# Polyglot Data Management: State of the Art & Open Challenges

Felix Kiehn
University of Hamburg, Germany
felix.kiehn@uni-hamburg.de

Mareike Schmidt
University of Hamburg, Germany
mareike.schmidt-3@uni-hamburg.de

Daniel Glake
University of Hamburg, Germany
Hamburg University of Applied Science, Germany
daniel.glake@uni-hamburg.de

Fabian Panse
University of Hamburg, Germany
fabian.panse@uni-hamburg.de

Wolfram Wingerath
University of Oldenburg, Germany
wolle@uni-oldenburg.de

Benjamin Wollmer
University of Hamburg, Germany
Baqend, Germany
benjamin.wollmer@uni-hamburg.de

Martin Poppinga
University of Hamburg, Germany
martin.poppinga@uni-hamburg.de

Norbert Ritter
University of Hamburg, Germany
norbert.ritter@uni-hamburg.de

## ABSTRACT

Due to the increasing variety of the current database landscape, polyglot data management has become a hot research topic in recent years. The underlying idea is to combine the benefits of different data stores behind a predefined set of common interfaces and thus address use cases that individual stores cannot meet. This can be accomplished using different approaches which vary greatly in terms of capabilities, functionality, and architectural concepts. This tutorial provides a detailed overview of the current state of research in polyglot data management. We motivate its use by showing the high diversity of existing data stores and discussing three use cases in which individual stores are insufficient. Thereafter, we present different taxonomies for classifying polyglot data systems and give a detailed review of a number of selected systems. Finally, we compare these systems based on their features and discuss open challenges that still need to be addressed in future research.

**PVLDB Reference Format:**
Felix Kiehn, Mareike Schmidt, Daniel Glake, Fabian Panse, Wolfram Wingerath, Benjamin Wollmer, Martin Poppinga, and Norbert Ritter. Polyglot Data Management: State of the Art & Open Challenges. PVLDB, 15(12): 3750 - 3753, 2022.
doi:10.14778/3554821.3554891

## 1 INTRODUCTION

Data is nowadays processed and managed in many different contexts with often completely different requirements (e.g., consistency vs. scalability, real-time vs. batch, write heavy vs. read heavy workloads, key-based access vs. analytical queries). As a logical consequence, many new data management and (distributed) data processing systems[1] have been developed over the last two decades, and have established themselves alongside the still widely used relational database management systems. Examples of such systems are key-value stores (e.g., Redis), document stores (e.g., MongoDB), wide-column stores (e.g., Cassandra), graph databases (e.g., Neo4J), time-series databases (e.g., Influx), real-time databases (e.g., Firebase), array-databases (e.g., SciDB), full-text search engines (e.g., Elasticsearch), and distributed computation frameworks (e.g., Apache Spark). All these systems have been designed with different applications in mind and it is up to the users to decide which system is best suited for their specific use cases. However, the sheer volume of potential candidates and the everlasting evolution of use cases poses several challenges to such a selection process:

- **Confusing System Space:** Even as an expert, it is hard to keep an overview of all these systems and their respective sweet spots and limitations (although some guidelines [9] are available).
- **Complex Trade-Offs:** Complex applications may have subcomponents with contradictory requirements so that using a single data store always ends up in a trade-off between them.
- **In-Flux Requirements:** As applications change and evolve, so do their requirements. Thus, the initially selected data store may not remain the best fitting one over time. However, those changes are often not immediately visible to the users and may therefore remain undetected for a long time.

The approach of *polyglot data management* [25] aims to solve these problems by combining the benefits of several data stores (and their underlying techniques) without adopting their drawbacks and to hide the automatic coordination across these stores behind single or multiple interfaces. This has several advantages:

- **Unified Access:** Users only need to acquire knowledge about a few interfaces, which in the best case are also standardized.
- **Transparent System Evolution:** One can react internally – preferably automatically – to changing requirements by migrating data between data stores or even exchanging entire stores without users being affected.
- **Ease-of-Extensibility:** Newly developed data stores can be easily integrated into the existing polyglot data system, without having to completely rebuild existing structures.


Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554891

[1] In the following we summarize both kinds of systems under the term *data store*.

Over the last decade, this basic idea has been implemented in the form of various systems, such as HadoopDB [1], BigIntegrator [34], PolyBase [6], Myria [31], FORWARD [23], BigDAWG [8], RHEEM [2], CloudMdsQL [17], or ESTOCADA [3]. Although most of these systems are based on the mediator-wrapper architecture [7], there are major differences between their implementations and limitations. In this tutorial, we give an overview of the current state-of-the-art in polyglot data management. We will (i) introduce key terminology, taxonomies, and techniques, (ii) discuss the features, advantages, and disadvantages of different existing systems, and (iii) present open challenges.

## 2 TUTORIAL OUTLINE

This tutorial is split into six parts:

(1) **Motivation & Use Cases.** We introduce the underlying idea of polyglot data management and motivate work in this area by presenting three real-life use cases where individual data management solutions are not capable of meeting all of the given requirements.
(2) **Terminology & Taxonomies.** We introduce pivotal terms and present various taxonomies according to which polyglot data systems can be classified and compared.
(3) **Overview Database Landscape.** We give an overview of the different classes of currently used data stores. In addition, we delineate their respective advantages and disadvantages, and discuss how their individual design characteristics are linked to concrete functional and non-functional application requirements.
(4) **Basic Techniques & Concepts.** We discuss several concepts and techniques from distributed data management [24] and data integration [7] that are reused in solutions for polyglot data management, such as the mediator-wrapper architecture and operators for distributed data computation.
(5) **Current Systems.** In the main part of this tutorial, we present the key concepts of several existing polyglot data systems, such as PolyBase [6] and RHEEM [2]. We provide an overview of their respective architecture, functionality, and capabilities. We also analyze their respective strengths and limitations, from which we then derive potential use cases.
(6) **Open Challenges.** We close with a set of desiderata for polyglot data management, evaluate existing systems based on these desiderata, and discuss open problems based on the results of this evaluation.

## 3 GOALS AND OBJECTIVES

This section provides details about the tutorial parts.

### 3.1 Part 1: Motivation & Uses Cases

We first motivate polyglot data management based on three real-world use cases, each covering a different type of application:

**E-Commerce and Customer Management:** An online store provides users with products using text descriptions, ratings, short films, and pictures, with a high availability requirement to serve purchasing services constantly. The different components require different levels of consistency (e.g., while product ratings tolerate low consistency in favor of partition tolerance and high availability, payments require high consistency). The high read throughput created by users requires a read-friendly distribution scheme. In addition to the typical OLTP traffic, analytical tasks (data mining & OLAP) are required to compute product recommendations and calculate annual business numbers.

**Agent-Based Simulation:** A mobility simulation for cities needs to process different kinds of agents (e.g., citizens or cars) which move in a spatial environment or along road networks, and whose decisions are affected by stationary objects, events, and other agents. Thus, it combines data about different agent types with graph and grid data, and requires complex data operations, such as routing or spatial filtering. Depending on the selected time-scale and spatial extension, the simulation produces an extensive set of diverse simulation results and thus a high write throughput. In addition, to display aggregated indicators and visualize moving agents on a map, data objects are streamed through the system at runtime.

**Healthcare Data Management:** A hospital needs to store and query detailed information on patients, laboratory, and radiology results, as well as doctors' and nurses' notes regarding observations and treatments. Thus, medical data (e.g., MIMIC II [19]) ranges from structured and unstructured data (patient data, lab results) to images and waveform data (CT and MRI images, electrocardiograms). The analysis of images and waveform data requires complex operations such as Fourier-Transformations (e.g., to find regions containing abnormal tissues). In addition, text search and graph algorithms are required (e.g., to find similar examination reports or analyze correlations between symptoms). To keep patients in an intensive care unit under continuous surveillance, real-time monitoring is essential. In case of emergencies, it is necessary that vital data, e.g., blood types, are available at any time.

### 3.2 Part 2: Terminology & Taxonomies

Like federated databases [24] (and in contrast to multi-model data stores [20]), polyglot data systems provide a unified access on multiple data stores. However, in contrast to federated databases, they are designed to deal with data-store-heterogeneity on technical (e.g., architecture) and conceptual levels (e.g., data model, query language). Despite this common ground, the term *polyglot data management* has often been used with different meanings. To get a clear overview of potential interpretations, we present three taxonomies that assign existing systems to clear classes.

The first taxonomy distinguishes them based on the number of query interfaces they provide:

- **Multistores** provide access through a single query language.
- **Polystores** allow users to submit queries in different languages (e.g., the native languages of the individual stores).

The second taxonomy takes the internal architecture of these systems into account. Here we distinguish between:

- **Loosely-coupled** systems correspond to a loose network of already existing and often independently managed data stores where the mediator has read rights, but no rights to write data or reconfigure the individual stores. In general, they bear a strong resemblance to virtual integration systems.
- **Tightly-coupled** systems were often designed from the beginning as one overall system. The mediator has write permissions on the individual stores and is therefore also able to migrate

data between them. In this setting, the individual stores are typically not autonomous, so that the mediator is able to adapt their configurations to its own requirements.
- **Hybrid** systems are those in which some stores are tightly coupled and others are loosely coupled.

Both taxonomies are used to classify existing systems in Table 1. The last taxonomy, not included in Table 1 due to space limitations, classifies these systems based on the degrees of (i) **heterogeneity**, (ii) **autonomy**, (iii) **transparency**, (iv) **flexibility**, and (v) **optimality** they support [28].

### 3.3 Part 3: Overview Database Landscape

Since the emergence of various application areas with unique requirements has led to the development of many highly specialized systems, the current data management landscape is more diverse and heterogeneous than ever before. The focus of these individual systems can vary wildly: While some abandon consistency and querying power in favor of schema-freeness and scalability, others handle specific data types (e.g., spatio-temporal), data models (e.g., graphs), or combine transactional with analytic processing (HTAP). This potpourri of different systems provides the possibility to adapt data management to the requirements of the own application. However, it also complicates the selection of a suitable system, since an overview of advantages and disadvantages of these systems is hard to procure due to their sheer mass. In addition, there are – as our examples in Section 3.1 show – numerous use cases in which requirements of different subareas are combined and thus none of these systems can be used in these use cases standalone.

In this part of the tutorial, we give an overview of the current data management landscape, including different classes of data stores (NoSQL, real-time, etc.) and data processing frameworks (e.g., Hadoop, Spark, Flink). Moreover, we present the strengths and weaknesses of the different classes of NoSQL data stores regarding different user requirements (functional and non-functional) [9].

### 3.4 Part 4: Basic Techniques & Concepts

The idea of polyglot data management is usually implemented using the mediator-wrapper architecture, which is well-known from virtual data integration systems [7]. Thus, the schemas of the individual data stores are typically mapped to a (virtual) mediated schema, e.g., via tuple-generating dependencies [29] or views [15]. These mappings are used to decompose user queries into several subqueries, each forwarded to another data store, and to combine their results to a single (consistent) set of query answers. The result of such a query rewriting is one or more *query execution plans* (QEPs), each following predefined optimization goals and determining the most efficient join-order to integrate the relevant data of the respective stores. When building the QEPs, the mediator must decide where to execute the individual operators based on the store-specific location of the processed data, also known as the *operator-placement problem* [27]. Since this problem is NP-hard, the optimal placement needs to be approximated [18].

In this part of the tutorial, we present basic knowledge on (i) the mediator-wrapper architecture, (ii) schema mapping languages, (iii) operators that allow an efficient combination of data across stores, such as bind and skew-aware joins [13], (iv) typical cost models used for optimization, and (v) several heuristics to solve the operator-placement problem during query planning.

**Table 1: Existing systems for polyglot data management. The selected representatives are in bold.**

| | *multistore* | *polystore* |
|---|---|---|
| *loosly-coupled* | **PolyBase** [6]<br>BigIntegrator [34]<br>FORWARD [23]<br>Apache Drill (Calcite) [4]<br>QoX [26]<br>QUEPA [22]<br>Odyssey [14] | Myria [31] |
| *tightly-coupled* | **RHEEM** [2]<br>MuSQLE [11]<br>HadoopDB [1] | **ESTOCADA** [3]<br>Polypheny-DB [30] |
| *hybrid* | **CloudMdsQL** [17]<br>SparkSQL [5] | **BigDAWG** [8] |

### 3.5 Part 5: Current Systems

Polyglot data management is a subject of intense research, and various systems have been developed in recent years. Table 1 lists some popular systems classified based on the first two taxonomies we have introduced in Section 3.2. In this part, we give a deeper insight into the architecture and functionality of some of these systems. The selected systems implement highly diverse concepts and cover different categories from Table 1. These systems are:

- **PolyBase**: a loosely-coupled multistore that allows the integration of unstructured (HDFS) data from Hadoop clusters with relational and NoSQL data by using SQL [6]. PolyBase is now an integral part of MS SQL Server[2].
- **RHEEM** (now **Apache Wayang**[3]): a tightly-coupled multistore that utilizes ML techniques to learn costs for data shipping operators from already existing execution logs [2, 16, 18].
- **ESTOCADA**: a tightly coupled polystore that automatically distributes data across multiple non-autonomous data stores (relational and NoSQL) to optimize query performance [3].
- **CloudMdsQL**: a hybrid multistore whose query interface uses a nesting approach to combine several native query languages within a single one [17].
- **BigDAWG**: A hybrid polystore with support for several data models (including their respective query languages) that is able to migrate data between individual stores [8].

After describing the individual systems, we assess for which use cases they are particularly well suited and for which rather less.

### 3.6 Part 6: Open Challenges

In the final part of the tutorial, we describe a set of properties desirable for polyglot data systems and illustrate them using the three use cases from Section 3.1. This includes (i) **preserving the non-functional capabilities** (e.g., scalability) of each underlying store despite unified access across multiple stores, (ii) support of **ad-hoc data manipulation** through a global interface, (iii) **automatic**

[2]https://docs.microsoft.com/en-us/sql/relational-databases/polybase/ (29.06.2022)
[3]https://wayang.apache.org/ (29.06.2022)

**detection of changes** in system requirements or workloads, and corresponding **adjustments** of cross-store data distribution (permanent data migration between stores) and store-specific **reconfigurations**, (iv) efficient cross-system **query planning and execution** including cost-efficient **data shipping** and **optimal operator placement** under consideration of functional and non-functional system properties, (v) the ability to process data in **real-time** in a polyglot setup, and (vi) **multi-model schema management**.

Thereafter, we compare the systems of Section 3.5 based on these characteristics and highlight common deficits. Finally, we summarize the current status of polyglot data management and discuss the open challenges we have identified in this tutorial.

## 4 RELATIONSHIP TO PRIOR TUTORIALS

The tutorial has not been given before. The only past tutorial addressing a similar subject was held by Lu et al. 2018 [21]. However, while we discuss polyglot data management from the perspective of several functional and non-functional properties, Lu et al. consider it in comparison to multi-model databases and thus focus on the collective use of different data models. Moreover, we present basic knowledge that has not been covered by Lu et al., such as the operator-placement problem and cross-system join operators. Finally, we present five systems in more detail (see Section 3.5). Of these, two were barely (BigDAWG) or not at all (CloudMdsQL) addressed by Lu et al. In addition and like polyglot data management in general, the other three systems (PolyBase, RHEEM, ESTOCADA) have been further developed in the meantime, and thus contain many novel aspects of interest to the audience.

## 5 PRESENTER BACKGROUND

Our team has a solid background in data management through more than 20 years of research at the University of Hamburg, the Hamburg University of Applied Science, and the University of Oldenburg. We address the problems of polyglot data management in both research and practice, both in the multi-agent system MARS[4] and in the area of web technologies in the Backend-as-a-Service company Baqend[5]. Our knowledge on NoSQL, real-time, stream, and polyglot data management has been published in several surveys [9, 12], tutorials [32], and books [10, 33].

## REFERENCES


[1] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. 2009. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *Proc. VLDB Endow.* 2, 1 (2009), 922–933.

[2] Divy Agrawal, Sanjay Chawla, Bertty Contreras-Rojas, Ahmed K. Elmagarmid, Yasser Idris, Zoi Kaoudi, Sebastian Kruse, Ji Lucas, Essam Mansour, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, Saravanan Thirumuruganathan, and Anis Troudi. 2018. RHEEM: Enabling cross-platform data processing - may the big data be with you! -. *Proc. VLDB Endow.* 11, 11 (2018), 1414–1427.

[3] Rana Alotaibi, Damian Bursztyn, Alin Deutsch, Ioana Manolescu, and Stamatis Zampetakis. 2019. Towards scalable hybrid stores: Constraint-based rewriting to the rescue. In *SIGMOD*. ACM, 1660–1677.

[4] Apache Software Foundation. 2022. *Apache Drill.*

[5] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. 2015. Spark SQL: Relational data processing in Spark. In *SIGMOD*. ACM, 1383–1394.

[6] David J. DeWitt, Alan Halverson, Rimma V. Nehme, Srinath Shankar, Josep Aguilar-Saborit, Artin Avanes, Miro Flasza, and Jim Gramling. 2013. Split query processing in polybase. In *SIGMOD*. ACM, 1255–1266.

[7] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. 2012. *Principles of Data Integration.* Morgan Kaufmann.

[8] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stanley B. Zdonik. 2015. The BigDAWG polystore system. *SIGMOD Rec.* 44, 2 (2015), 11–16.

[9] Felix Gessert, Wolfram Wingerath, Steffen Friedrich, and Norbert Ritter. 2017. NoSQL database systems: A survey and decision guidance. *Comput. Sci. Res. Dev.* 32, 3-4 (2017), 353–365.

[10] Felix Gessert, Wolfram Wingerath, and Norbert Ritter. 2020. *Fast and Scalable Cloud Data Management.* Springer.

[11] Victor Giannakouris, Nikolaos Papailiou, Dimitrios Tsoumakos, and Nectarios Koziris. 2016. MuSQLE: Distributed SQL query execution over multiple engine environments. In *Big Data*. IEEE, 452–461.

[12] Daniel Glake, Felix Kiehn, Mareike Schmidt, Fabian Panse, and Norbert Ritter. 2022. Towards polyglot data stores: Overview and open research questions. *CoRR* abs/2204.05779 (2022).

[13] Ankush M. Gupta, Vijay Gadepally, and Michael Stonebraker. 2016. Cross-engine query execution in federated database systems. In *HPEC*. IEEE, 1–6.

[14] Hakan Hacigümüs, Jagan Sankaranarayanan, Jun'ichi Tatemura, Jeff LeFevre, and Neoklis Polyzotis. 2013. Odyssey: A multi-store system for evolutionary analytics. *Proc. VLDB Endow.* 6, 11 (2013), 1180–1181.

[15] Alon Y. Halevy. 2001. Answering queries using views: A survey. *VLDB J.* 10, 4 (2001), 270–294.

[16] Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, Bertty Contreras-Rojas, Rodrigo Pardo-Meza, Anis Troudi, and Sanjay Chawla. 2020. ML-based cross-platform query optimization. In *ICDE*. IEEE, 1489–1500.

[17] Boyan Kolev, Patrick Valduriez, Carlyna Bondiombouy, Ricardo Jiménez-Peris, Raquel Pau, and José Pereira. 2016. CloudMdsQL: Querying heterogeneous cloud data stores with a common language. *Distributed Parallel Databases* 34, 4 (2016), 463–503.

[18] Sebastian Kruse, Zoi Kaoudi, Bertty Contreras-Rojas, Sanjay Chawla, Felix Naumann, and Jorge-Arnulfo Quiané-Ruiz. 2020. RHEEMix in the data jungle: A cost-based optimizer for cross-platform systems. *VLDB J.* 29, 6 (2020), 1287–1310.

[19] Joon Lee, Daniel J. Scott, Mauricio Villarroel, Gari D. Clifford, Mohammed Saeed, and Roger G. Mark. 2011. Open-access MIMIC-II database for intensive care research. In *EMBC*. IEEE, 8315–8318.

[20] Jiaheng Lu and Irena Holubová. 2019. Multi-model databases: A new journey to handle the variety of data. *ACM Comput. Surv.* 52, 3 (2019), 55:1–55:38.

[21] Jiaheng Lu, Irena Holubová, and Bogdan Cautis. 2018. Multi-model databases and tightly integrated polystores: Current practices, comparisons, and open challenges. In *CIKM*. ACM, 2301–2302.

[22] Antonio Maccioni, Edoardo Basili, and Riccardo Torlone. 2016. QUEPA: QUerying and Exploring a Polystore by Augmentation. In *SIGMOD*. ACM, 2133–2136.

[23] Kian Win Ong, Yannis Papakonstantinou, and Romain Vernoux. 2014. The SQL++ semi-structured data model and query language: A capabilities survey of SQL-on-Hadoop, NoSQL and NewSQL databases. *CoRR* abs/1405.3631 (2014).

[24] M. Tamer Özsu and Patrick Valduriez. 2020. *Principles of Distributed Database Systems, 4th Edition.* Springer.

[25] Pramod J. Sadalage and Martin Fowler. 2012. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* Pearson Education.

[26] Alkis Simitsis, Kevin Wilkinson, Malú Castellanos, and Umeshwar Dayal. 2012. Optimizing analytic data flows for multiple execution engines. In *SIGMOD*. ACM, 829–840.

[27] Utkarsh Srivastava, Kamesh Munagala, and Jennifer Widom. 2005. Operator placement for in-network stream query processing. In *PODS*. ACM, 250–258.

[28] Ran Tan, Rada Chirkova, Vijay Gadepally, and Timothy G. Mattson. 2017. Enabling query processing across heterogeneous data models: A survey. In *BigData*. IEEE, 3211–3220.

[29] Balder ten Cate and Phokion G. Kolaitis. 2010. Structural characterizations of schema-mapping languages. *Commun. ACM* 53, 1 (2010), 101–110.

[30] Marco Vogt, Alexander Stiemer, and Heiko Schuldt. 2018. Polypheny-DB: Towards a distributed and self-adaptive polystore. In *Big Data*. IEEE, 3364–3373.

[31] Jingjing Wang, Tobin Baker, Magdalena Balazinska, Daniel Halperin, Brandon Haynes, Bill Howe, Dylan Hutchison, Shrainik Jain, Ryan Maas, Parmita Mehta, Dominik Moritz, Brandon Myers, Jennifer Ortiz, Dan Suciu, Andrew Whitaker, and Shengliang Xu. 2017. The Myria big data management and analytics system and cloud services. In *CIDR*.

[32] Wolfram Wingerath, Felix Gessert, Erik Witt, Steffen Friedrich, and Norbert Ritter. 2018. Real-time data management for big data. In *EDBT*. 524–527.

[33] Wolfram Wingerath, Norbert Ritter, and Felix Gessert. 2019. *Real-Time & Stream Data Management - Push-Based Data in Research & Practice.* Springer.

[34] Minpeng Zhu and Tore Risch. 2011. Querying combined cloud-based and relational databases. In *CSC*. IEEE, 330–335.


[4] https://www.mars-group.org/

[5] https://www.baqend.com