

# Optimizing Federated Queries Based on the Physical Design of a Data Lake

Philipp D. Rohde

TIB Leibniz Information Centre for Science and  
Technology  
Hannover, Germany  
philipp.rohde@tib.eu

Maria-Esther Vidal

TIB Leibniz Information Centre for Science and  
Technology  
Hannover, Germany  
maria.vidal@tib.eu

## ABSTRACT

The optimization of query execution plans is known to be crucial for reducing the query execution time. In particular, query optimization has been studied thoroughly for relational databases over the past decades. Recently, the *Resource Description Framework* (RDF) became popular for publishing data on the Web. As a consequence, federations composed of different data models like RDF and relational databases evolved. One type of these federations are *Semantic Data Lakes* where every data source is kept in its original data model and semantically annotated with ontologies or controlled vocabularies. However, state-of-the-art query engines for federated query processing over Semantic Data Lakes often rely on optimization techniques tailored for RDF. In this paper, we present query optimization techniques guided by heuristics that take the physical design of a Data Lake into account. The heuristics are implemented on top of *Ontario*, a SPARQL query engine for Semantic Data Lakes. Using source-specific heuristics, the query engine is able to generate more efficient query execution plans by exploiting the knowledge about indexes and normalization in relational databases. We show that heuristics which take the physical design of the Data Lake into account are able to speed up query processing.

## 1 INTRODUCTION

Advances in the technologies for data generation and ingestion facilitate the collection of large volumes of data from where valuable knowledge can be extracted. However, the wide variety of formats and data management systems available for storing and processing the collected data, hamper interoperability and data integration. The problem of integrating data collected from different data sources has been extensively treated in the literature [2, 11]; the mediator and wrapper architecture proposed by Wiederhold [23] and the data integration system approach presented by Lenzerini [17], represent the basis for the state of the art in data integration [7, 16, 19] and query processing over heterogeneous data sets or polystores [3, 5, 8, 15, 20]. Albeit the rich variety of solutions, the problem of efficiently querying heterogeneous data sources remains still open because data sources may differ in many various parameters, e.g., the physical implementation of the databases that store the data. In order to effectively solve interoperability and take advantage of the huge amount of available data, novel query processing solutions able to exploit not only logical characteristics of the data but also their physical representation, are demanded.

**Motivation.** Considering the query in Figure 1a two different query execution plans (QEP) can be generated. On the one hand, the QEP in Figure 1b is unaware of the physical design. Therefore, as many operations as possible are performed at the level of the query engine. On the other hand, the QEP in Figure 1c is aware of the physical design. Hence, as many operations as possible are pushed to the data sources. In the example query, the information about genes and diseases is from the *Diseasome* data set and stored in a single source. Therefore, the join can be pushed down. The filter expression for the scientific name of the species in the *Affymetrix* data set is always performed at the query engine because it is not indexed. No index is created since there are values that are present in more than 15% of the records.

This paper is organized as follows. Preliminary concepts and heuristics for optimizing federated queries are discussed in Section 2. Section 3 provides a preliminary analysis. Related work is presented in Section 4. We conclude in Section 5.

## 2 OUR APPROACH

### 2.1 Preliminaries

In this section, we present basic concepts required to understand this work. SPARQL queries can be partitioned into groups of acyclic patterns that share exactly one variable [22]. In the common case they represent a class of instances that share the same properties. Decomposing a query based on these groups leads to a QEP with *star-shaped sub-queries* (SSQ) in the leaves. In the motivating example (cf. Figure 1a) the SSQs are indicated with colored brackets. Following the idea of star-shaped groups over subjects *RDF Molecule Templates* (RDF-MT) [4] are an abstract description of the properties of the entities in an RDF data set. Each RDF-MT represents one class of instances, e.g., drugs from *Diseasome* or genes from TCGA. A *Data Lake* is a collection of heterogeneous data sets. The data sets do not necessarily share the same data model. If data models that do not have semantics by nature, e.g., relational databases, are annotated with semantics, the collection of data sets is called *Semantic Data Lake*. RDF and relational databases are amongst the most frequent data models present in Semantic Data Lakes. In our case, the query engine receives a SPARQL query and translates sub-queries to the native query language, e.g., SQL, of the data source.

### 2.2 Source-Specific Heuristics

One problem to tackle during query processing over a Data Lake is the variety of data models used throughout the Data Lake. State-of-the-art query engines use generalized optimization techniques or rely on heuristics tailored for one specific data model. Hence, they lose further opportunities for improving the query performance. In order to enable the maximal capability of optimizing the query execution plans, the physical design of the

```
SELECT * WHERE {
```

```

?probe  rdf:type          am:Probeset ,
         am:x-geneid      ?gene ,
         am:species-scientific-name "Homo sapiens",
         am:chromosomal-location ?loc .

?gene   tcga:chromosome ?chromosome ,
         tcga:start      ?chrstart ,
         tcga:stop       ?chrstop ,
         tcga:Strand     ?strand .

?dgene  disease:geneId  ?gene ,
         disease:hgncId ?hgnc ,
         owl:sameAs   ?dgenesame ,
         rdfs:label     ?dgenelabel .

?disease disease:associatedGene ?dgene ,
          disease:name          ?dn ,
          disease:possibleDrug  ?drug ,
          disease:class         ?diseaseclass .

```

```
}
```

(a) SPARQL Query

affymetrix

TCGA

Disease Gene

Disease Disease

(b) QEP without using indexes

TCGA

TCGA

TCGA

affymetrix

(c) QEP using indexes

**Figure 1: Query execution plans (QEP) for the same query (a); not considering indexes (b) and considering indexes (c). Optimizing QEPs with respect to the physical design of the Data Lake allows to find more efficient plans with fewer operations needed to be performed at the query engine level.**

Data Lake needs to be considered. This includes optimizing sub-queries for the different data models present in the Data Lake. We propose two heuristics designed for relational databases to show the impact of respecting the physical design. The proposed heuristics assume that the relational tables are normalized in 3NF. Further we expect that the subjects of a SPARQL query are modeled as the primary keys of the tables. Jozashoori and Vidal [14] showed that this is the best case scenario for running star-shaped sub-queries against relational databases.

**Heuristic 1 (Pushing down joins).** *Given two star-shaped sub-queries over the same RDB endpoint, combine those sub-queries into one sub-query if the join attribute is indexed.*

Heuristic 1 is proposed since joins over indexed attributes in relational databases are considered to be fast as long as the number of joins is kept reasonable. In the data we are working on, star-shaped sub-queries are usually represented by less than four relational tables. Assuming the worst case for a star-shaped sub-query, three relational tables contribute to the answer. Two of those tables are connected with the remaining table via foreign keys. Therefore, joining two star-shaped sub-queries leads to a six-way join in the worst case. Hence, the number of joins can be considered as reasonable. In order to decide if the number of joins is kept reasonable, a later version should consider the number of relational tables involved. Not only the join performance of RDB engines justifies this heuristic but also the possibility of a reduced size of the intermediate result. Heuristic 1 improves the query performance by reducing the time needed to perform the join as well as possibly decreasing the intermediate result.

**Heuristic 2 (Pushing up instantiations).** *Given a star-shaped sub-query over a relational database, perform filters on query engine level unless there is an index on the filtered attribute and the network speed is low.*

From our experience filtering string data at the query engine performs faster compared to executing the filters in the relational database. Therefore, Heuristic 2 is expected to speed up the query execution, even though a larger intermediate result has to be transferred to the query engine. However, if the network speed is low, the intermediate result has to be minimized and, therefore, the instantiation is performed at the relational database. Heuristic 2 leads to faster query execution through

speeding up the filter evaluation in case of fast networks at the cost of transferring larger intermediate results.

The proposed heuristics follow common knowledge about relational databases and network delays. Relational databases are designed to find effective and efficient query execution plans for joins and filter expressions exploiting indexes if beneficial. Even if the execution time at one source is increased by combining sub-queries into one, the overall query performance might be improved in the case of a slow network by reducing the intermediate result. Hence, the heuristics are very well suited to investigate the impact of considering the physical design of the Data Lake during the query optimization.

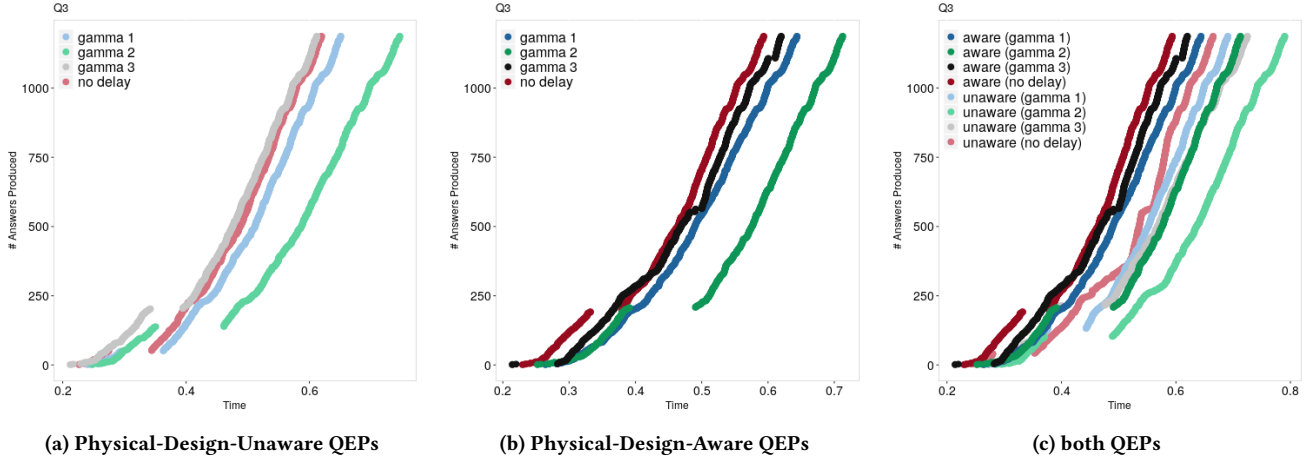
### 3 EXPERIMENT

We empirically study two different kinds of query execution plans in order to evaluate the proposed heuristics. The QEPs are as follows: **a) Physical-Design-Unaware QEP:** A QEP not respecting the physical design of the Data Lake and, therefore, not using the generated indexes to optimize the query execution. **b) Physical-Design-Aware QEP:** A QEP that considers the indexes present in the relational database. The source code and the data used for the experiment as well as the results are available at GitHub<sup>1</sup>.

**Data Sets.** In this experiment, we use the data sets from the LSLD benchmark [12] which is composed of ten real-world data sets from the life sciences domain of the Linked Open Data (LOD) cloud. The RDF version of each data set is transformed into relational tables. These tables are then normalized to 3NF. Indexes are created for the primary keys. Furthermore, additional indexes for some attributes that are used for joins or selections in the queries used are generated to evaluate the impact of the proposed heuristics. The data of each LSLD data set are uploaded into a dedicated MySQL 5.7.24 Docker container.

**Queries.** The queries provided for the LSLD benchmark do not contain the possibility of pushing down the join of two star-shaped sub-queries. Therefore, we do not use the provided queries and created five queries tailored for the heuristics to show their impact on query performance. The following parameters were considered during the development of the new queries: **a)** the selectivity of the query, **b)** filter expressions over indexed

<sup>1</sup><https://github.com/SDM-TIB/Ontario-SEADData2020>



**Figure 2: Answer Traces for Q3 with no delay and three different delays according to the gamma distributions. Answer traces show the generation of answers over time (in seconds). (a) Physical-Design-Unaware QEP not using indexes; (b) Physical-Design-Aware QEP using indexes whenever possible; and (c) both QEPs in comparison. Slow networks have a higher impact on physical-design-unaware QEPs. Respecting the physical design improves query performance.**

attributes, and c) possible joins of star-shaped sub-queries over indexed attributes. Another factor that impacts on the performance of a query is the size of the intermediate result.

**Network Simulation.** We used four different network settings which simulate the following networks: **a) No Delay:** perfect network with no or negligible latency. **b) Gamma 1:** fast network with a gamma distribution ( $\alpha = 1, \beta = 0.3$ ) of response latency resulting in an average latency of 0.3 milliseconds. **c) Gamma 2:** medium fast network with an average latency of 3 milliseconds resulting from a gamma distribution ( $\alpha = 3, \beta = 1$ ). **d) Gamma 3:** slow network with a gamma distribution ( $\alpha = 3, \beta = 1.5$ ) leading to an average latency of 4.5 milliseconds per message.

**Setup.** For the purpose of the experiment, Ontario [5] was modified to run physical-design-aware QEPs and physical-design-unaware QEPs. Network delays are simulated within the SQL wrapper of Ontario; delaying the retrieval of the next answer from the source. The duration of the delay is calculated using the `numpy.random.gamma()` function and the delay is produced using the Python `time.sleep()` function. Like the data sources, Ontario is running in a Docker container. All Docker containers were running at the same server. Hence, network costs other than introduced by the network simulation can be neglected. The experiments were executed on an Ubuntu 16.04.6 LTS 64 bit machine with two Intel(R) Xeon(R) Platinum 8160 2.10 GHz CPUs, and 755 GiB DDR4 RAM.

**Preliminary Results.** The experiment conducts of eight different configurations in total, i.e., both QEP types are evaluated using all four simulated network conditions. In doing so, we enable analyzing the impact of different network conditions and not only the impact of physical-design-aware execution plans. The analysis shows that the impact of network delays is higher in the case of physical-design-unaware query execution plans. An analysis of the results suggests that the proposed heuristics have potential to improving the query performance. However, the heuristics need to be evaluated more thoroughly and revised. The evaluation of Heuristic 1 is currently limited due to the query translation of Ontario. The translation of SPARQL queries into

SQL queries is not optimized for combining star-shaped sub-queries. This leads to an increase in the query execution time if the join is pushed down. Forcing Ontario to send the optimized SQL query for Q2 approx. halves the execution time compared to the physical-design-unaware QEP.

Even though Heuristic 2 seems to be correct from our experience, a deeper study on the difference of the filter execution performance between relational database and query engine is needed. On the one hand, the results of Q1 support our experience and suggest to follow Heuristic 2. On the other hand, the results of Q3 suggest otherwise. Figure 2 shows the answer generation for Q3 over time. It can be seen that executing the filter at the relational database (physical-design-aware QEP) is faster for this query. Therefore, more studies on the filter execution need to be done. Additionally, the experiment shows that the proposed heuristics are impacted by the implementation of other optimizations that are performed by Ontario. The impact of the heuristics on the query performance is not only influenced by the physical design of the Data Lake and the network conditions but also by the implementation of the query engine and wrappers.

## 4 RELATED WORK

**Federated Databases.** The problem of integrating data from dissimilar data sources has been addressed in the literature by implementing the mediator and wrapper architecture proposed by Wiederhold [23]. Several federated query engines have been defined in the context of relational database [6, 10, 13, 24], as well as diverse of integration frameworks [9]. We focus mainly on approaches that implement strategies to address the problem of source selection and decomposition of SPARQL queries, although, we recognize the tremendous advance that the Database community has done to the general problem of data integration in the last fifteen years. Existing approaches are grouped according to the amount of knowledge that describes the data sources, and that is exploited during source selection and query decomposition to enhance the quality of the generated query decompositions.

**Federations of RDF.** With the rise of the Resource Description Framework (RDF) new federated query engines were proposed to optimize query processing over the new data model.

*FedX* [21] is one of those query engines. *FedX* aims at minimizing the number of requests to be sent to the sources by identifying groups of triple patterns that can be exclusively evaluated by a single endpoint. *ANAPSID* [1] stores a list of predicates that each endpoint is able to answer. Queries are decomposed into star-shaped sub-queries. *ANAPSID* introduces adaptive physical operators to generate results as soon as they arrive from the sources. These operators perform better than the traditional blocking operators. *MULDER* [4] is based on *ANAPSID* and describes the sources in terms of RDF Molecule Templates (RDF-MTs). *MULDER* is able to reduce the query execution time and increase answer completeness by using semantics in the source descriptions during decomposition and source selection. Query engines for federations of RDF sources can benefit from the semantics of the metadata and received data.

**Polystores.** More recently, the research focus shifted towards query processing against heterogeneous data sources. Different approaches have been proposed on how to store, integrate, and query heterogeneous federations. *SeBiDA* [18] is a proof-of-concept for a semantified big data architecture. Data sets are differentiated in semantic, annotated with semantics, and non-semantic. The latter can optionally be lifted with semantics if mappings are provided. *SeBiDA* uses *Apache Spark* to reformat the data according to classes. The data is reformatted in *Apache Parquet* tables. Therefore, the data is integrated in a centralized or clustered manner and can be queried using SQL. Contrary to *SeBiDA*, *PolyWeb* [15] and *Ontario* [5] keep the data sources in their original data model. Data sources are queried in their native query language while the user sends SPARQL queries to the query engine. *PolyWeb* uses the same cost-based model as *FedX* does and *predicate-based join groups* to reduce the number of local joins. Other than *PolyWeb*, *Ontario* is based on *MULDER* and uses the same plan generator extended with heuristics for better optimization potential. *Ontario* also uses the physical operators of *ANAPSID*. Several query processing engines have been proposed, but most of them focus on a single data model for query optimization and therefore miss optimization opportunities.

## 5 CONCLUSION

In this paper we present two rather simple heuristics that aim at improving the query performance by considering the physical design of the Data Lake compared to state-of-the-art query execution plans. Our heuristics take the **a)** presence of indexes, and **b)** network condition into account. Even though the heuristics and their implementation are in an early stage, we can conclude that the query performance in a Data Lake can be improved when considering the characteristics of each data model. In future work, we plan to overcome the described limitations, e.g., improving the quality of the translated SQL queries. Furthermore, we will investigate the performance of different implementations of relational databases in order to gain a deeper understanding of why filter expressions seem to perform better at query engine level in most cases even though the intermediate results are larger in that case. Additionally, studying different kinds of query decomposition (e.g., triple-based instead of star-shaped sub-queries) and not normalized tables is part of our plans.

## ACKNOWLEDGMENTS

This work has been partially supported by the EU H2020 RIA funded projects *QualiChain* (No 822404) and *iASiS* (No 727658).

## REFERENCES

- [1] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. 2011. *ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints*. In *The Semantic Web – ISWC 2011. ISWC 2011. Lecture Notes in Computer Science*, Vol. 7031. Springer, Berlin, Heidelberg, 18–34.
- [2] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. 2012. *Principles of Data Integration*. Morgan Kaufmann.
- [3] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. 2015. The BigDAWG Polystore System. *SIGMOD Rec. 44*, 2 (Aug. 2015), 11–16.
- [4] Kemele M. Endris, Mikhail Galkin, Ioanna Lytra, Mohamed Nadjib Mami, Maria-Esther Vidal, and Sören Auer. 2018. Querying Interlinked Data by Bridging RDF Molecule Templates. In *Transactions on Large-Scale Data and Knowledge-Centered Systems XXXIX. Lecture Notes in Computer Science*, Vol. 11310. Springer, Berlin, Heidelberg, 1–42.
- [5] Kemele M. Endris, Philipp D. Rohde, Maria-Esther Vidal, and Sören Auer. 2019. Ontario: Federated Query Processing against a Semantic Data Lake. In *Database and Expert Systems Applications. DEXA2019. Lecture Notes in Computer Science*, Vol. 11706. Springer, Cham, 379–395.
- [6] Daniela Florescu, Alon Levy, and Alberto Mendelzon. 1998. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Rec. 27*, 3 (Sept. 1998), 59–74.
- [7] Behzad Golshan, Alon Y. Halevy, George A. Mihaila, and Wang-Chiew Tan. 2017. Data Integration: After the Teenage Years. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*. 101–106.
- [8] Rihan Hai, Sandra Geisler, and Christoph Quix. 2016. Constance: An Intelligent Data Lake System. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. 2097–2100.
- [9] Alon Halevy, Anand Rajaraman, and Joann Ordille. 2006. Data Integration: The Teenage Years (VLDB '06). VLDB Endowment, 9–16.
- [10] Alon Y. Halevy. 2001. Answering Queries Using Views: A Survey. *The VLDB Journal* 10, 4 (Dec. 2001), 270–294.
- [11] Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. 2006. Data Integration: The Teenage Years. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*. 9–16.
- [12] Ali Hasnain, Qaiser Mehmood, Syeda Sana e Zainab, Muhammad Saleem, Claude Warren, Durre Zehra, Stefan Decker, and Dietrich Rehbholz-Schuhmann. 2017. BioFed: federated query processing over life sciences linked open data. *Journal of Biomedical Semantics* 8, 1 (15 Mar 2017), 13:1–13:19.
- [13] Zachary G. Ives, Alon Y. Halevy, Peter Mork, and Igor Tatarinov. 2004. Piazza: mediation and integration infrastructure for Semantic Web data. *J. Web Semant.* 1 (2004), 155–175.
- [14] Samaneh Jozashoori and Maria-Esther Vidal. 2019. MapSDI: A Scaled-Up Semantic Data Integration Framework for Knowledge Graph Creation. In *On the Move to Meaningful Internet Systems: OTM 2019 Conferences. OTM 2019. Lecture Notes in Computer Science*, Vol. 11877. Springer, Cham, 58–75.
- [15] Yasar Khan, Antoine Zimmermann, Alokumar Jha, Vijay Gadepally, Mathieu D'Aquin, and Ratnesh Sahay. 2019. One Size Does Not Fit All: Querying Web Polystores. *IEEE Access* 7 (01 2019), 9598–9617.
- [16] Craig A. Knoblock and Pedro A. Szekely. 2015. Exploiting Semantics for Big Data Integration. *AI Magazine* 36, 1 (2015), 25–38.
- [17] Maurizio Lenzerini. 2002. Data Integration: A Theoretical Perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*. 233–246.
- [18] Mohamed Nadjib Mami, Simon Scerri, Sören Auer, and Maria-Esther Vidal. 2016. Towards Semantification of Big Data Technology. In *Big Data Analytics and Knowledge Discovery: DaWaK 2016. Lecture Notes in Computer Science*, Vol. 9829. Springer, Cham, 376–390.
- [19] Michalis Mountantonakis and Yannis Tzitzikas. 2019. Large-scale Semantic Integration of Linked Data: A Survey. *ACM Comput. Surv.* 52, 5, Article 103 (Sept. 2019), 40 pages.
- [20] Christoph Quix, Rihan Hai, and Ivan Vatov. 2016. GEMMS: A Generic and Extensible Metadata Management System for Data Lakes. In *28th International Conference on Advanced Information Systems Engineering (CAISE 2016)*. 129–136.
- [21] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. 2011. *FedX: Optimization Techniques for Federated Query Processing on Linked Data*. In *The Semantic Web – ISWC 2011. ISWC 2011. Lecture Notes in Computer Science*, Vol. 7031. Springer, Berlin, Heidelberg, 601–616.
- [22] Maria-Esther Vidal, Edna Ruckhaus, Tomas Lampo, Amadis Martinez, Javier Sierra, and Axel Polleres. 2010. Efficiently Joining Group Patterns in SPARQL Queries. In *The Semantic Web: Research and Applications*, Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 228–242.
- [23] Gio Wiederhold. 1992. Mediators in the Architecture of Future Information Systems. *IEEE Computer* 25, 3 (March 1992), 38–49.
- [24] Vladimir Zadorozhny, Louiqa Raschid, Maria Esther Vidal, Tolga Urhan, and Laura Bright. 2002. Efficient Evaluation of Queries in a Mediator for Web-Sources (*SIGMOD '02*). Association for Computing Machinery, New York, NY, USA, 85–96.