Polystore Systems and DBMSs: Love Marriage or Marriage of Convenience?

Marco Vogt, David Lengweiler, Isabel Geissmann, Nils Hansen,
Marc Hennemann, Cédric Mendelin, Sebastian Philipp, Heiko Schuldt

Databases and Information Systems Research Group
Department of Mathematics and Computer Science, University of Basel, Switzerland
{firstname.lastname}@unibas.ch

Abstract. Polystore systems allow to combine different heterogeneous data stores in one system and also offer different query languages for accessing data. While this addresses a large number of requirements especially when providing access to heterogeneous data in mixed workloads, most polystore systems are somewhat limited in terms of their functionality. In this paper, we make the case to 'upgrade' polystore systems towards full-fledged databases systems, leading to the notion of *PolyDBMSs*. We summarize the features of such PolyDBMSs and exemplify the implementation on the basis of our PolyDBMS Polypheny-DB.

Keywords: Polystore Systems · Database Management Systems.

1 Introduction

In the last years, *polystore systems* have become popular as an attempt to bridge the heterogeneity of data models and to combine the best-of-breed in a single system – by seamlessly integrating the concepts of multistore databases and polyglot persistence [3]. A multistore database system combines heterogeneous data stores and manages data across these stores by offering a single query interface and a single query language. Polyglot persistence offers different query languages for accessing data. Most existing polystore systems focus on selected aspects but do not provide the full-fledged feature set of a database system.

According to a summary given in [2], Ted Codd identified the following functionality a full-fledged database system has to provide: (i) *Storage* of data (ii) *Retrieval* and *update* of data (iii) *Access* support from remote locations (iv) User accessible *metadata catalog or data dictionary* (v) Support for *transactions and concurrency* (vi) Facilities for *recovering* the database in case of damage (vii) Enforcing *constraints*, and (viii) Support for *authorization* of access and update of data. When deploying polystore systems in real world applications, it has turned out that the full DBMS functionality is required, not (only) the support for heterogeneous data stores and different query languages.

In this paper, we make the case for upgrading polystore systems to full-fledged databases – for which we introduce the term *PolyDBMSs* – and we discuss the challenges for the different database functionality. We exemplify this on the basis

of Polypheny-DB [4, 5], the polystore database system we have introduced in our previous work. Polypheny-DB has been published under an open source license[1] and participates in 2021 to the Google Summer of Code (GSoC) program.

The contribution of the paper is twofold: first, we identify the challenges polystore systems have to meet to provide the features of a full-fledged DBMS. Second, we exemplify based on Polypheny-DB how these feature can be provided.

2 From Polystore Systems to "PolyDBMSs"

In this section, which is organized along Codd's DBMS features as summarized in [2], we discuss the challenges for polystore systems in general, leading to a novel kind of *PolyDBMS*, and how they are addressed in Polypheny-DB.

2.1 Storage of Data

PolyDBMSs need to support different data storage engines optimized for various types of data and workloads. These data stores are internally based on different data models (relational, documents, wide-columns, key-values, graphs, etc.) and are queried using different query languages and methods. This is an inevitable feature all PolyDBMSs have to provide intrinsically.

Polypheny-DB currently supports relational, document, and wide-column stores and different data sources. The connection to data stores and data sources is handled by *adapters*. *Data Stores* are used as physical storage and execution engines and are fully maintained by and under exclusive control of Polypheny-DB. In order to be able to guarantee correctness, the stores are only accessed through Polypheny-DB. *Data Sources* allow mapping data on (remote) database systems into the schema of Polypheny-DB. There are also adapters for querying file systems or CSV files. Data source adapters are less complex than data store adapters and usually only support a subset of the functionality. Polypheny-DB allows that data sources are queried by other systems in parallel. Hence, Polypheny-DB does not provide support for constraints or data replication/partitioning on entities originating from data sources, only for the ones from data stores. The optimization offered by a storage system can be leveraged when pushing down a complete query (or at least parts of it) whenever possible. In order to optimize the data transfers, query results are read on demand.

2.2 Retrieval and Update of Data

PolyDBMSs intrinsically need to support the retrieval of data using multiple query languages and methods. Furthermore, PolyDBMSs should also offer data modification queries – which usually goes beyond the feature set of polystores.

Polypheny-DB supports DML and DDL operations. The most mature query language supported by Polypheny-DB is its own SQL dialect *PolySQL*. It features a common set of operations including `JOIN`, `GROUP BY` and `HAVING` clauses,

[1] https://github.com/polypheny/Polypheny-DB

set operations, inner queries and `WITH` clauses. Additionally, it provides a large set of query and aggregation functions [2] and it comes with functions specifically for media and blob data. Furthermore, Polypheny-DB supports a distance function for k-NN similarity search. Polypheny-DB also supports the *MongoDB Query Language*. Moreover, support for the Contextual Query Language is currently being added. With the *Explore-by-Example* interface and the *Dynamic Query Builder*, Polypheny-DB also supports two innovative query methods [4].

2.3 Access Support from Remote Locations

PolyDBMSs should offer the query functionality identified in Section 2.2 also from remote locations by offering appropriate APIs and query interfaces.

The JDBC interface of *Polypheny-DB* supports the retrieval of meta data and the control of transactions. It also provides prepared statements and batch inserts and updates. The REST-based query interface allows accessing and modifying data using `GET`, `POST`, `PATCH`, and `DELETE` requests. Results are returned as JSON.

2.4 User Accessible Metadata Catalog or Data Dictionary

In addition to the usual metadata maintained by a DBMS, *PolyDBMSs* also need to keep metadata on data distribution across different data stores.

Polypheny-DB comes with a data dictionary that has a browser-based user interface (Polypheny-UI). It allows to view and alter the schema. and it can be used to browse and modify the data, manage data stores and data sources, and execute queries using the supported query methods and languages. In addition to accessing schema information using Polypheny-UI, it is also possible to retrieve the schema using the JDBC meta functions provided by our JDBC driver.

2.5 Support for Transactions and Concurrency

PolyDBMSs need to offer transaction support at their interface. This is particularly relevant when data accessed within an application is internally spread across several data stores.

Polypheny-DB supports concurrent queries, guaranteeing atomicity and isolation using transactions for data stored on its underlying data stores. For data stored on data sources, support for transactions can be limited and depends on the capabilities of the data source. The isolation of concurrent transactions is ensured on the polystore level. Due to data partitioning and replication, only the polystore has the necessary information for ensuring the isolated execution of transactions. Locking on the underlying data stores is deactivated for performance reasons whenever possible. Polypheny-DB uses *strong strict two-phase locking (SS2PL)* [1] for the isolation of concurrent transactions. The SS2PL implementation in Polypheny-DB comes with the necessary deadlock detection.

[2] https://polypheny.org/documentation/PolySQL/Operators/

2.6 Facilities for Recovering the Database in Case of Damage

PolyDBMSs need to support two types of failure cases: (i) failures of the PolyDBMS as a whole and (ii) failures of single data stores / data sources.

Polypheny-DB distinguishes between data recovery in the underlying data stores, and schema recovery, which includes data placement (i.e., the physical schema mapping). For *data recovery*, Polypheny-DB assumes that the selected underlying data stores work correctly and thus delegates recovery there. For the data stores integrated in Polypheny-DB (e.g., file store), a proper recovery mechanism is implemented in the adapter. *Schema recovery* is under the responsibility of Polypheny-DB. It leverages the whole catalog containing the schema information which is persistently stored using a transactional storage system featuring a write-ahead log. On start-up, all persistent placements of an entity are restored. For entities without a persistent placement, only the schema is restored.

2.7 Enforcing Constraints

PolyDBMSs need to enforce constraints that span two or more data sources, not just constraints within a single store that are natively enforced there.

Polypheny-DB enforces primary key, foreign key, and uniqueness constraints. The enforcement is done on the polystore level by extending the query plan. The major challenge with implementing constraint enforcement on the polystore level is that constraints need to be enforced even if data is stored on data stores that do not natively support constraints. Furthermore, data can be partitioned across multiple data stores which makes the full delegation of constraint enforcement to the underlying data stores unfeasible. Hence, constraint enforcement may only (partly) be delegated to underlying stores whenever applicable.

2.8 Support for Authorization of Access and Update of Data

In a *PolyDBMSs*, each single store is supposed to provide necessary mechanisms for authorizing accesses. In addition, this support also needs to be provided globally at the PolyDBMS level.

Polypheny-DB supports basic authentication, but there is not yet a complete mechanism for a role or user-based authorization of specific actions.

3 Conclusion

Polystore systems combine several distributed and potentially heterogeneous data stores underneath one or several interfaces. Usually, even though the data stores might be full-fledged database systems, polystore systems lack one or several features of a complete DBMS. In this paper, we make the case to 'upgrade' polystore systems to full-fledged DBMSs, leading to the notion of *PolyDBMSs*. We have surveyed the requirements at the PolyDBMS level and we have briefly presented how these challenges have been implemented in Polypheny-DB, which combines the advantages of a polystore system with the ones of a DBMS.

Acknowledgments: This work has been partly funded by the Swiss National Science Foundation, project Polypheny-DB (contract no. 200021_172763).

References

1. Bernstein, P., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley Longman, Boston, MA, USA (1987)
2. Connolly, T., Begg, C.: Database Systems: A Practical Approach to Design, Implementation, and Management. Pearson (2014)
3. Tan, R., Chirkova, R., Gadepally, V., Mattson, T.G.: Enabling Query Processing Across Heterogeneous Data Models: A Survey. In: Proceedings of the 2017 IEEE International Conference on Big Data (BigData 2017). pp. 3211–3220. IEEE, Boston, MA, USA (2017). https://doi.org/10.1109/BigData.2017.8258302
4. Vogt, M., Hansen, N., Schönholz, J., Lengweiler, D., Geissmann, I., Philipp, S., Stiemer, A., Schuldt, H.: Polypheny-DB: Towards Bridging the Gap Between Polystores and HTAP Systems. In: Heterogeneous Data Management, Polystores, and Analytics for Healthcare. pp. 25–36. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-71055-2_2
5. Vogt, M., Stiemer, A., Schuldt, H.: Polypheny-DB: Towards a Distributed and Self-Adaptive Polystore. In: 2018 IEEE International Conference on Big Data. pp. 3364–3373. IEEE (2018). https://doi.org/10.1109/BigData.2018.8622353