# PolyPipe: Merging Data Pipelines and Multi-Model Databases

David Lengweiler
University of Basel
Basel, Switzerland
david.lengweiler@unibas.ch

Tobias Weber
University of Basel
Basel, Switzerland
tm.weber@stud.unibas.ch

Heiko Schuldt
University of Basel
Basel, Switzerland
heiko.schuldt@unibas.ch

Marco Vogt
University of Basel
Basel, Switzerland
marco.vogt@unibas.ch

## Abstract

Modern data is characterized by its high volume and inherent heterogeneity, primarily managed by systems tailored to three distinct modeling paradigms: the relational model, which enforces strict schema and high structural integrity; the document model, which offers schema flexibility for semi-structured data; and the graph model, which prioritizes modeling complex relationships between entities. While the database industry is trending toward multi-model systems that incorporate features from all paradigms, data management practices still lag behind. Data scientists rely on manual, multi-stage and labor-intensive workflows to integrate disparate data sources. This process forces users to switch tools, results in high data shipping costs, and forfeits database-level optimizations and structural guarantees, leading to complex, brittle and non-reusable “one-off” solutions. We argue that embedding data pipelines directly into a multi-model database offers significant benefits, including streamlining, simplification, and improved maintainability, by utilizing declarative, database-native operators. This paper presents PolyPipe, an extension to the Polypheny multi-model database system. PolyPipe integrates data pipeline functionality as a first-class citizen, allowing the construction of complex pipelines using a hybrid of database and classical operators within a single system.

## Keywords

Heterogenous Data Management, Data Pipelines, ETL, Multi-Model Databases

## 1 Introduction

Modern data is characterized by its high volume, diverse formats, and varying logic. To model this logic, several distinct paradigms have evolved. Today, the data landscape is dominated by a mixture of three primary models: the *relational model*, which enforces strict schema constraints and inter-table relationships; the *semi-structured document model*, which offers schema flexibility for unstructured data but lacks strict connectivity; and the *graph data model*, which prioritizes complex relationships between entities. To fully utilize their structural and relational aspects, these models rely on specialized query languages. *SQL* leverages the structural guarantees of the relational model for efficient retrieval; *MongoDB Query Language (MQL)* manages the variability of document collections; and *Cypher* excels at modeling and traversing complex relationships within graphs.

While dedicated systems have established themselves for these specific models (e.g., PostgreSQL for relational, MongoDB for documents, Neo4j for graphs), the industry is shifting toward convergence. Most systems now incorporate features from other paradigms, such as PostgreSQL’s JSON capabilities or MongoDB’s schema validation. This trend is spearheaded by the rise of multi-model databases, which aim to provide diverse modeling and querying capabilities within a single system.

However, current data management practices lag behind this convergence. Data scientists must frequently integrate disparate sources, ranging from raw files to databases, using manual integration processes and multiple query languages. Although the specifics vary, these workflows generally follow four distinct stages:

- **i. Exploration:** The initial assessment of data structure and quality. In heterogeneous environments, this forces the user to switch between disparate tools (e.g., SQL clients, JSON viewers, graph visualizers) just to understand the available datasets, preventing a holistic view of the data landscape.
- **ii. Integration:** The cleaning, transformation, and joining of data sources. This is often the most labor-intensive phase; without a unified model, it typically requires extracting data into application-layer scripts (e.g., Python or R), resulting in high data shipping costs and the loss of database-level optimizations.
- **iii. Analysis:** The application of complex logic or aggregation. When pipelines are decoupled from the storage engine, analytical queries cannot leverage the structural guarantees or indexes of the underlying data models, significantly reducing performance.
- **iv. Visualization:** The presentation of results. In disjointed pipelines, the link between the final visual output and the raw source data is often weakened or severed, making it difficult to trace anomalies or verify data lineage dynamically.

Creating pipelines to bridge these heterogeneous landscapes is often complex, tedious to maintain, and results in “one-off” solutions with limited reusability.

We argue that by migrating data pipelines directly into a multi-model database, the process can be streamlined, simplified, and made more maintainable:

- **Streamlining:** Rather than building pipelines with ad-hoc tools, using database-native operators standardizes the process and minimizes custom programming logic.
- **Simplification:** While query languages introduce specific operators, their declarative nature makes it easier to analyze and comprehend pipeline logic compared to custom code.

*EDBT ’26, Tampere (Finland)*

10.48786/edbt.2026.72

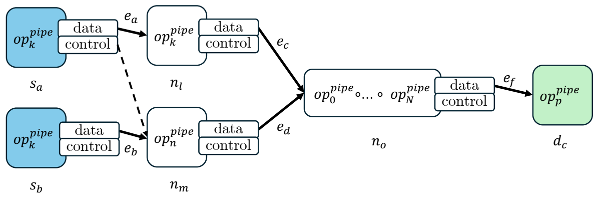


**Figure 1: Structure of PolyPipe with singular and chained pipeline operators.**

**Maintainability:** Integrating pipelines into the database reduces the need for meticulous documentation of external tools and creates a unified environment for storage, retrieval, and transformation.

In this paper, we present *PolyPipe*, an extension to the Polypheny system that integrates data pipeline functionality as a first-class citizen within a multi-model database. This combination enables the construction of complex pipelines using a hybrid of database and classical operators, leveraging symbiotic optimizations for high performance and direct data access. Polypheny is a PolyDBMS [9] that supports multiple data models and uses various DBMS as storage and execution engines.

The remainder of the paper is structured as follows. In Section 2 we introduce the data model of PolyPipe and present the graphical editor in Section 3. The overview of the system is given in Section 4, followed by an overview of relevant work in Section 5. We summarize the work in Section 6.

## 2 Architecture Model

The proposed model establishes a unified architecture by integrating multi-model database operators with data pipeline semantics, positioning pipelines as first-class citizens within a PolyDBMS. This approach is based on the PolyAlgebra [8], extending it to incorporate pipeline concepts such as control flow and checkpoints.

### 2.1 Multi-Model Data and Operators

The core of the model is an extension of relational algebra to a multi-model setting. Data is organized into entities ($Ent$), which are annotated with their underlying data model (m) (e.g., relational tuples, JSON documents, or graph nodes/edges). Operators are similarly model-aware: A multi-model operator ($OP^m$) consumes one or more input entities (which may come from different models) and produces a single output entity whose model is defined by the operator itself. For example, a relational join ($Join^{rel}$) requires and produces relational entities, while a cross-model transformation ($DocToRel$) consumes a document entity and yields a relational one. As in classical algebra, operators can be composed into queries. The critical difference here is that operators must explicitly state and respect the data model of both their inputs and their expected outputs, enabling cross-model transformations as an inherent part of the system.

### 2.2 Pipeline Model

Data pipelines are formally modeled as Directed Acyclic Graphs (DAGs) consisting of a set of nodes ($N$) and directed edges ($E$) between them. Each node represents an operator defined by sets of input ($I$) and output ports ($O$), and a transformation ($t$) applied upon execution. Edges connect the output of one node to the input of another, defining the flow of entities, which can carry either entity data (data edges) or signals (control edges). Control edges trigger execution or reroute failed records for recovery. Pipelines connect to external data via sources ($S$) and sinks/destinations ($D$). Sources are special nodes corresponding to scan operators ($Scan^m$) that read data, while sinks correspond to modify operators ($Modify^m$) that insert data. Finally, the notion of an operator is generalized to a pipeline operator ($OP^{p,m}$), which, unlike a standard database operator that produces a single entity, can produce multiple output entities. This generalization allows complex database queries to be embedded directly as pipeline nodes while also supporting pipeline features like branching and multi-output operators common in ETL (Extract, Transform, Load) workloads. The embedding of database operators into pipelines and the general structure can be seen in Figure 1.

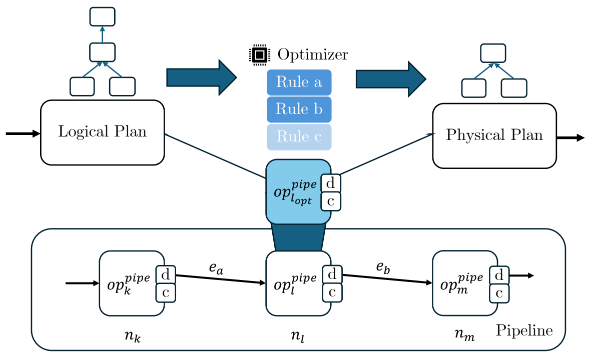


**Figure 2: The offline preparation of a pipeline operator going through the underlying query optimizer.**

### 2.3 Execution

Query execution in databases typically follows the same structure $Query \rightarrow LogicalQueryPlan \rightarrow PhysicalQueryPlan \rightarrow Result$. The exact process between the *Logical* and the *Physical* plans heavily depends on the database. In the PolyDBMS model, the transition from logical to physical plans includes two additional phases: (i) a *Routing* phase, where a router selects the internal sources that can provide the required data, and (ii) an *Optimization* phase. First, the optimizer builds a rule set $R$ by requesting available transformation rules from all involved data engines and from the PolyDBMS engine itself. Each rule consists of a target operator and a precondition under which the transformation is valid. In a second step, the optimizer iteratively applies these rules to the logical query plan, gradually transforming it into a physical plan. The optimizer continues applying rules as long as it discovers transformations that reduce a cost function, or until no further improvement has been observed for a predefined number of iterations. If the complete plan can be transformed, the resulting physical plan is executed; otherwise, the system reports an error to the user. Pipeline execution roughly follows three steps: (i) extract the relevant set of data entries from each source in the pipeline; (ii) propagate this set along all outgoing edges, piping it to each connected target node; (iii) for each node, consume its input set(s), apply the transformation, and produce new output sets, which are again propagated to downstream nodes. This online process continues until no node produces further output. Before a newly defined pipeline can be executed, our model applies a similar process to classical query optimization: it compiles each logical operator node into a physical query

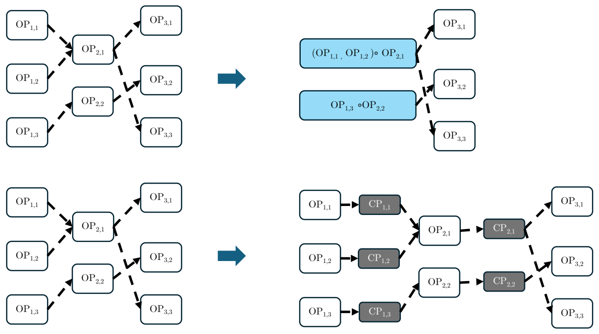


**Figure 3: top-to-bottom: Optimization by fusing chainable operators and automatic or manual insertion of checkpoints.**

plan. The result is a pipeline whose nodes are physical query plans connected via edges. The offline preparation and the online execution process are illustrated in Figure 2.

## 2.4 Storage/Checkpointing and Optimization

An important aspect of data pipelines is their full or partial replayability. To achieve this, we use the established idea of checkpointing, where the results between operators are stored persistently. But checkpoints come with some caveats as they increase the required time for the execution of a pipeline as well as the required space. When pipelines are part of a multi-model database, they can directly reuse the database's storage engines for checkpointing, avoiding additional communication and data transfer to external storage systems. Depending on the use case, a data scientist might not require maximum durability and replayability, and may prefer to trade some guarantees for higher execution speed. As mentioned in Section 2.3, the extended query optimizer is used to rewrite operator nodes into more efficient physical query plans. This idea can be pushed even further with *node fusion* which merges two or more following operator nodes together resulting in a larger query providing a large space of possible optimizations. A second optimization is the ability to lend ideas from stream processing by using *pipelining*. A special set of operators can be identified that, in contrast to collection-based operators like aggregation, which requires the complete set to be present during execution, can operate on entries separately. This allows to execute the next operator already, even if the whole collection is not yet finished with the last operator. A good example of this is the $Projection$ operator, which extends each entry of a set of data entries with an additional field. Both optimizations can rely on a slightly extended traditional database query optimizer during the optimization step. The visual representation of the two optimizations can be seen in Figure 3.

## 3 Graphical Pipeline Editor

Constructing data pipelines is a tedious task, often requiring an iterative workflow in which intermediate results are repeatedly validated until a satisfactory result is achieved. As data pipelines lend themselves to visual presentation due to their graph structure, a visual interface is an ideal fit for constructing and maintaining them, especially when working with a large set of heterogeneous sources and operators. To assist in this process, we provide a graphical editor for PolyPipe, shown in Figure 4. The editor enables the iterative construction of complex data pipelines via a simple drag-and-drop interface and supports partial pipeline execution for validating intermediate results. It also enables exploration of all available data sources. The editor offers a large set of predefined operators and clearly indicates the preconditions that must be satisfied for each operator to be applicable. Furthermore, it provides convenient tools for analyzing potential error cases, fixing problems, and re-executing the pipeline. The visual interface also lists all available pipelines, their current status, and execution metrics, offering practical pointers for future optimizations.

## 4 PolyPipe

PolyPipe is integrated into the Polypheny ecosystem to provide a robust framework for managing complex data lifecycles. This section details how the system handles multi-model data integration and pipeline orchestration through a realistic e-commerce deployment scenario. The system's utility is showcased through three operational phases: multi-model data discovery, dynamic pipeline adaptation, and high-scale heterogeneous processing.

### 4.1 Multi-Model Data Integration

In production environments, PolyPipe serves as the central orchestration layer for disparate data streams. The system's effectiveness is validated using a comprehensive e-commerce dataset that spans three primary data models.

**Cross-Model Data Management** PolyPipe manages delivery networks (graph), product manifests (document), and customer transactions (relational) concurrently. The system allows for querying across these model boundaries without requiring expensive, manual pre-conversion steps, maintaining data integrity at the storage layer.

**Pipeline Transparency** The PolyPipe visual interface provides real-time monitoring of active pipelines. This allows for the inspection of individual operators, monitoring of intermediate result representations, and analysis of execution metrics to identify potential bottlenecks in the data flow.

### 4.2 Dynamic Pipeline Evolution

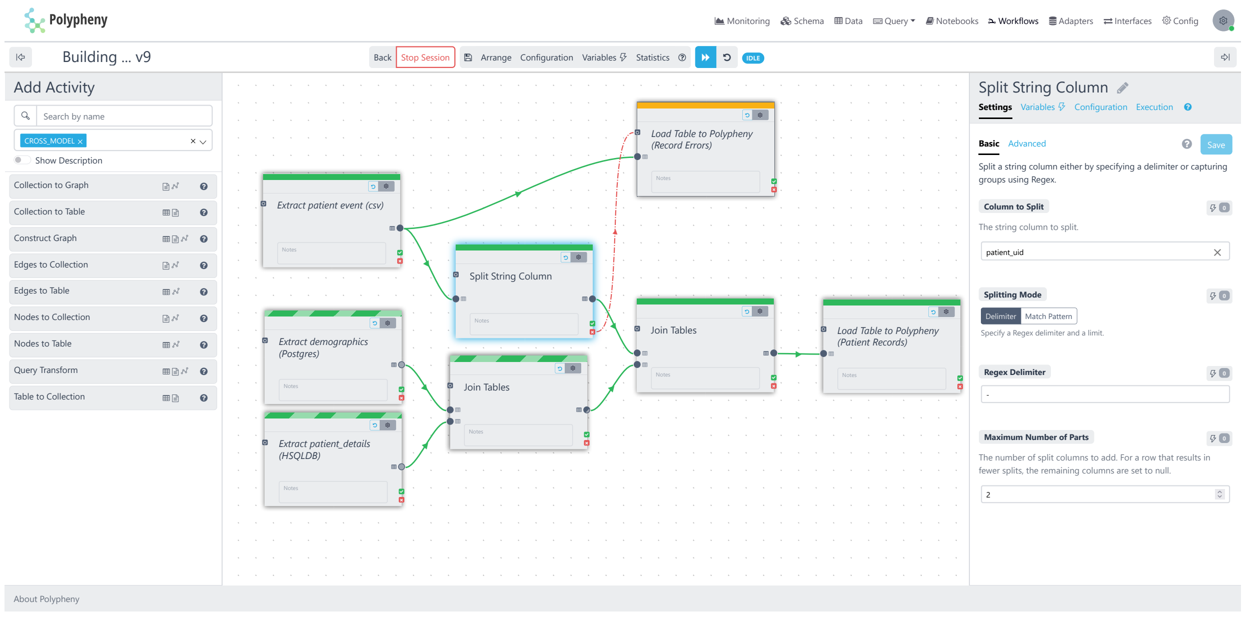
The flexibility of PolyPipe is demonstrated through its ability to adapt to changing requirements, such as the integration of new data providers or schema evolutions.

**Workflow Extensibility** When new inventory sources are introduced, PolyPipe allows for the seamless extension of existing workflows. New operators can be integrated into the data path to normalize and route incoming data to the most suitable storage engine.

**Optimization & Execution** The system evaluates multiple execution plans based on user-defined optimization strategies. This allows for a comparison between low-latency processing and high-throughput batching, ensuring that the pipeline configuration aligns with specific performance requirements.

### 4.3 High-Volume and Heterogeneous Workloads

PolyPipe is designed to maintain stability under stress, particularly when dealing with non-uniform data structures. In deployment scenarios, the system is frequently tasked with processing entities that vary significantly in scale and complexity.



Split String Column

Settings Variables Configuration Execution

Basic Advanced Save

Split a string column either by specifying a delimiter or capturing groups using Regex.

Column to Split

The string column to split.

patient_uid

Splitting Mode

Delimiter Match Pattern

Specify a Regex delimiter and a limit.

Regex Delimiter

-

Maximum Number of Parts

The number of split columns to add. For a row that results in fewer splits, the remaining columns are set to null.

2

**Figure 4: Screenshot of the visual editor of PolyPipe accessible directly within Polypheny.**

By processing data streams where entities (e.g., product metadata) contain varying depths of nesting and diverse attribute sets, PolyPipe demonstrates resilience against structural heterogeneity. Observations of the system under high-volume inputs confirm that it maintains runtime stability and predictable performance, even as the complexity of the non-relational data increases.

## 5 Related Work

Our work lies at the intersection of heterogeneous data management and ETL optimization. Heterogeneous data management systems, ranging from Polystores [5] and Multi-model DBMSs [6] to PolyDBMSs [9], successfully simplify querying across models but typically treat data integration as an external concern. Conversely, ETL research, dating back to EXPRESS [3], has focused on conceptual modeling [7], parallelization (e.g., Pig [2], CloudETL [1]), and cost-based optimization [4]. However, these ETL approaches generally treat the database as a black box, disconnecting transformation logic from database-internal planning. We bridge this gap by embedding algebra-aware pipelines directly into a PolyDBMS, enabling optimizations such as operator fusion that are impossible in external systems.

## 6 Conclusion

This paper introduced PolyPipe, a model and framework that extends multi-model databases with pipeline functionality. This unification removes the need for expensive data transfers between systems and allows users to create, store, and maintain pipelines within a single system. The combination of pipeline support for heterogeneous data and the heterogeneous nature of a multi-model database enables robust pipelines that leverage all available query languages and are easily reusable. As part of this work, we also presented an intuitive graphical user interface that assists users in constructing and modifying complex pipelines with ease.

## Acknowledgments

This work is partly funded by the Swiss National Science Foundation, project Polypheny-DDI (contract no. 200020_213121 / 1), and Innosuisse, project SwissRenov (contract no. 107.512 FS-EE).

## References

[1] Xiufeng Liu, Christian Thomsen, and Torben Bach Pedersen. 2014. CloudETL: Scalable Dimensional ETL for Hive. *Proceedings of the 18th International Database Engineering & Applications Symposium on - IDEAS '14* (2014), 195–206. doi:10.1145/2628194.2628249

[2] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. 2008. Pig Latin: A Not-so-Foreign Language for Data Processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*. Association for Computing Machinery, New York, NY, USA, 1099–1110. doi:10.1145/1376616.1376726

[3] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. 1977. EXPRESS: A Data EXtraction, Processing, and Restructuring System. *ACM Trans. Database Syst.* 2, 2 (June 1977), 134–174. doi:10.1145/320544.320549

[4] A. Simitsis, P. Vassiliadis, and T. Sellis. 2005. State-Space Optimization of ETL Workflows. *IEEE Transactions on Knowledge and Data Engineering* 17, 10 (Oct. 2005), 1404–1419. doi:10.1109/TKDE.2005.169

[5] Michael Stonebraker and Uğur Çetintemel. 2005. "One Size Fits All": An Idea Whose Time Has Come and Gone. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*. IEEE, Tokyo, Japan, 2–11. doi:10/ctkd2n

[6] ArangoDB Development Team. 2025. ArangoDB: Multi-Model Database for Your Modern Apps. https://github.com/arangodb/arangodb. Accessed: 2025-10-03.

[7] Panos Vassiliadis, Alkis Simitsis, and Spiros Skiadopoulos. 2002. Conceptual modeling for ETL processes. In *Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP* (McLean, Virginia, USA) *(DOLAP '02)*. Association for Computing Machinery, New York, NY, USA, 14–21. doi:10.1145/583890.583893

[8] Marco Vogt. 2022. *Adaptive Management of Multimodel Data and Heterogeneous Workloads*. Ph. D. Dissertation. University of Basel. doi:10/j44k

[9] Marco Vogt, David Lengweiler, Isabel Geissmann, Nils Hansen, Marc Hennemann, Cédric Mendelin, Sebastian Philipp, and Heiko Schuldt. 2021. Polystore Systems and DBMSs: Love Marriage or Marriage of Convenience?. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare – VLDB Workshops, Poly 2021 and DMAH 2021*, Vol. 12921. Springer, Copenhagen, Denmark, 65–69. doi:10/gn8qvm