On the Optimization of Query Plans in Multistores

(Discussion Paper)

Chiara Forresi[1,*], Matteo Francia[1], Enrico Gallinucci[1] and Matteo Golfarelli[1]

[1]*University of Bologna, Cesena, Italy*

Abstract

Multistores are data management systems that facilitate query processing across databases based on different data models; in addition to distributing data, integration and data fusion activities are necessary to address complexities such as schema heterogeneity and data replication. Our multistore solution relies on a dataspace to provide the user with an integrated view of the available data and enables the formulation and execution of GPSJ queries. In this paper, we outline a technique to optimize the execution of GPSJ queries by formulating and evaluating different execution plans on the multistore. In particular, we identify different strategies to carry out joins and data fusion by relying on different schema representations; then, a self-learning black-box cost model is used to estimate execution times and select the most efficient plan. The experiments assess the effectiveness of the cost model in choosing the best execution plan.

Keywords

Multistore, NoSQL, Query optimization, Cost model

1. Introduction

The decline of the *one-size-fits-all* paradigm has pushed researchers and practitioners towards the idea of *polyglot persistence* [1], where a multitude of databases is employed to support data storage and querying. The motivations are manifold, including the exploitation of the strongest features of each system, the off-loading of historical data to cheaper DBMS, and the adoption of different storage solutions by different branches of the same company. This trend has influenced the discipline of data science, as analysts are steered away from traditional data warehousing and towards a more flexible and lightweight approach to data analysis.

Multistores are characterized by 1) the replication of data across different storage systems (i.e., there is no sharp horizontal partitioning) with possibly conflicting records (e.g., the same customer with a different country of residence in different databases), and 2) a high level of schema heterogeneity: records of the same real-world entity may be represented with different structures, using different naming conventions for the same kind of data. The large volume and the frequent evolution of these data hinder the adoption of a traditional integration approach.

SEBD 2023: 31st Symposium on Advanced Database System, July 02–05, 2023, Galzignano Terme, Padua, Italy

*Corresponding author.

chiara.forresi@unibo.it (C. Forresi); m.francia@unibo.it (M. Francia); enrico.gallinucci@unibo.it (E. Gallinucci); matteo.golfarelli@unibo.it (M. Golfarelli)

0000-0001-5652-2455 (C. Forresi); 0000-0002-0805-1051 (M. Francia); 0000-0002-0931-4255 (E. Gallinucci); 0000-0002-0437-0725 (M. Golfarelli)

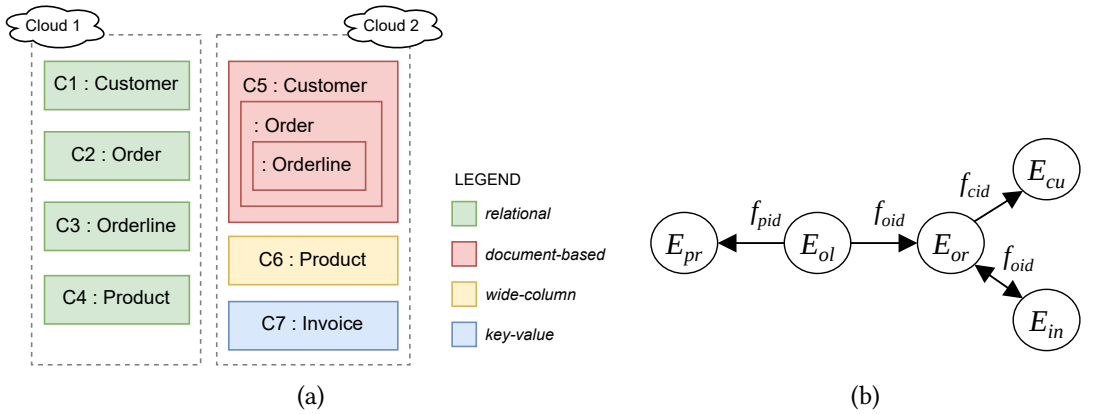


Figure 1: A graphical representation of the physical implementation of the case study; different colors represent different databases with different data models (a). The dataspace of the case study (b).

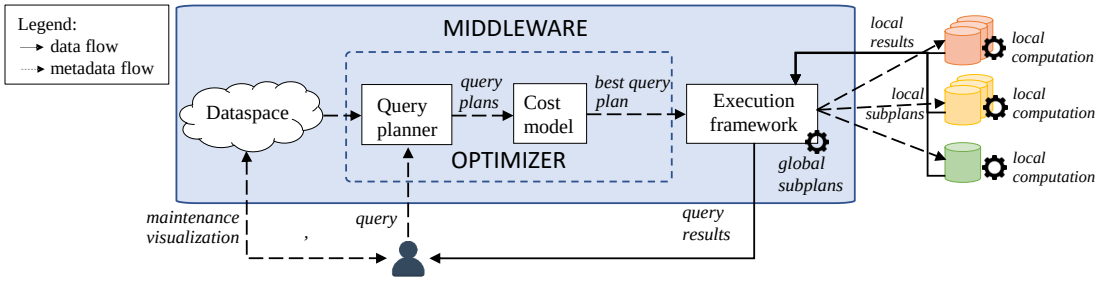


Figure 2: Overview of our multistore.

In recent work [2, 3, 4, 5] we have proposed a multistore solution that relies on a dataspace to provide the user with an integrated view of the data. A *dataspace* is a lightweight integration approach providing basic query expressiveness on a variety of data sources, bypassing the complexity of traditional integration approaches and possibly returning best-effort or approximate answers [6]. The dataspace is built in accordance with a *pay-as-you-go* philosophy, i.e., by applying simple matching rules to recognize relationships between data structures and by letting the users progressively refine the dataspace as new relationships are discovered [7]. Users exploit the dataspace to formulate GPSJ (generalized projection, selection, and join) queries, i.e., the most common class of queries in analytical applications [8]. Queries are translated into execution plans that consist of many local computations (carried out by the single databases) and a global computation (carried out by the middleware layer).

In this paper, we outline a technique to optimize the execution of GPSJ queries by finding the most efficient execution plan on the multistore and experimentally assess its efficiency.

2. Overview and multistore formalization

We consider a *multi-cloud architecture* case study, where different branches of the same holding rely on different storage systems to store overlapping data on the same domain. The physical implementation is depicted in Figure 1a, with C_1 to C_7 representing the collections of data and the ":" notation indicating the entities contained in each collection (notice that the document-based database contains a single collection which uses nested structures, e.g., to embed orders and order lines within customers). While Cloud 1 employs a relational database, Cloud 2 satisfies the need for data variety support by relying on NoSQL systems and also stores orders' invoices. As the two branches belong to the same holding, both customers and products are partially overlapped in the two cloud environments. Figure 1b shows the dataspace of the case study.

The multistore is described by a *dataspace*, i.e., an abstract global representation of the data scattered across different databases. It is composed of two main concepts: *entities*, corresponding to the real-world entities in the multistore (e.g., customers, products), and *features*, corresponding to the attributes that describe entities (e.g., the name of customers, the brand of products). These concepts are built in a pay-as-you-go fashion by analyzing the schemas in the data and detecting relationships between attributes.

Figure 2 provides a functional overview of the multistore system and the supported user interactions. Most importantly, users interact with the dataspace to formulate GPSJ queries, which are well-suited for data analysis; a typical analytical query consists of a group-by set (i.e., the features used to carry out an aggregation), one or more numerical features to be aggregated by some function (e.g., sum, average), and (possibly) selection predicates. Based on the user's query, the system's Optimizer defines the query plan to be executed on the multistore in two steps: first, the Query planner generates multiple query plans, then a Cost model is used to choose the most convenient one. Query plans are decomposed into subplans, each identified by *macro operators* that embed a tree of operations. *Local subplans* are computed directly on the local databases; *global subplans* are computed on the middleware's execution framework to combine the partial results from local subplans and obtaining the final result to be returned.

In the dataspace, entities are identified by a key that corresponds to the feature that uniquely distinguishes the instances (e.g., the feature identifying orders in E_{or} is f_{oid}). Relationships are expressed between two entities E_i and E_j on a feature f. Most importantly, *many-to-one* relationships are indicated with $E_i \xrightarrow{f} E_j$. It is $E_i \Rightarrow E_k$ if there exists a path of many-to-one relationships from E_i to E_k.

A collection C contains data that refer to one or more dataspace entities, indicated with $\mathcal{E}_C$; the portion of the dataspace described by C is called a *collection graph* (CG_C). Depending on the way that entities $\mathcal{E}_C$ are modeled in C, we recognize three kinds of *schema representations*.

- *Normal* (NoR), composed by a single entity.
- *Nested* (NeR), composed by at least two entities connected in a single path of many-to-one relationships from E_i to E_k ($E_i \Rightarrow E_k$). Each instance in C is identified by the key of E_k, and contains features about the other entities in the form of nested arrays. For example, C_i in Figure 3 is a fully nested collection, showing an instance of E_{cu} containing an array of instances of E_{or}, each containing an array of instances of E_{ol}.
- *Flat* (FlR), which is also composed of at least two entities where $\exists E_i \in \mathcal{E}_C$ such that

C_i

```
{ "cid":"C001",
  "firstName":"Alice",
  "orders":[{
    "oid":"O010",
    "orderDate":"2020-01-01",
    "orderLines":[{
      "olid":"OL100",
      "asin":"B00794N76O",
      "qty":94
    },{ "olid":"OL101",
      "asin":"B004PYML90",
      "quantity":80
}]}]}
```

C_j

```
{ "oid":"O010",
  "orderDate":"2020-01-01",
  "cid":"C001",
  "firstName":"Alice"
  "orderLines":[{
    "olid":"OL100",
    "asin":"B00794N76O",
    "qty":94
  },{ "olid":"OL101",
    "asin":"B004PYML90",
    "quantity":80,
  }]}
```

C_k

```
{ "olid":"OL100",
  "asin":"B00794N76O",
  "qty":94,
  "oid":"O010",
  "orderDate":"2020-01-01",
  "cid":"C001",
  "firstName":"Alice"
},{ "olid":"OL101",
  "asin":"B004PYML90",
  "quantity":80,
  "oid":"O010",
  "orderDate":"2020-01-01",
  "cid":"C001",
  "firstName":"Alice"}
```

Figure 3: Three examples of collection graphs representing the same data with the same set of entities using different schema representations.

$\forall E_j \in \mathcal{E}_C \setminus E_i$ it is $E_i \Rightarrow E_j$. Each instance in C is identified by the key of E_i and instances of E_i also contain features of the other entities. For example, C_k in Figure 3 is a fully flat collection showing two instances of E_{ol} with the corresponding features of E_{or} and E_{cu}. Notice that FlR implies the duplication of values from E_{or} and E_{cu}.

If a collection graph fully conforms to one of these schema representations, we indicate it with $rep(CG_C) \in \{\text{NoR}, \text{NeR}, \text{FlR}\}$. If a collection mixes different schema representations (for instance C_j in Figure 3, which mixes NeR and FlR), then $rep(CG_C) = \varnothing$.

3. Multistore algebra

3.1. NRA and data fusion operations

The query execution plans are formulated in Nested Relational Algebra (NRA) extended with the merge operator (⟗) to support data fusion operations, handling overlap between collections and resolving schema heterogeneity and record overlapping [4, 5]. Its goal is to retain as much information as possible, both from the extensional and the intensional points of view. The merge operator (⟗) answers this need by (i) avoiding any loss of records, (ii) providing output in terms of features instead of attributes, and (iii) resolving conflicts whenever necessary. The operation essentially involves a full-outer join between the collections, followed by the resolution of the columns that represent the same feature.

3.2. Entity views

To simplify the discussion on query plans, we introduce the notion of entity views as high-level abstraction operations. An entity view (EV) is a runtime-computed collection that provides a standard representation for the records modeling a set of entities.

Definition 1 (Entity view). *An entity view is a collection χ whose records represent the features of a given set of entities in accordance to a schema representation. Its collection graph CG_χ is such that $rep(CG_\chi) \in \{NoR, NeR, FlR\}$.*

An EV is either local or global. A *local* entity view (LEV) is obtained from collections belonging to the same database, thus it may provide a partial representation of a set of entities. A *global* entity view (GEV) provides a complete and cleansed representation of a set of entities in the multistore. The operations on EVs are defined as *EV operators*, i.e., macro-NRA operators (distinguished from simple ones by the hat ˆ symbol) that embed a tree of NRA operations.

- **LEV creation**: $\hat{\pi}(\mathcal{C}_\chi, F_\chi, p_\chi, CG_\chi)$. This operation creates a LEV χ from a set of collections $\mathcal{C}_\chi$ from the same database, projects a set of features F_χ, and applies the optional selection predicates p_χ; the structure of the result is defined by CG_χ.
- **GEV creation**: $\hat{\sqcup}(X, p_X)$, where X is a set of LEVs, $|X| \geq 2$, and p_X is an optional conjunction of selection predicates. This operation creates a GEV χ' by resolving conflicts between duplicated records from two or more LEVs sharing the *same* collection graph CG', i.e., $\forall \chi \in X$ it is $CG_\chi = CG'$. Essentially, this macro-operator produces a left-deep tree of binary merge operations between LEVs. Once all LEVs have been merged, the optional selection predicates are applied.
- **Join of GEVs**: $\hat{\bowtie}(X)$, where X is a set of GEVs, $|X| \geq 2$. The obtained GEV χ' is the result of join operations between the GEVs in X representing connected but non-overlapping sets of entities. Like $\hat{\sqcup}$, this macro-operator produces a left-deep tree of binary join operations between two GEVs. The result is the GEV that provides a cleansed representation of all the records in the multistore that are required to answer the query.

EV operations implement logical rules to produce an optimized NRA tree (e.g., push-down of selection predicates, join operation reordering), but we overlook them due to space limitations.

4. Query planning

The execution plans of GPSJ queries are defined in terms of EV operations as follows.

Definition 2 (Query plan). *A query plan P is a rooted tree of entity view operations, where (i) the root is a GEV join operation ($\hat{\bowtie}$), (ii) the root is preceded by one or more GEV creation operations ($\hat{\sqcup}$), and (iii) each of the latter is preceded by one or more (parallel) LEV creation operations ($\hat{\pi}$). The root is possibly extended with an NRA aggregation operation (γ).*

Example 1. *Figure 4 shows a sample plan for a query that computes, for each gender, the average quantities bought for products of brand "BrandABC". In the upper part, two LEV creation operations compute an EV in NeR with customers, orders, and order line records from the collection in the document-based database (i.e., C_5) and the tables in the relational one (i.e., C_1 to C_3), respectively; in particular, the latter is the one hiding the most complexity, as multiple join and nest operations are required to compute the NeR representation. The two LEVs are then merged in a GEV creation operation, that returns a cleansed FlR representation of the same data and projects the only features required by subsequent operations. Similarly in the lower part, two other LEV creation operations*

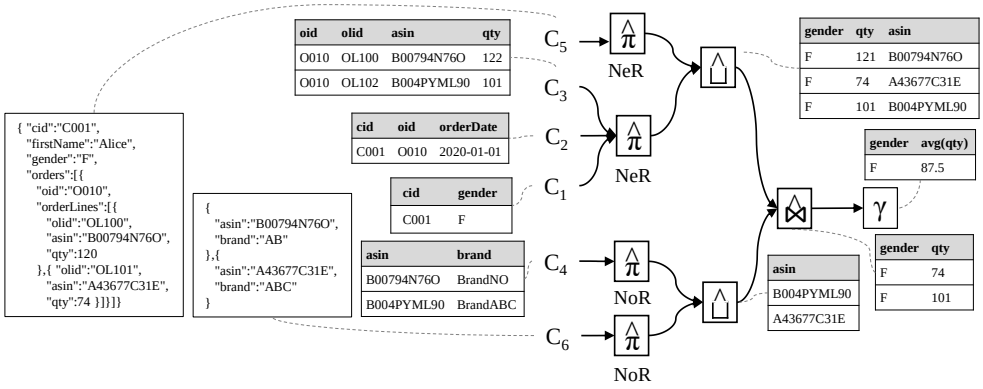


Figure 4: Example of a full query plan expressed with EV operations. Data flows are represented by full arcs while dotted lines link collections and EVs to sample data.

compute an EV in NoR with product records from C_6 and C_4, respectively. The subsequent GEV creation operation merges the products and applies the filter on the reconciled records. Ultimately, the GEV join operation combines the produced GEVs, while the aggregation operation computes the final result.

Several query plans can be devised for the same query. The factors that determine the number of alternative query plans are summarized below.

#1 *LEV creation.* Different query plans can be created by choosing different schema representations to create the LEVs.

#2 *GEV creation.* A query plan may include several alternative combinations of GEVs (e.g., considering a query that involves two entities E_{cu} and E_{or}, the options are to create (i) a GEV for each entity and joining them, or (ii) a single GEV in NeR or FlR).

#3 *LEV allocation.* Each LEV creation operation can be executed either directly by the middleware or pushed down to the database storing the respective data.

For a given query, all feasible query plans are enumerated. Due to space constraints, we refer the reader to [5] for the detailed algorithms to enumerate query plans. Among all possible query plans, the most efficient one is identified by the cost model discussed in Section 5.

5. Cost model

Finding the most efficient query plan is crucial and challenging due to the heterogeneity of different DBMSes and the variability in terms of DBMS's resources. In [3], we relied on existing literature to model the cost of each NRA operation on each engine in terms of read and written disk pages. While this worked well on the simple example considered by [3], (i) it did not consider resources allocation, (ii) it made simplistic assumptions about the parallelization of the computation, (iii) it considered execution costs related to disk I/O only, (iv) it required

Profile Feature	Domain	Engine supp.
Entity view operation	$\{\hat{\pi}, \hat{\sqcup}, \hat{\bowtie}\}$	R D W K M
Source schema representation	{NeR, NoR, FlR}	R D W K M
Target schema representation	{NeR, NoR, FlR}	R D W K M
Number of records	$\mathbb{N}$	R D W K M
Selectivity	$[0, 1]$	R D W K M
Aggregation rate	$[0, 1]$	R D - - M
Number of unnest operations	$\mathbb{N}$	R D - - M
Number of join operations	$\mathbb{N}$	R D - - M
Number of union operations	$\mathbb{N}$	- - - - M
Number of merge operations	$\mathbb{N}$	- - - - M
Number of selections exploiting indexes or source partitioning	$\mathbb{N}$	R D W K M
Number of selections not exploiting indexes or source partitioning	$\mathbb{N}$	R D W K M
Number of nested selections exploiting indexes	$\mathbb{N}$	R D - - M
Number of nested selections not exploiting indexes	$\mathbb{N}$	R D - - M
Number of aggregations (nest, group by)	$\mathbb{N}$	R D - - M

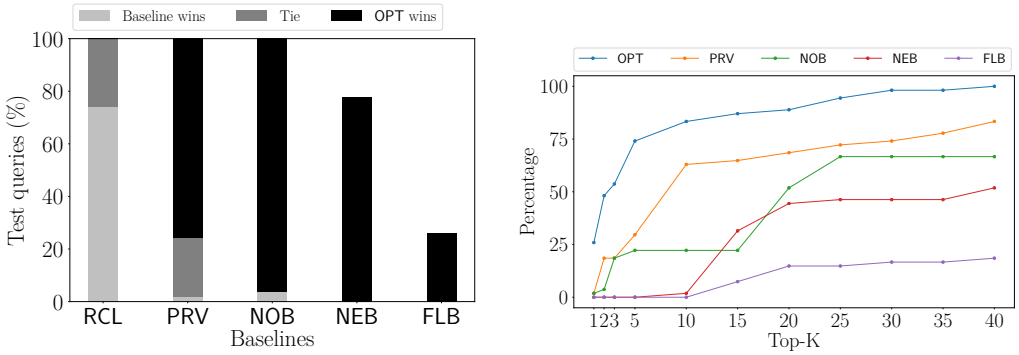
Table 1
The regression models' features; engines are Relational, Document, Wide-column, Key-value, Middleware.

an advanced knowledge about the internal details of each engine and related algorithms that reduces its extensibility.

We overcome these limitations by adopting a self-learning cost model, which implicitly captures the aforementioned aspects without requiring explicit and complex modeling of execution costs [9, 10]. Inspired by [11], the cost model is composed by a set of multi-regression models $H = \{h_0(), ..., h_n()\}$, one for each of the n execution engines composing the multistore including the middleware denoted by $h_0()$. The query plan P is partitioned in a set subplans SP, each corresponding to the execution of an EV operation on an engine. A multi-regression model $h_{eng(P')}(P')$ estimates the execution time for the subplan P' on the corresponding engine $eng(P')$ based on a plan profile. Table 1 shows the list of the features captured by the profile; some of them are directly obtained from the plan (e.g., number of unnest operations embedded in a $\hat{\pi}$ or $\hat{\sqcup}$ operation), while others also require basic statistics on the local databases (e.g., indexes, collections' cardinalities, and attributes' histograms to compute selectivity and aggregation rate). The execution time for P is estimated by composing the execution time of its subplans SP as follows: $Time(P) = \sum_{P' \in SP|eng(P')=0} h_0(P') + \max_{i \in [1,n]} \sum_{P' \in SP|eng(P')=i} h_i(P')$. Models' drift is detected through an error threshold, and new regression trees must be built for drifted engines or new databases.

6. Related Work and Conclusions

The variety in terms of data models responds to different requirements of modern data-intensive applications, but providing transparent querying mechanisms to query large-scale collections on heterogeneous data stores is an active research area [12]. Multistore and polystore systems have emerged as solutions to provide integrated access and querying to several heterogeneous stores through a mediator layer (middleware) [12]. The difference between multistores and polystores lies in whether they offer a single or multiple querying interfaces, respectively. Among the most notable are BIGDAWG [13], TATOOINE [14], and CloudMDsQL [15]. However, these systems do not provide direct support for data fusion. To effectively query a heterogeneous system with



(a) (b)

Figure 5: Pair-wise comparison between plan-selection strategies (a). Percentage of times that the plan chosen by the strategy is one of the top-K plans (b).

overlapping records, data fusion techniques [16] are necessary, but limited proposals consider this scenario in a polyglot system [17, 18]. As for the cost model, BIGDAWG [13, 19] uses black-box models for optimization within each engine, TATOOINE [14] makes no mention of cost optimization, and CloudMDsQL [15] blends rule-based and white/black-box cost modeling without giving details. Our multistore uses rule-based optimization and a black-box cost model with active learning, overcoming challenges of white-box models in complex environments; indeed, black-box models automatically learn and fine-tune a system behavior model, freeing the user from the task of modeling query costs.

In this paper, we have outlined a cost-based optimization of execution plans in a multistore by devising and evaluating different strategies to carry out joins and data fusion in presence of data replication. The execution plans are generated in terms of a multistore algebra extended from NRA and are based on different schema representations, so as to possibly take advantage of the original modeling of the data in the local databases. Experiments on different multistore benchmarks[1] have revealed the factors that drive the performance of different execution plans, demonstrating the need to evaluate alternative plans. Two key factors impacting execution plan performance are: (i) the need to solve record overlapping, which affects schema representation choice, and (ii) preserving the original modeling of data usually translates to faster executions.

Figures 5a and 5b show the effectiveness of the cost model (OPT) by comparing it with five baseline strategies: RCL is the oracle that always selects the optimal plan; PRV is based on a previous multistore implementation [4]; NOB, NEB, and FLB adopt a simple strategy to choose the plan that maximizes both computation push-down and the creation of LEVs in a given schema representation (respectively NoR, NeR, and FlR). The results show that OPT outperforms all baseline strategies and is more likely to choose the optimal (or a sub-optimal) plan.

Future work aims to enhance the multistore data platform by adding support for the graph data model and incorporating advanced features (e.g., data profiling, provenance investigation, application pipeline orchestration [20]). We plan to improve the system's efficiency by exploring data aggregation push-down to local databases and developing cost-effective execution plans.

[1]Available at https://big.csr.unibo.it/multistore

References

[1] P. J. Sadalage, M. Fowler, NoSQL distilled: a brief guide to the emerging world of polyglot persistence, Pearson Education, 2013.

[2] H. Ben Hamadou, E. Gallinucci, M. Golfarelli, Answering GPSJ queries in a polystore: A dataspace-based approach, in: Proceedings of Conceptual Modeling - 38th Int. Conf., ER 2019, volume 11788, Springer, 2019, pp. 189–203.

[3] C. Forresi, M. Francia, E. Gallinucci, M. Golfarelli, Optimizing execution plans in a multistore, in: Advances in Databases and Information Systems - 25th European Conference, ADBIS 2021, Tartu, Estonia, August 24-26, 2021, Proceedings, Springer, 2021, pp. 136–151.

[4] C. Forresi, E. Gallinucci, M. Golfarelli, H. B. Hamadou, A dataspace-based framework for olap analyses in a high-variety multistore, The VLDB Journal (2021) 1–24.

[5] C. Forresi, M. Francia, E. Gallinucci, M. Golfarelli, Cost-based optimization of multistore query plans, Information Systems Frontiers (2022).

[6] M. J. Franklin, A. Y. Halevy, D. Maier, From databases to dataspaces: a new abstraction for information management, SIGMOD Record 34 (2005) 27–33.

[7] S. R. Jeffery, M. J. Franklin, A. Y. Halevy, Pay-as-you-go user feedback for dataspace systems, in: 2008 ACM SIGMOD Int. Conf. on Management of Data, ACM, 2008, pp. 847–860.

[8] M. Golfarelli, D. Maio, S. Rizzi, The dimensional fact model: A conceptual model for data warehouses, Int. J. Cooperative Inf. Syst. 7 (1998) 215–247.

[9] L. Baldacci, M. Golfarelli, D. Lombardi, F. Sami, Natural gas consumption forecasting for anomaly detection, Expert systems with applications 62 (2016) 190–201.

[10] C. Loader, Local regression and likelihood, Springer Science & Business Media, 2006.

[11] M. Golfarelli, S. Graziani, S. Rizzi, An active learning approach to build adaptive cost models for web services, Data Knowl. Eng. 119 (2019) 89–104.

[12] R. Tan, R. Chirkova, V. Gadepally, T. G. Mattson, Enabling query processing across heterogeneous data models: A survey, in: 2017 IEEE Int. Conf. on Big Data, IEEE Computer Society, 2017, pp. 3211–3220.

[13] V. Gadepally, P. Chen, J. Duggan, A. J. Elmore, B. Haynes, J. Kepner, S. Madden, T. Mattson, M. Stonebraker, The bigdawg polystore system and architecture, in: 2016 IEEE High Performance Extreme Computing Conference, HPEC 2016, Waltham, MA, USA, 2016, pp. 1–6.

[14] R. Bonaque, et al., Mixed-instance querying: a lightweight integration architecture for data journalism, Proc. VLDB Endow. 9 (2016) 1513–1516.

[15] B. Kolev, et al., Cloudmdsql: querying heterogeneous cloud data stores with a common language, Distributed and Parallel Databases 34 (2016) 463–503.

[16] J. Bleiholder, F. Naumann, Data fusion, ACM Comput. Surv. 41 (2008) 1:1–1:41.

[17] A. Maccioni, R. Torlone, Augmented access for querying and exploring a polystore, in: 34th IEEE Int. Conf. on Data Engineering, ICDE 2018, IEEE Computer Society, 2018, pp. 77–88.

[18] E. Gallinucci, M. Golfarelli, S. Rizzi, Approximate OLAP of document-oriented databases: A variety-aware approach, Inf. Syst. 85 (2019) 114–130.

[19] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden,

D. Maier, T. Mattson, S. B. Zdonik, The bigdawg polystore system, SIGMOD Rec. 44 (2015) 11–16.

[20] M. Francia, E. Gallinucci, M. Golfarelli, A. G. Leoni, S. Rizzi, N. Santolini, Making data platforms smarter with MOSES, Future Gener. Comput. Syst. 125 (2021) 299–313.