An integrated approach for column-oriented database application evolution using conceptual models

Pablo Suárez-Otero[1], Michael J. Mior[2], María José Suárez-Cabal[1] and Javier Tuya[1]

[1] University of Oviedo, Gijón, Spain
{suarezgpablo, cabal, tuya}@uniovi.es[1]
[2] Rochester Institute of Technology, Rochester NY, USA
mmior@cs.rit.edu[2]

Abstract. Schema design for NoSQL column-oriented database applications follows a query-driven strategy where each table satisfies a query that will be executed by the client application. This strategy usually implies that the schema is denormalized as the same information can be queried several times in different ways, leading to data duplication in the database. Because the schema does not provide information such as where the data is duplicated or the relationships between conceptual entities, developers must use additional information when evolving the database. One strategy for accessing this information is to use a conceptual model that must be synchronized and kept consistent with the physical schema. In this work, we propose evolving a column-oriented database application after a schema change with a combination of methods that consists of four sequential stages: 1) reflect the schema change in the conceptual model, 2) take the necessary actions in the schema to maintain consistency between the new conceptual model and the schema, 3) maintain data integrity through migration of data and 4) update and adapt the client application to the new schema.

Keywords: evolution, column-oriented database, data integrity, program repair

1 Introduction

The design of a database is based on both the requirements of applications and the characteristics of the DBMS. For instance, the schema of a relational database is designed based on the data that will be stored, with emphasis on producing a normalized model. On other hand, schema design for NoSQL databases follows different strategies based on how the data is expected to be used by applications that modify and display the data. For instance, column-oriented databases follow a query-driven approach to schema design to achieve high performance when executing queries, where each table commonly satisfies a query of the client application.

This query-driven approach implies a denormalized model as the same information can be queried several times in different ways. This leads to the storage of the same information in different tables of the database. Therefore, evolution of a schema that changes a particular table or column might affect other parts of the database where the same information is stored. Failure to apply the required changes to the rest of the

database may result in problems regarding consistency, increasing the risk of columns storing incorrect data instead of the data that they were intended to store.

In relational databases, where the schema is usually normalized, each table stores the information of a single entity or relationship, so the schema is very similar to the conceptual model. Due to this, a common practice is to evolve the database directly via changing the schema. However, in NoSQL databases, and particularly in a column-oriented database, evolution of the schema requires more information. If a developer evolves one of these databases only considering the information provided by the schema, they risk committing mistakes during this evolution because of the lack of relevant information such as how and where the data is duplicated or the relationships that exist between conceptual entities.

This information can be provided by a conceptual model [3, 7] that, in order to remain useful, must always be synchronized with the schema. An incorrect evolution of the database may affect this synchronization by having database structures that contradict the conceptual model, also provoking issues related to data integrity. Client applications are also affected, as they may contain bugs caused by this incorrect evolution, such as code referring to entities that are either not considered in the schema or have different properties. There also have been approaches to infer a conceptual model from the database schema [1, 2], helping the evolution of database schemas that had been designed without employing a conceptual model.

To properly evolve a column-oriented database application from a change in the schema, we propose a collection of methods where each one solves a particular issue using conceptual models. These methods will determine the actions required to keep the conceptual model and the schema synchronized, the data integrity of the database and the database client applications updated to the current version of the schema.

The remainder of this paper is structured as follows. Section 2 contains motivation for this work based on a real scenario. Section 3 details the combination of methods and Section 4 contains the conclusion and future work.

2 Motivation

We have studied the evolution of the schema in several open-source projects that use column-oriented databases to store their data [10]. One of these projects is PowSybl[1], which is a framework for real and simulated power systems that has had 35 schema versions during its lifetime. In prior work [10] we focused on issues caused by an incorrect evolution of the database due to lack of consideration of additional information contained in the conceptual model, such as where data are duplicated.

One of the identified changes that is prone to cause these issues is the addition of a new column. In the PowSybl project, there was a version[2] where 15 tables were modified by adding the same two columns to each of them, “bus” and “connectableBus”,

[1] https://github.com/powsybl/powsybl-network-store

[2] https://github.com/powsybl/powsybl-network-store/blob/3c962d5c8f78f56c7be6e7729be2a2ca910c2bcf/network-store-server/src/main/resources/iidm.cql

which store information of a new entity named “BusBreaker” and relationships between it and other entities. The week following this evolution, a bug was detected and reported to the developers of the project, warning about an issue related to the schema, which did not have the database structures required to store relationships between “BusBreaker” and “DanglingLine”. The developers of PowSybl fixed this bug[3] by adding the columns “bus” and “connectableBus” to a table that was not part of the original 15 tables. As NoSQL column-oriented database applications are focused on storing big data, a mistake like this one can imply the loss of great amounts of data, as all the insertions of relationships between instances of “BusBreaker” and “DanglingLine” would be lost during that week. This scenario is illustrated in **Fig. 1**.

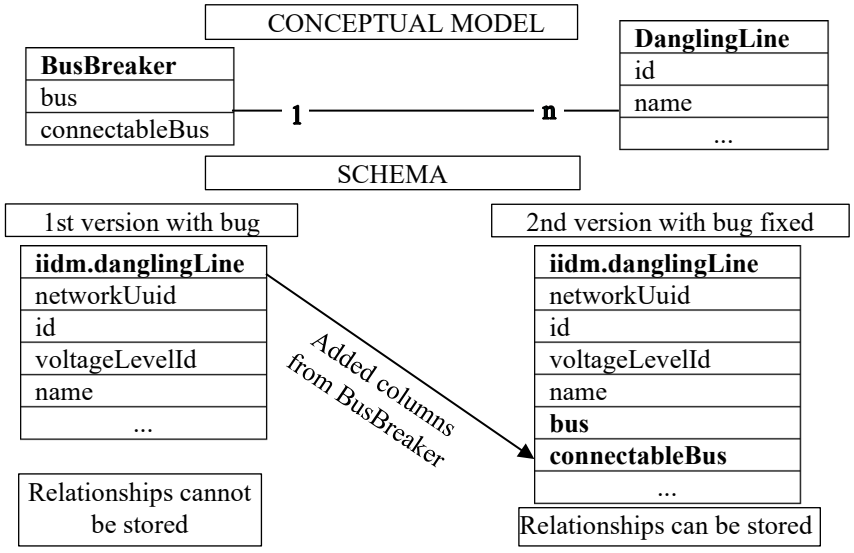

Fig. 1. Fix of the schema to include a new conceptual requirement

The addition of a new column can result from several possible scenarios from a conceptual point of view such as a new conceptual entity, a new relationship, or new attributes. Depending on the type of the change in the conceptual model, developers must perform a specific set of actions to evolve the database, which might involve performing changes in other tables and columns from the schema, data migrations, and changes in the applications that work with the database. Our objective in this paper is to help developers avoid problems related to the evolution of column-oriented database applications when there is a change in the schema by using more information in addition to the schema. In the next section, we propose a combination of methods that uses a conceptual model synchronized with the schema that provides the required information to evolve the database and avoid mistakes.

[3] https://github.com/powsybl/powsybl-network-store/blob/49a2745c43b510cd89ada8de9950e1b33e3c8e2e/network-store-server/src/main/resources/iidm.cql

3 Evolution procedure

We propose a procedure to evolve the schema that starts with a change in the database structures of the schema and uses a conceptual model that is synchronized with the schema. For this synchronization to be useful, inter-model consistency must be maintained, which assures that the conceptual model is a normalized version of the schema.

The initial information required for the proposed procedure are the current conceptual model, the current schema, a mapping between conceptual model and schema and the change in the schema that triggers this procedure. This procedure is composed of four stages that are illustrated in **Fig. 2** along with their interactions with each other. Each stage resolves a particular issue through specific steps:

1. Change the conceptual model to maintain the inter-model consistency:
   1.1. Determine the changes in the conceptual model required to reflect the change in the schema.
   1.2. Apply the changes determined in 1.1 to the conceptual model.
2. Change the schema to maintain the inter-model consistency:
   2.1. Determine the changes in the rest of the schema required to maintain inter-model consistency with the new conceptual model obtained in 1.
   2.2. Apply the modifications from 2.1.
3. Migrate the required data to the created or modified database structures to maintain data integrity:
   3.1. Determine tables that might exhibit data integrity problems.
   3.2. Determine the migration of data required to maintain data integrity, identifying the tables from where to get the data.
   3.3. Execute the migrations determined in 3.2.
4. Update the client application database statements to the new schema:
   4.1. Identify the code and database statements that need to be updated.
   4.2. Modify the queries and code of the client application identified in 4.1.

In the following paragraphs we describe each stage in detail, detailing their objective and the methods that we propose to use in them.

Stage 1 aims to maintain inter-model consistency after a change to the schema by performing the required updates to the conceptual model to reflect the change in the schema. We propose using existing work that studies the renormalization of the schema in a normalized conceptual model [8, 9]. These works propose generating a normalized conceptual model based on a column-oriented schema. We use past work focused on column-oriented databases [8] as a guideline for determining the specific changes to be performed in the conceptual model for each change in the schema.

Stage 2 aims to regain inter-model consistency by applying in the required changes to schema in order to adapt it to the conceptual model generated in stage 1. For instance, an update to the cardinality of a relationship needs to be replicated to every table that stores the relationship. In past work [10], we defined a method which, given a conceptual model, provides the instructions required to reflect the change in the schema. Due to being focused on evolution, we propose using this method for stage 2. To complete the scenarios to cover, we will also use the ones focused on designing a column-oriented

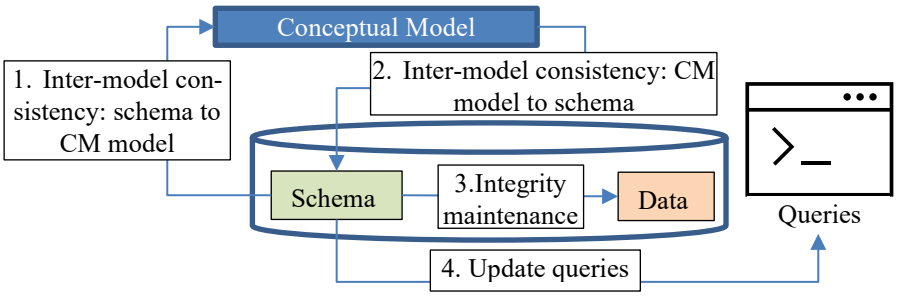

Fig. 2. Full evolution of the database procedure

database schema from a conceptual model and the queries that are going to be executed by the client application [3, 4, 7].

When the change in the schema does not modify the conceptual model, stages 1 and 2 are skipped. For instance, there could be a new table that satisfies a query that retrieves information already defined in the conceptual model. In this case however, there could be problems related to data integrity, so stages 3 and 4 are still required.

Stage 3 aims to maintain the data integrity of the database. This integrity is jeopardized due to data duplication caused by the denormalization of the schema, as data needs to be kept consistent. To address this problem, we will use a combination of several methods. To identify the scenarios where migrations of data are required to maintain the data integrity after the change in the schema, we will use the method defined in [10], which describes the specific migrations required for each schema change type. The identification of the source tables to get the data to be migrated will be based on the method defined in [11], where we used conceptual models to address an automatic maintenance of the data integrity in scenarios where it was jeopardized. The migration of data process will be performed following the previously defined strategies [6] in order to achieve the best performance possible during the migration process.

The objective of stage 4 is to update the client application by adapting it to the new schema. Although there is no existing work that studies this specific problem, program repair approaches [5] are an interesting option. These approaches aim to fix a bug or solve an inconsistency in software. We propose using a similar approach that updates the application in both the database statements that are embedded in the application and the application code that prepares the database statements and process the result of the execution of these database statements.

4 Conclusion and future work

The incorrect evolution of a database causes important problems like data loss or the creation of inconsistencies between the schema and the conceptual view of the data system, which were shown in a real-world scenario. In this work, we proposed a combination of methods that perform the complete evolution of a column-oriented database application after a change in the schema, approaching the schema, the conceptual model, data integrity and client applications. To the best of our knowledge, this is the

first work that approaches the evolution of column-oriented database applications in all stages. At the moment, we have developed the solutions for the maintenance of the inter-model consistency from changes in the conceptual to the schema, as well as the identification of the data migrations required to maintain the data integrity.

As future work, we intend to focus on stages whose methods either need to be adapted or that have yet to be developed. For stage 1 we intend to adapt previous work [8] to orientate it for schema evolution. For stage 4 we plan to develop a new method based on program repair is able to update a client application for the new schema. Finally, we aim to automate the process by integrating all methods, connecting the outputs of each stage with the input of the following one.

Acknowledgments

This work was supported by the TestBUS project (PID2019-105455GB-C32) of the Ministry of Science and Innovation, Spain and the TESTEAMOS project (TIN2016-76956-C3-1-R) of the Ministry of Economy and Competitiveness, Spain.

References

1. Abdelhedi, F., Brahim, A. A., Ferhat, R. T., Zurfluh, G. Reverse Engineering Approach for NoSQL Databases. In International Conference on Big Data Analytics and Knowledge Discovery, pp. 60-69. Springer, Cham (2020).
2. Akoka, J. and Comyn-Wattiau, I. Roundtrip engineering of NoSQL databases. Enterprise Modelling and Information Systems Architectures, 13, 281-292 (2018)
3. Chebotko, A., Kashlev, A., Andrey, L., Lu, S. A big data modeling methodology for Apache Cassandra. In 2015 IEEE International Congress on Big Data, pp. 238-245 (2015)
4. de la Vega, A., García-Saiz, D., Blanco, C., Zorrilla, M., Sánchez, P. Mortadelo: Automatic generation of NoSQL stores from platform-independent data models. Future Generation Computer Systems 105, pp. 455-474 (2020)
5. Gopinath, D., Khurshid, S., Saha, D., Chandra, S. Data-guided repair of selection statements. Proceedings of 36th International Conference on Software Engineering, pp.243-253 (2014)
6. Hillenbrand, A., Störl, U., Levchenko, M., Nabiyev, S., Klettke, M. Towards Self-Adapting Data Migration in the Context of Schema Evolution in NoSQL Databases. In 2020 IEEE 36th International Conference on Data Engineering Workshops, pp. 133-138 (2020)
7. Mior, M. J., Salem, K., Aboulnaga, A., Liu, R. NoSE: Schema design for NoSQL applications. IEEE Transactions on Knowledge and Data Engineering. 29(10), 2275-2289 (2017)
8. Mior, M. J., and Salem, K. Renormalization of NoSQL database schemas. In International Conference on Conceptual Modeling, pp. 479-487. Springer, Cham (2018).
9. Ruiz, D.S., Morales, S.F. and Molina, J.G. Inferring versioned schemas from NoSQL databases and its applications. In International Conference on Conceptual Modeling, pp. 467-480. Springer, Cham (2015)
10. Suárez-Otero, P., Mior, M. J., Suárez-Cabal, M. J., Tuya, J. Maintaining NoSQL Database Quality During Conceptual Model Evolution. In 2020 IEEE International Conference on Big Data (Big Data), pp. 2043-2048 (2020)
11. Suárez-Otero, P., Suárez-Cabal, M. J., Tuya, J. Leveraging conceptual data models to ensure the integrity of Cassandra databases. Journal of Web Engineering. 18 (6), 257-286 (2019)