, and a selection query $\sigma_{A=\text{“}s\text{”}}R$ on the clustering attribute $A$ the similarity-based answer is

$$\{F_i \mid head_i = argmax_{j=1,\ldots,n} sim(s, head_j)\}.$$

If there is more than one cluster head with maximal similarity, we choose one of them at random. In this case it is sufficient to execute the query only locally at a single server which hosts the single relevant data fragment. An alternative to this approach would be to return the union of all fragments with most similar head elements.

*Example 4.7.* Assuming that $sim(Asthenopia, Keratitis) = 0.25$ being the highest similarity of all cluster heads to *Keratitis*, we can answer all queries having an appropriate selection condition on the Disease attribute (that is, *Disease=Keratitis*) in a similarity-based way by returning the fragment `Ill_0` – potentially joined with the collocated derived fragments.

## 5 IMPLEMENTATION

To evaluate the clustering-based fragmentation, we comparatively benchmark three implementation alternatives: (1) Basic Ignite, (2) Materialized Fragments and (3) Partition Number as described in the following subsections. The source code is available in a Github repository[1].

### 5.1 Similarity Calculation and Clustering

As the underlying input to our approach, we have to provide the pairwise similarities, $sim(a, b)$, for two terms $a$, $b$ of the MeSH taxonomy. Our implementation is based on the path length measure. Depending on the use case, other semantic similarities can be used as well as other similarities or distance measures that can be applied to numeric data [21].

*Definition 5.1.* The path length measure is calculated as

$$\frac{1}{length(path(a, b))}$$

[1] https://github.com/l-wiese/SiFAMIS

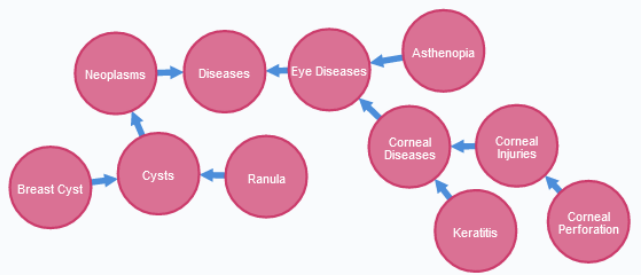


**Figure 1: A snippet of MeSH in Neo4J**

where $length(path(a, b))$ denotes the number of all *nodes* on the shortest path between $a$ and $b$.

The clustering, which is required to induce our fragmentation strategy as described in Section 4.3, was implemented in Java. We apply it to obtain appropriate clusters of the MeSH disease terms based on pairwise similarities.

In order to determine the path length between the terms found in MeSH [23], we imported the taxonomy into the graph database *Neo4j*. We used the MeSH taxonomy in the provided RDF N-Triples format. The RDF triples had to be transformed into the property graph model as implemented by *Neo4j* (see Figure 1 for a snippet). To this end, the *neosemantics* plugin was utilized [3]. The RDF triples were hence transformed in the following manner. Each subject of a triple is modeled as a node. Any predicate attached to the subject is only modeled as a relationship if the attached object again is a resource itself (and hence is modeled as a node). If in contrast the object is a literal then the predicate is mapped to the corresponding subject node as a property with the object literal as the property value. Once this mapping has been established, we can calculate the similarity between two disease nodes by finding the shortest path between their respective *Topical Descriptor* nodes using the provided relationships. Naturally, not all relationships qualify as valid paths and we therefore only included those of the type *treeNumber*, *parentTreeNumber*, and *broaderConcept*. We apply a corresponding query in the *Neo4J* query language Cypher using the shortest path algorithm provided by *Neo4J*. In our benchmark in Section 6 we tested a term set of varying size up to covering the whole MeSH taxonomy. We randomly choose a subset of the respective term set for our benchmark dataset.

### 5.2 Basic Ignite

This basic implementation of the medical information system is achieved by creating partitioned tables by the default (similarity-agnostic) Ignite partitioning methodology. The primary table (in our case, `ILL`) is partitioned horizontally based on a hash function applied to its affinity key (that is, attribute `ID`); the data are partitioned and distributed in an arbitrary way without respecting similarity of disease terms. The other tables are collocated via their shared attribute, the patient ID, such that personal information and the diseases and anamnesis of a patient are stored together. That is, only collocation via the attribute `ID` is guaranteed to gain some efficiency as costly data transfer between the nodes (servers) of the cluster can be avoided if the data to be joined are available locally –

but no similar disease terms are collocated because this approach is implemented without the clustering-based fragmentation.

### 5.3 Materialized Fragments

The materialized approach stores the horizontal fragments of the primary table (as induced by the precomputed clustering of the disease terms based on similarity) as separate relations in the database – with different table names for the different fragments. This implies that fragments are now separate tables and not longer belong together syntactically from the DDBS's point of view. To achieve the distribution of the horizontal fragments to the servers, a modified affinity function has to be explicitly provided and made available on all nodes. This affinity function handles the mapping of the materialized fragments to the servers; it also provides the similarity function for two disease terms that is required for both the clustering and the query rewriting and answering. Furthermore, the collocation of table fragments via a derived horizontal fragmentation is achieved by storing materialized fragments of the other tables on the same server. For example, personal information of patients (*Info*-tuples) at each server may be joined with information about a disease that this patient has; that is, for each materialized *Ill* fragment there is also a materialized *Info* fragment present for appropriate collocation. Note that the materialization of the horizontal fragments as separate tables is transparent for the user: the clustering and the rewriting of queries is done automatically by our system.

### 5.4 Partition Number

The partitions approach also materializes the fragments. However, in contrast to the Materialized Fragments approach, the fragmentation is invisible from the SQL perspective: instead of using different relation names, we assign different partition numbers to the different fragments of the relations via the affinity function API. The partitions are then distributed and mapped to the servers. By doing so, the queries do not require to be rewritten as in the Materialized Fragments approach because the query processing is almost completely done by Ignite's query processor; this is because – instead of having different relation names for each fragment – the different fragments are distinguished by their partition numbers. Additionally, the fragmentation is again fully transparent for the user. Collocation of derived partitions is also achieved with the help of the affinity function and the identification of the correct partition number that is inferred from the clustering.

### 5.5 Query Rewriting

Based on term similarity, intelligent query answering by relaxing the selection condition on the appropriate attribute can be performed as query rewriting for all three implementations. Hence, the developed approaches are able to rewrite given SQL queries such that they match the underlying clustering-based fragmentation.

- The Basic Ignite implementation substitutes the selection condition of the clustering attribute (in our case, a disease term) with a *set* of constant symbols (other *similar* disease terms). The original SQL query is translated into one with a SQL `IN`-clause containing the similar disease terms from the appropriate cluster.
- In case of the Materialized Fragments approach where the fragments of the relations are materialized as separate SQL tables, it is required to rewrite queries according to that materialization – that is, all the table names in the queries have to be adapted to the corresponding fragment. If there is a selection condition on the appropriate attribute in the query, the single relevant fragment is determined and is substituted for the table name. If there is no such selection condition, a UNION query has to be executed in order to answer the query correctly by considering all existing fragments.
- In case of the Partition Number approach where the fragmentation is done by assigning a partition number to each fragment and mapping each partition to one of the servers, substituting the table names is not required because the partitions are known to Ignite's query processor already and the query contains the correct relation names. Nevertheless, selection conditions on the chosen attribute are answered in a similarity-based manner. Our implementation is capable of executing queries only on the server that hosts the corresponding partition by identifying the most similar cluster first and then using Ignite's SQL API in order to set the partition number for the query according to the identified fragment.

In summary, the first approach needs a substitution of the selection condition with a more general expression corresponding to the addressed cluster. In the latter two approaches, the selection condition is simply omitted and the query is adapted by restricting it to the fragment that belongs to the cluster with the relevant diseases, such that then all answers only need to be obtained from this fragment. Thus, for the materialized fragment approach only the relation names have to be changed to the corresponding fragment – whereas for the partition number approach only the corresponding partition number has to be set via the SQL API. Furthermore, too many irrelevant answers (so-called overgeneralization [25]) are avoided as the attribute values of the obtained answers are restricted to terms of the same cluster providing a semantically close result set instead of a result set that also includes too general answers.

## 6 BENCHMARK

### 6.1 Data Set

We generated a synthetic data set to comparatively investigate the behavior of our two implementation variants with the basic Ignite behavior. We analyzed scalability of our approach by varying both the data set size and the size of the term set (that is, active domain of the clustering attribute) on which similarities are calculated. Our test data set is modelled according to the three tables in our running example. The three data set tables are obtained as follows. The *Ill* table is our primary table; the patient IDs are generated in increasing order; the diagnosis attribute contains disease terms extracted randomly from the MeSH data set. The Anamnesis table contains the patient IDs and randomly generated string data as the examination. The Info table contains one random address string for each patient as well as a randomly generated age.

| | |
|---|---|
| $Q_1$ | `SELECT p.name, p.age, p.address FROM ILL i, INFO p WHERE i.id = p.id AND i.disease='Asthenopia'` |
| $Q_2$ | `SELECT p.name, p.age, p.address FROM ILL i1, ILL i2, INFO p WHERE i1.id = p.id AND i2.id = p.id AND i1.disease='Asthenopia' AND i2.disease='Ranula'` |
| $Q_3$ | `SELECT a.examination FROM ILL i,ANAMNESIS a WHERE a.id = i.id AND i.disease = 'Asthenopia'` |
| $Q_4$ | `SELECT p.name, p.age, a.examination FROM ILL i,ANAMNESIS a, INFO p WHERE a.id = i.id AND i.id = p.id AND i.disease = 'Asthenopia'` |
| $Q_5$ | `SELECT a.examination FROM ILL i, ILL i2, ANAMNESIS a WHERE i.id = i2.id AND i.id = a.id AND i.disease = 'Asthenopia' AND i2.disease = 'Corneal Perforation'` |

**Table 2: Benchmark queries (disease terms are chosen randomly)**

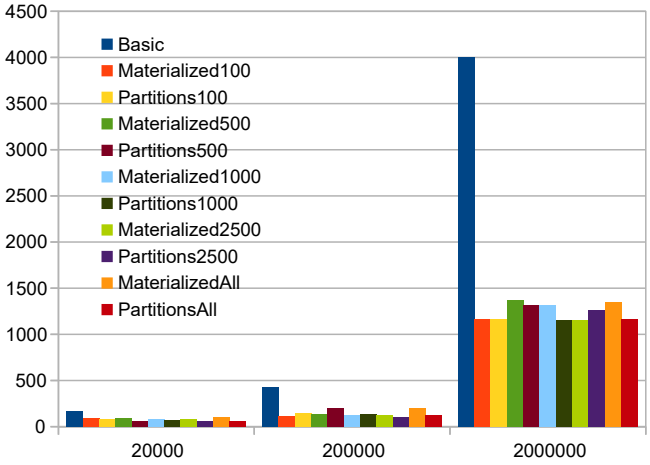


**Figure 2: Average query execution time in milliseconds (y-axis) of $Q_1$ when scaling the amount of tuples in the database (x-axis) and the amount of disease terms (color bars).**

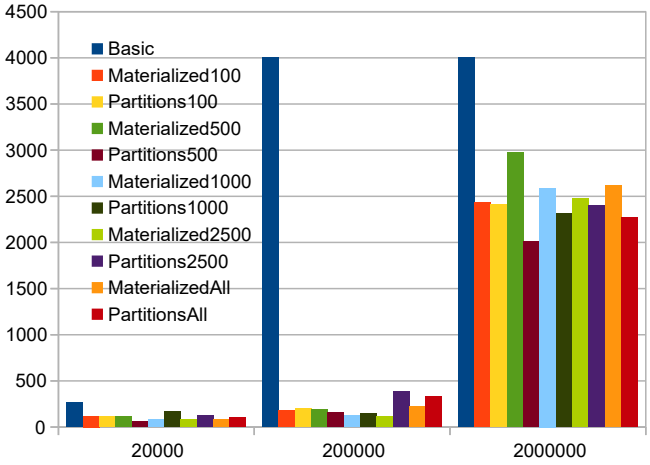


**Figure 3: Average query execution time in milliseconds (y-axis) of $Q_2$ when scaling the amount of tuples in the database (x-axis) and the amount of disease terms (color bars)**

Scaling of the data set was obtained by a default number of tuples for each of the tables multiplied by a scaling factor. The default size of the Ill table is 100 tuples and the default size of both Anamnesis and Info table is 50 tuples – that is, an average of two disease entries per person plus one sample examination. For a given scaling factor $s$ the dataset is expanded to a total of $s \cdot (100 + 50 + 50) = 200 \cdot s$ tuples. We vary $s$ from 100 over 1000 up to 10000, so that we have datasets with 20000, 200000 and 2000000 tuples each. The MeSH term set was divided into smaller, randomly chosen subsets ranging from a minimum of 100 terms, over 500, 1000 and 2500 up to all 4798 terms from which the Diagnosis column of the *Ill* table is filled. The similarity threshold of the clustering was chosen manually to achieve a relatively balanced clustering in terms of the number and size of clusters. Values below 0.1 and above 0.2 produce imbalanced clusterings with one huge cluster for all terms or one separate cluster per term. Thus, the similarity threshold was set to 0.12 for all term sets resulting in 15 clusters with 7 terms in each of them (on average) for 100 terms, 7 clusters with 71 terms in each of them for 500 terms, 13 clusters with 76 terms per cluster for 1000 terms, 22 clusters with 113 terms per cluster for 2500 terms and 16 clusters with 299 terms per cluster for all 4798 terms.

## 6.2 Queries

Recall that *Ill* is our primary table. While in the first Basic Ignite approach data allocation is up to hashing of the patient id, in the two other approaches data allocation is based on the clustering executed on the diagnosis attribute. In all three cases derived fragments of the Anamnesis and Info tables are allocated together with the matching primary fragments. Our benchmark queries $Q_1$ to $Q_5$ (see Table 2) consist of executing joins (between primary and secondary fragments) with a selection condition on the diagnosis attribute. Selection conditions are randomly chosen from the term set selected from the underlying taxonomy; selection conditions need not be contained in the randomly generated data set. Similarity-based query execution includes finding the fragment that is closest to the selection condition (in terms of similarity to the cluster head that is contained in the diagnosis attribute of the fragment). In other words, query execution extracts the selection condition, applies the query rewriting described in Section 5.5 and returns the obtained fragment (or joins of fragments, respectively) as the set of related answers. Note that queries Q2 and Q5 both have two selection conditions on which similarity-based query answering is applied. The difference between the two queries is that in Q2 both selection conditions are not in the same cluster and data have to be retrieved from different fragments before joining them; whereas in Q5 the selection conditions come from the same fragment (and the tuples are hence collocated).

## 6.3 Results

Our benchmark is evaluated in a network of three Apache Ignite nodes where each of the nodes runs in a JVM and is hosted by one of three servers. Each server has 4 processors and is equipped with 8GB of memory. We tested the basic approach (independent

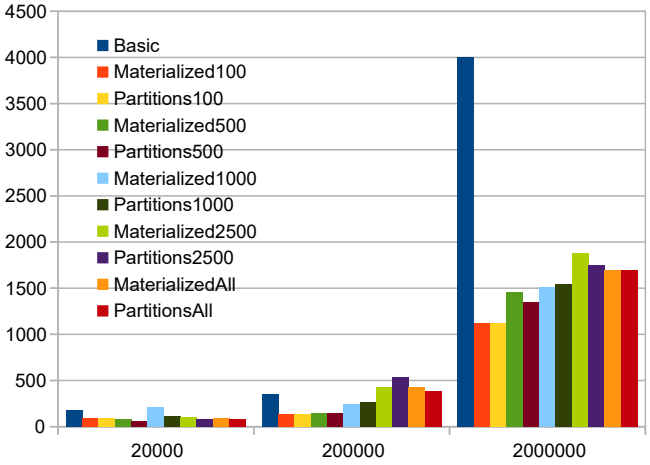


**Figure 4: Average query execution time in milliseconds (y-axis) of $Q_3$ when scaling the amount of tuples in the database (x-axis) and the amount of disease terms (color bars)**

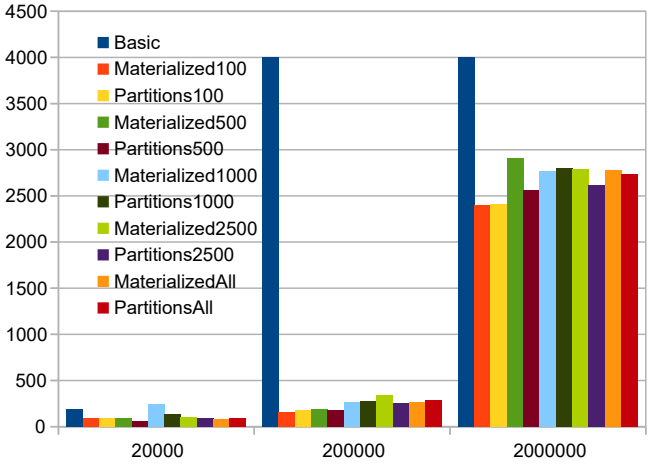


**Figure 6: Average query execution time in milliseconds (y-axis) of $Q_5$ when scaling the amount of tuples in the database (x-axis) and the amount of disease terms (color bars)**

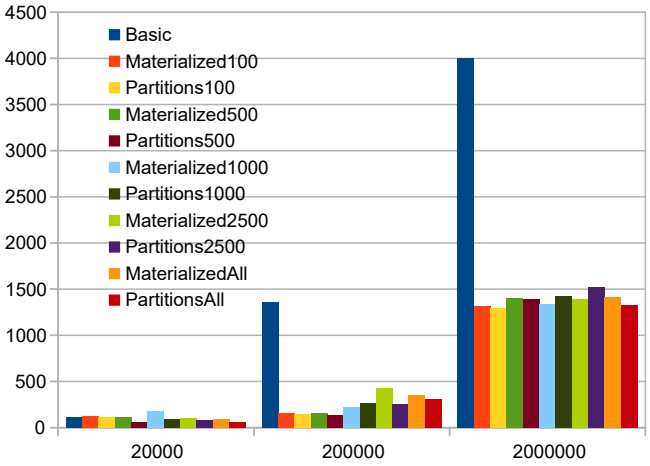


**Figure 5: Average query execution time in milliseconds (y-axis) of $Q_4$ when scaling the amount of tuples in the database (x-axis) and the amount of disease terms (color bars)**

of any similarity-based fragmentation) and compared it to both the similarity-based approaches: materialized fragments and partition numbers in Ignite's affinity function. Figures 2 to 6 show the results of executing our five benchmark queries (Table 2) when scaling the amount of tuples in the database from 20000 over 200000 to 2000000. All three approaches return the same set of results. For the two similarity-based approaches we tested in addition different amounts of underlying disease terms from MeSH that are used in the Diagnosis column: we tested 100, 500, 1000, and 2500 randomly chosen as well as all (4798) MeSH terms. We ran the five queries three times and averaged the runtimes. Whenever the Basic Ignite approach runtime significantly exceeded the runtime of the other approaches, we cut off its measurements after 4000*ms*.

Comparing the Materialized Fragments and the Partion Number approach, the performance is on average quite similar with both approaches. Only with the largest data set, the Partition Number approach shows a slight advantage over the Materialized Fragments approach: in this case, on average the Partition Number approach is 5.12 % faster than the Materialized Fragments approach.

One interesting thing to note is that for the entire MeSH term set (4798 terms) the runtime is often less than the one for measurements with fewer MeSH terms. This is due to the fact that the more MeSH terms are available in the dataset, the smaller the resulting fragments become. In other words, if there are less MeSH terms underlying the clustering, then more patients will have a tuple in the *Ill* relation with a certain disease term (like Asthenopia) as the data are generated completely randomly and diseases are picked randomly from the chosen term set. This results in more tuples matching the selection condition and also more tuples that have to be joined. While the join is still pretty fast due to collocation, a presumably bigger result set has to be transferred back to the client.

Another interesting observation is that joining two subresults from two different fragments in $Q_2$ performs better (due to parallelization of the subresult computation on two different servers) than joining two subresults from the same fragment in $Q_5$.

In sum, with both our similarity-based approaches the join computation is much less a bottleneck than for the basic off-the-shelf implementation. In roughly half of our tests the basic implementation exceeded the time limit of 4 seconds and had to be stopped prematurely; in contrast, our two implementations show a good scalability and a decent performance in all test runs.

## 7 DISCUSSION AND CONCLUSION

The previously depicted results of the query executions in the different implementations show that the clustering-based fragmentation of the data improves the execution time of cohort identification queries against the DDB significantly when comparing to the reference implementation that provides only an arbitrary, but balanced

horizontal fragmentation of the data. The intelligent similarity-based query answering enables the DDBS to speed up the execution of queries containing selection conditions on the relaxation attribute as the underlying fragmentation is not only meaningful but also more efficient when answering queries. The intelligent similarity-based answering hence enables the DDBS to speed up the execution of queries containing selection conditions on the clustering attribute. In addition to this, both clustering-based techniques scale well with the size of the database, whereas the query execution time of the basic implementation does not.

As already mentioned in the introduction, in future work we aim to support more sophisticated notions of similarity (like a combination of semantic as well as numeric as in [21, 24]) in order to identify not only patients that suffer from similar diseases but also whole patient profiles based on the similarity of their personal characteristics (e.g. their age or weight) and some other recorded measurements (e.g. body temperature or blood parameters). Another open issue is modification (by either insertions, deletions or updates) of the data set. The mechanisms for cluster adaption have been theoretically discussed in [25] but have not yet been tested in our implementation. In addition the clustering-based fragmentation could also be considered for more flexible non-relational data models [1, 26]. Moreover, fragmentation can indeed be used to protect sensitive content from unauthorized access [2, 5, 18]; in future work we aim to further elaborate the security enforcement by fragmentation for specific applications in the medical domain.

## ACKNOWLEDGEMENTS

This work was partially supported by the Fraunhofer Internal Programs under Grant No. Attract 042-601000.

## REFERENCES

[1] Ahmed Al-Ghezi and Lena Wiese. 2018. Adaptive workload-based partitioning and replication for RDF graphs. In *International Conference on Database and Expert Systems Applications*. Springer, 250–258.

[2] Mazhar Ali, Kashif Bilal, Samee U Khan, Bharadwaj Veeravalli, Keqin Li, and Albert Y Zomaya. 2015. DROPS: division and replication of data in cloud for optimal performance and security. *IEEE Transactions on Cloud computing* 6, 2 (2015), 303–315.

[3] J. Barrasa. 2019. neosemantics. https://github.com/jbarrasa/neosemantics version 3.4.0.2.

[4] Mary Regina Boland, George Hripcsak, Yufeng Shen, Wendy K Chung, and Chunhua Weng. 2013. Defining a comprehensive verotype using electronic health records for personalized medicine. *Journal of the American Medical Informatics Association* 20, e2 (09 2013), e232–e238. https://doi.org/10.1136/amiajnl-2013-001932 arXiv:https://academic.oup.com/jamia/article-pdf/20/e2/e232/6113544/20-e2-e232.pdf

[5] Ferdinand Bollwein and Lena Wiese. 2018. Keeping secrets by separation of duties while minimizing the amount of cloud servers. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXVII*. Springer, 1–40.

[6] Mirko Michele Dimartino, Andrea Calì, Alexandra Poulovassilis, and Peter T Wood. 2019. Efficient Ontological Query Answering by Rewriting into Graph Queries. In *International Conference on Flexible Query Answering Systems*. Springer, 75–84.

[7] Saurabh Gombar, Alison Callahan, Robert Califf, Robert Harrington, and Nigam H Shah. 2019. It is time to learn from patients like mine. *npj Digital Medicine* 2, 1 (2019), 16.

[8] T. F. Gonzalez. 1985. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science* 38 (1985), 293 – 306.

[9] B. Haarbrandt, B. Schreiweis, S. Rey, U. Sax, et al. 2018. HIGHmed – An Open Platform Approach to Enhance Care and Research across Institutional Boundaries. *Methods of Information in Medicine* 57, S 01 (Jul 2018), e66–e81.

[10] Donald Kossmann. 2000. The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)* 32, 4 (2000), 422–469.

[11] Choong Ho Lee and Hyung-Jin Yoon. 2017. Medical big data: promise and challenges. *Kidney research and clinical practice* 36, 1 (2017), 3.

[12] Wei Lu, Jiajia Hou, Ying Yan, Meihui Zhang, Xiaoyong Du, and Thomas Moscibroda. 2017. MSQL: efficient similarity search in metric spaces using SQL. *The VLDB Journal* 26, 6 (2017), 829–854.

[13] B Middleton, DF Sittig, and A Wright. 2016. Clinical decision support: a 25 year retrospective and a 25 year vision. *Yearbook of medical informatics* 25, S 01 (2016), S103–S116.

[14] Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. 2017. Decomposing federated queries in presence of replicated fragments. *Journal of Web Semantics* 42 (2017), 1–18.

[15] Shawn N Murphy, Griffin Weber, Michael Mendis, Vivian Gainer, Henry C Chueh, Susanne Churchill, and Isaac Kohane. 2010. Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2). *Journal of the American Medical Informatics Association* 17, 2 (2010), 124–130.

[16] Abdul Quamar, Jannik Straube, and Yuanyuan Tian. 2020. Enabling Rich Queries Over Heterogeneous Data From Diverse Sources In HealthCare. In *CIDR*. cidrdb.org.

[17] Susan Rea, Jyotishman Pathak, Guergana Savova, Thomas A. Oniki, Les Westberg, Calvin E. Beebe, Cui Tao, Craig G. Parker, Peter J. Haug, Stanley M. Huff, and Christopher G. Chute. 2012. Building a robust, scalable and standards-driven infrastructure for secondary use of EHR data: The SHARPn project. *Journal of Biomedical Informatics* 45, 4 (2012), 763 – 771. https://doi.org/10.1016/j.jbi.2012.01.009 Translating Standards into Practice: Experiences and Lessons Learned in Biomedicine and Health Care.

[18] A. Roy and A. Olmsted. 2017. Distributed query processing and data sharing. In *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*. 221–224.

[19] Charles Safran, Meryl Bloomrosen, W Edward Hammond, Steven Labkoff, Suzanne Markel-Fox, Paul C Tang, and Don E Detmer. 2007. Toward a national framework for the secondary use of health data: an American Medical Informatics Association White Paper. *Journal of the American Medical Informatics Association* 14, 1 (2007), 1–9.

[20] Kai-Uwe Sattler. 2018. Distributed Query Processing. In *Encyclopedia of Database Systems (2nd ed.)*. Springer.

[21] Araek Tashkandi, Ingmar Wiese, and Lena Wiese. 2018. Efficient In-Database Patient Similarity Analysis for Personalized Medical Decision Support Systems. *Big data research* 13 (2018), 52–64.

[22] Caetano Traina Jr, Andre Moriyama, Guilherme Rocha, Robson Cordeiro, Cristina DA Ciferri, and Agma Traina. 2019. The SimilarQL framework: similarity queries in plain SQL. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 468–471.

[23] U.S. National Library of Medicine. 2019. Medical Subject Headings. https://www.nlm.nih.gov/mesh/

[24] Ingmar Wiese, Nicole Sarna, Lena Wiese, Araek Tashkandi, and Ulrich Sax. 2018. Concept acquisition and improved in-database similarity analysis for medical data. *Distributed and Parallel Databases* (2018), 1–25.

[25] Lena Wiese. 2014. Clustering-based fragmentation and data replication for flexible query answering in distributed databases. *Journal of Cloud Computing* 3, 1 (2014), 18.

[26] Lena Wiese. 2015. *Advanced Data Management for SQL, NoSQL, Cloud and Distributed Databases*. DeGruyter/Oldenbourg.

[27] Lena Wiese, Armin O. Schmitt, and Mehmet Gültas. 2019. Big Data Technologies for DNA Sequencing. In *Encyclopedia of Big Data Technologies*. Springer.

[28] Hongjiu Zhang, Fan Zhu, Hiroko H Dodge, Gerald A Higgins, Gilbert S Omenn, Yuanfang Guan, and the Alzheimer's Disease Neuroimaging Initiative. 2018. A similarity-based approach to leverage multi-cohort medical data on the diagnosis and prognosis of Alzheimer's disease. *GigaScience* 7, 7 (07 2018). https://doi.org/10.1093/gigascience/giy085 arXiv:https://academic.oup.com/gigascience/article-pdf/7/7/giy085/25192114/giy085.pdf giy085.

[29] Weiguo Zheng, Lei Zou, Wei Peng, Xifeng Yan, Shaoxu Song, and Dongyan Zhao. 2016. Semantic SPARQL similarity search over RDF knowledge graphs. *Proceedings of the VLDB Endowment* 9, 11 (2016), 840–851.