Curating Variational Data in Application Development

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

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

Abstract—Building applications for processing data lakes is a software engineering challenge. We present Darwin, a middleware for applications that operate on variational data. This concerns data with heterogeneous structure, usually stored within a schema-flexible NoSQL database. Darwin assists application developers in essential data and schema curation tasks: Upon request, Darwin extracts a schema description, discovers the history of schema versions, and proposes mappings between these versions. Users of Darwin may interactively choose which mappings are most realistic. Darwin is further capable of rewriting queries at runtime, to ensure that queries comply with legacy data. Alternatively, Darwin can migrate legacy data to reduce the structural heterogeneity. Using Darwin, developers may thus evolve their data in sync with their code. In our hands-on demo, we curate synthetic as well as real-life datasets.

Index Terms—NoSQL databases; schema evolution; schema management; data migration; query rewriting; variational data

I. INTRODUCTION

With data accumulating in data lakes, there is a rejuvenated interest in data governance for semi-structured and heterogeneous data. Recent research efforts on so-called variational data [1] include the extraction of a schema description [2], [3] or integrity constraints [4], or even handling multiple sets of schemas when several applications require access [5].

In this paper, we focus on a specific scenario within this context that agile developers are facing in their daily work, namely an application evolving along several versions. This brings about evolutionary changes in the structure of persisted data, often as simple as added fields, but also involving more complex changes.

For applications that need to be available 24/7, NoSQL databases can be an appealing architectural choice: The new version of the application can be deployed against the same database that has also been serving the predecessor versions, without application downtime due to database migration.

While the database now stores records that adhere to the structure expected by the latest version of the application, it also stores legacy records, created by some earlier version of the application. Even though the database itself may not manage any schema (as the schema is implicit in the application code), in its essence, this can be considered an instance of the problem of schema evolution.

II. DARWIN SYSTEM OVERVIEW

Figure 1 shows the system architecture of Darwin. Darwin is a middleware for Java applications and a database storing variational data:

- To the top of the application stack is the Java application. It stores its data in a NoSQL database, interacting with the system-independent Darwin Persistence API.
- Via the Darwin WebApp, application developers may trigger data curation tasks directly.
- The Darwin Core REST API interfaces with the modules for schema extraction, handling schema evolution, data migration, and query rewriting.

We next describe the interplay of the components of Darwin along the workflow sketched in Figure 2.

1While we have demoed an earlier version of Darwin [6], features (2) through (4) above are new material. In particular, features (2) and (3) are now partly automated, as opposed to being manually specified by the users.
Mission. Among the nodes of a property hierarchy information, as well as the timestamps of all entities of this entity type. The graph further contains a node for each property, recording data type and hierarchy information, as well as the timestamps of all entities carrying this property. For instance, the node for Player pid has the timestamp list [10, 14], whereas the node for score has the timestamp list [14].

Step 2: The Schema Evolution Manager analyzes the timeline inherent in the schema version graphs. It recognizes schema versions and proposes schema evolution operations as mappings between succeeding schema versions. We refer to our companion paper [7] for details.

Darwin is able to derive single-type operations (such as add, remove, and rename) affecting the properties of entities of the same entity type. Comparing entities of different entity types, Darwin can detect multi-type operations for copying and moving properties between entity types. We have introduced these operations in earlier work [8]. Darwin pre-formulates a join condition for copy or move operations, and leaves it to the user to specify the join predicate, as seen in Figure 5.

There may be alternative schema evolution operations mapping between schema versions. In our example, we detect a new property score in entity type Mission at timestamp 14. This change may be described by an add operation, or a copy operation from entity type Player, joining on the player id. Darwin therefore compiles a decision table, as shown in Figure 5, listing alternatives for the software developers to choose from, as discussed next. After all, they have the necessary domain knowledge to resolve ambiguities, e.g., remembering how the software has evolved in the past.

Step 3: The software developers now resolve ambiguities in the decision table. This yields a sequence of schema evolution operations, describing the schema evolution history and thus the lineage of the variational data.

Darwin visualizes this history in Figure 6: The JSON schemas for Mission are shown in their two latest versions. The changes w.r.t. the previous schema version are marked up by a shaded background, e.g., Mission has a newly added property score in the version shown to the right.

Taming variational data in a data lake is a big challenge with dynamic requirements and heterogeneous applications. The comprehensive view on the schema evolution history may gain valuables insights for the developers already at this stage.

Notes on Performance and Scalability: Extracting and analyzing the entire data instance is a one-time effort. After the initial schema extraction, newly added entities can be analyzed incrementally and on-the-fly.

In extracting schema version graphs and deriving candidate mappings between schema versions, Darwin also proceeds incrementally. Since Darwin does not load the entire data instance into main memory, but only incremental batches [7], Darwin may safely handle large volumes of data. Our production installation of Darwin is running on the Big Data Cluster at Darmstadt University of Applied Sciences, comprising 40 physical nodes with installations of the NoSQL databases Cassandra, Couchbase, and MongoDB.

III. LEVERAGING THE SCHEMA EVOLUTION HISTORY

Having restored the schema evolution history, the software developers may pursue two different strategies in curating variational data with Darwin:

1) Leave the variational data as it is (e.g., due to compliance reasons), and have Darwin rewrite queries at runtime to account for the structural heterogeneity of the data.

2) Have Darwin migrate the variational data to a single, homogeneous structure.

Both strategies allow the application developers to code against the latest schema, without having to worry about variational...
data. We briefly sketch these approaches.

**Query Rewriting:** With the first approach, the database stores variational data, yet the application developers write code as if all records adhere to a global (virtual) schema. Provided that queries are simple enough, queries can be automatically rewritten by Darwin such that they account for the structural variations in legacy data. To this end, we have adapted the approach from [9], based on the Chase algorithm, for selection and projection queries.

Let us illustrate this with a simple example. The application issues a query that assumes the latest schema, where all Mission entities carry a score property:

```
select title from Mission
where score > 10
```

Darwin has detected that starting with timestamp 14, score properties are copies from the Player owning the mission. The query is now automatically rewritten:

```
(select title from Mission
where score > 10 and ts >= 14)
union
(select title from Mission M, Player P
where P.score > 10 and P.id = M.pid and M.ts < 14)
```

**Data Migration:** Alternatively, developers may migrate the data to adhere to a single schema: Since Darwin is aware of the historical schema versions, as well as the sequence of mappings, Darwin can reliably carry out the required steps.

IV. DEMONSTRATION SCENARIO

Our demo follows the workflow outlined in Figure 2 and walks through the steps that software developers may take from there, as sketched in Section III. We have prepared several datasets containing versioned data to illustrate different challenges. Besides the synthetic gaming data used in this paper, we also present biological data with observations of species in the Baltic sea [7], and real world configuration data from the Wendelstein 7-X plasma experiments [2], [10].

**Restoring the Schema Evolution History:** First, we select and register the data source in the Darwin WebApp. We let Darwin extract the schema versions and propose candidate evolution operations. The developers then interactively resolve any ambiguities. Figure 5 shows a screenshot of Darwin with the alternative schema evolution operations add and copy from our running example. The user of the Darwin WebApp is editing the join condition for the copy operation.

**Visualizing the Schema Evolution History:** Now we can browse the schema evolution history, as seen in Figure 6.

**Data Inspection:** We may also sample data from the database, to examine its structural heterogeneity. Figure 7 shows Mission entities in versions 5 and 11.

**Query Rewriting:** In the Darwin WebApp, we issue ad-hoc queries that assume that the entire data instance adheres to the latest schema. For instance, we assume that Missions carry a property score. We inspect the rewritten queries as well as the results of evaluating them on the variational data.

**Data Migration:** Darwin can carry out data migration to reconcile all entities with the schema expected by the latest
One of the prominent systems tackling schema evolution in relational databases, and extends them to handling hierarchical data. We plan to publish the details in a future paper.

While we find several connections to related work (especially given the long-standing history of schema evolution research [14]), the workflow presented in this demo is novel: Darwin is the first system to implement this end-to-end support for curating variational data in agile application development.

VI. CONCLUSION

We presented Darwin, a middleware that assists application developers in curating variational data. Using Darwin, developers can not only extract a schema, but also reconstruct a plausible schema evolution history. This already can contribute new insights. The developers may further choose to have Darwin rewrite queries at run-time, to account for variational data, or they may homogenize their data instance.

This kind of tool support frees up time, so that the developers may focus on implementing new application features, rather than writing low-level data curation scripts. Tools like Darwin can thus substantially simplify the long-term maintenance of variational data gathering in data lakes.

ACKNOWLEDGMENTS

This project was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant #385808805.

We thank O. Häller, T. Landmann, T. Lehwald, K. Möchel, H. Nkwincu, M. Richter, and M. Shenavai from Darmstadt University of Applied Sciences for contributing to the Darwin code base.

REFERENCES

[5] B. Chandramoniu, J. Gehrke, J. Goldstein, D. Kossmann, J. J. Levan-
zation of RDF graphs,” PVLDB, vol. 8, no. 12, 2015.