CONTENTS

EasyBDI: Near Real-Time Data Analytics over Heterogeneous Data Sources

Bruno Silva
IEETA
University of Aveiro
Aveiro, Portugal
bsilva@ua.pt

José Moreira
DETI-IEETA
University of Aveiro
Aveiro, Portugal
jose.moreira@ua.pt

Rogério Luís de C. Costa
CIIC
Polytechnic of Leiria
Leiria, Portugal
rogerio.l.costa@ipleiria.pt

ABSTRACT

The large volume of currently available data creates several opportunities for sciences and industry, especially with the application of data analytics. But also raises challenges that make unfeasible the use of batch-based ETL processes. Indeed, near real-time data analytics is a requirement in several domains as an alternative to traditional data warehouses. In the last years, big data platforms have been developed to enable query execution over distributed data sources. However, they do not deal with subject-oriented analysis, do not provide data distribution transparency, or do not assist with schema mapping and integration.

In this demonstration, we present EasyBDI. It's a near real-time big data analytics prototype that enables users to run queries over heterogeneous data sources based on global logical abstractions created by the system and provides some usual concepts of data warehouse systems, like facts and dimensions. We use two motivating scenarios, one based on three years of real data on photovoltaic energy production and consumption, and the other based on the SSB+ benchmark. We will also present implementation challenges, issues, solutions, and insights.

KEYWORDS

Distribution transparency, data analytics, near real-time data warehousing

1 INTRODUCTION

For several years, data analytics has been based on large data warehouses. Such warehouses are mostly centralized databases whose data is periodically extracted from OLTP databases and load into the warehouse as part of an ETL (extract, transform, and load) process [10]. This traditional warehouse structure is not suitable for most of the current big data analytics environments. On the other hand, near real-time operations have become a requirement in several current IT contexts, like in IoT, where several sensors generate data streams and users need to process and analyze the most recent data [10].

This demonstration presents EasyBDI (*Easy Big Data Integration*), a prototype for logical integration of distributed and heterogeneous data sources (including NoSQL ones, like MongoDB, Kafka, and Redis) into a global database and global star schemas. The integration is logical, i.e., there is no materialized global database, and data source autonomy is maintained. Analytical queries specified over global star schemas are transformed and executed by the distributed and heterogeneous sources.

Building a global schema requires finding and matching syntactic and semantic similarities between the data structures of distinct local sources. This can be achieved either by looking at the structural organization of data (i.e., schema-based matching) or to the contents and meaning of data (i.e., instance-based matching) [7]. Also, each local element should be mapped into a global element. For instance, local structures identified as semantically identical in the schema matching process should be mapped into a single global entity. Partitioning of (logical) global structures across distinct databases should also be handled [6]. Creating a global schema over distributed databases is challenging, particularly in the context of NoSQL and heterogeneous databases. EasyBDI gets the data organization on participating sources and uses a combination of techniques to automatically propose a global schema, which can be fine-tuned by the users.

EasyBDI runs over a distributed query execution engine (Trino, formerly PrestoSQL [4, 8]) and adds some levels of abstraction, namely data location and fragmentation transparency, and specialized subject-oriented analysis. The system uses schema matching and integration techniques to automatically design a global model and allows users to build subject-oriented cubes over such model. Non-expert users may use drag-and-drop to submit analytical queries over global cubes, but advanced features (e.g. based on SQL language) are also available for experts.

Big data frameworks and polystore systems (e.g. Apache Drill [5], Presto [8], BigDAWG [3]) provide a unified query language that can be used to access distributed data. But big data frameworks commonly lack providing distribution transparency, while polystores are tightly integrated, managing all sources together, including in terms of data location and data replication [9]. Our system maintains source autonomy, uses a global schema to provide distribution transparency (location, replication, and fragmentation), is extensible, and supports a wide range of data sources.

In the demonstration, two scenarios will be made available for participants. The first one is based on more than 3 years of real data on photovoltaic panel production/consumption in Sydney, Australia, and nearby areas. The second uses the SSB+ benchmark [2], which contains persistent and streaming data on retail store's sales, deliveries and popularity in social media.

Participants will understand how EasyBDI deals with some key challenges, like how to (i) explore the local data models to identify entities of the global model that are partitioned across multiple data sources, (ii) implement the automatic schema matching, mapping, and integration procedures to support the users in designing global models for a large number objects and data sources, and (iii) execute queries on a high-level star schema model abstracting several distributed, heterogeneous and autonomous data sources. They will also see the global SQL queries generated by EasyBDI and their translation into queries to the local sources. Implementation challenges and issues, adopted solutions and insights will be discussed, making this demonstration helpful to researchers and practitioners.

2 EASYBDI PROTOTYPE

EasyBDI is a framework for near real-time analytics that provides logical integration of multiple data sources while also enabling the creation of star schemas over global (logical) entities. Local databases are assumed to be autonomous (operate independently) and heterogeneous (in terms of data models and query languages, e.g., relational, graph and document databases, or semi-structured and unstructured data sources).

Data sources are accessible through a distributed query engine (Trino). The role of EasyBDI is to add additional levels of transparency, namely, location and fragmentation transparency, to allow building global schemas providing unified and high-level representations of the data sources, and to enable the execution of analytical queries by subject-experts. The architecture and the main components of EasyBDI are depicted in Figure 1.

API communication handler This component handles the interaction with the external systems: Trino and SQLite.

The *Distributed Query Execution Engine Interface* uses Java and JDBC, and implements all the logic regarding the integration of EasyBDI with Trino. Even though Trino provides a single representation of the underlying data, queries to Trino must contain the data source identifier for each data fragment. Thus, this component must also convert the queries specified using a global schema (see Database Integration below) into queries supported by Trino. This requires adding data source identifiers, as well as union and join operators, according to the mapping used between the global and local schemas.

The *Schema Metadata Storage* is responsible for the storage and management of the metadata of all schemas: local schema view, global schema, and star schema. Currently, these (meta)data are stored in an SQLite database.

Configuration Manager This component is responsible for generating the configuration files that allow Trino to communicate with the data sources (e.g., data source type, data source URL, username, password, and other parameters that depend on the data source type). It also creates local schema views. The procedure consists of iterating each data source and retrieving (meta)data about their schemas, namely, tables and columns information (or the equivalent concepts depending on the data source), using Trino commands. These (meta)data are stored in an SQLite database through the *Metadata Storage Interface*.

Database integration This layer deals with the creation of the global schema. A global schema contains a set of global tables, each containing a mapping to logically related data fragments. The integration is logical, i.e., the global schema is entirely virtual and not materialized. The main tasks to build a global schema are *schema matching*, *schema integration* and *schema mapping*.

The *schema matching* defines a mapping of concepts in a schema with concepts in another schema. EasyBDI uses a schema-based method with linguistic and constraint-based criteria [6]. The algorithm starts by finding tables with similar names using the Levenshtein distance. For each pair of matching tables, EasyBDI does columns matching using the Levenshtein distance to compare names and a similarity measure to compare data types.

The *schema integration* defines the global tables and their columns, using the correspondences found in the schema matching. EasyBDI uses the stepwise binary integration method [6].

The *schema mapping* defines how to combine data from one or more local data sources (data fragments) into a single global table while keeping consistency and semantic coherence. EasyBDI

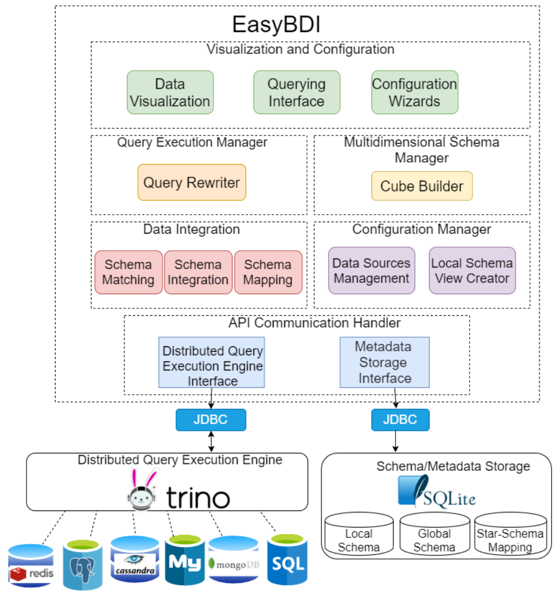


Figure 1: Main Components of EasyBDI architecture

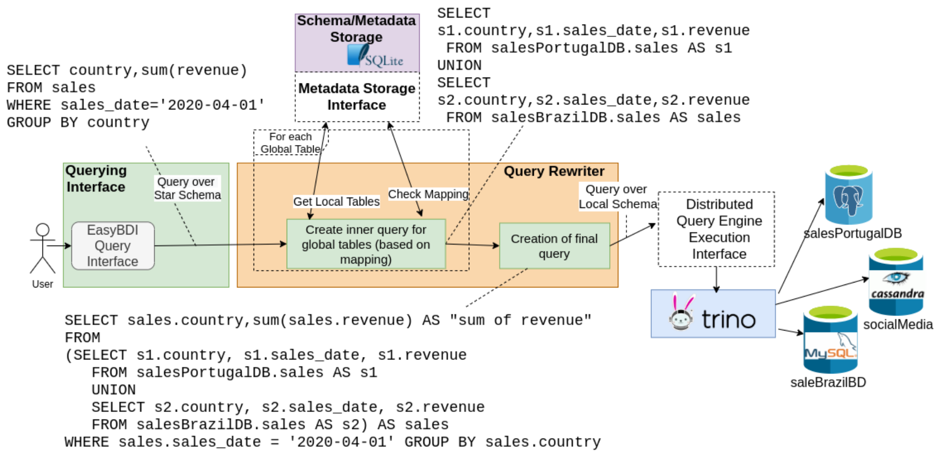
analyzes the matching between global and local entities, and identifies the type of partitioning (horizontal, vertical, or none) used. If a global table corresponds to only one local entity, then there is no partitioning. If a global table has a correspondence with several local tables, and the number of matching columns and their data types in all tables are the same, then the local tables are considered horizontally partitioned. The current implementation of EasyBDI uses foreign key-primary key relationships to find whether two or more tables are vertically partitioned, but this is only feasible when it is possible to get constraint information from the catalog of the data sources.

The methods presented above are automatic and may lead to incomplete or semantically incorrect results. Thus, EasyBDI allows users to review and edit the global schema generated automatically (e.g., removing replicated data sources) using an intuitive GUI interface.

Multidimensional Schema Manager This component allows the design of data cubes (star schemas) and the use of abstractions like facts and dimensions to perform analytical queries so that users focus on data analysis rather than technical details regarding data organization. Data cubes are built over the global schema,i.e., fact and dimensions tables are based on global entities. A start query is basically a join between a fact table and some dimensions, possibly with filters and aggregations.

Query Execution Manager This component rewrites the queries on the global schema into the queries submitted to Trino to get data from the local data sources. It handles vertical and horizontal data partitioning. Queries on global tables are automatically translated into queries on local tables using union and join operations of data fragments. The framework can handle multiple aggregations and joins at the same time. The queries that merge partitioned data are written as nested queries. An outer query contains the operators specified by the user (e.g., filters and aggregations) and other implicit joins needed between the facts table and dimensions. It is also possible to deal with complex operations such as pivoting and unpivoting data.

Figure 2 exemplifies the query submission process, which starts with a user interacting with the interface and issuing a



```
SELECT country,sum(revenue)
FROM sales
WHERE sales_date='2020-04-01'
GROUP BY country
```

```
SELECT
s1.country,s1.sales_date,s1.revenue
 FROM salesPortugalDB.sales AS s1
UNION
SELECT
s2.country,s2.sales_date,s2.revenue
 FROM salesBrazilDB.sales AS sales
```

```
SELECT sales.country,sum(sales.revenue) AS "sum of revenue"
FROM
(SELECT s1.country, s1.sales_date, s1.revenue
   FROM salesPortugalDB.sales AS s1
   UNION
   SELECT s2.country, s2.sales_date, s2.revenue
   FROM salesBrazilDB.sales AS s2) AS sales
WHERE sales.sales_date = '2020-04-01' GROUP BY sales.country
```

Figure 2: Query transformation - from an initial query over global schema to a query over local schemas

query over the star schema. Then the system generates an SQL query based on the global schema, which is translated into a query on the local schema and submitted to Trino. Notice that users do not need to know any query language: EasyBDI provides an intuitive interface and analytical queries are specified using drag-and-drop. Finally, Trino prepares and sends the queries that must be executed by the local data sources. Thus, users can create queries using global names and cubes and do not need to know about the format, organization and distribution of the data, nor a query language.

The *Query Execution Manager* also transforms the results provided by Trino according to the global schema and sends them to the visualization layer.

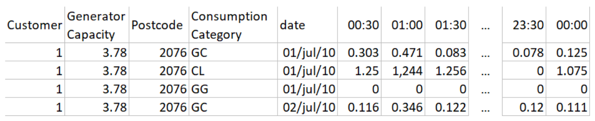
Visualization and Configuration This layer comprises several tools related to querying and configuration, including wizards to provide guided operations to users, like the cube builder, query builder, global schema editor, and data source configuration.

3 DEMONSTRATION

This demonstration has two case studies to highlight different features of EasyBDI. It will cover the selection of data sources, automatic generation of a global schema, creation of a star schema and execution of drag-and-drop (and user-edited) analytic queries.

Case study 1 uses data on energy production and consumption of 300 homes equipped with photovoltaic panels in Sydney over the span of 3 years. The data are available in three CSV files. The lines represent the customers, generator capacity, dates and type of consumption or production (GC, CL, and GG), and the columns represent the values recorded every 30 minutes (Figure 3). This case is interesting because the organization of the data is far from a typical data organization in relational databases.

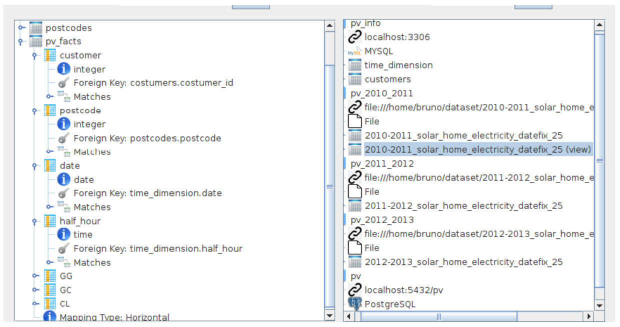
We consider that the data on the customers' location (postal codes related data) are in a PostgreSQL database and the temporal data are in a MySQL database, just for experimentation purposes.



Customer	Generator Capacity	Postcode	Consumption Category	date	00:30	01:00	01:30	...	23:30	00:00
1	3.78	2076	GC	01/jul/10	0.303	0.471	0.083	...	0.078	0.125
1	3.78	2076	CL	01/jul/10	1.25	1,244	1.256	...	0	1.075
1	3.78	2076	GG	01/jul/10	0	0	0	...	0	0
1	3.78	2076	GC	02/jul/10	0.116	0.346	0.122	...	0.12	0.111

Figure 3: Sample of a CSV file with photovoltaic data

After the automatic generation of the global schema, we will manually edit it to show some advanced features. In particular, we will show how to use virtual tables and user-defined commands to unpivot the data in the CSV files, how to specify constraints and change the data types of global schema columns, and how to create a mapping between the global schema and the data sources using virtual tables (Figure 4).



```
postcodes
pv_facts
  customer
    integer
    Foreign Key: costumers.costumer_id
    Matches
  postcode
    integer
    Foreign Key: postcodes.postcode
    Matches
  date
    date
    Foreign Key: time_dimension.date
    Matches
  half_hour
    time
    Foreign Key: time_dimension.half_hour
    Matches
  GG
  GC
  CL
  Mapping Type: Horizontal
```

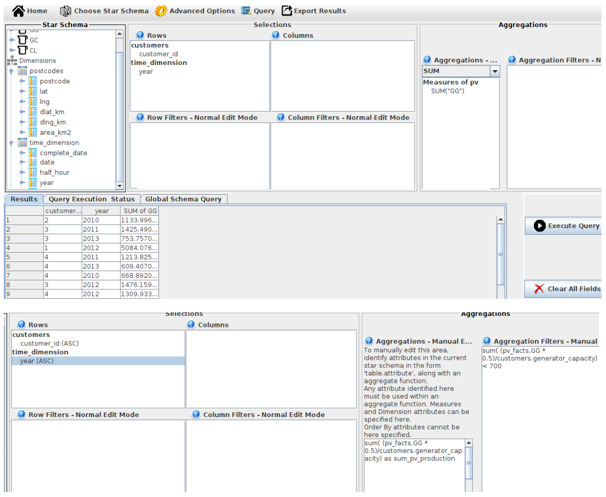
```
pv_info
  localhost:3306
  MYSQL
  time_dimension
  customers
pv_2010_2011
  file:///home/bruno/dataset/2010-2011_solar_home_e
  File
  2010-2011_solar_home_electricity_datefix_25
  2010-2011_solar_home_electricity_datefix_25 (view)
pv_2011_2012
  file:///home/bruno/dataset/2011-2012_solar_home_e
  File
  2011-2012_solar_home_electricity_datefix_25
pv_2012_2013
  file:///home/bruno/dataset/2012-2013_solar_home_e
  File
  2012-2013_solar_home_electricity_datefix_25
pv
  localhost:5432/pv
  PostgreSQL
```

Figure 4: Global schema (left panel) and local schema views (right panel) for the photovoltaic datasets

Then, a star schema will be created with a fact table (pv_facts), three measures (GC, CL and GG) and three dimensions (customer_dim, time_dim and postalcodes_dim). We will show how to run queries on the global schema using only drag-and-drop how to edit the commands manually (Figure 5). We will also illustrate the transformation of queries on a global schema (Listing 1) into queries on the local schema views that must be executed by Trino (Listing 2). The "<user query>" in Listing 2 denotes the query used to transform the data structure depicted in Figure 3 into a virtual table and is omitted because of its size.

Listing 1: Code generated for the global query depicted in Figure 5 (top) **(datatype casting operators were removed).**

```
SELECT     c.customer_id, t.year,
           SUM(pv.GG) AS "SUM_of_GG"
FROM       customers c, time_dimension t, pv
WHERE      ( t.half_hour = pv.half_hour
AND          c.customer_id = pv.customer
AND          t.date = pv.date )
GROUP BY   ( t.year, c.customer_id)
```



customers
customer_id
time_dimension
year

Measures of pv
SUM("GG")

	customer...	year	SUM of GG
1	2	2010	1133.996...
2	3	2011	1425.490...
3	3	2013	753.7570...
4	1	2012	5084.076...
5	4	2011	1213.825...
6	4	2013	609.4070...
7	4	2010	668.8920...
8	3	2012	1476.159...
9	4	2012	1309.933...

customers
customer_id (ASC)
time_dimension
year (ASC)

To manually edit this area, identify attributes in the current star schema in the form 'table.attribute', along with an aggregate function.
Any attribute identified here must be used within an aggregate function. Measures and Dimension attributes can be specified here.
Order By attributes cannot be here specified.

```
sum( (pv_facts.GG *
0.5)/customers.generator_cap
acity) as sum_pv_production
```

```
sum( (pv_facts.GG *
0.5)/customers.generator_capacity)
< 700
```

Figure 5: A global query made with drag-and-drop (top) and a global query edited manually (bottom)

Listing 2: Translation of the code in Listing 1 into a query over the local schema views **(the name of the data source "mysql_localhost_3306_time" is abbreviated to "mysql").**

```
SELECT customers.customer_id, time_dimension.year,
    SUM(GG) AS "SUM_of_GG"
FROM
  (SELECT mysql.pv_schema.customers.customer_id
    FROM mysql.pv_schema.customers) AS customers,
  (SELECT mysql.pv_schema.time_dimension.half_hour,
   mysql.pv_schema.time_dimension.year,
   mysql.pv_schema.time_dimension.date
   FROM mysql.pv_schema.time_dimension) AS
   time_dimension,
SELECT customer, date, half_hour, GG FROM
  (<user query> UNION <user query>
  UNION <user query>) AS pv
WHERE (time_dimension.half_hour = pv.half_hour
  AND customers.customer_id = pv.customer
  AND time_dimension.dated = pv.date)
GROUP BY(time_dimension.year, customers.customer_id)
```

Case study 2 uses the SBB+ benchmark [2] and represents a more conventional scenario in a big data environment. The SSB+ data model has two star schemas, one for batch Online Analytical Processing (OLAP) and the other for streaming OLAP. The batch OLAP is an adaptation of the schema proposed in TPC-H. The streaming OLAP is based on social media data and represents the popularity of retail stores (and their sales and deliveries).The data for OLAP batch storage are stored in Hive, while the facts data for streaming OLAP are stored in Cassandra (a NoSQL database that uses a wide-column store model), and conceptual relationships exist between data stored in both systems, as represented in Figure 6. We use the code available in [1] to populate the data sources.

In this case, EasyBDI's automatic matching, integration and mapping was mostly correct. Only a few manual edits are needed and no user-defined commands are necessary. We also show how to create the star schemas for batch OLAP and streaming OLAP and how to execute queries. SSB+ has a listing of analytical queries that we used to test the functionality of EasyBDI and we also use some of these queries in this demonstration. Regarding EasyBDI's performance, the overhead associated with query

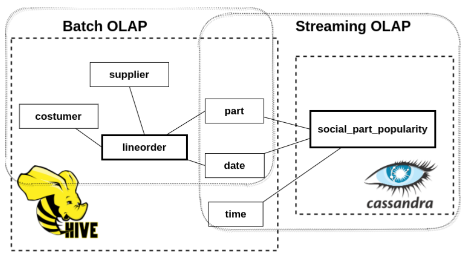


Figure 6: Case study 2: batch and streaming OLAP – overview of local schemas

generation is small (≈ 1% of query execution time). Query execution time depends mostly on query complexity, data sources' efficiency, and on the resources available for Trino.

4 SUMMARY

In this work, we present EasyBDI, a framework for near real-time data analytics that uses logical data integration to provide a high-level abstraction of data distribution and heterogeneity, while keeping the autonomy of the data sources. Automatic schema matching and configuration wizards are used to support the addition of new data sources. Multidimensional data organization is used to enable query analytics by subject-experts. We present two scenarios, one based on real data and the other on the use of a benchmark that simulates sales data and social media data. We also present several implementation issues, discussing adopted solutions that may be helpful to researchers and practitioners. EasyBDI is publicly available on Github *https://github.com/bsilva3/EasyBDI*.

ACKNOWLEDGMENTS

This work is partially funded by National Funds through the FCT (Foundation for Science and Technology) in the context of the projects UIDB/04524/2020, UIDB/00127/2020 and POCI-01-0247-FEDER-024541.

REFERENCES

[1] Carlos Costa. 2019. Big Data Benchmarks. https://github.com/epilif1017a/bigdatabenchmarks Accessed = 2021-02-01.
[2] Carlos Costa and Maribel Yasmina Santos. 2018. Evaluating several design patterns and trends in big data warehousing systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 10816 LNCS. Springer Verlag, 459–473. https://doi.org/10.1007/978-3-319-91563-0_28
[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] Trino Software Foundation. 2021. Trino - Distributed SQL Query Engine for Big Data. https://trino.io/ Accessed = 2021-02-01.
[5] Michael Hausenblas and Jacques Nadeau. 2013. Apache Drill: Interactive Ad-Hoc Analysis at Scale. *Big data* 1, 2 (2013), 100–104.
[6] M. Tamer Özsu and Patrick Valduriez. 2020. *Principles of distributed database systems, fourth edition.* Springer. 1–674 pages.
[7] Erhard Rahm and Philip A. Bernstein. 2001. A survey of approaches to automatic schema matching. *VLDB Journal* 10, 4 (dec 2001), 334–350.
[8] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, and Christopher Berner. 2019. Presto: SQL on everything. *Proceedings - International Conference on Data Engineering* 2019-April (2019), 1802–1813.
[9] Dimitris Stripelis, Chrysovalantis Anastasiou, and José Luis Ambite. 2018. Extending Apache Spark with a Mediation Layer. In *Proceedings of the International Workshop on Semantic Big Data* (Houston, TX, USA) *(SBD'18)*. Association for Computing Machinery, New York, NY, USA, Article 2, 6 pages.
[10] Ran Tan, Rada Chirkova, Vijay Gadepally, and Timothy G. Mattson. 2017. Enabling query processing across heterogeneous data models: A survey. In *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017.* IEEE Computer Society, 3211–3220.