Datalution: A Tool for Continuous Schema Evolution in NoSQL-Backed Web Applications

Stefanie Scherzinger
OTH Regensburg, Germany
stefanie.scherzinger
@oth-regensburg.de

Stephanie Sombach
OTH Regenburg, Germany
stephanie.sombach
@st.oth-regensburg.de

Katharina Wiech
OTH Regensburg, Germany
katharina1.wiech
@st.oth-regensburg.de

Meike Klettke
University of Rostock, Germany
meike.klettke
@uni-rostock.de

Uta Störl
Darmstadt University
of Applied Sciences, Germany
uta.stoerl@h-da.de

ABSTRACT

When an incremental release of a web application is deployed, the structure of data already persisted in the production database may no longer match what the application code expects. Traditionally, eager schema migration is called for, where all legacy data is migrated in one go. With the growing popularity of schema-flexible NoSQL data stores, *lazy* forms of data migration have emerged: Legacy entities are migrated on-the-fly, one at-a-time, when they are loaded by the application. In this demo, we present *Datalution*, a tool demonstrating the merits of lazy data migration. *Datalution* can apply chains of pending schema changes, due to its Datalog-based internal representation. The *Datalution* approach thus ensures that schema evolution, as part of continuous deployment, is carried out correctly.

CCS Concepts

•**Information systems** → **Database utilities and tools;**
•**Software and its engineering** → *Software evolution;*

Keywords

Software evolution; Schema evolution; NoSQL data stores

1. INTRODUCTION

New releases of an application can be disruptive when the application code no longer matches the structure of data already stored in the production database [1]. Thus, the application may be facing serious robustness and therefore quality issues. Traditionally, legacy data is migrated *eagerly*, meaning all legacy data is migrated in one go. Popular tools like Flyway and Liquibase (for relational databases), or Mongeez (for MongoDB) follow this approach, usually triggering migration at application startup. Eager migration requires careful deployment, to avoid application downtime, and may be costly when it is causing billable reads and writes in a cloud-hosted database.

With schema-flexible NoSQL data stores, application engineers may proceed *lazily* instead: At the time of a new release, legacy entities remain unchanged. Rather, they are migrated on demand, one at-a-time, when they are loaded into the application. When only a smaller share of the legacy data will still be accessed by the application (often called "hot" as opposed to "cold" data), the migration costs in terms of billable reads and writes are effectively reduced.

In this paper, we present *Datalution*, a tool that makes the tradeoffs between eager and lazy schema evolution transparent. *Datalution* simulates the schema-flexible NoSQL backend of a web application, where entities are stored in JSON format. For the given application release, the schema of entities is fixed. This assumption is only natural when object mappers implicitly declare the current schema.

The schema may change with a new release. *Datalution* supports adding, renaming, or removing attributes. It further allows to copy or move attributes between entities, and thus changes that involve more than one entity. The application itself adds, updates, and loads entities from its backend. When a legacy entity is loaded into the application, the challenge is to return it in the schema required by the current application release. *Datalution* achieves this by capturing the state of the data store, the pending schema changes, as well as the application accessing data, with Datalog rules. Datalog is a programming language for deductive databases. Programs can be evaluated bottom-up or top-down, and sophisticated algorithms avoid computing unrelated intermediate results in top-down evaluation. Our tool makes use of this: In carrying out schema changes eagerly, we evaluate Datalog rules bottom up. This migrates all legacy entities. For lazy migration, we evaluate rules top-down, thus migrating only the legacy entities that are actually requested by the application. One strong point of *Datalution* is that it can lazily roll forward chains of schema changes.

As a downside, lazy migration increases the latency of loading entities. As experiments in related work show, this overhead is affordable for adding and removing attributes [2].

History of Datalution: The theory behind *Datalution* is described in [3,4]. While *Datalution* has been presented in a poster session [5], this is the first tool paper on *Datalution*.

QUDOS'16, July 21, 2016, Saarbrücken, Germany
© 2016 ACM. 978-1-4503-4411-1/16/07...$15.00
http://dx.doi.org/10.1145/2945408.2945416

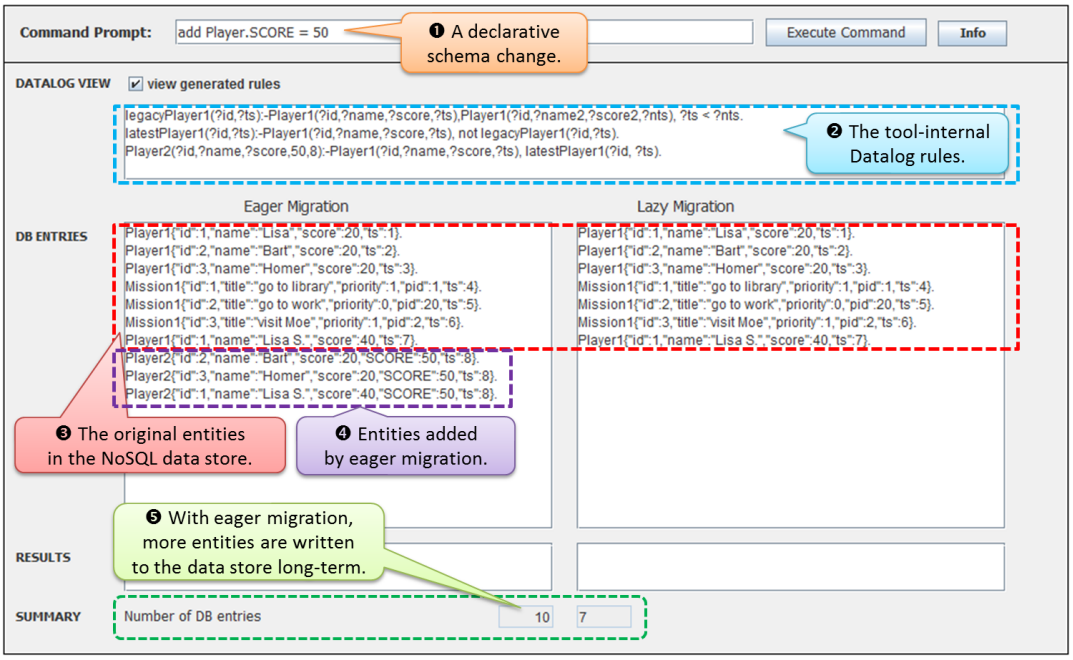

Figure 1: Screenshot of *Datalution*, showing the data store contents for eager and lazy data migration.

2. TOOL DEMO OUTLINE

Figure 1 shows a screenshot of *Datalution*. The setting is a gaming application persisting entities in a schema-flexible NoSQL data store. The outline of our tool demo is as follows:

1. We show the initial entity set, consisting of players and the missions that they are pursuing. The entities are shown in JSON format in section ❸ of Figure 1.
2. The application makes changes to the persisted entities. Simulating new application releases, we declare schema changes, such as adding, removing, and renaming attributes in legacy entities. The case for adding an attribute *SCORE* with an initial value of 50 is shown in section ❶. Our schema evolution language [3] also allows copying and moving attributes between entities.
3. Upon a schema change, *Datalution* carries out eager and lazy data migration side by side, displaying the data store contents in both scenarios:
   With *eager* migration, schema changes are carried out immediately (c.f. ❹). This requires reading and writing all legacy entities, which may incur high costs with cloud providers charging by a pay-as-you-go scheme.
   With *lazy* migration, schema changes are carried out when the application requests an entity. *Datalution* ensures that pending changes are only applied to the subset of legacy entities actually affected. In general, eager evaluation tends to cause more writes (c.f. ❺).
   In the tradition of NoSQL data stores BigTable and HBase, all entities are timestamped and versioned, rather than modified in place.
4. We can make the internal Datalog rules visible (c.f. ❷).
5. By loading entities into the application, we can convince ourselves that the loaded entity is returned in the same version, regardless of eager or lazy migration.

3. OUTLOOK

Our next step is to implement the *Datalution* engine on top of an industrial-strength NoSQL data store. The tool is currently implemented as a main memory-based application. A key challenge is to account for the distributed architecture of these systems, avoiding to introduce an architectural bottleneck. Further, it is crucial that lazy data migration is carried out correctly in multi-user environments. Since NoSQL data stores commonly provide few transactional guarantees, this requires careful engineering.

A further point for future work is to integrate the *Datalution* approach with a cost model to help choose between eager and lazy data migration.

4. REFERENCES

[1] Curino, C., Moon, H. J., Tanca, L., and Zaniolo, C. Schema Evolution in Wikipedia - Toward a Web Information System Benchmark. In *ICEIS'08* (2008).

[2] Saur, K., Dumitras, T., and Hicks, M. W. Evolving NoSQL Databases Without Downtime. *CoRR abs/1506.08800* (2015). Technical report.

[3] Scherzinger, S., Störl, U., and Klettke, M. Managing Schema Evolution in NoSQL Data Stores. In *Proceedings DBPL'13* (2013).

[4] Scherzinger, S., Störl, U., and Klettke, M. A Datalog-based Protocol for Lazy Data Migration in Agile NoSQL Application Development. In *Proceedings DBPL'15* (2015).

[5] Wiech, K., Sombach, S., and Scherzinger, S. A Datalog-based Tool for Schema Evolution in NoSQL Databases. Poster presentation at XP'16, 2016.