

Constraint Management and Data Placement in the Context of Polyglot Persistence

Felix Kiehn^{1,*†}, Mareike Schmidt^{1,†} and Fabian Panse²

¹Universität Hamburg, Vogt-Kölln-Str. 30, 22527, Hamburg, Germany

²Hasso Plattner Institute, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

Abstract

Novel database architectures, data models and technologies have long challenged the *one size fits all* paradigm. Poly- and MultiStores emerged as a solution to combine these new data stores, each with their own strengths and weaknesses. Unfortunately, these systems come with their own set of problems such as a lack of support for adaptivity, data updates, or migration processes. In this work, we propose solutions intended as first steps towards solving these problems. In detail, we look at two main topics, (1) Data Placement and Workload Analysis as well as (2) Data Migration and Integrity Constraint Management, and discuss the challenges associated with them.

Keywords

Polyglot Persistence, Poly-/MultiStore, Data Placement, Constraint Management, Data Migration, Adaptivity

1. Introduction

Since the late 2000s novel ways to deal with an ever-growing amount of (oftentimes unstructured) data have emerged and continue to inspire new database systems and data models. All of these new systems, whether NoSQL or more advanced relational systems, have their own strengths and weaknesses [1]. Given the large amount of different systems optimized for different use cases, developers are often confronted with keeping track of where these systems excel or where not [2]. Additionally, the traditional way of using a single database solution for a project and the associated *one size fits all* mindset is no longer valid [3, 4]. Furthermore, beside so-called microservices [5], smaller functional sub-units of larger projects, each with its own database solution, there have also been efforts in recent years to combine several heterogeneous data stores (database systems). In an effort to combine the various strengths of these systems, they are unified within a mediation system and exposed to users as a single data store system [1]. The

focus of these systems, coined Multi- and PolyStores [6], range from aiming to offer the full data store experience (e.g., Polypheny-DB [7], BigDAWG [8]) to cross-platform data processing systems (e.g., Apache Wayang [9]). Most of these systems hold a static, manually designated mapping to place data to the respective data stores and lack any form of flexibility. However, to appropriately react to (possibly changing) requirements based on workloads and users' service level agreements (SLAs), these systems should be able to determine an optimal data placement including an appropriate handling of potential constraints. Therefore, we want to discuss two main problems and outline possible solutions and ideas: (1) How to calculate an optimal data placement based on current data, workload, SLAs, and deployed data stores, and (2) how to model and validate integrity constraints in data stores as well as implement efficient update/insert processes in a Multi-/PolyStore setting.

In Section 2, we describe the concept of PolyStores in detail and present the state of the art in Section 3. In Sections 4 and 5, we focus on our two main topics of Data Placement and Constraint Management, respectively. Finally, we provide a brief conclusion in Section 6.

2. Multi-/PolyStores

Before we dive deeper into the aforementioned problems, we want to establish what Poly- and MultiStores are in detail first. They can both be classified as "[s]ystems federating specialized data stores and enabling query processing across heterogeneous data models [...]" [6].

The main difference between these systems is the amount of query interfaces they offer. MultiStores have one single query interface whereas PolyStores incorporate multiple interfaces (Figure 1). However, both are

35th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), May 22-24, 2024, Herdecke, Germany.

*Corresponding author.

†These authors contributed equally.

✉ felix.kiehn@uni-hamburg.de (F. Kiehn);

mareike.schmidt-3@uni-hamburg.de (M. Schmidt);

fabian.panse@hpi.de (F. Panse)

🌐 <https://www.informatik.uni-hamburg.de/vsis/members/look/2291>

(F. Kiehn); <https://www.informatik.uni-hamburg.de/vsis/members/look/2290>

(M. Schmidt);

<https://hpi.de/naumann/people/dr-fabian-panse.html> (F. Panse)

🆔 0000-0003-2345-8551 (F. Kiehn); 0000-0002-7330-6276

(M. Schmidt); 0000-0002-0675-4116 (F. Panse)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).



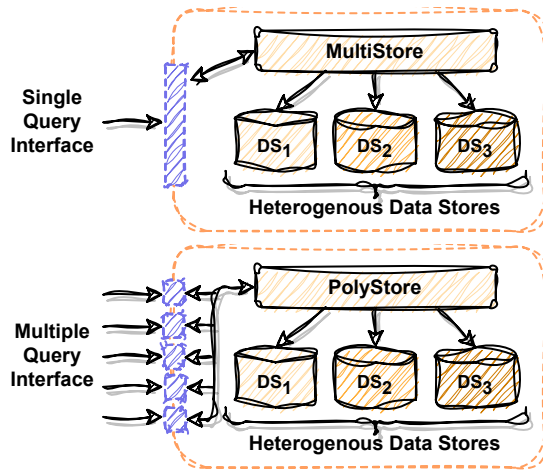


Figure 1: Difference between Multi- and PolyStores (Based on: Tan et al.[6])

built upon heterogeneous data stores. This distinguishes them from other system classes such as federated systems or polylingual¹ systems, which use a homogeneous selection of data stores as well as single (federated) and multiple (polyglot) query interfaces respectively [6]. Given that the issues addressed in our research impact both MultiStores and PolyStores alike, we will employ the term 'PolyStores' to encompass both system classes throughout the subsequent discourse for clarity and consistency.

Regarding their architecture, most PolyStores use a Mediator-Wrapper Architecture [10] and can be distinguished based on (1) how tightly-coupled the mediation layer is with the underlying data stores, (2) how they model the relationships between the different schemas, (3) how they combine data from different stores using different join algorithms, or (4) how they optimize queries across multiple stores to leverage the stores' strengths [11, 12].

3. State of the Art

In the following, we want to deep dive into an exemplary selection of PolyStores to highlight how these systems work, what they offer, and in which areas the current research lacks.

Apache Wayang, originally developed as Rheem [13], is a platform-agnostic data analytics framework which decouples the application from the underlying data store and places itself in the middle to perform the cross-platform data processing [9]. Data processing jobs are written as Wayang plans which are then mapped from

platform-agnostic to platform-specific operators and optimized based on cost estimates. The migration of data is done without materialized views or permanent data migration but on-demand if data is needed to be shipped from one store to another. The migration and transformation of the data is implemented by using a directed conversion graph, which defines the transformation rules from one model to the other.

BigDAWG, a PolyStore developed at MIT and Northwestern University, is designed as a 3-layer architecture [8]. The first layer contains applications and visualizations whereas the second layer represents a mediation layer responsible for query planning and optimization. The third layer is formed of so-called islands of information. Each island is built of a data model, a query language and a set of underlying data stores.

Without a consideration of write operations or data placement, the sole focus of the system is the processing of complex, island-overarching read operations. While querying a specified island using so-called SCOPE-operations, data shipping can be manually initiated between different islands using CAST-operations. However, transformations from the data model of one island to the model of another island have to be defined manually by the user. Query planning and optimization in BigDAWG is achieved by building a performance catalog of query plans. In addition, She et al. introduce the concept of **semantic equivalence** for query planning according to which "Semantically equivalent queries [...] are substitutable" [14].

Polypheny was first introduced in [7] as a project of the University of Basel and later transformed into a company². From their original vision, a highly adaptive PolyStore, the current release³ already supports a variety of data stores with different data models such as PostgreSQL for multimodel, MongoDB for document and Neo4j for graph data. Schema information is managed using name spaces, each of which has its own data model. Datasets are assigned manually to one of these name spaces as well as one or several data stores. Queries can be formulated using several languages such as SQL, CQL, and Cypher. Additionally, it is possible to formulate name space or store overarching queries for some languages (e.g., SQL). However, the processing of these queries was, at least in our test of the available prototype, unstable and the transformation between different data models rudimentary. For instance, when querying data residing in a document store (e.g., MongoDB) with SQL, the data is transformed into the relational model in a way that leads to a table with one column containing complete documents.

¹Polylingual system are named Polyglot Systems in [6] and some other publications.

²<https://polypheny.com/en/>

³<https://github.com/polypheny/Polypheny-DB/releases/tag/v0.9.1> (v0.9.1)

4. Data Placement

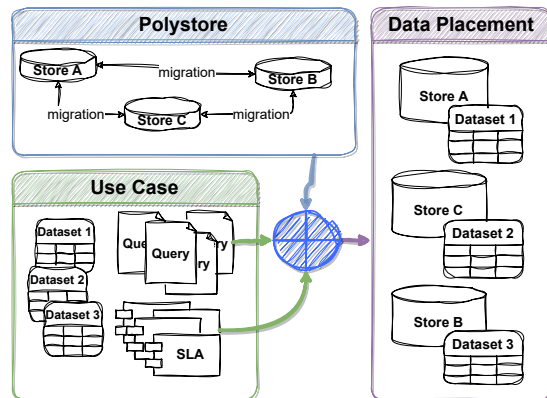


Figure 2: Operational costs of the PolyStore and use case information are used to compute the best data placement.

While the placement of operators on data stores for PolyStores has already been the topic of current research [15, 16, 9], a suitable assignment of datasets (e.g., relational tables, MongoDB collections, or graph nodes with the same label) to data stores is still one of the key challenges. A poor choice in this regard might prevent the polyglot system from exploiting all the capacities of the underlying data stores. Choosing the right store for a dataset comes into play in the following situations: (1) A new dataset is inserted, (2) existing data records are updated or deleted significantly changing a dataset’s characteristics, (3) the workload (and thus the access patterns) changes, (4) the users’ SLAs, such as certain consistency guarantees or runtime requirements, change, or (5) the underlying architecture changes (e.g., a new data store is added).

In all cases, the best assignment for a given dataset depends on (1) the architectural characteristics of the system, (2) the intended workload for this dataset and (3) predefined SLAs. The impact of these three aspects as well as first ideas to combine them within one model, are illustrated in Figure 2. Within this section, we will first present an attempt to tackle the problem of data placement and then discuss all three aspects in more detail with respect to our proposed approach.

4.1. Defining a Data Placement Model

To be able to provide an optimized placement of datasets in data stores, a model has to be created that incorporates the characteristics of the related PolyStore and requirements defined by the intended use case. In a first attempt, the placement itself can basically be described as a binary assignment problem. In [17], Tu et al. already used a sim-

ilar approach for the assignment of encryption methods to columns of relational tables.

In our context, we propose the following formulation of a mathematical binary optimization problem to compute an optimal set of dataset assignments X :

$$\operatorname{argmin}_X \sum_{d \in D} \sum_{s \in S} \sum_{o \in O} w_{do} c_{os} x_{ds} \quad (1)$$

Here, D contains the datasets that should be assigned, S is the set of available data stores and O is the set of supported operators. Furthermore, c_{os} are the costs derived from the PolyStore measurements (Subsection 4.2) and the weights w_{do} model the distribution of operations for each dataset d (Subsection 4.3). x_{ds} is the resulting assignment being 1 if dataset d is assigned to data store s and 0 otherwise.

In order to ensure that every dataset is assigned to exactly one data store, the following constraint is added to the minimization:

$$\forall d \in D: \sum_{s \in S} x_{ds} = 1 \quad (2)$$

Up to this point, only the characteristics of the PolyStore and the workload of the use case define the data placement. To also ensure compliance with the provided SLAs, further constraints are added to the minimization. These will be discussed in more detail in Subsection 4.4.

4.2. Cost Measurement of PolyStore Characteristics

To be able to estimate the costs of operations within a polyglot system, it has to be determined if and how an operation can be processed. Based on results from [12], the analyzed PolyStores use up to four possible ways of computing operations:

1. One of the underlying data stores is able to serve the operation directly so that it can be pushed down and computed there.
2. A data store does not offer the given operation but it can perform a semantically equivalent sequence of operations instead.
3. There exists a semantically equivalent sequence where different parts of the sequence can be performed on different data stores with data being shipped between these stores.
4. There exists a semantically equivalent sequence where parts of the sequence can be performed within some of the underlying data stores and the final operations can be performed within the mediation layer.

```

1 SELECT a.row_num, b.col_num,
2        SUM(a.value * b.value)
3 FROM a, b
4 WHERE a.col_num = b.row_num
5 GROUP BY a.row_num, b.col_num;

```

(a) using relational databases

```

1 multiply(a, b)

```

(b) using SciDB

Figure 3: Semantically equivalent formulations of a multiplication of matrices a and b

Choosing between these four possibilities is a key challenge in query optimization and is highly intertwined with the placement of datasets in data stores. To be able to choose the best solution, costs such as operation runtime, memory consumption, latency, and even transformation and migration costs have to be considered during the assignment of datasets. We are currently working on assessing these costs for different PolyStores.

4.3. Workload Analysis

Information on how a dataset will be processed for a given use case is an important factor for the optimality of a data placement. The question here is not only how an operation can be performed but also how often it occurs compared to others. Therefore, it is necessary to extract the distribution of operations for each dataset from a predefined sample workload.

To assess a given workload while staying independent of the chosen query language and its individual capacities, a **unifying abstract model** for queries is necessary. Hence, we currently develop a tree model that is enriched by optional schema information. In order to be able to correctly combine the weights (w_{do}) extracted from the tree with the operational costs, we enable the system to **find semantically equivalent sequences** (e.g., using pattern matching) and to **replace them by a single, more complex operation**. Figure 3 shows two formulations of a matrix multiplication. Here, the sequence of cross product, selection, and aggregation will be replaced within our model by a multiply operation.

Factoring more complex operations into the distribution derived from the workload, will facilitate the calculation of operation costs as it enables the system to replace the costs for one operation by the costs of a sequence of operations and migrations. Therefore, all ways of computing operations described in Section 4.2 are considered in the overall minimization.

4.4. Service Level Agreements

As outlined in Subsection 4.1, necessary characteristics of data processing have to be considered when optimizing the placement of datasets. A broad overview of SLAs can be found in [2]. While the need for some of these functionalities, such as the applicability of graph algorithms or the processing of joins, are already captured in the weights extracted from the workload, other demands such as transactions or consistency guarantees have to be incorporated differently. Within our proposed data placement model, we guarantee that the resulting data placement satisfies all defined SLAs by adding the following set of constraints:

$$\forall d \in D: \forall c \in C_d: \sum_{s \in S \setminus S_c} x_{ds} = 0 \quad (3)$$

Here, D contains all datasets to be assigned, whereas C_d is the set of SLAs required for a certain dataset d . S is the set of supported data stores, S_c is the set of stores supporting a given SLA c and x_{ds} represents the assignment. The given formulation implies that the sum of assignments of d to data stores not supporting a given SLA (e.g., strong consistency) is 0. Therefore, datasets cannot be assigned to these data stores.

4.5. Usability, Practicability and Future Directions

With our proposed optimization model, we present a first solution to tackle the problem of data placement in PolyStores. It can be used for both, the initial placement of data by solving the optimization in advance and for adapting the placement to a changing workload by measuring the workload during a predefined time interval and adjusting the weights accordingly.

Since the number of data stores $|S|$ supported by one PolyStore is rather small and the number of datasets $|D|$ is unlikely to exceed a few hundred, the computational effort for the optimization is manageable in practice. Especially in the case where a dataset is assigned to exactly one store, there are only $|S| \times |D|$ possible solutions.

Nevertheless, there are some drawbacks, that have to be solved in future work: (1) Currently, the costs are only defined by data store and operation. Here, (pre-)known characteristics of the datasets and its attributes with regard to certain operations, such as selectivity, have to be incorporated. (2) Replication and vertical partitioning of datasets across different data stores might be desirable.

5. Constraint Management and Data Migration

As motivated in Section 1, developer are confronted with a multitude of data stores and a demand for using different stores in parallel to benefit from their respective strengths [2, 5]. PolyStores pose a possible solution for developers who don't have the resources or knowledge to be well-versed with all these different data stores and their quirks by managing all data stores transparently under a mediation layer [12]. However, transparency introduces a new problem that PolyStores have to deal with: What if users want guarantees (e.g., foreign key or check constraints) but know nothing about the underlying data store landscape, the capabilities of the stores, or even the location of their data? In this case, PolyStores need mechanisms to either delegate these requirements to the stores or implement missing constraints in the mediation layer. In the following section, we will discuss challenges that arise in this context and discuss possible solutions for them.

5.1. Constraint Modeling

For a PolyStore to delegate constraint handling or support store capabilities, we need a better grasp of what constraints are supported by which data stores. We focus on popular data stores, such as MongoDB, Neo4J, and PostgreSQL, to cover a broad selection of data models. Especially NoSQL systems lack support for constraints on their data, as these systems focus more on flexible or loosely connected data models [2]. Additionally, we have to consider support for constraints that concern multiple data stores such as foreign key or check constraints that reference data in other stores.

For this purpose, we propose a mediation layer component orchestrating constraints by adding support (1) for missing constraints on stores who lack them and (2) for cross-data-store constraints not achievable on a single store. As these data models oftentimes were not designed with full integrity control in mind in order to maximize case-specific performance benefits, such an approach will add overhead to the whole system's performance. To minimize this overhead a strong priority lies on implementing missing constraints by methods provided by the stores themselves if possible.

To better handle the constraint management in the mediation layer some kind of catalog of the data residing in the stores is required. Meta-modeling the data is a promising approach which can be used to (1) catalogue the data in a single schema, (2) transform the data (and potentially the constraints as well) efficiently between stores by avoiding direct store-to-store transformations (3) and finally build the foundation of migrating data as well as the imposed constraints from store to store. In

recent years multiple approaches for meta-models for polyglot persistence were proposed, such as U-Schema [18], M-Schema [19], or the Canonical Models used by [20] for data transformation and migration. It is our goal to implement our constraint approach into a suitable meta model to build upon already available work and enhance its capabilities to our needs. For this, a meta-model should fulfill a number of criteria including: (1) Given the focus on the data stores PostgreSQL, Neo4J and MongoDB, the meta-model must include the relational, the document, and the graph model. (2) For the focus on integrity constraints as well as an efficient validation of them, the transformation between meta-model and data store model should generate very "natural" data in the sense that e.g. data transformed from graph to relational is similar to data that was intended and designed for the relational model in mind. If graph nodes and edges were to be put into two simple tables (one for nodes and one for adjacency), the validation of constraints – which are typically defined on single entity types – will become very inefficient because the corresponding tuples would have to be identified first.

Migrating data between data stores to place it where it can be handled most efficiently is a key factor to enable PolyStores to adapt themselves to changing requirements. Ultimately, the need for migration stems from the data placement problem already discussed in Section 4. In combination with the previously established reasons for introducing constraint handling into these systems (e.g., transparency), one has to preserve and transform constraints in the migration process as well. A further challenge with the migration process lies in how to verify that the transformed constraints are still semantically equivalent to the old ones. When transforming data between different data models while keeping the constraints intact one must take the different semantics of the models into account (e.g., null values). The authors of the query language SQL++ which is part of the MultiStore FORWARD [21] include variable semantic definitions per query to bridge the different data model semantics. As SQL++ aims to provide an SQL-like, implementation-agnostic experience which is detached from the actual data store, the addendum at the end of each query seems like a suitable solution. However, if implemented as part of a PolyStore mediation layer such semantics can be baked directly into the system. This shows that a migration process with constraints needs to consider the different semantics of the underlying data stores and their data models to ensure equivalence to work properly. From this, different challenges can be identified:

1. Developing an abstract language for modeling constraints that can be integrated into the meta model and translated to the languages of the underlying stores (e.g., SQL check constraints).

2. Developing a mechanism to rewrite constraints affected by a data migration process so that the resulting set of constraints is semantically equivalent to the original set of constraints (i.e., the rewriting is accurate and complete).

5.2. Constraint Validation

As already discussed in [11] and [12], efficient updates over multiple data stores themselves are a hard problem as these involve intermediary results and query steps.

As seen in the two round approach illustrated in Figure 4, in order to execute the update command the mediator has to work in steps: (1) The mediator determines all records that need to be updated and calculates a table with intermediary results (depicted D in Line 2). (2) The intermediary results are then used to update the original record based on their IDs (see Lines 6-7). This approach leads to an updated state in our PolyStore but begs the question of whether a more efficient one-round variant may be possible. This is especially important if we consider more advanced and complicated queries which may need more than two rounds to be solvable. Furthermore, there are few to no PolyStores out there, that support update/insert queries over their heterogeneous data stores.

While the previous section mostly focused on the topic of how to model constraints in a polyglot persistence environment, we have to consider the validation process of constraints, too. This mainly matters during inserting or updating data in our PolyStore and boils down to two core problems we want to focus on:

1. Finding an efficient way to update/insert data across multiple data stores.
2. Incorporating the constraint validation into a polyglot data update/insert process.

The last challenge can be further divided into:

- Efficient validation of constraints involving data distributed across different data stores.
- Consistent execution of referential actions across different data stores under reuse of existing mechanisms within the individual stores.

A naive validation of constraints in the mediator may require loading large amounts of data from the individual data stores. Efficient validation strategies must therefore attempt to minimize such loading with the help of indices (e.g., for unique constraints) and clever query generation.

In summary, developing efficient update/insert as well as constraint validation processes within PolyStores presents considerable challenges. As we move forward, it is imperative to develop mechanisms that not only ensure efficient data handling but also uphold referential integrity and consistency across multiple data stores.

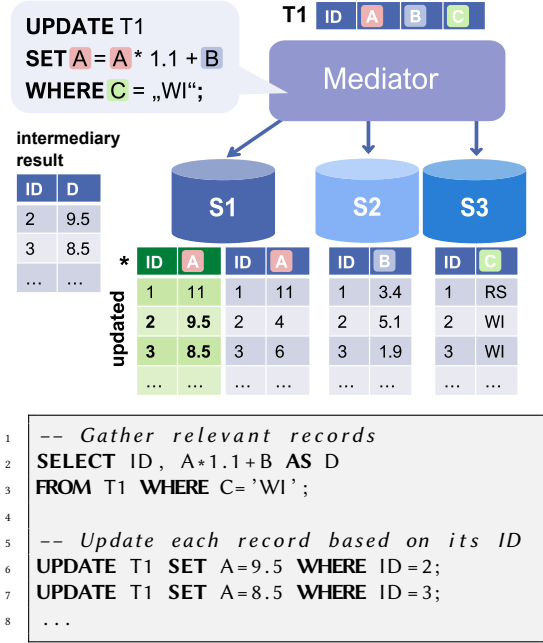


Figure 4: Example for a two-round approach for updates on a PolyStore (Adapted from [11])

6. Conclusion

In this paper, we explained the concepts of Multi- and PolyStores and the open challenges within these systems with respect to data placement and constraint management. Based on our research and the current development direction of state-of-the-art PolyStores, we identified adaptivity as a key factor for future research in this field. To ensure adaptivity, we propose research ideas in the topics of (i) data placement and workload analysis as well as (ii) data migration and constraint management.

For data placement, we plan to incorporate a unifying abstract model into the process of workload analysis. The results of the analytical process and the characteristics of the PolyStore are then cast into an optimization model, that computes an assignment of datasets to data stores.

For a migration process in PolyStores we introduced the idea of an overarching constraint management system as well as the incorporation of a meta-model. Furthermore, we discussed the need for efficient update processes for constraint validation.

Both topics are important step stones on the way to a fully adaptive PolyStores which can fulfill their initial promise to enable users to benefit from modern data store solutions without knowing every nook and cranny of the used technologies.

References

- [1] P. J. Sadalage, M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*, Pearson Education, Boston, Massachusetts, USA, 2012.
- [2] F. Gessert, W. Wingerath, S. Friedrich, N. Ritter, NoSQL database systems: a survey and decision guidance, *CSRD* 32 (2017) 353–365. doi:10.1007/s00450-016-0334-3.
- [3] M. Stonebraker, U. Çetintemel, "one size fits all": An idea whose time has come and gone (abstract), in: K. Aberer, M. J. Franklin, S. Nishio (Eds.), *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, IEEE, 2005, pp. 2–11. doi:10.1109/ICDE.2005.1.
- [4] M. Stonebraker, C. Bear, U. Çetintemel, M. Cherniack, T. Ge, N. Hachem, S. Harizopoulos, J. Lifter, J. Rogers, S. B. Zdonik, One size fits all? part 2: Benchmarking studies, in: *Third Biennial Conference on Innovative Data Systems Research (CIDR)*, 2007, pp. 173–184. URL: <http://cidrdb.org/cidr2007/papers/cidr07p20.pdf>.
- [5] W. Hasselbring, G. Steinacker, Microservice architectures for scalability, agility and reliability in e-commerce, in: *IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 243–246. doi:10.1109/ICSAW.2017.11.
- [6] R. Tan, R. Chirkova, V. Gadepally, T. G. Mattson, Enabling query processing across heterogeneous data models: A survey, in: *IEEE Big Data*, 2017, pp. 3211–3220. doi:10.1109/BigData.2017.8258302.
- [7] M. Vogt, A. Stiemer, H. Schuldt, Polypheny-db: Towards a distributed and self-adaptive polystore, in: *IEEE Big Data*, 2018, pp. 3364–3373. doi:10.1109/BigData.2018.8622353.
- [8] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, S. Zdonik, The BigDAWG Polystore System, *ACM SIGMOD Record* 44 (2015) 11–16. doi:10.1145/2814710.2814713.
- [9] K. Beedkar, B. Contreras-Rojas, H. Gavrilidis, Z. Kaoudi, V. Markl, R. Pardo-Meza, J.-A. Quiané-Ruiz, Apache wayang: A unified data analytics framework, *ACM SIGMOD Record* 52 (2023) 30–35. doi:10.1145/3631504.3631510.
- [10] A. Doan, A. Halevy, Z. Ives, *Principles of Data Integration*, Elsevier, Amsterdam, Netherlands, 2012.
- [11] F. Kiehn, M. Schmidt, D. Glake, F. Panse, W. Wingerath, B. Wollmer, M. Poppinga, N. Ritter, Polyglot data management: state of the art & open challenges, *Proc. VLDB Endow.* 15 (2022) 3750–3753. URL: <https://vldb2022.dbis.hamburg/>. doi:10.14778/3554821.3554891.
- [12] D. Glake, F. Kiehn, M. Schmidt, F. Panse, N. Ritter, Towards polyglot data stores – overview and open research questions, 2022. arXiv:2204.05779.
- [13] D. Agrawal, L. Ba, L. Berti-Equille, S. Chawla, A. Elmagarmid, H. Hammady, Y. Idris, Z. Kaoudi, Z. Khayyat, S. Kruse, M. Ouzzani, P. Papotti, J.-A. Quiane-Ruiz, N. Tang, M. J. Zaki, Rheem: Enabling multi-platform task execution, in: *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)*, ACM, 2016, p. 2069–2072. doi:10.1145/2882903.2899414.
- [14] Z. She, S. Ravishankar, J. Duggan, Bigdawg polystore query optimization through semantic equivalences, in: *IEEE High Performance Extreme Computing Conference (HPEC)*, 2016, pp. 1–6. doi:10.1109/HPEC.2016.7761584.
- [15] D. Glake, M. Schmidt, F. Kiehn, F. Panse, U. Lenfers, T. Clemen, N. Ritter, Operator placement for spatio-temporal tasks, in: *2022 IEEE International Conference on Big Data (Big Data)*, 2022, pp. 281–290. doi:10.1109/BigData55660.2022.10020279.
- [16] Z. Kaoudi, J.-A. Quiané-Ruiz, B. Contreras-Rojas, R. Pardo-Meza, A. Troudi, S. Chawla, ML-based cross-platform query optimization, in: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 1489–1500. doi:10.1109/ICDE48307.2020.00132.
- [17] S. L. Tu, M. F. Kaashoek, S. R. Madden, N. Zeldovich, Processing analytical queries over encrypted data, *Proceedings of the 39th International Conference on Very Large Data Bases (PVLDB)* (2013). doi:10.14778/2535573.2488336.
- [18] C. J. F. Candel, D. Sevilla Ruiz, J. J. García-Molina, A unified metamodel for nosql and relational databases, *Information Systems* 104 (2022) 101898. doi:<https://doi.org/10.1016/j.is.2021.101898>.
- [19] A. Conrad, P. Utzmann, M. Klettke, U. Störl, Meta-models to support database migration between heterogeneous data stores, in: *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems (MODELS): Companion Proceedings*, ACM, 2022, p. 546–551. doi:10.1145/3550356.3561574.
- [20] T. Vanhove, M. Sebrechts, G. Van Seghbroeck, T. Wauters, B. Volckaert, F. De Turck, Data transformation as a means towards dynamic data storage and polyglot persistence, *International Journal of Network Management* 27 (2017) e1976. doi:<https://doi.org/10.1002/nem.1976>.
- [21] K. W. Ong, Y. Papakonstantinou, R. Vernoux, The SQL++ semi-structured data model and query language: A capabilities survey of sql-on-hadoop, nosql and newsql databases, 2014. arXiv:1405.3631.