# AI-Powered Orchestration of Multi-Model Data

Jáchym Bártík
supervised by Irena Holubová
Faculty of Mathematics and Physics, Charles University
Prague, Czech Republic
jachym.bartik@matfyz.cuni.cz

## ABSTRACT

Multi-model databases are an increasingly popular solution to today's data management challenges of Big Data. However, their inherent complexity and lack of standardization stand in the way of their widespread adoption. In our research, we focus on reducing the complexity by automating the management of such databases. The goal is to provide a robust framework capable of unified modelling, transformation, querying, and evolution management of multi-model data and to leverage AI techniques to optimize data distribution among the database systems.

**VLDB Workshop Reference Format:**
Jáchym Bártík. AI-Powered Orchestration of Multi-Model Data. VLDB 2024 Workshop: VLDB Ph.D. Workshop.

## 1 INTRODUCTION

More than 2/3 of the 50 most widely used database management systems (DBMSs)[1] fall under the category of *multi-model*.[2] The multi-model data is organised in various mutually interlinked formats and models, often with contradictory features [17]. In addition, its structure may change over time, and its size can grow to the extremes of Big Data. These aspects create one of the most complex challenges of effective data management.

As handling such a complex task manually is impossible, we focus on the automatic management of dynamic multi-model Big Data. We want to create a robust framework capable of accepting different types of data, queries, changes, and propagation strategies. Based on such rich input, the system will learn to provide self-adapting evolution management, ensuring a complete, correct, and efficient propagation of changes. Particularly, it will support the following features:

- *Multi-Model Modeling*: We need to model the data in one unified and formally backed schema. The model can either (1) be created manually or (2) automatically inferred from sample data. We can also combine these approaches, i.e., infer a reasonable schema and then manually improve it.
- *Multi-Model-to-Multi-Model Transformations*: Transformation from one model to another is a simple process. But, we must be able to migrate the data between different combinations of models represented by different database systems.
- *Cross-Model Querying*: We need to query over the whole dataset, not just a single database system. Also, the queries should be independent of the underlying data models so that we can use the same query language for the whole system and thus not force the user to learn different languages.
- *Multi-Model Evolution Management*: As we have mentioned, each system evolves over time, whereas in the case of multi-model data, the evolution must cover all combined models. Primarily, we want to be able to update the model, the data itself, and the queries. And, when possible, automatically.

Several solutions have already implemented these features, many of which are widely used. However, they all have one thing in common: they are either tightly coupled with the underlying database systems or too limited to fully model multi-model data. For example, the UML and ER models are industry standards. But, they cannot generally model complex properties, maps, or graphs.

*Outline.* In Section 2, we discuss the current functionalities of our framework consisting of a family of tools. In Section 3, we describe our planned steps. In Section 4, we outline the open problems.

## 2 INITIAL FRAMEWORK

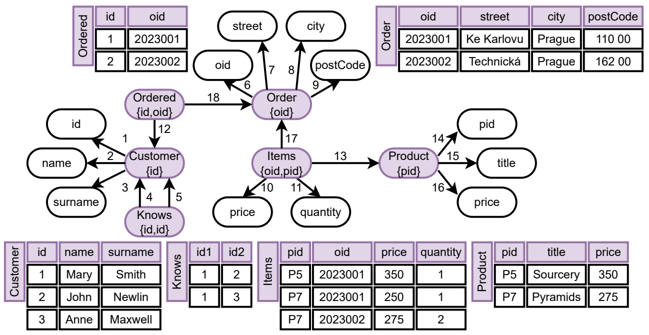
In our research group, we have proposed several solutions to selected aspects of unified and efficient multi-model data management. We have also implemented tools for their experimental verification. This toolset represents the initial framework we currently intend to enhance by exploiting AI to automate data management.

First, we needed a sufficiently abstract approach to handle all the conflicting requirements because we deal with varied data models and database systems. Therefore, we proposed a system-independent representation based on category theory [12]. We can view a *category* as a directed multigraph for simplicity. The nodes (called *objects*) represent entities, and the edges (called *morphisms*) represent relationships between them. For example, object $A$ represents a User and object $B$ represents a Name. Then, we can have a morphism $f : A \longrightarrow B$, meaning that a User has a Name. We can create structures representing arrays, sets, weak-entity types, etc. In our framework, we call such category a *schema category*.

This unifying representation enables us to "grasp" any combination of models and to process it in a system-independent manner. When a particular operation has to be done at this abstract level, it is propagated to the underlying database system.

*Example 2.1.* An example of a schema category can be found in Fig. 1. The schema category is mapped only to the relational database model (denoted using the violet colour). On the other hand, in Fig. 2, we can see the same schema category after an evolution of the mapping. It is now mapped to relational (violet) and document (green) models. □

[1]https://db-engines.com/en/ranking
[2]I.e., consisting of multiple data models (relational, document, graph, ...).



Ordered

| id | oid |
|---|---|
| 1 | 2023001 |
| 2 | 2023002 |

Order

| oid | street | city | postCode |
|---|---|---|---|
| 2023001 | Ke Karlovu | Prague | 110 00 |
| 2023002 | Technická | Prague | 162 00 |

Customer

| id | name | surname |
|---|---|---|
| 1 | Mary | Smith |
| 2 | John | Newlin |
| 3 | Anne | Maxwell |

Knows

| id1 | id2 |
|---|---|
| 1 | 2 |
| 1 | 3 |

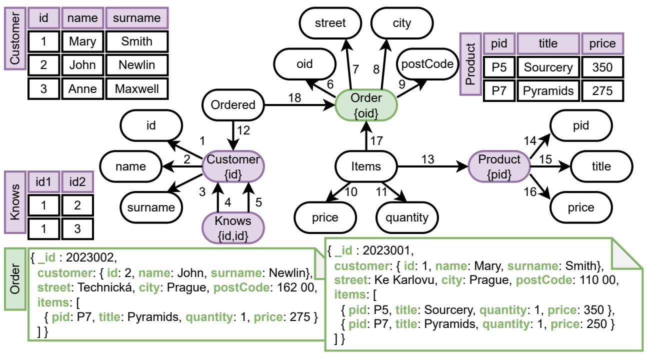
Items

| pid | oid | price | quantity |
|---|---|---|---|
| P5 | 2023001 | 350 | 1 |
| P7 | 2023001 | 250 | 1 |
| P7 | 2023002 | 275 | 2 |

Product

| pid | title | price |
|---|---|---|
| P5 | Sourcery | 350 |
| P7 | Pyramids | 275 |

**Figure 1: A sample schema category. Each violet node represents a complex object. In this case, all of them are mapped to respective tables of the relation model, e.g., PostgreSQL.**



Customer

| id | name | surname |
|---|---|---|
| 1 | Mary | Smith |
| 2 | John | Newlin |
| 3 | Anne | Maxwell |

Product

| pid | title | price |
|---|---|---|
| P5 | Sourcery | 350 |
| P7 | Pyramids | 275 |

Knows

| id1 | id2 |
|---|---|
| 1 | 2 |
| 1 | 3 |

Order

```
{ _id : 2023002,
  customer: { id: 2, name: John, surname: Newlin},
  street: Technická, city: Prague, postCode: 162 00,
  items: [
   { pid: P7, title: Pyramids, quantity: 1, price: 275 }
  ] }
```

```
{ _id : 2023001,
  customer: { id: 1, name: Mary, surname: Smith},
  street: Ke Karlovu, city: Prague, postCode: 110 00,
  items: [
   { pid: P5, title: Sourcery, quantity: 1, price: 350 },
   { pid: P7, title: Pyramids, quantity: 1, price: 250 }
  ] }
```

**Figure 2: The example schema category from Fig. 1 after an evolution. The schema category did not change, but the affected tables (*Items* and *Orders*) were replaced by a collection of documents (*Order*), e.g., in MongoDB. The change introduced redundancy to query orders more efficiently.**

Our toolset involves two tools that enable the creation of a schema category: *MM-cat* [14] enables the modelling of the schema category manually, as well as the creation of its decomposition and mapping of the selected components to particular logical models. *MM-infer* [13] enables one to infer a schema category from the given sample multi-model data (semi-)automatically.

## 2.1 Transformations

As indicated in the introduction, not only do we need to be able to model the multi-model data using a unified representation, but we also need to be able to transform them. In particular, we need a support for transforming any combination of the (sub-)models to any other combination of (sub-)models.

For this purpose, as a part of *MM-cat*, we have developed algorithms that leverage mapping between the schema category and the logical models to transform data between them [12]. To work with each database system in the same way, we have to create wrappers for each of them. Then, we can use the same algorithm to transform data between any two database systems.

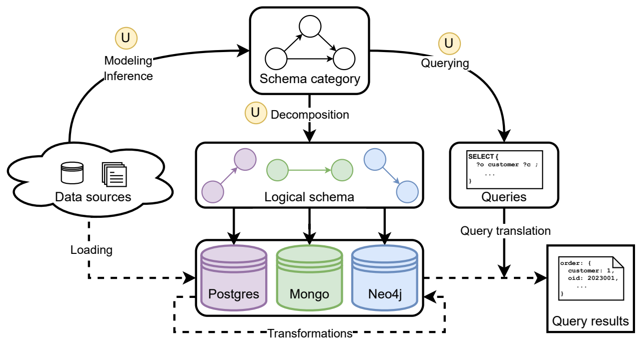
This is an important distinction from other approaches. For example, the ETL (*Extract-Transform-Load*) process is usually tightly coupled with the underlying database systems. In our solution, we can use the same algorithm to transform data between any two database systems, even multi-model ones. Another example are *data lakehouses* which can store data in multiple (single-model) formats, but they do not provide a unified way to work with them.

## 2.2 Querying

Querying the data is an essential feature of any database. However, this is a much more challenging task in multi-model databases as each system has its own set of supported models and their combinations [17] and a specific query language [4]. There is no general standard for multi-model querying except for the standards SQL/XML [7] and SQL/JSON [8] for relational/document models.

We have proposed the *Multi-Model Query Language* (MMQL), a query language based on the SPARQL syntax that enables one to query over the schema category. Then, within a tool called *MM-quecat* [11], we developed the query-evaluation algorithm that uses a similar approach as the transformation algorithms: First, the query is parsed and mapped to the schema category. Then, we use the mappings to split it into *query parts* that can be executed in the specific underlying systems. We perform the maximum amount of work in the databases, minimising cross-database joins. Finally, we combine the intermediate results.

The global workflow with the indicated functionalities of the initial framework is depicted in Fig. 3. First, the schema category is inferred from sample data or modelled manually. Then, it has to be decomposed into the system-specific sub-models. Finally, the user can specify queries over the schema category. All these actions require user input. On the other hand, all data transformations and query resolutions are automated.



```
SELECT {
  ?o customer ?c ;
    ...
}
```

```
order: {
  customer: 1,
  oid: 2023001,
    ...
}
```

**Figure 3: Workflow of the initial framework. The full arrows represent the flow of information with the U symbols marking user inputs. The dashed lines represent the flow of data.**

## 2.3 Evolution Management

On the most basic level, evolution means changing a database schema over time. However, when the schema changes, the data and the queries must also be updated. Thus, in the context of our framework, we do not see evolution as just another feature but as a fundamental quality of each part of the system.

The change usually starts with a modification of the schema category. First, we have identified several key *schema modification operations* (SMOs) [1], e.g., creating a new object or a new morphism. Then, we can build more complex operations from them, such as grouping several objects into a new one or moving objects around the graph. Then, we again leverage the mappings to propagate the changes to the system-specific models and the data. Lastly, the changes are propagated to the queries in two ways. On the one hand, we have to update the queries to reflect the changes in the schema. On the other hand, we use the mappings to decompose and translate the queries into the system-specific languages, which are also affected by the changes.

However, the change might start with the mappings (compare Figs. 1 and 2). For example, we might want to alter the way we store data in a specific database system or add a new system altogether. In that case, we have to update the mappings and then propagate the changes to the queries, though the schema category is unaffected.

*Example 2.2.* The evolution process is depicted in Fig. 4. The user changes the schema category, the mappings, or both. The changes are then automatically propagated to the rest of the system.

The propagation of the changes to the data can be, in some cases, mediated by the querying feature. For example, suppose we want to split an object (e.g., an Address) into two (e.g., a Street and a City) based on a specific rule. In that case, we may first create the new objects and then internally use a query with a `SPLIT` function to extract the data about the street and the city from the original object. □

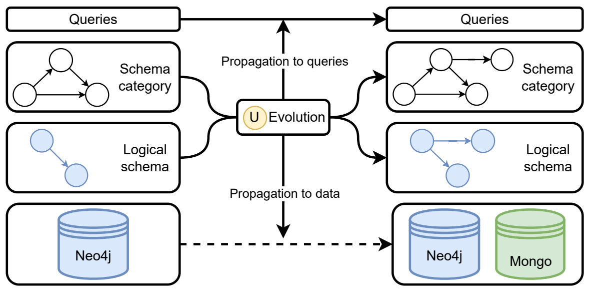


**Figure 4: Evolution in the framework.**

Currently, evolution is managed in the framework by two tools: *MM-evocat* [10] (dealing with propagation of SMOs to logical schemas and data instance) and *MM-evoque* [2] (dealing with propagation of SMOs to MMQL and system-specific query languages).

## 3 FUTURE PLANS

Currently, we have a solid theoretical background and proof-of-concept tools to integrate and extend to the final robust solution. It may seem that the only work left is to optimize the algorithms and finish the implementation. And that would be the case in the world of single-model databases, where we can rely on a skilled database administrator to manage the whole system. In the case of multi-model data, however, we still have to deal with different database systems, models, optimization strategies, etc. Our framework can shield the users from all databases' complexity and specific (often contradictory) features. But the principal question now becomes: "*How to optimally distribute data between the particular data models and respective database systems with requirements changing over time?*".

We cannot rely only on human experts for more complex use cases. Therefore, our main future aim is an AI-powered solution. The general idea is that the user would design the system-independent schema while the framework would decide how to distribute the data among the available database systems. And over time, the framework will propose modifications to the schema category.

Generally, there are three possible approaches – *rule-based*, *search-based*, and *model-based*. The first relies on a hard-coded rule set to generate optimal database settings [20]. The second one (e.g., [23]) searches the space of all possible configurations. The last one utilizes novel techniques such as deep reinforcement learning (as discussed in Section 3.1). However, to our knowledge, none of these approaches has been considered for multi-model databases. The rule-based strategy is not very flexible and extensible, so it would not be easy to cover all possible scenarios of the multi-model world. Similarly, the search-based one would be too expensive as the search space grows exponentially with the number of models and database systems. Hence, the model-based one is the most promising because it should be able to adapt to all possible scenarios, although it would require a lot of data to train the model.

### 3.1 AI for DB

The cooperation of AI with database systems can be divided into two categories [15]. The *DB4AI* approaches use databases to improve AI models. For example, the AI-native DBMS openGauss [16] supports native AI computing engine, model management, AI operators, native AI execution plan, etc.

Conversely, the *AI4DB* techniques leverage AI to improve DBMSs. These approaches include learning-based methods that utilize reinforcement or deep learning to optimize database functionality. Examples include configuration tasks such as index selection [9], partitioning advisory [5], and general knob tuning [22]. Additionally, AI4DB techniques extend to optimizing query performance [21], join selection [19], and influencing database design through learned indexes [3] or key/value design [6]. The applications also involve predicting, e.g., query arrival rates [18].

Our planned extension of the framework falls into the second category. We aim to optimize the management of multi-model databases by exploiting AI. All user inputs necessary in the current framework are depicted in Figs. 3 and 4. The schema category has to be modelled by the user (unless we can infer it from the data, which is still not fully automatic). Similarly, only the user knows what to query (but there are already techniques to infer this information from the application code, e.g., various ORM frameworks). Hence, we will primarily focus on automating the decomposition process and the evolution (in the form of a gradual optimization of how the schema is decomposed).

### 3.2 Data for Training

The quality of an AI model is highly dependent on the quality of training data. In our case, the expected data will be primarily logs from real-world databases. A member of our research group is already working on a wrapper for commonly used database systems that collect metadata for each query, ranging from the execution time, through the number of entities, to the percentage of entities that were filtered out in each step of the execution plan. The dataset collected during the process will have a value on its own. It can be used to train various AI models utilised for data management purposes or just as an insight into the performance of underlying database systems of our toolset.

There are also many open datasets and even some query datasets. But they usually lack the statistical information we need. Moreover, they tend to be not very diverse, usually focusing on a single single-model database system, which is quite the opposite of what we need. Hence, another task being solved by a member of our research group is a tool for transforming single-model real-world data sets to pseudo-realistic multi-model versions.

Both these approaches solve us the core problem of AI-based approaches – where to gain high-quality training data.

## 4 OPEN PROBLEMS

Besides identifying the data management tasks where AI can be utilised to lower user interaction and to select the optimal approach, we face more specific challenges.

*Extensible Approach.* Most contemporary AI4DB approaches are based on the ER model, thus suitable for relational databases only. The question is how challenging it will be to adapt them to the multi-model world. Generally, we want to adjust the existing approaches for the graph model because the schema category is, in essence, a graph. There are some similarities that we can exploit (e.g., both models are aggregate-ignorant), but there are also many differences that we must consider.

*Information Aggregation.* Primarily, we want to use the above-specified database logs as training data as they are not extensive. However, it is unclear what parts of the logs are the most important. Also, we are not sure if we need all the logs or just some kind of aggregation (e.g., the average execution time for each type of query). If we aggregate the data, we can fit much more information into the model, but we might lose some essential details.

*Data Quality.* To clean the real-world data sets, we will use verified techniques like anomaly detection to improve the data quality. Again, there are not many applications of such techniques in the multi-model world yet. Each data model brings challenges, and their solutions might require decisions from domain experts. For example, it is a common practice to remove a whole row if we find a single anomaly. However, if we apply the same approach to a document database with large documents, we might remove many otherwise valid data.

*Model Scale.* There are two options for the scale of the AI model. We can create a single general model trained on statistical data from many multi-model database instances. Alternatively, each instance can have its model and be trained on its data. The first option is more efficient in justifying a larger model. On the other hand, the second one might give us better results because of its customization. There is also a third option – create a single general model but fine-tune it for each instance.

In each of the cases, we plan to compare the most promising approaches experimentally and identify optimal approaches together with their parameters.

## ACKNOWLEDGMENTS

This work was supported by the GAUK grant no. 292323.

## REFERENCES

[1] Jáchym Bártík, Pavel Koupil, and Irena Holubová. 2024. Modelling and Evolution Management of Multi-Model Data. In *SAC '24: The 39th ACM/SIGAPP Symposium on Applied Computing (accepted)*. ACM.

[2] Jáchym Bártík, Pavel Koupil, and Irena Holubová. 2024. *MM-evoque*: Query Synchronisation in Multi-Model Databases. In *Submitted, under review*.

[3] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, et al. 2020. ALEX: An Updatable Adaptive Learned Index. In *Proc. of SIGMOD '20*. ACM, 969–984.

[4] Qingsong Guo, Chao Zhang, Shuxun Zhang, and Jiaheng Lu. 2023. Multi-model query languages: taming the variety of big data. *Distributed and Parallel Databases* (31 May 2023). Publisher Copyright: © 2023, The Author(s).

[5] Benjamin Hilprecht, Carsten Binnig, and Uwe Röhm. 2020. Learning a Partitioning Advisor for Cloud Databases. In *SIGMOD 2020*. ACM, 143–157.

[6] Stratos Idreos, Niv Dayan, Wilson Qin, Mali Akmanalp, et al. 2019. Design Continuums and the Path Toward Self-Designing Key-Value Stores that Know and Learn. In *Proc. of CIDR 2019*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p143-idreos-cidr19.pdf

[7] ISO. 2011. ISO/IEC 9075-14:2011 Information technology – Database languages – SQL – Part 14: XML-Related Specifications (SQL/XML).

[8] ISO. 2017. ISO/IEC TR 19075-6:2017 Information technology — Database languages — SQL — Part 6: SQL support for JavaScript Object Notation (JSON).

[9] Jan Kossmann, Stefan Halfpap, Marcel Jankrift, and Rainer Schlosser. 2020. Magic Mirror in My Hand, Which Is the Best in the Land? An Experimental Evaluation of Index Selection Algorithms. *Proc. of VLDB Endow.* 13, 11 (2020), 2382–2395. http://www.vldb.org/pvldb/vol13/p2382-kossmann.pdf

[10] Pavel Koupil, Jáchym Bártík, and Irena Holubová. 2022. *MM-evocat:* A Tool for Modelling and Evolution Management of Multi-Model Data. In *Proc. of CIKM 2022, Atlanta, GA, USA*. ACM, 4892–4896.

[11] Pavel Koupil, Daniel Crha, and Irena Holubová. 2023. MM-quecat: A Tool for Unified Querying of Multi-Model Data. In *Proc. of EDBT '23*. OpenProceedings.org, 831–834.

[12] Pavel Koupil and Irena Holubová. 2022. A Unified Representation and Transformation of Multi-Model Data using Category Theory. *J. Big Data* 9, 1 (2022), 61.

[13] Pavel Koupil, Sebastián Hricko, and Irena Holubová. 2022. MM-infer: A Tool for Inference of Multi-Model Schemas. In *Proc. of EDBT '22*. OpenProceedings.org, 2:566–2:569.

[14] Pavel Koupil, Martin Svoboda, and Irena Holubová. 2021. MM-cat: A Tool for Modeling and Transformation of Multi-Model Data using Category Theory. In *Proc. of MODELS '21*. IEEE, New York, NY, USA, 635–639.

[15] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. AI Meets Database: AI4DB and DB4AI. In *Proc. of SIGMOD '21*. ACM, 2859–2866.

[16] Guoliang Li, Xuanhe Zhou, Ji Sun, Xiang Yu, et al. 2021. openGauss: An Autonomous Database System. *Proc. of VLDB Endow.* 14, 12 (2021), 3028–3041. http://www.vldb.org/pvldb/vol14/p3028-li.pdf

[17] Jiaheng Lu and Irena Holubová. 2019. Multi-model Databases: A New Journey to Handle the Variety of Data. *ACM Comput. Surv.* 52, 3, Article 55 (2019), 38 pages.

[18] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, et al. 2018. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *Proc. of SIGMOD '18*. ACM, 631–645.

[19] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *Proc. of aiDM@SIGMOD 2018*. ACM, 3:1–3:4.

[20] Andy Pavlo. 2018. *What is a Self-Driving Database Management System?* A. Pavlo blog. https://www.cs.cmu.edu/~pavlo/blog/2018/04/what-is-a-self-driving-database-management-system.html

[21] Ji Sun and Guoliang Li. 2019. An End-to-End Learning-based Cost Estimator. *Proc. of VLDB Endow.* 13, 3 (2019), 307–319.

[22] Ji Zhang, Ke Zhou, Guoliang Li, Yu Liu, et al. 2021. CDBTune+: An Efficient Deep Reinforcement Learning-Based Automatic Cloud Database Tuning System. *VLDB J.* 30, 6 (2021), 959–987.

[23] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, et al. 2017. BestConfig: Tapping the Performance Potential of Systems via Automatic Configuration Tuning. In *Proc. of SoCC '17*. ACM, 338–350.