# Enabling Rich Queries Over Heterogeneous Data From Diverse Sources In HealthCare

Abdul Quamar
IBM Research - Almaden
ahquamar@us.ibm.com

Jannik Straube
IBM Germany
Jannik.Straube@de.ibm.com

Yuanyuan Tian
IBM Research - Almaden
ytian@us.ibm.com

## ABSTRACT

The digitalization of healthcare has created abundant and rich health-related data. To exploit the wealth of information in these healthcare data, modern applications often need to support *rich* queries that access *heterogeneous* data from *diverse* sources. This raises a number of data management challenges on data placement, data integration, and data querying. In this paper, we demonstrate how to address these challenges using an example healthcare application, which helps physicians match drugs against patient conditions. Three datasets are collected and placed into three disparate stores: a relational database, a text search engine, and a graph database. Domain specific data integration methods are applied to link the different pieces of data together. And finally, a simple polystore architecture is developed to support rich queries across the different datasets stored in disparate stores.

## CCS Concepts

•**Information systems** → **Mediators and data integration;**

## Keywords

Rich Queries; Heterogeneous data; Data Integration; Data Placement

## 1. INTRODUCTION

The digitalization of healthcare has created abundant and *diverse* health-related datasets, ranging from patient electronic medical records (EMR), physiologic signals from medical devices, IoT data from wearables, pharmaceutical informaton, and genomic data, to curated medical ontologies. Data coming from such diverse data sources are often stored under *heterogeneous* data models, including structured data that fit in the relational model, text data (e.g. clinic notes and drug side effects), graph data (e.g. medical ontologies and protein interaction networks), time series data (e.g. ECG signals), sequence data (e.g. genome data), and image data (e.g. X-rays). Sometimes, even a single dataset needs to store data under different data models. For example, Mimic [16] is such a heterogeneous dataset that contains structured data, text data, and time series data, etc.

To exploit the wealth of information in the healthcare data, applications often need to support *rich* queries that access data from diverse sources and under heterogeneous data models. For example, to find patients with similar diseases, one needs to refer to the disease relationships (e.g. type 2 diabetes is a type of diabetes) in the ontology graph to define disease similarity (e.g. sibling or cousin nodes in the disease hierarchy are considered similar), when querying structured records of patients (e.g. EMR). As another example, the textual clinical notes often need to be accessed together with the structured records of a patient to produce a personalized treatment plan.

The need to enable rich queries against heterogeneous data from diverse sources creates a number of data management challenges as described below.

**Data placement**. The first challenge is on deciding what data models and storage engines best fit the data. This requires a deep understanding of the data, the expected workload of the target healthcare application, and the query processing capabilities of the underlying data stores. Heterogeneous data requires different processing capabilities. Structured data is best suited to be stored in databases and queried through SQL; text data is mostly indexed and retrieved through search engines, like Elasticsearch [2] and Solr [1]; graph data is better analyzed using graph query languages (e.g. Gremlin [17] and Cypher [12]) in graph databases, like JanusGraph [5] and Neo4j [6]. Although one could argue to transform all the data into a single data model and analyze using a single query engine to address the issue of heterogeneity. However, we believe that this would require a tremendous amount of pre-processing effort in data transformation and often might lead to sub-optimal results in terms of supported queries, quality, and performance. Essentially, *one size does not fit all* [19]. As such, this approach of using the best store based on its querying capabilities and the supported data model is consistent with the recent polystore approaches like BigDAWG [11].

**Data integration.** The second challenge is on how to link different pieces of data together. For example, to better provide personalized preventive medicine, applications need to link a patent's EMR records with his/her IoT data from the wearable devices. This requires matching a patient to a user of a mobile app, through entity resolution [13]. Actu-


*CIDR* '2020 Amsterdam, Netherlands

ally, data integration issues also arise when a single dataset is broken up into pieces under different data models. For example, when the textual clinic notes are stored separately from the structured part of the patient records, integration points are still need to be maintained to reconstruct the full patient records later on. Of course, existing techniques in data integration [14] and to some extent graph based linking approaches like [10] that are limited to keyword search across different stores can be applied, but we argue that data integration especially in the healthcare domain requires an additional deep understanding of the semantic information in the domain schemas, with help from domain specific ontologies, taxonomies and dictionaries.

**Querying across disparate data sources.** The final challenge is on querying data placed in disparate data stores. Enabling rich queries in the healthcare domain requires accessing heterogenous data across disparate storage engines with different query languages/interfaces and processing capabilities. This essentially calls for a polystore solution [19]. Having said that, we would like to emphasize that any implementation of such a polystore solution would require addressing issues such as query optimization, data transformation between different data models, and integration of results obtained from different stores. For e.g. Estocada [8], allows for querying data fragments across a heterogeneous set of stores. It is based on view based query rewriting and requires a separate integration engine to combine results across different stores.

In this paper, we demonstrate how we address the above three challenges of data placement, integration, and querying, to enable rich queries over heterogeneous data from diverse sources for a particular healthcare application. This application aims at helping physicians match drugs against patients' conditions. To achieve this goal, three distinct datasets are collected: a patient EMR dataset, a drug dataset, and a disease ontology dataset. These datasets are analyzed and stored into three disparate stores, the Db2 relational database [3], Elasticsearch [2], and JanusGraph [5], based on the corresponding data models that best fit the data. We then apply *domain specific data integration* methods to connect the different pieces of data together. Afterwards, a simple polystore architecture is developed to support rich queries across the different datasets stored in disparate stores.

Note that although we implemented our solution for a Healthcare application, our proposed architecture and techniques for data placement, integration, and querying are generic, applicable across different domains and can benefit many of the existing polystore architectures mentioned above.

## 2. DATASET PLACEMENT

In this section, we describe the datasets collected for the demo application and their placement based on the their characteristics, the expected workload and the data models supported by the underlying stores.

**MDX Dataset [4].** This dataset describes a set of drugs in terms of the diseases or medical conditions that they treat, the dosage and administration details for different age groups, information about the adverse effects, precautions to be taken, how each drug is supplied, etc. The domain schema of the dataset is captured by an ontology consisting of 43 concepts, 78 properties and 58 relationships providing a semantically rich representation of the meta-data associated with the dataset.

This dataset is a collection of heterogeneous data. First of all, it contains structured information about drugs such as their names and IDs, the conditions they treat, administration dosage for adults and pediatric use, drug efficacy, approval authority (like FDA), etc. We extract and store instance data corresponding to these concepts in the ontology into a relational database (Db2). Storing such information in the relational database allows us to efficiently support queries that require joins and aggregations as well as integration with other structured data sources (Ref Section 3).

This dataset also has extensive textual information, e.g. the adverse effects associated with the drug such as *increased risk of bleeding*, drug precautions such as the drugs should not be taken by patients with *hepatic dysfunction*, whether the drug should be administered *(intravenously or orally)*, etc. We index such textual information associated with each drug in a text search index (Elasticsearch) in JSON format. Using a text search engine enables us to utilize fuzzy matching and ranking capabilities to answer queries that require context from the textual data associated with the drugs.

**Patient Dataset:** This is an internal dataset that contains patient EMR records, including basic information about patients, lab observations, medical conditions, and medication prescribed for each medical condition.

The EMR records in this dataset are available as structured data, and as such we place them in the relational database enabling a rich set of point and aggregation queries against the dataset. This also enables us to integrate the dataset with the structured portion of the MDX dataset enabling derivation of useful insights across the two integrated datasets (Ref Section 3).

**SNOMED Ontology [7].** SNOMED is a well known repository of medical terminology, represented in the form of an ontology graph that provides a rich semantic representation of clinical terms, disease conditions and their relations to each other, including hierarchies and compositions.

The SNOMED ontology graph is stored in a graph database (JanusGraph) to facilitate execution of a rich set of graph queries such as reachability queries (whether two medical conditions or diseases are related), path queries to see how two diseases are connected, similarity queries to see if two diseases are similar (e.g. sibling or cousin nodes in the disease hierarchy or nodes within a k-hop neighborhood in the SNOMED ontology are considered similar).

## 3. DATASET INTEGRATION

In this section we provide a high level overview of our approach for integrating domain specific datasets collected from different sources and placed onto multiple heterogeneous stores. We follow a two step approach.

First, we utilize the semantic information embedded in the domain schema, often described in the form of ontologies and augmented with taxonomies and dictionaries. These domain-specific ontologies provide an entity-centric view of the domain schema, in terms of the entities and concepts relevant to the domain and the relationships between them. We apply standard entity resolution techniques [13] over this meta data to identify candidate concept pairs that seman-

tically represent the same information. These concept pairs are considered as potential data integration points between two datasets.

Second, from the candidates, we determine the minimal set of concept pairs that are viable points of integration. For doing so, we find support at the data instance level based on the matches between the candidate concept pairs. We compute the Jaccard similarity score for each such pair (Ratio of the number of instance level matches to the total number of possible instance pairs). A naive approach to compute the match between data instances would be to do an exact data equality match. However, as the datasets are collected from varied sources, an exact match technique does not have a good recall. We, therefore, utilize domain specific vocabularies, available as dictionaries, taxonomies and smart term generators, to create variants of the data instance values representing the same information, and use these to determine the number of matches between candidate concept pairs. For example, in order to determine a match between, say, a particular disease like *Renal impairment* treated by a drug and the medical condition *kidney failure* associated with a patient, our relaxation technique would indicate a match, as both these terms are variants of the same clinical term. All Concept pairs that have a Jaccard similarity (computed based on their instance level matches) above a certain threshold (empirically determined) are considered as the minimal set of points of integration between the two datasets. Next we describe the different scenarios for dataset integration.

## 3.1 Integration of Heterogeneous Data

This scenario handles the case when a single dataset contains a mix of different types of data like MDX. Figure 1 shows the healthcare drug dataset MDX, available as XML documents. The structured portion of the drug data is stored in Db2 and the textual information associated with each drug is indexed in Elasticsearch. The point of integration between the two portions is via the drug identifiers.

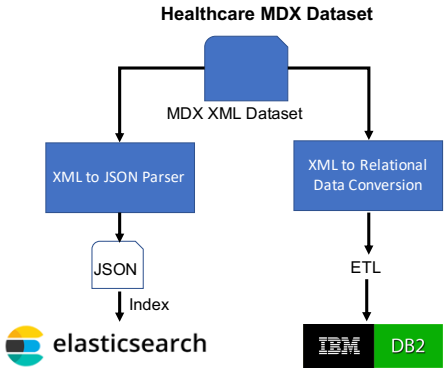


**Figure 1: MDX Dataset: Structured and text data integration**

## 3.2 Integration of data from different sources

**Use Case 1:** This use case handles the scenario of integrating datasets having different data models and collected from different sources. Figure 2 shows the patient dataset and the SNOMED ontology, stored in Db2 and JanusGraph, respectively. The point of integration between the two datasets across the two different stores is provided by a SNOMED ID which uniquely identifies a particular disease in the ontology (represented by a node in the SNOMED ontology graph) and the medical condition that a patient is suffering from (stored as a column in the relational database).

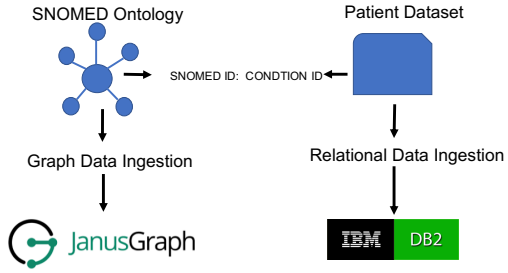


**Figure 2: Patient Dataset and SNOMED Ontology: Structured and graph data integration**

**Use Case 2:** This use case handles the scenario of integrating two datasets having the same data model but collected from different sources. Figure 3 shows two different structured datasets: Patient and MDX(structured portion). In order to identify the point of integration, we follow our two step approach mentioned above and identify candidate concept pairs: the patient medical *condition* concept in the Patient dataset and the concepts *Indication* and *Finding* in the Drug dataset that pertain to medical conditions treated by a drug. We then determine the instance level Jaccard similarity between the candidate pairs using the relaxed approach as described above and choose the *Indication* concept as the best match with the patient *condition* concept providing the necessary point of integration.

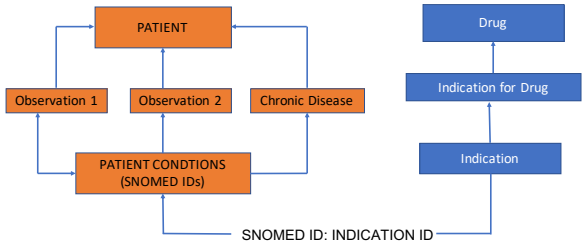


**Figure 3: MDX Drug - Patient EMR dataset integration**

# 4. SYSTEM ARCHITECTURE

In this section we introduce our system architecture (Figure 4) that enables the federation of disparate data stores and provides a mechanism of querying an integrated healthcare dataset. In our simple approach, we use a relational engine (Db2 [3]) as the main query processor and augment its capabilities with complex text search and graph queries through *User Defined Functions* (UDFs) that query, transform and ingest data from a text search engine (Elasticsearch [2]) and a graph database (JanusGraph [5]).

The choice of using UDFs was largely driven by, 1) their availability in most database systems and, 2) the flexibility that they can provide in terms of interfacing with disparate data stores, data ingestion and transformation from different data models, all of which are essential for implementing our

polystore architecture. In particular, we use *Table UDFs* in our implementation which return a record set, essentially a table which can then be joined with other tables in the SQL query to produce a combined result. As a result users can

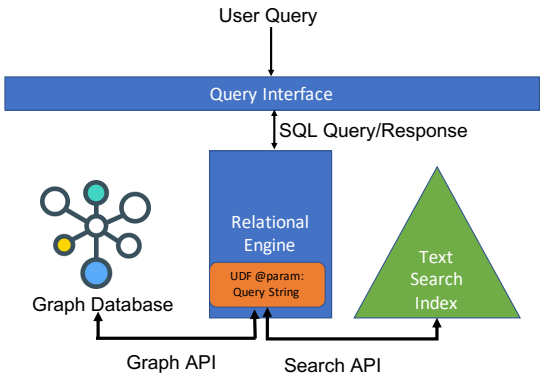


**Figure 4: System Architecture**

submit SQL queries against the relational query interface of any standard relational database system. This precludes any requirements of making changes to the SQL query language itself to accommodate constructs for querying graphs, text or semi-structured data. UDFs in the SQL query allow sub-parts of the query to be executed across different stores and results to be combined in the relational database and returned to the user.

In order to generate an optimal plan, query optimizers depend on cardinality estimations. It is hard to estimate the cardinality of tables returned using the UDF mechanism for data ingested from other stores based on a dynamic query. However, most relational engines do support cardinality hints. We exploit this ability and provide hints to the query optimizer using clues from queries against the search and graph stores. For example, if the text search queries contains a *'count=1000'* clause or a graph query has a *'limit(10)'* clause, we extract this information to provide a cardinality hint. In the absence of such clues, we can leverage existing techniques for cardinality estimation over text-search indexes [9, 15] and graph stores [18]. In practice, we observed that most queries that require context from the search and graph engines do limit the number of results using count and limit clauses enabling us to provide necessary inputs to the relational engine to generate optimal query execution plans.

# 5. USE CASES

In this section we provide a detailed walk through using different use case scenarios. We first describe the polystore setup, including the placement of data across different stores. We then provide details of queries issued against the polystore for the different scenarios described below using the system query interface shown in Figure 5.

## 5.1 Polystore Setup

We place and integrate the healthcare datasets described in Section 3 in three different stores. The MDX dataset is placed in a relational database (DB2) and text search index (Elastic Search). The Patient dataset is placed in the relational database (DB2) and the SNOMED ontology is

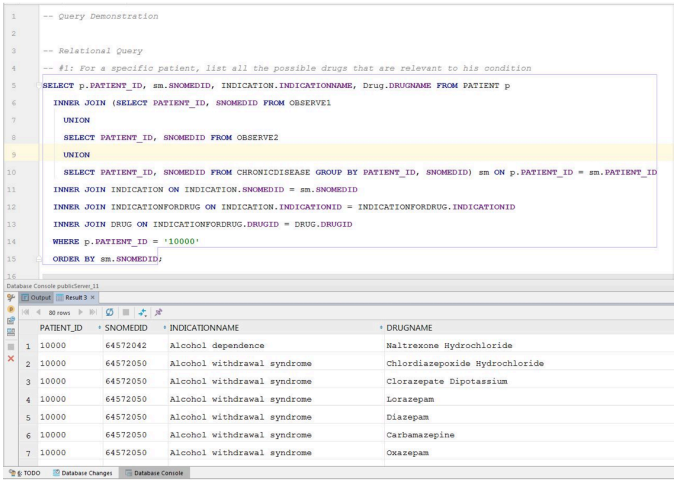


**Figure 5: User Interface**

placed in the graph database (Janus Graph). We register UDFs with DB2 that allow it to ingest and transform data from the other two data stores over the network.

## 5.2 Use case scenarios

**Scenario 1: Query against multiple data sources in a single relational database**

The following query searches for a patient the possible drugs that are relevant to the patient's medical conditions.

```
SELECT PATIENT.PATIENT_ID, CHRONICDISEASE.SNOMEDID,
  INDICATION.INDICATIONNAME, DRUG.DRUGNAME
FROM PATIENT INNER JOIN CHRONICDISEASE
    ON PATIENT.PATIENT_ID = CHRONICDISEASE.PATIENT_ID
  INNER JOIN INDICATION
    ON INDICATION.SNOMEDID = CHRONICDISEASE.SNOMEDID
  INNER JOIN INDICATIONFORDRUG
    ON INDICATION.INDICATIONID
      = INDICATIONFORDRUG.INDICATIONID
  INNER JOIN DRUG
    ON INDICATIONFORDRUG.DRUGID = DRUG.DRUGID
WHERE PATIENT.PATIENT_ID = '10000'
ORDER BY CHRONICDISEASE.SNOMEDID;
```

This query can be executed completely in the relational store using multiple joins over different tables of the integrated Patient and MDX dataset. Scenario 1 demonstrates the benefits in terms of the richness of queries than can be supported by the integration of two different structured data sets using the two step approach mentioned in Section 3. The absence of such an integrated solution would necessitate applications to reason about and combine results from queries against two different datasets.

**Scenario 2: Query across the relational database and the text search engine**

The following query finds all possible drugs relevant to a patient condition and that should be administered intravenously.

```
SELECT PATIENT.PATIENT_ID, CHRONICDISEASE.SNOMEDID,
  INDICATION.INDICATIONNAME, DRUG.DRUGNAME,
  CASE WHEN (R.DRUGIDENTIFIER IS NULL)
    THEN 0 ELSE 1 END INTRAVENOUS, R.HIGHLIGHT
FROM PATIENT INNER JOIN CHRONICDISEASE
    ON PATIENT.PATIENT_ID = CHRONICDISEASE.PATIENT_ID
  INNER JOIN INDICATION
    ON INDICATION.SNOMEDID = CHRONICDISEASE.SNOMEDID
  INNER JOIN INDICATIONFORDRUG
```

```
    ON INDICATION.INDICATIONID
        = INDICATIONFORDRUG.INDICATIONID
  INNER JOIN DRUG
    ON INDICATIONFORDRUG.DRUGID = DRUG.DRUGID
  LEFT JOIN TABLE(POLYSTORE.SEARCH('filter=
administration.type:intravenous&count=1000')) R
    ON DRUG.DRUGIDENTIFIERINT = R.DRUGIDENTIFIER
WHERE PATIENT.PATIENT_ID = '10000'
ORDER BY CHRONICDISEASE.SNOMEDID;
```

The drugs relevant for a specific patient's conditions are obtained from the relational store. The specific drugs that can be administered intravenously are obtained by doing a text search on drug related text data. The drug identifier returned by the search index for the drugs that match the search condition are combined with the drugs prescribed for specific patient conditions to get the desired result.

Scenario 2 benefits from multiple stages of data set integration. First, is the patient and drug data set integration (integration of two structured datasets) that allows the extraction of drugs relevant to the patient condition. Second, Drugs relevant to a particular type of administration mechanism are obtained from a search store through the powerful UDF (described in Section 4) mechanism which enables querying, data transformation and ingestion of data from a text search engine.

**Scenario 3: Query across the relational and graph databases**

The following query searches all patients suffering from a particular medical condition as well as the conditions similar to it.

```
SELECT PATIENT.PATIENT_ID, CHRONICDISEASE.SNOMEDID
FROM PATIENT INNER JOIN CHRONICDISEASE
  ON PATIENT.PATIENT_ID = CHRONICDISEASE.PATIENT_ID
WHERE CHRONICDISEASE.SNOMEDID IN(
  SELECT SNOMEDID
  FROM TABLE(POLYSTORE.GRAPH('{"gremlin":
"g.V().has('SNOMEDID',381).store('d').both()
.store('d').both().store('d')).cap('d').dedup()
.values('SNOMEDID')"}'));
```

This query requires the processing capabilities of a relational store to find patients suffering from a particular condition and a graph store to find conditions similar to a disease identified by a SNOMED ID. The similar diseases to a particular disease is defined as the diseases within its 2-hop neighborhood.

Scenario 3 highlights the benefits accrued from the integration of two datasets having different data models and collected from different sources. A structured dataset (patient dataset) and a graph data set (SNOMED ontology) stored under different data models across two different data stores. Again, the scenario demonstrates the effective use of UDFs in our polystore architecture for the required data transformation and ingestion to support a rich set of queries.

In our current system, a user has to explicitly express what sub-queries are issued against the different stores and how their results are combined together. However, the burden of writing explicit queries in the UDF calls for the individual stores can be alleviated by a higher-level abstraction layer that exposes a natural language query (NLQ) interface and hides the complexity of explicit queries from the users. Given inputs of data location and underlying store capabilities, such an abstraction layer could then translate a user query/request into appropriate sub-queries and route them against the appropriate underlying data stores. We would like to emphasize that our UDF-based polystore architecture proposed in this paper would serve as a basis for building such a high-level abstraction layer.

# 6. CONCLUSION

In this paper, we have demonstrated how to effectively support a rich set of queries against an integrated healthcare dataset consisting of structured, text and graph data collected from a diverse set of data sources. We highlight the challenges in building a system for querying and analyzing such a dataset placed on a heterogeneous set of data stores that support different data models and have varying query processing capabilities. We build and demonstrate a polystore solution to address the challenges in data placement, data integration and querying across disparate data sources in the healthcare domain, and provide several useful real-world scenarios to highlight the effectiveness of our proposed system in gaining valuable insights from the underlying dataset.

# 7. REFERENCES


[1] Apache Solr. http://lucene.apache.org/solr.

[2] Elasticsearch: RESTful, Distributed Search & Analytics. https://www.elastic.co/products/elasticsearch.

[3] IBM Db2. https://www.ibm.com/analytics/us/en/db2.

[4] IBM Micromedex. http://truvenhealth.com/Products/Micromedex/Product-Suites/Clinical-Knowledge.

[5] JanusGraph. http://janusgraph.org.

[6] Neo4j. https://neo4j.com.

[7] SNOMED International. http://www.snomed.org.

[8] R. Alotaibi, D. Bursztyn, A. Deutsch, I. Manolescu, and S. Zampetakis. Towards scalable hybrid stores: Constraint-based rewriting to the rescue. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1660–1677, 2019.

[9] A. Broder, M. Fontura, V. Josifovski, R. Kumar, R. Motwani, S. Nabar, R. Panigrahy, A. Tomkins, and Y. Xu. Estimating corpus size via queries. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pages 594–603, New York, NY, USA, 2006. ACM.

[10] C. Chanial, R. Dziri, H. Galhardas, J. Leblay, M.-H. L. Nguyen, and I. Manolescu. Connectionlens: Finding connections across heterogeneous data sources. *Proc. VLDB Endow.*, 11(12):2030–2033, Aug. 2018.

[11] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. Zdonik. The bigdawg polystore system. *SIGMOD Rec.*, 44(2):11–16, Aug. 2015.

[12] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1433–1445, 2018.

[13] L. Getoor and A. Machanavajjhala. Entity resolution: Theory, practice & open challenges. *PVLDB*, 5:2018–2019, 2012.

[14] B. Golshan, A. Y. Halevy, G. A. Mihaila, and W. C. Tan. Data integration: After the teenage years. In *PODS*, 2017.

[15] F. Islam, A. Hassaine, A. Jaoua, G. Das, and N. Zhang. Individual query cardinality estimation using multiple query combinations on a search engine's corpus. In *2017 International Conference on Computer and Applications (ICCA)*, pages 312–316, Sep. 2017.

[16] A. E. W. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark. Mimic-iii, a freely accessible critical care database. *Scientific Data*, 3, 2016.

[17] M. A. Rodriguez. The gremlin graph traversal machine and language (invited talk). In *Proceedings of the 15th Symposium on Database Programming Languages*, DBPL 2015, pages 1–10, 2015.

[18] G. Stefanoni, B. Motik, and E. V. Kostylev. Estimating the cardinality of conjunctive queries over RDF data using graph summarisation. *CoRR*, abs/1801.09619, 2018.

[19] M. Stonebraker. The case for polystores. https://wp.sigmod.org/?p=1629, 2015.