B. Mitschang et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2017),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 619

InVerDa – The Liquid Database[1]

Kai Herrmann[2], Hannes Voigt[1], Thorsten Seyschab[1], Wolfgang Lehner[1]

Abstract: Multiple applications, which share one common database, will evolve over time by their very nature. Often, former versions need to stay available, so database developers find themselves maintaining co-existing schema versions of multiple applications in multiple versions—usually with handwritten delta code—which is highly error-prone and explains significant costs in software projects. We showcase InVerDa, a tool using the richer semantics of a bidirectional database evolution language to generate all the delta code automatically, easily providing co-existing schema versions within one database. InVerDa automatically decides on an optimized physical database schema serving all schema versions to transparently optimize the performance for the current workload.

Keywords: Database evolution, Database versioning, Co-existing schema versions.

1 Introduction

Current relational database management systems (DBMS) do not support co-existing schema versions within the same database properly. However, this is a very common requirement in today's information systems. While software developers use agile techniques to quickly evolve running applications and frequently publish improved versions, the database is the millstone around the neck. Current DBMSes force developers to migrate a database completely in one haul to a new schema version. Keeping other schema versions alive before and after such a migration typically requires manually written and maintained delta code either in the database (views and triggers) or in the application. Further, finding an optimal physical database schema that serves all co-existing schema versions is notoriously hard, since the mix of mainly accessed schema versions changes over time. With all the described challenges, handling co-existing schema versions within one database is very costly, error-prone, and significantly slows down agile developers of information systems.

This demo[3] showcases InVerDa (Integrated Versioning of Database schemas), which provides a solution to this dilemma. InVerDa uses a declarative Database Evolution Language (DEL) called BiDEL. With BiDEL developers can evolve an existing schema to add a new schema version to a database. InVerDa makes the database instantly available through all **co-existing schema versions** within the DBMS. Data can be read and written through all schema versions; writes in one version are reflected in all other versions. To account for changing workload mixes, InVerDa **transparently optimizes the physical database schema** to continuously ensure a high performance without any further interaction of the developer. Hence, InVerDa greatly simplifies handling co-existing schema versions.

[1] This work is partly funded by the German Research Foundation (DFG) within the RoSI RTG (1907).

[2] Technische Universität Dresden, Database Systems Group, Nöthnitzer Str. 46, 01187 Dresden, Germany
<firstname>.<lastname>@tu-dresden.de

[3] Sneak preview and demonstrator available at `https://wwwdb.inf.tu-dresden.de/research-projects/projects/inverda`

BiDEL—InVerDa's evolution language—provides Schema Modification Operators (SMOs) to create, drop, and rename both tables and columns as well as splitting and merging tables both vertically and horizontally. BiDEL is similar to established DELs [Cu13, He15], with the distinction that its SMOs are specifically designed to be bidirectional. Based on a BiDEL-specified evolution, InVerDa is able to generate all required delta code to make the database instantly available through the new schema version and to propagate data both forward and backward between all schema versions. BiDEL combines standard SMOs for forward evolution with strategies to fill missing information and resolve ambiguity occurring in backward evolution, essentially by adding to each SMO the arguments of its inverse SMO. Since many SMOs are not information-preserving, InVerDa manages additional auxiliary tables to keep the otherwise lost information. We have formally validated the bidirectionality of BiDEL's SMOs by showing that payload data from any schema version N survives a round trip to version $N + 1$ and back to N unchanged and that the same holds vice versa. Hence, each schema version appears like a full-fledged single-schema database.

As applications evolve over time, the user's behavior—and hence the actual workload mix—changes as well. InVerDa continuously monitors the workload: after significant changes in the workload, InVerDa heuristically determines an optimized physical database schema, migrates the data accordingly, and adapts all involved delta code to keep all schema versions accessible. This automatic migration happens transparently to the users and developers without any required interaction. The physical database schema materializes a subset of all tables in all versions. InVerDa ensures that a specific table can be either accessed locally (iff the table version is physically stored) or by propagating the data access through SMOs to other physically stored table versions. Due to the guaranteed bidirectionality of BiDEL's SMOs, we can be sure that no data will be lost, regardless of the physical database schema.

The contributions of InVerDa are the (1) bidirectional DEL BiDEL, (2) automatic generation of co-existing schema versions, and (3) transparent migration to an optimized physical database schema—all on display in the demo. In the remainder, we introduce InVerDa using a comprehensible example (Section 2) and outline demo details (Section 3).

2 InVerDa – User Perspective

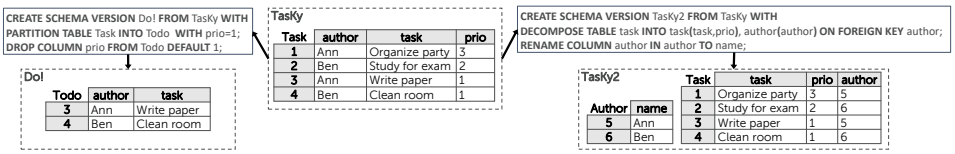

Fig. 1: Example.

We illustrate InVerDa's co-existing schema version support, using the example of a simple task management system called TasKy (cf. Figure 1). TasKy runs as a desktop application backed by a central database. Users can create new tasks and list, update, or delete them. Each task has an author and a priority with 1 being the most urgent priority. In the first release, TasKy stores all its data in a single table `Task(author,task,prio)`.

Development Time As TasKy gets widely accepted, we extend it by a third party phone app called Do! to list the most urgent tasks. Do! uses a different database schema than TasKy. The Do! schema consists of a table `Todo(author,task)` that contains merely tasks of priority 1. Obviously, the initial schema version needs to stay alive for the broadly installed TasKy. InVerDa greatly simplifies this job as it generates all the necessary delta code automatically. The developer simply executes the BiDEL script for Do! (cf. Figure 1), which instructs InVerDa to derive schema Do! from schema TasKy by creating a horizontal partition of `Task` with `prio=1` and dropping the priority column. Executing the script creates a new schema including the view `Todo` with delta code for propagating data changes. When a user adds a new entry into `Todo`, this will automatically insert a corresponding task with priority 1 to `Task` in TasKy. Equally, updates and deletes are propagated to the TasKy schema as well. In sum, the TasKy data is immediately available to be read and written through the newly incorporated Do! app by simply executing the three lines of BiDEL script.

Over time, the TasKy application is further refined and improved. For the next release, called TasKy2, it is decided to normalize the table `Task` into `Task` and `Author`. For an incremental roll-out of TasKy2, the old version TasKy has to remain functional until all clients are updated. Again, InVerDa does the job. When executing the BiDEL script as shown in Figure 1, InVerDa creates the schema version TasKy2 and decomposes the table version `Task` to separate the tasks from their authors while creating a foreign key to maintain the dependency. Additionally, the column `author` is renamed to `name`. InVerDa generates delta code to make the TasKy2 schema immediately available. Write operations to any of the three schema versions are now propagated to all other schema versions.

Operating Time The physical tables initially used for storing the data are the unevolved table versions. All other table versions are implemented with the help of delta code. The delta code introduces an overhead on read and write accesses to new schema versions. The more SMOs are between schema versions, the more delta code is involved and the higher is the overhead. In the case of the task management system, the schema versions TasKy2 and Do! require delta code to propagate data accesses to the physical table `Task`. Assume, some weeks after releasing TasKy2 the majority of the users has upgraded to the new version and heavily uses the mobile phone app, so that TasKy is still accessed but merely by a minority of users. Hence, it is appropriate to migrate data physically e.g. to the table versions of the TasKy2 schema and potentially replicating the data also for Do!'s `Todo`.

Traditionally, the database administrator decides on such a new physical database schema and developers would have to write a migration script, which moves the data and implements new delta code. All that can accumulate to some hundred lines of code, which need to be tested intensively in order to prevent them from messing up the data. InVerDa automatically optimizes the physical database schema and transparently runs the data migration, maintaining transaction guarantees, and updates the involved delta code of all schema versions. No developers need to be involved. All schema versions stay available; read and write operations are merely propagated to different physical tables. In sum, InVerDa allows users to continuously use all schema versions and developers to continuously develop the applications without caring about the physical database schema for a single second.

3 InVerDa – Demonstrator

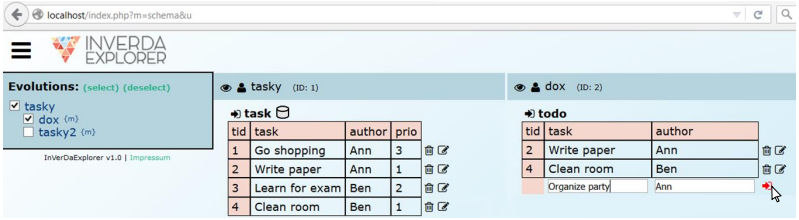

Fig. 2: InVerDa Explorer with TasKy and Do!.

For the sake of demonstration, InVerDa consist of three parts: (1) the InVerDa Console, an Eclipse Plugin to write and execute BiDEL evolution scripts, (2) the InVerDa Explorer, an application for conveniently browsing all **co-existing schema versions**, and (3) the InVerDa Optimizer, a tool running in the background to continuously monitor the workload and **automatically optimize the physical database schema** when necessary. These components operate on top of a PostgreSQL database with common SQL statements. Figure 2 shows the InVerDa Explorer with the versions TasKy and Do!; Task from TasKy is physically stored as indicated by the small cylinder. The InVerDa Explorer allows to manipulate data in any schema version and observe the effects in the other versions. Inserting a task in Do! as shown, also adds the task in TasKy.

During the demo, we will present the prepared TasKy example. However, InVerDa is a working system and we encourage participants to test it interactively with own scenarios. We will execute BiDEL evolution scripts with the InVerDa Console and demonstrate the genuinely co-existing schema versions by manipulating data with the InVerDa Explorer. Participants can experience both the developer perspective by writing own BiDEL scripts, and the user perspective by probing the generated delta code. They can also have a look behind the scenes at the generated delta code. Further, we have prepared dynamic workloads to show how InVerDa automatically adapts the physical database schema. We want to stimulate discussions with practitioners about e.g. use cases and missing aspects as well as with researchers about e.g. technical details and future research questions.

In sum, InVerDa allows multiple applications in multiple versions to share one common database, each having an individual view on the data, while the physical database schema is transparently adapted to a changing workload in the background. Using the richer semantics of a DEL, InVerDa unburdens database administrators and developers from the costly and error-prone tasks of managing co-existing schema versions manually.

References

[Cu13] Curino, Carlo; Moon, Hyun Jin; Deutsch, Alin; Zaniolo, Carlo: Automating the database schema evolution process. VLDB Journal, 22(1):73–98, 2013.

[He15] Herrmann, Kai; Voigt, Hannes; Behrend, Andreas; Lehner, Wolfgang: CoDEL - A Relationally Complete Language for Database Evolution. In: ADBIS 2015, Poitiers, France. volume 9282 of LNCS. Springer, pp. 63–76, 2015.