Towards Mixed-Model Database Management Systems

Martin Vahlensieck, Marco Vogt, and Heiko Schuldt

Department of Mathematics and Computer Science, University of Basel, Switzerland
{firstname.lastname}@unibas.ch

In recent years data management has become increasingly complex. The days when the needs of a business could be met by a single relational database management system (DBMS) are long gone [1]. In today's data landscape, businesses need to combine different DBMSs and different data models. As an example, consider a web shop that is using a relational database for customer information and a document database for products; the analytics department is using a graph database, the logistics department is using a column store, etc. Unfortunately, the data stored in the various databases is not independent. Customers information is needed by multiple applications and is therefore stored in different databases, where records in one database may reference records in another database.

The problem is that different DBMSs differ (among other things) in data consistency features, support for referential integrity and transactional guarantees. That means that it is up to the applications to ensure these properties (i.e., when a customer updates their address, the web shop must update all databases that have a copy of the address). This approach does not scale [2].

These problems are only symptoms of the underlying problem, which is a missing single source of truth: what data do we have, how does it look like, where is it stored and what invariants must be upheld. In the example above, this knowledge is implicit and split between different applications.

PolyDBMSs offer a part of the solution [3]. They provide a variety of query interfaces with unified access to different data models. However, while these systems offer cross-model queries (e.g., using SQL to query document data), the data stays in the original data model. This is problematic because it prevents having referential integrity between data models, like referencing a document from a relational tuple. There is also no conceptual component to the schema, so this information must be maintained elsewhere. Further, the mappings between data models for cross-model queries are fixed, while ideally the mapping is chosen based on the data and application needs.

In this paper, we propose an extension of the PolyDBMS concept that features a unified schema, independent of any single data model. This unified schema contains a conceptual model, together with mappings to different data models for storage and mappings to different data models for use with applications (used for queries). This approach also enables us to express constraints and invariants in the conceptual part and the PolyDBMS, if applicable, pushing these down to the DBMSs storing the data.

For this unified schema we propose to use constructs from set theory. The constructs from set theory can be used efficiently and concisely to create a conceptual description. Popular data models are also described with these constructs, making it easy to express the mappings between data models.

Combining PolyDBMSs with a unified, data model independent schema is an important step towards true mixed-model data management.

References

[1] Stonebraker, M., & Çetintemel, U. (2018). "One size fits all": An idea whose time has come and gone. *Making Databases Work: The Pragmatic Wisdom of Michael Stonebraker*, 1st ed., 441–462.

[2] Hutter, A., et al. (2025). Model once, represent everywhere: UDA (Unified Data Architecture) at Netflix. *Netflix Technology Blog*. Retrieved November 12, 2025, from `https://netflixtechblog.com/uda-unified-data-architecture-6a6aee261d8d`.

[3] Vogt, M., et al. (2021). Polystore systems and DBMSs: Love marriage or marriage of convenience? *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, Vol. 12921, 65–69.