Supporting Schema Evolution in Hybrid Database Systems

Maxime Gobert
University of Namur, Belgium
Supervised by Anthony Cleve
maxime.gobert@unamur.be

ABSTRACT

Relational database management systems (RDBMS) and NoSQL database systems have been built in order to meet different requirements. Organisations are therefore increasingly using hybrid data persistence architectures, that we call *hybrid database systems* or *hybrid polystores*. The evolution of such systems is more complex due to potential data duplication, migration across different technologies and data heterogeneity. Existing work on NoSQL database evolution exposes the different problems regarding data heterogeneity and mainly provide technology-specific solutions. In this PhD research we propose to apply a schema engineering approach to handle *hybrid database system* evolution at a conceptual level. Our starting point is an existing unified conceptual data model for hybrid databases. Our thesis objectives include (1) to enrich this conceptual data model with generic schema evolution operators, (2) to specify the semantics of those operators depending on the underlying data models, (3) to design a generic evolution framework and (4) to develop its proof-of-concept implementation. Our approach will help to propagate conceptual schema changes to all impacted software artefacts, namely native data structures, database contents, queries and programs.

1. RESEARCH CONTEXT

According to Sadalage et al. [12] the ideal solution for data management in concrete real life scenarios will consist of multiple technology usage . Database evolution in such *hybrid database system* or *polystore* involves the joint modification of multiple software artefacts and heterogeneous data models. It is therefore error prone and lead to database inconsistencies or to failures in the application programs relying on them. In such a context, there is a significant gap between the ideal solution and the current situation faced by database administrators and software developers. Below, we present the different challenges to address nowadays when evolving a hybrid database system.

Proceedings of the VLDB 2020 PhD Workshop, August 31st, 2020. Tokyo, Japan. Copyright (C) 2020 for this paper by its authors. Copying permitted for private and academic purposes.

In the current situation the developer is left alone in the evolution process. Developers have first to identify which system artefacts are impacted by the considered schema evolution scenario, and how to propagate the schema evolution operations in terms of data migration, native data structure changes and query adaptations. Let us consider an industrial hybrid data-intensive system made up of several application programs that jointly access a document-oriented database, a relational database and a key-value store. Let us assume a simple schema evolution scenario: changing the name of an attribute in an entity type called *products*. This change would possibly require the adaptation of the native database structures (if the products are stored in a relational database), of the database contents (if the products are stored as a document database collection). It would also involve the rewriting of all queries manipulating the products with an explicit reference to the renamed attribute. Furthermore, some programs may also access the products via mapped data access objects or have some programming logic relying on the renamed attribute. The related code fragments would therefore also require some adaptations. The context of hybrid database system adds another layer of difficulty, as in the above scenario we could imagine that the products are stored *both* in the document-oriented database and in a relational database, i.e., with some (partial) overlapping between both data stores.

We can see, already through a simple evolution scenario, that schema evolution in a hybrid database system may prove complex, time-consuming and error-prone for database administrators and developers, as multiple and heterogeneous software artefacts have to be consistently modified.

We argue that this evolution process would be easier to handle by means of an integrated evolution framework, that would define generic hybrid schema evolution operations. Those evolution operations should be expressed on top of a unified, conceptual data model, covering multiple heterogeneous data modeling paradigms. This model is the entry point of the evolution process as it enables the propagation operations, to be applied automatically to the underlying native database(s) and to the related software artefacts (data, queries, programs).

We identify the following research questions that we will try to answer during this PhD research:

RQ 1. How can we specify generic schema evolution operators for hybrid database systems? In this question we identify which conceptual schema evolution operators are needed to evolve a hybrid database system.

RQ 2. How can we design an evolution framework propagating such evolution operations to all impacted software artefacts? An evolution operator may have multiple impacts on several artefacts of the system. This question will establish a generic impact matrix that specifies atomic, paradigm-specific propagating changes to apply depending on the mapping between the conceptual data model and the native backends.
RQ 3. To what extent can this framework actually help users in evolving hybrid database systems? We plan to develop a proof-of concept implementation of our framework and to evaluate, in realistic conditions, its usability, completeness, correctness, and fitness-for-use.

2. STATE OF THE ART

NoSQL data models and design

NoSQL systems can be grouped into four main different data models, each having its specific requirements and advantages. We can cite *key value stores*, *document databases*, *column databases* or *graph databases* and reference Hecht and Jablonski [6] for further details on their data models.

NoSQL data models are mostly considered as schema-less models as they are oriented towards flexibility. However Sadalage et al. [12] argue that there is always a schema that consists of the assumptions made on the data in the application code. Therefore poor design decisions may significantly affect scalability, performance and data consistency.

Several authors have proposed a generic model combining heterogeneous NoSQL data models. Atzeni et al. [2] introduced NoAM (NoSQL Abstract Model), an abstract data model for NoSQL databases, which exploits the commonalities of various NoSQL data models. Database design is based on aggregate identification and partitioning, *entity per aggregate object*, *entity per atomic value* and *entry per top level field* are identified as specific modeling strategies that can apply to heterogeneous data models. Another proposal by Abdelhedi et al. [1] relies on a model-driven approach, where transformation rules expressed on a conceptual database schema are used create NoSQL (specifically a column-oriented) logical and physical schemas.

Our work relies on a TyphonML [21], a unified modeling language bringing a concrete method for designing and developing hybrid polystores. TyphonML combines relational model and NoSQL data models. It allows users to define the conceptual entities that the hybrid polystore has to manage, and to specify how those entities have to be actually stored in terms of native databases. This model constitutes the starting point of our evolution framework as we extend it with evolution operators.

Database evolution & migration

Curino et al. [4] developed an automatically-supported approach to relational database schema evolution, called the *PRISM* framework. They specified *Schema Modification Operators* representing atomic schema changes, and they link each of these operators with native modification functions for both data and queries. Our work aims at developing a similar evolution framework, but designed for *hybrid* database systems, relying on several different relational and NoSQL technologies.

Database evolution according to Cleve et al.[3] is classified according to three dimensions: The structural, the semantic and the language/platform dimension. The platform dimension characterises *intra-paradigm* database evolution and *inter-paradigm* database evolution. This reflects the fact that the evolution scenario may involve, or not, the migration of the data towards another data model. Inter-paradigm database evolution, a.k.a. database migration, has been widely studied. Some approaches [9, 23, 22] rely an intermediary model to migrate data from relational to NoSQL data models, while other ones [8, 11, 18] follow direct one-to-one mapping migration strategies. Our approach will reuse some of those migration techniques in order to allow users to easily migrate from one platform to another.

NoSQL database evolution

Existing research on NoSQL database evolution is more recent, yet several related problems have already been explored. Some approaches are dedicated to how to specify and formalize NoSQL database evolution, i.e, NotaQL [17]. Other authors focus on data heterogeneity problems, caused by the co-existence of multiple data versions in the same system [15]. *Data migration* research is targeted on how data can be migrated to another representation, with a wide spectrum of solution such as lazy, eager or predictive data migration [20, 16, 7], intermediary tools [5, 13, 14], intermediary language or even Object NoSQL mappers [19, 10].

3. EVOLUTION FRAMEWORK

The core expected contribution of our PhD thesis is a generic framework for supporting schema evolution in hybrid database systems. We describe this framework below.

Inputs M_{Source}, represents the hybrid database schema at both conceptual and logical levels. $\{EO\}$ is a set of schema evolution operators to apply to M_{Source}. $\{DB_{Source}\}$ is the set of source native data structures and data instances. $\{Q_{Source}\}$ is the set of existing queries expressed on M_{Source}.
Outputs M_{Target} is the source model adapted according to the evolution operators in $\{EO\}$. $\{DB_{Target}\}$ represents the set of database structures and instances adapted according to those operators. $\{Q_{Target}\}$ gives, when possible, equivalent queries as $\{Q_{Source}\}$, but expressed on M_{Target} . $\{DAO\}$ is a set of database access classes on specific drivers.

Figure 1 summarizes the different components of our framework. The rest of this section will further detail each of those components.

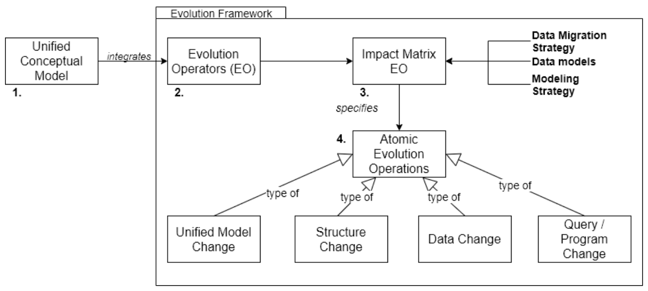

Figure 1: Evolution Framework

Unified Conceptual Model (1)

Our approach starts from a data model that allows one to design hybrid polystore systems. The design principles described in this section can be applied to any model and several were listed above in Section 2. In our work we used the Typhon Modeling Language (TyphonML) developed by one of our research partners of the Typhon project (see Section 5). Using such a unified model to evolve hybrid database systems reduces the complexity by keeping software artefacts consistent with each other. We implement a top-down evolution approach of evolution from model to the related software artefacts (data, queries, programs). The model is extended to allow users to trigger the execution of evolution operations.

Evolution Operators (2)

The first step of our PhD thesis was to define and specify the evolution operations that a user can apply to the hybrid polystore data model. Those operations are the entry point of the whole evolution process. An Evolution Operator (EO) is a combination of a domain model object and an evolution operation. The domain model object can be either *entity type, attribute, relationship*, or *identifier.* The available evolution operations span from simple changes, manipulating a single domain object *add, remove, rename, modify* to more complex ones such as *horizontal split, vertical split, merge*, or *migrate.* Figure 2 gives an example list of EOs, that can be expressed in our extended data model syntax.

```
evolutionOperators [
  add attribute rating : int to Review,
  add relation responses to Review -> Comment[0..*],
  merge entities User CreditCard as User
]
```

Figure 2: Example of evolution operators expressed within the TyphonML unified model

Impact Matrix EO (3)

The impact matrix is the central point of our framework as it defines the atomic operations that need be applied to actually propagate the polystore schema evolution operators to related software artefacts. The rows of the matrix correspond to the available Evolution Operators. The columns of the matrix then characterize each of the operators according to several dimensions; each influencing the propagating operations that should be applied. The first dimension specifies the underlying data models of the object(s) subject to evolution. The possible values of this dimension include *relational, document, key-value* or *graph* databases. The second dimension concerns the chosen modeling strategy, as explained in Section 2. The third dimension relates to the chosen data migration strategy. Each cell of the impact matrix specifies the set of atomic, paradigm-specific operations to apply to the polystore artefacts (native structures, data, queries, etc.), in order to propagate the requested EOs according to the dimensions' values.

Atomic Evolution Operations (4)

The set of atomic operations will be an exhaustive list of function specifications. Their goal is to enable the actual transformation required to propagate the desired schema evolution. They are classified into four categories, depending on which software artefacts are involved.

Unified conceptual schema change

As we want to keep the polystore conceptual model and the related polystore artefacts consistent with each other, each evolution scenario has to be triggered by means of a list of *evolution operators* expressed on top of the conceptual polystore schema. Our framework applies changes corresponding to each evolution operator specified in the $\{EO\}$ list. Each time an EO is executed, a new intermediary conceptual model is produced. This intermediary model is then used as the source schema for the following operators. The final output is a new conceptual model resulting from the application of all evolution operators.

Propagation to native data structures

According to the underlying logical mappings some EOs require the adaptation of the native data structures that are mapped with the conceptual schema objects subject to evolution. This adaptation, in the case of a relational database, translates as SQL *DDL* commands, for example *create table, alter table, create index, add constraint,. . . .* Equivalent structure manipulation for NoSQL data models include *create collection, create index* for document data model, *create table, create column family* for column databases. Key value and graph data models are pure data and do not have equivalent structure objects. Note that in our work we have not planned to take into account the specific physical aspects of NoSQL databases, meaning that we do not plan to handle the physical storage of data and cloud computing aspects.

Propagation to data instances

The propagation of EOs to data instances aims at transforming/migrating the data in order to make them comply with the target polystore data model/paradigm. As an example, a possible strategy (EAO, see Section 2) to represent a relationship between two conceptual entities in a document-oriented database is to use two specific collections, one for each entity type. The source entity of the relationship will then have a reference attribute referring to the identifier of the other collection. In this case, renaming an attribute of the target entity type would necessitate to rename the corresponding attribute for each data object in the target collection. Another representation strategy (ETF) of this relationship would consist in using a single document collection, with each data object including the referenced data object as a nested object. The same renaming operation would, instead, require the adaptation of the nested objects.

Propagation to database queries and programs

EO operations may also require the adaptation of existing database queries, that would become invalid due to the applied schema changes. Depending on the semantics-preserving nature of each EO involved in the scenario, it will be possible or not to translate the source database queries into equivalent queries expressed on the target polystore schema, using rule based transformations. Simple *Data Access Object* classes in programs can also be generated or modified according the received EOs.

4. CURRENT RESULTS AND NEXT STEPS

The schema evolution framework presented in this paper has been partially implemented in the context of the Typhon EU H2020 research project. A total of 18 evolution operators are now fully specified and implemented. At this stage of our research, we only consider mappings to relational and document-oriented databases, and we follow an eager data migration strategy. In the next coming months, our framework implementation will be used by four industrial partners, by considering realistic use-case scenarios. Future work in this PhD thesis includes (1) the specification of more complex and composite evolution operators, (2) the implementation of other data migration strategies (lazy, predictive or query based) in the impact matrix, and (3) the extension of our framework to key-value and column-oriented data models.

5. ACKNOWLEDGEMENTS

This research was partly carried out in the context of the EU H2020 Typhon research project, the goal of which is to provide an industry-validated methodology and integrated technical offering for designing, developing, querying, evolving, analyzing and monitoring scalable hybrid data persistence architectures. This PhD thesis research is supported by the Excellence of Science project 30446992 SECO-ASSIST, funded by the F.R.S.-FNRS and the FWO.

6. REFERENCES

[1] F. Abdelhedi, A. Ait Brahim, F. Atigui, and G. Zurfluh. Processus de transformation mda d'un schéma conceptuel de données en un schéma logique nosql. In *34e Congrès Informatique des Organisations et Systèmes d'Information et de Décision 2016.*

[2] P. Atzeni. Data modelling in the nosql world: A contradiction? In *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016*, CompSysTech '16, pages 1–4. ACM.

[3] A. Cleve. *Program analysis and transformation for data-intensive systems evolution.* PhD thesis, University of Namur, 2009.

[4] C. A. Curino, H. J. Moon, A. Deutsch, and C. Zaniolo. Automating the database schema evolution process. *The VLDB Journal*, 22(1):73–98, Feb. 2013.

[5] F. Haubold, J. Schildgen, S. Scherzinger, and S. Deßloch. Controvol flex: Flexible schema evolution for nosql application development. *Datenbanksysteme für Business, Technologie und Web (BTW 2017).*

[6] R. Hecht and S. Jablonski. Nosql evaluation: A use case oriented survey. In *2011 International Conference on Cloud and Service Computing*, pages 336–341.

[7] M. Klettke, U. Störl, M. Shenavai, and S. Scherzinger. Nosql schema evolution and big data migration at scale. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 2764–2774.

[8] C. Li. Transforming relational database into hbase: A case study. In *2010 IEEE international conference on software engineering and service sciences*, pages 683–687.

[9] D. Liang, Y. Lin, and G. Ding. Mid-model design used in model transition and data migration between relational databases and nosql databases. In *2015 IEEE International Conference on Smart City/SocialCom/SustainCom*, pages 866–869.

[10] A. Ringlstetter, S. Scherzinger, and T. F. Bissyandé. Data model evolution using object-nosql mappers: Folklore or state-of-the-art? In *proc. of the 2nd International Workshop on BIG Data Software Engineering*, pages 33–36. ACM, 2016.

[11] L. Rocha, F. Vale, E. Cirilo, D. Barbosa, and F. Mourão. A framework for migrating relational datasets to nosql. *Procedia Computer Science*, 51:2593–2602, 2015.

[12] P. J. Sadalage and M. Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence.* Pearson Education, 2013.

[13] K. Saur, T. Dumitraş, and M. Hicks. Evolving nosql databases without downtime. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 166–176. IEEE.

[14] S. Scherzinger, M. Klettke, and U. Störl. Cleager: Eager schema evolution in nosql document stores. *Datenbanksysteme für Business, Technologie und Web (BTW 2015).*

[15] S. Scherzinger, M. Klettke, and U. Störl. Managing schema evolution in nosql data stores. *arXiv preprint arXiv:1308.0514*, 2013.

[16] S. Scherzinger, U. Störl, and M. Klettke. A datalog-based protocol for lazy data migration in agile nosql application development. In *proc. of the 15th Symposium on Database Programming Languages*, pages 41–44. ACM, 2015.

[17] J. Schildgen, T. Lottermann, and S. Deßloch. Cross-system nosql data transformations with notaql. In *proc. of the 3rd ACM SIGMOD Workshop on Algorithms and Systems for MapReduce and Beyond*, page 5. ACM, 2016.

[18] L. Stanescu, M. Brezovan, and D. D. Burdescu. Automatic mapping of mysql databases to nosql mongodb. In *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 837–840. IEEE, 2016.

[19] U. Störl, T. Hauf, M. Klettke, and S. Scherzinger. Schemaless nosql data stores-object-nosql mappers to the rescue? *Datenbanksysteme für Business, Technologie und Web (BTW 2015).*

[20] U. Störl, D. Müller, A. Tekleab, S. Tolale, J. Stenzel, M. Klettke, and S. Scherzinger. Curating variational data in application development. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1605–1608.

[21] The University of L'Aquila. Deliverable D2.3 - Hybrid Polystore Modeling Language (Final Version), 2018.

[22] J. Yoo, K.-H. Lee, and Y.-H. Jeon. Migration from rdbms to nosql using column-level denormalization and atomic aggregates. *Journal of Information Science & Engineering*, 34(1), 2018.

[23] G. Zhao, Q. Lin, L. Li, and Z. Li. Schema conversion model of sql database to nosql. In *2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 355–362. IEEE.