open proceedings

PolyBooks: A System for Interactive Multi-Model Querying with Provenance and Reuse

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

Data exploration, integration, organization, and analysis are critical workflows for data scientists. In recent years, tools like Jupyter Notebooks have gained significant traction by incorporating these steps into a unified repository, allowing users to modify, extend, and document complex analytical processes with ease. However, while these tools streamline analysis, they often leave data integration to the user, frequently resulting in the execution of processes on stale data. Furthermore, standard notebooks lack robust support for persisting large datasets, forcing data scientists to rely on file-based storage or ephemeral memory. Databases, especially multi-model databases, offer a convenient repository for consolidating diverse data formats and providing simplified access. In this paper, we present the combination of these two paradigms. We show how integrating notebooks with multi-model databases leverages established data models to access and persist analytical data, ultimately improving performance for data science use cases.

Keywords

Heterogenous Data Management, Computational Notebooks, Interactive Data Exploration, Multi-Model Databases

1 Introduction

Data scientists, regardless of their specific field, must deal with extensive data. Their work involves creating complex script books to derive information from a large corpus of data, often containing complex, use-case-specific logic expressed across different script repositories in various programming languages.

To enhance the digestibility, reusability, and collaborative potential of these scripts, visual computing has gained significant traction, becoming a widely used resource for data scientists, especially in fields like Machine Learning and AI. This trend was further amplified by the increased popularity of programming languages like Python and the creation of Jupyter notebooks. These notebooks offer a streamlined environment for defining complex analytical logic, allowing for the documentation of logic and the static and dynamic visualization of results. Their tight integration with Python enables easy access to a plethora of libraries, offering integration and analytical shortcuts.

However, despite abstracting away many tedious workflow steps, notebooks face critical limitations:

Poor Discovery and Exploration: Data scientists often lack unified tools for data discovery and exploration, heavily disincentivizing the timely incorporation of up-to-date information.

Manual Data Integration: Retrieving external data requires substantial manual work, often leading to the use of disconnected data files and separate, tedious extraction tasks.

Persistence Deficiencies: Notebooks lack effective mechanisms for data persistence, forcing scientists to devise their own solutions. This frequently results in long-running, all-or-nothing applications that prevent the simple persistence of partial results.

Despite the promise of notebooks, existing solutions to these shortcomings, like integrating specialized data tools or leveraging standard single-model databases, often fail to solve the core problem efficiently. These approaches usually require additional know-how and significant adaption effort, resulting in fragmentation that counters the initial goal of notebooks of simplifying the workflow. While data scientists could benefit from having direct access to databases without manual setup to overcome *Persistence Deficiencies*, a traditional single-model database could only partially address these shortcomings. This is because real-world data is inherently heterogenous. As data scientists frequently deal with a mix of fully structured relational data, semi-structured documents (e.g. JSON files) and graph structures. Trying to force all these diverse data formats into a single-model structure only worsens the challenge of *Manual Data Integration*. Multi-model database, however, are uniquely suited to overcome these shortcomings. They are designed to support different data models natively, providing separate optimized engines to efficiently store the most common data models. Crucially, they provide unified tools, often a singular or complementary set of query languages, to seamlessly interact with and formulate complex queries across heterogenous data.

To truly unify the data access layer, a new approach is necessary. This approach builds on the polystore concept, a system that acts as a direct middle-layer over existing, distributed data resources without requiring physical migration. This is the foundation of the *PolyDBMS*: a system that not only supports retrieval access to underlying databases but also supports modification operations. The direct integration of notebooks with a PolyDBMS system can, therefore, effectively and completely overcome all the mentioned critical limitations.

In this paper, we propose PolyBooks, an extension of the computational notebook concept that embeds visual notebooks as part of PolyDBMS multi-model databases. We argue that this

EDBT '26, Tampere (Finland)

10.48786/edbt.2026.71

integration offers mutual benefits. Firstly, notebooks gain direct access to the diverse data stored within the multi-model database, which drastically reduces the effort required during the exploration phase. Secondly, PolyBooks allows for simple persistency methods to store analytical or partial results directly from within the notebooks on the the underlying database, eliminating the need for data scientists to acquire specialized knowledge about database storage behaviour. As part of this paper, we also provide an implementation of PolyBooks within the multi-model database Polypheny.

The remainder of this paper, is structured as follows. First we describe the data model of PolyBooks in Section 2 and present the graphical user interface in Section 3. We show details of the system in use in Section 4, followed by a summary of relevant work in Section 5. Section 6 concludes.

2 Data Model

Computational notebooks serve as centralized, ideally well-documented repositories for a data science workflow. This section outlines the PolyBooks data model and the improvements it provides for data science workflows in three key components *i.) Exploration and Integration*, *ii.) Data Access and Storage*, and *iii.) Analytical Operation Access and Abstraction*, explaining how notebooks, powered by PolyBooks, can significantly improve established data analytics processes.

2.1 Exploration and Integration

The initial phase of any workflow involves exploration and integration: accessing data from external sources (static files, databases, APIs) and conforming it to a usable format.

In traditional data exploration, data scientists face significant overhead: they must manually connect to diverse resources or download files, and interpreting the data's value often requires immediate, manual integration. This effort is compounded by the need for specialized knowledge for each unique data source (e.g., database-specific query languages).

PolyBooks leverages the power of PolyDBMS (Poly-model Database Management Systems) to integrate multiple different data formats (*relational*, *document*, *graph*, etc.) into a unified system.

Diverse Data Models: Multi-model databases support different data models, each defined by a set of possible types and a specific form of data value collection.

Unified Query Target: Although different query languages (like SQL for relational or MongoQL for document) exist, they all expose a base operation to retrieve a data model entity (e.g., a simple `SELECT * from <Table>` or `db.<collection>.find({},{})`), which allows for simple retrieval of various entities over a generic retrieval command.

Low-Effort Exploration: By connecting diverse data resources to the PolyDBMS where only minimal integration is necessary, data scientists gain easy access to the data resources through a unified layer. This dramatically lowers the effort required to create exploration scripts that assist in building future aggregation logic.

2.2 Data Access and Storage

A major challenge in data analytics is the *Impedance Mismatch*: the discrepancy between data representations across different systems. This is often discussed regarding database drivers

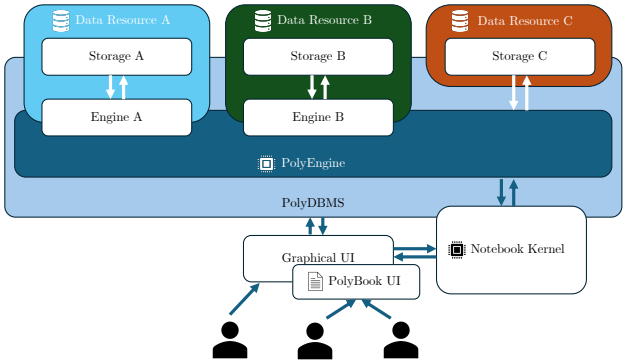

Figure 1: Architecture of PolyBooks, embedded in a PolyDBMS system, providing access to the underlying engine.

(like JDBC), but it extends throughout the entire analytical process:

Obtain: Data is obtained in some original form.

Transform (Storage): Transformed into a storage-specific format (e.g., in a database).

Extract & Transform (Notebook): Extracted, and often immediately transformed into a programming language-specific format (e.g., a Python DataFrame).

Save (File-based): When saving results, data scientists often keep it close to the programmatic representation and opt for file-based storage.

This continuous data transformation is inefficient and makes it difficult to validate large results or extract additional metrics consistently. PolyBooks lightens this process by significantly reducing the amount of data transformation required for the analytical process. It improves the storing and accessing of data during computational scripts:

Integrated Storage: PolyBooks encourages and facilitates the storage of partial results directly within the underlying multi-model database, instead of disparate file-based storage.

Reduced Overhead: By maintaining data formats closer to the database's native models, PolyBooks minimizes the costly programmatic transformations that typically occur between the database, the notebook environment, and file system storage.

2.3 Database Operation Access and Abstraction

Computational notebooks use a programming language to express operations on target data. PolyBooks optimizes these operations by leveraging the power of the database engine. While complex operations are often easier to express in a programming language, databases are highly optimized to perform simpler or model-specific operations significantly faster on their native data. PolyBooks allows for the utilization of database operations to outsource (or "offload") such computations onto the underlying query engine. This leads to much faster operation execution compared with purely programmatic operations executed within the notebook environment.

Speed Advantage: Even seemingly simple operations are magnitudes faster when expressed within a specific database. For example, a simple counting operation performed programmatically in a notebook can take marginally more

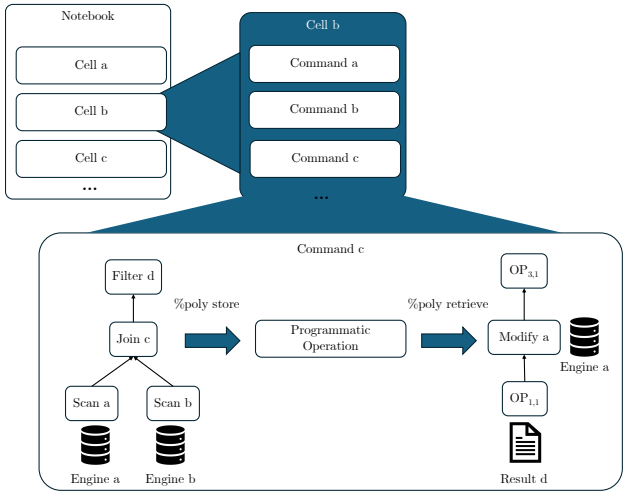

Figure 2: Direct access of multi-model operations and storage from within PolyBooks.

time than a simple `COUNT` query in SQL or a `count()` query in MongoDB, which are executed directly by the optimized engine.

Complex Aggregations: This benefit extends beyond simple counts. PolyBooks can access the underlying engines and execute complex aggregations (like joins, grouping, and statistical calculations) directly on the source data, delivering only the concise, final result back to the notebook for immediate use. This drastically reduces the amount of data transferred and processed programmatically.

3 Graphical User Interface

We integrated PolyBooks within the PolyDBMS Polypheny, an extension to the multi-model database paradigm. Part of this implementation is a graphical user interface, which follows similar design guidelines like most available notebook editors or platforms. Additionally, we provide an IPython extension removing the need to access Polypheny via graphical UI. The graphical user interface structures available notebooks in a data system like structure allowing to create notebook files but also directly uploaded and connect other files. To directly query Polypheny one can use the `%poly` magic command or the visual `query` block provided in the visual interface, followed by the required query language and the query: `%poly sql: <query>`. On execution of the notebook block this command directly executes the query through the PolyDBMS engines of Polypheny and provides the result back into the notebook. A screenshot of the visual editor can be seen in Section 3.

4 System Application and Workflows

The proposed system architecture extends traditional computational notebooks to serve as a unified control plane for the Polypheny multi-model database. By integrating specialized middleware into the notebook kernel, the system bridges the gap between ephemeral exploratory analysis and robust, persistent data management. The system's utility is organized into three functional layers: unified multi-model exploration, persistence orchestration, and multi-lingual command execution.

4.1 Unified Multi-Model Data Exploration

At its core, the system provides a unified environment for the immediate integration and manipulation of diverse data sources without requiring external drivers or complex boilerplate code. The architecture allows the notebook kernel to maintain a single session that can simultaneously address relational tables, document collections, and file-based data stores.

This capability ensures that users can perform native schema inspections and data acquisitions using the query language best suited for the underlying model. A critical feature of this layer is the automatic unification of heterogeneous results into standard data structures (e.g., Pandas DataFrames). This allows for immediate cross-model analysis and visualization, ensuring that the user's primary focus remains on data exploration rather than the technical overhead of integration setup.

- **Schema Inspection:** Visual exploration of heterogeneous data schemas stored within Polypheny directly from the notebook.
- **Kernel Unification:** Seamless merging of diverse data models into standard Python objects for unified analysis.
- **Resource Integration:** Dynamic connection of additional data resources by incorporating them directly into the active session.

4.2 Persistent Pipeline Orchestration

The system architecture specifically addresses the transition from exploratory, in-memory computations to persistent database workflows. This functional layer allows users to treat the notebook not just as a sandbox, but as an orchestration tool for the database's persistence layer.

Through the implementation of specialized magic commands, such as `%poly store`, the system enables the low-friction transformation of ephemeral script steps into durable database entities. For instance, a complex aggregate calculated in-memory can be instantly persisted back into Polypheny as a new document collection or relational table. This design facilitates iterative refinement, allowing users to modify notebooks to retrieve pre-calculated results via native database reads, thereby utilizing the database's performance and data integrity guarantees.

- **Feature Engineering:** Implementation of complex transformations using in-memory Python operations.
- **Persistence Integration:** Direct storage of transformed datasets into the multi-model store for long-term availability.
- **Workflow Staging:** The ability to bypass redundant computations by referencing previously persisted database stages.

4.3 Multi-Lingual Command Center

The final component of the system is its role as a multi-lingual command center for specialized, native database operations. In a heterogeneous environment, certain operations, such as DDL instructions or complex graph traversals, are best handled using the specialized query languages of the underlying storage engines.

The system exposes these capabilities directly within the notebook interface, allowing for the execution of non-standard operations like creating indexes or issuing Cypher queries for relationship-heavy data. This prevents the "lowest common denominator" problem often found in unified interfaces; instead, the system harnesses the unique strengths of every component within the

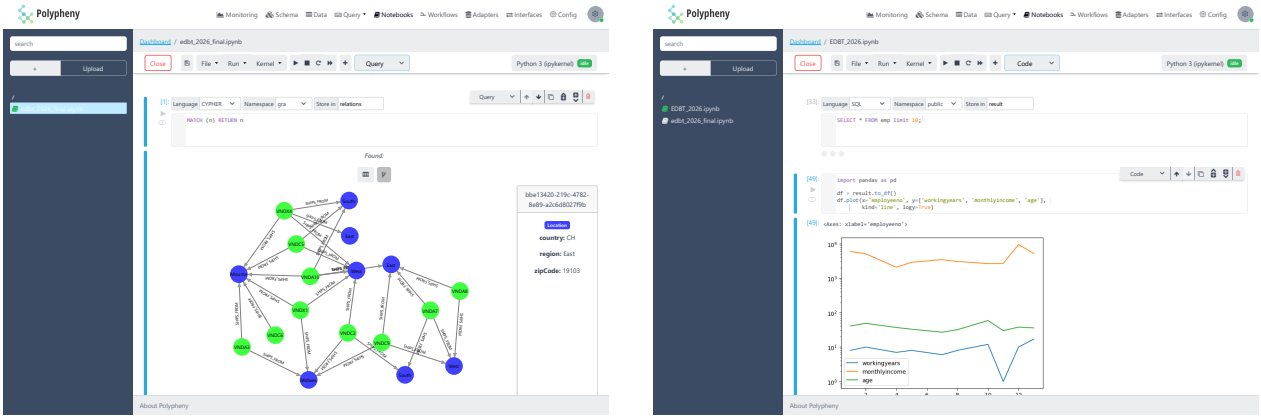

Figure 3: Screenshots of visual editor of PolyBooks.

multi-model architecture. By delegating complex aggregation and structural logic to the server-side engines, the notebook maintains high performance even when dealing with massive datasets.

- **Index and Management:** Execution of DDL instructions to manage the physical and logical structure of the data.
- **Specialized Querying:** Utilizing native languages like Cypher for high-performance relationship traversals.
- **Server-Side Delegation:** Optimizing resource use by triggering complex aggregation queries directly on the database engines.

5 Related Work

PolyBooks combine the concept of multi-model data management and querying with the collaborative idea of computational notebooks.

5.1 The Collaborative Aspect

Early implementations of digital notebooks focused on the aggregation of diverse information sources. The *Virtual Notebook System* utilized distributed hypertext to allow researchers to organize heterogeneous data collaboratively [5]. Similarly, the *Collaboratory Notebook* focused on supporting open-ended inquiry, allowing users to structure collaborative learning and data sharing [1].

5.2 Multi-Model Visualization

Visualizing data from multi-model sources requires flexible interfaces. Pham et al. [4] demonstrated that effective interactive exploration requires multiple visual contexts that can adapt to different data models and dimensions, ensuring that the user interface does not restrict the complexity of the underlying data.

5.3 Performance and Backend Integration

To handle the latency inherent in connecting notebooks to large-scale data backends, recent work on the *Magpie* system [2] introduces optimization techniques such as lazy evaluation and code translation. This allows notebooks to function as high-performance interfaces for cloud-scale data, bridging the gap between interactive Python scripts and heavy database execution [3].

6 Conclusion

In this paper, we have presented a new kind of interactive notebooks, combining the concepts of multi-model databases and computational notebooks, within the Polypheny database management system. This integration accomplishes several pivotal objectives. First, it consolidates the various tools required for data analysis and visualization into a single, unified platform, thereby streamlining the workflow of data analysts. Second, it significantly enhances both reusability and collaborative potential, as analysts can now effortlessly share a common environment equipped with comprehensive features and a reusable schema mapping providing seamless access to all data in their organization. Furthermore, this combination allows for advanced optimization strategies. It is a step towards promoting the Polypheny PolyDBMS as a highly valuable and effective tool for data science.

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] Daniel C. Edelson, Roy D. Pea, and Louis M. Gomez. 1996. The Collaboratory Notebook. *Commun. ACM* 39, 4 (April 1996), 32–33. doi:10.1145/227210.227218

[2] Alekh Jindal, K Venkatesh Emani, Maureen Daum, Olga Poppe, Brandon Haynes, Anna Pavlenko, Ayushi Gupta, Karthik Ramachandra, Carlo Curino, Andreas Mueller, Wentao Wu, and Hiren Patel. 2021. Magpie: Python at Speed and Scale Using Cloud Backends. (2021).

[3] Hiren Patel. 2021. Magpie: Python at Speed and Scale Using Cloud Backends. (Jan. 2021).

[4] Phi Giang Pham, Mao Lin Huang, and Quang Vinh Nguyen. 2017. Interactive Data Exploration through Multiple Visual Contexts with Different Data Models and Dimensions. In *2017 21st International Conference Information Visualisation (IV)*. 84–89. doi:10.1109/iV.2017.53

[5] F. M. Shipman, R. J. Chaney, and G. A. Gorry. 1989. Distributed Hypertext for Collaborative Research: The Virtual Notebook System. In *Proceedings of the Second Annual ACM Conference on Hypertext - HYPERTEXT '89*. ACM Press, Pittsburgh, Pennsylvania, United States, 129–135. doi:10.1145/74224.74235