PROV-IDEA: Supporting Interoperable Schema and Data Provenance within Database Evolution

BEATRIZ PÉREZ and ÁNGEL LUIS RUBIO GARCIA, Department of Mathematics and Computer Science, University of La Rioja, Logroño, Spain
MARÍA A. ZAPATA, Department of Computer and Systems Engineering, University of Zaragoza, Zaragoza, Spain

Database evolution and data provenance are two closely related research fields. On the one hand, the registry (via provenance) of the schema evolution allows the maintenance of its version record. On the other hand, the origin of the data (i.e., its provenance) will always be affected by modifications (i.e., the evolution) in the schema on which they are based. Despite these interrelationships, there are few works in the literature that have proposed advances in that direction. In particular, to the best of our knowledge, there is no research that has resulted in a general and interoperable solution to the problem of managing database evolution using provenance. In this article, we present PROV-IDEA: a PROV-Interoperable Database Evolution Approach. This is a proposal that allows the simultaneous management of the provenance of schemas (of relational databases) and data, using the PROV standard as a way to guarantee interoperability. Furthermore, it is an adaptable and expandable approach (by using PROV templates), which allows a non-intrusive and seamless integration with existing applications, as well as different aspects of provenance information generation. These properties are demonstrated in the article by presenting a proof of concept built on top of a third-party relational database evolution tool.

CCS Concepts: • **Information systems** → **Data provenance**; • **Software and its engineering** → **Software configuration management and version control systems**; **Interoperability**;

Additional Key Words and Phrases: database schema and data evolution, schema modification operators, data modification operators, data provenance

ACM Reference format:
Beatriz Pérez, Ángel Luis Rubio Garcia, and María A. Zapata. 2025. PROV-IDEA: Supporting Interoperable Schema and Data Provenance within Database Evolution. *ACM Trans. Softw. Eng. Methodol.* 34, 3, Article 59 (February 2025), 27 pages.
https://doi.org/10.1145/3697008

All authors contributed equally to this research.
This work was supported in part by the Ministerio de Ciencia e Innovación/Agencia Estatal de Investigación, MCIN/AEI/10.13039/501100011033, and by the "NextGenerationEU"/Plan de Recuperación, Transformación y Resiliencia de España (PRTR), European Union, under Grant PDC2021-121128-I00 (ReCREA), and by project AFIANZA 2022/05 granted by the Autonomous Community of La Rioja.
Authors' Contact Information: Beatriz Pérez, Department of Mathematics and Computer Science, University of La Rioja, Logroño, Spain; e-mail: beatriz.perez@unirioja.es; Ángel Luis Rubio Garcia (corresponding author), Department of Mathematics and Computer Science, University of La Rioja, Logroño, Spain; e-mail: arubio@unirioja.es; María A. Zapata, Department of Computer and Systems Engineering, University of Zaragoza, Zaragoza, Spain; e-mail: mazapata@unizar.es.

ACM 1557-7392/2025/2-ART59
https://doi.org/10.1145/3697008

1 Introduction

Schema evolution has been on the research agenda of the database community for more than three decades. Nowadays, its study is still of interest, as both recent academic publications address it from different viewpoints (such as NoSQL databases [20], multi-model databases [22], or automation [52]), and modern software platforms (such as Flyway [16], Liquibase [28], or QuantumDB [15, 41]) aim to provide practical solutions to the problem. On the other hand, the study of provenance, and in particular the study of data provenance, is a more relatively recent area of research in which there have been many advances (especially around the establishment of PROV as a **World Wide Web Consortium (W3C)** standard [18, 31]).

It is undeniable that the two areas, schema evolution and data provenance, have important points in common. On the one hand, the provenance of the data forming a schema (i.e., the metadata) must be dealt with when the schema undergoes an evolution. On the other hand, data stored under an evolving schema will also, in the most general case, be subject to changes and transformations whose provenance will be relevant to deal with. This implies that schema evolution requires the simultaneous and coordinated management of (at least) two levels of information (metadata level and data level). Recording the provenance of all information related to schema evolution is therefore an interesting and challenging issue to investigate.

However, there are hardly any studies that have considered the relationships between schema evolution and provenance [1, 2, 17]. Furthermore, to our knowledge, no research work has explored the use of the W3C PROV standard as a means of enriching schema evolution approaches with provenance information, nor have solutions been proposed that would allow seamless and interoperable integration with existing database systems.

In this article, we advance in that direction, by presenting **PROV-Interoperable Database Evolution Approach (PROV-IDEA)**, an approach to integrate provenance information into relational database schema evolution, using PROV as provenance framework. Our proposal has several outstanding features, which, taken together, beggars the belief that there is no similar approach in the literature. PROV-IDEA takes into account the need to simultaneously manage two levels of provenance information (data and metadata) and guarantees its interoperability by making use of the PROV standard. Moreover, it is an adaptable and expandable proposal, thanks to its use of the *PROV-Template* approach [30], a technique that has proven useful in other contexts such as the *UML2PROV* framework [48, 49]. Our approach recognizes the relevance of providing a solution that allows for non-intrusive and as seamless as possible integration with other systems, yet is flexible enough to deliver different ways of obtaining provenance information. All these properties, as well as the feasibility of our proposal as a whole, are demonstrated in this article by presenting a proof of concept built on top of the QuantumDB [41] evolution tool. Therefore, the main contributions of this work are threefold:

(1) The central idea that information regarding the evolution of a database (both schema and data) can be managed using provenance techniques, in particular through the PROV standard and the mechanisms provided by PROV templates.
(2) The PROV-IDEA framework, which determines the complete steps (design, development, and consumption) that can be carried out to develop a provenance solution on database evolution.
(3) The verification of the feasibility of the PROV-IDEA framework, through the presentation of a fully implemented proof of concept on top of a third-party tool (QuantumDB).

The rest of the article is structured as follows: the background necessary for the understanding of the proposal (**Schema Modification Operators (SMOs)**, PROV, and the PROV-Template approach)

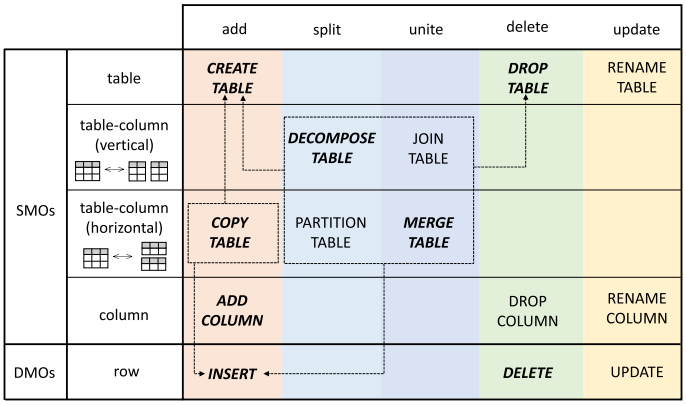


Fig. 1. SMOs and DMOs.

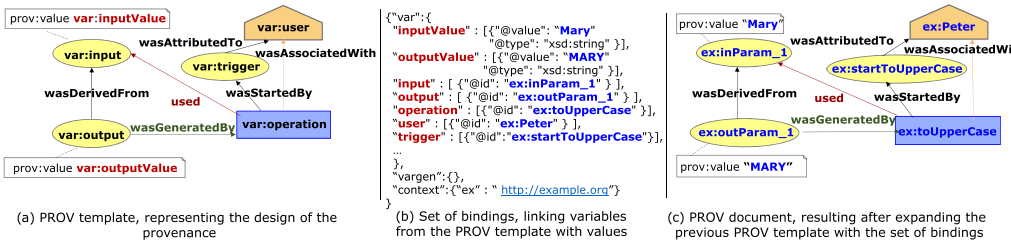
is presented in the next section. Section 3 presents the PROV-IDEA framework, detailing the design and capture of evolution provenance processes. The proof of concept implementation of the framework and its evaluation appear in Sections 4 and 5, respectively. Section 6 presents the discussion of our proposal in comparison with other related approaches. The article ends with conclusions and future work.

2 Background

In this section, we provide some background information about three aspects that deserve to be explained due to their relevance to our proposal: SMOs, the **PROV Data Model (PROV-DM)**, and the PROV-Template approach.

SMOs [13] have been proposed as standard functions for relational schema evolution. Each SMO performs a modification on some tables of a schema giving rise to a new schema version. The modifications performed by some SMOs update the data accordingly, that is, some SMOs implicitly perform appropriate data evolutions in the underlying database. These implicit data modifications (insert, delete, or update operations) can be expressed by means of **Data Modification Operators (DMOs)** [51]. In Figure 1 we have based on [19] to presente the SMOs and DMOs considered in the literature [13, 17, 19, 51]. More specifically, we have organized these operators according to both the type of operation (add, split,...) and the modified elements (table, vertical modification of tables and columns,...). The schema modification produced by some operators is straightforward. For example, `CREATE TABLE` or `DROP TABLE`, which, respectively, creates an empty table or deletes a table with its data. Other operators involve more complex schema modifications, which can be defined as a composition of simpler ones. For example, `COPY TABLE` causes the creation of a table (`CREATE TABLE`) whose values are taken from the table being copied (`INSERT`). Precisely, these composition relations between operations are represented in Figure 1 by arrows.

PROV [18] is a W3C standard that aims to enable the interoperable interchange of provenance information in heterogeneous environments. PROV defines a core data model (PROV-DM) for building representations of the entities, people, and processes involved in producing a piece of data or thing in the world. Figure 2(c) depicts a graphical example of a PROV document representing that `Peter` (agent) has executed the `toUpperCase` operation (activity), applied to the "`Mary`" input parameter value and giving rise to the "`MARY`" output parameter value (entities). This figure includes the three main concepts of PROV: (1) *Entities*, defined as physical, digital, conceptual, or other kinds of things, that can be described by means of different attributes (yellow oval); (2) *Activities*,



(a) PROV template, representing the design of the provenance

(b) Set of bindings, linking variables from the PROV template with values

(c) PROV document, resulting after expanding the previous PROV template with the set of bindings

Fig. 2. PROV-Template approach.

which are how entities come into existence and how their attributes change to become new entities (blue rectangle); and (3) *Agents*, which bear some form of responsibility for the activity taking place (orange pentagon). As can be seen in Figure 2(c), these concepts are related by relationships representing, for example, a new entity produced by an activity (*wasGeneratedBy*), a transformation of an entity into another (*wasDerivedFrom*), or an agent with some responsibility for an activity (*wasAssociatedWith*).

In order to facilitate the generation of provenance information, the *PROV-Template* approach [30] proposes a templating system for generating PROV-compliant provenance. On the one hand, designers specify the provenance to be generated by composing *templates*, expressed in PROV, containing variables that act as placeholders for values. For example, Figure 2(a) depicts a template representing the information that will be registered each time a user triggers an operation with input values, giving rise to output values. The name of the elements has the prefix `var` indicating they are variables that will be substituted by values. On the other hand, programmers write programs that log values at runtime and package them up in *sets of bindings* associating variables of the templates and values. For example, Figure 2(b) shows a set of bindings collected during the application execution, just when `Peter` has triggered the `toUpperCase` operation. Finally, the *expansion algorithm* [30] generates *PROV documents* from templates, by replacing all variables by values found in a set of bindings. More specifically, starting from the template of Figure 2(a) and the values associated to its variables given by the bindings of Figure 2(b), the expansion algorithm generates the expanded PROV document of Figure 2(c).

Templating technology allows provenance management at different levels of granularity. Thus, as stated in [30], the fine-grained provenance level is the one required to manage database provenance. This is because in databases it is necessary to have information at the attribute (value) level to manage the provenance of query results, in order to address questions related to explainability (why, how, and where provenance [1, 17]) or reproducibility [29]. In addition, PROV-Template offers great benefits concerning previous proposals in terms of development and maintenance effort, thanks to the separation of concerns between the design of the provenance to be generated (represented by PROV templates) and the collected provenance data (expressed by bindings).

3 PROV-IDEA

PROV-IDEA has as its main goal the management of data and schema provenance of relational databases evolution in a uniform way, giving rise to interoperable provenance information. Our approach considers that schema evolution is performed by means of SMOs, which produce a new version of the input schema and a migrated version of the database updated accordingly by making the necessary data modifications. As we will see later, aimed at providing a more adaptable approach, we advocate making explicit the DMOs triggered by the SMOs. From there, PROV-IDEA proposes the definition of PROV templates for SMOs and DMOs to structure the provenance to be generated,

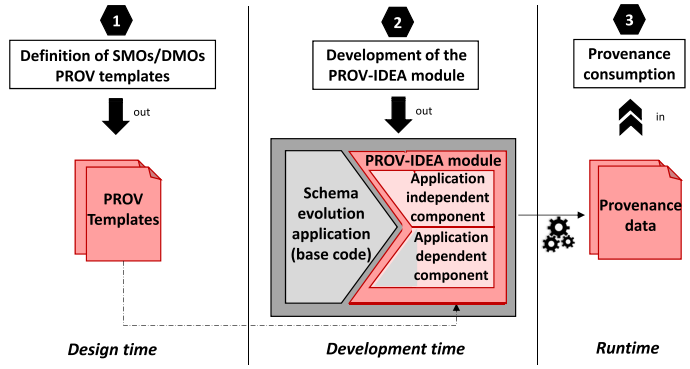


Fig. 3. The PROV-IDEA approach.

together with a tool devoted to capturing schema and data provenance during the execution of a schema evolution application (aiming at making such applications provenance-aware).

Figure 3 depicts the key processes and artifacts of PROV-IDEA, distinguishing the phase in which they are involved: design time, development time, or runtime. At design time, after analyzing the SMOs/DMOs' definitions, PROV-IDEA proposes the creation of PROV templates for these SMOs and DMOs. Once the PROV templates are designed, the following proposed step is the development of a PROV-IDEA module in charge of capturing schema and data provenance during the execution of a schema evolution application. This module will be integrated non-intrusively into the application, making it provenance-aware. Finally, the provenance data will be generated at runtime including high-quality information. This information would be ready for provenance consumption so that provenance consumers could exploit it as desired.

The most outstanding properties of PROV-IDEA, both because they are an essential part of the proposal and because they make our approach different from other proposals, are the following:

— *(P1) Join Management of Data and Schema Provenance Information.* Schema evolution implies the corresponding migration of the underlying database. For this reason, evolution schema provenance requires the simultaneous and coordinated management of data and schema provenance information.
— *(P2) Interoperability by Means of Provenance Standards.* PROV-IDEA proposes to store the provenance information captured during schema evolution by means of the W3C PROV standard. This fact facilitates the interoperability of the resulting evolution provenance.
— *(P3) Adaptable and Expandable.* The SMOs and DMOs are defined as conceptual standard operations which could be implemented by schema evolution applications following different strategies. To this respect, our proposal allows PROV templates to be combined like pieces of a puzzle to fit different evolution implementation approaches. Furthermore, the set of provided PROV templates is conceived as an open collection that can be extended to cover other variants of evolution operations.
— *(P4) Non-Intrusive and Seamless Integration.* The schema evolution application is enriched with an additional module, located apart from the application's source code. In this way, this module is non-intrusively integrated into the application, without requiring any modification of the source code. Moreover, the proposal can be applied independently of the programming language used to develop the application, as well as of the **Data Base Management Systems (DBMSs)** whose Data Basesevolution must be managed.

Table 1. SMO Sequence of the *Running Example*

1	**COPY TABLE** developer_sub_B **INTO** developer_sub_B_old
2	**MERGE TABLE** developer_sub_A, developer_sub_B **INTO** developer_sub_A_new
3	**DECOMPOSE TABLE** developer_sub_A_new **INTO** developer_sub_A (id, name, surname,...), contact (id, e-mail,...)

— *(P5) Flexible Provenance Computation.* Our proposal is agnostic in terms of *where* to compute the final provenance data (giving the possibility of storing provenance in different storage systems), *how* the provenance is serialized (considering a more or less verbose format), and of *when* to compute provenance (giving freedom to PROV-IDEA software developers to choose between following a *lazy*—computing provenance when it is required—or an *eager*—immediately—approach).

To illustrate our proposal in a practical way, throughout this article, we will use as *running example* the schema evolution shown in Table 1 applied to the database of a fictitious software company. Let us suppose that this company has two concrete subsidiaries (`Sub_A` and `Sub_B`) and, for both business and strategic reasons, one of the subsidiaries (`Sub_A`) absorbs the other (`Sub_B`). In this situation, the company decides to merge the developers of both subsidiaries (`developer_sub_A` and `developer_sub_B`) into a single table of developers (`developer_sub_A_new`), but it also wants to keep the developers table of the absorbed subsidiary (`developer_sub_B_old`) due to possible audit purposes. Considering that the semantics of the `MERGE TABLE` SMO involves deleting the source tables, first, table `developer_sub_B` is copied into table `developer_sub_B_old` (Table 1, line 1, `COPY TABLE` SMO). Once this operator has been performed, both source tables are merged into the new one `developer_sub_A_new` (Table 1, line 2, `MERGE TABLE` SMO). Finally, for optimal querying, the company decides to extract from the target table `developer_sub_A_new` the data referring to the developers' contact form, and performs a `DECOMPOSE TABLE` SMO (Table 1, line 3).

In the following subsections, we detail the characteristics of the PROV-IDEA approach, differentiating between provenance *analysis*, provenance *design*, and provenance *capture*.

3.1 Evolution Provenance Analysis

As stated in Section 1, and reflected in property (*P1*), a central idea of our approach is that all information related to the evolution of a database (both schema and data) can be managed by provenance techniques. This idea is shared by other authors who have addressed provenance in databases. These authors stress the importance of "preserving the state of the database" when changes ("derivations") are made [8] or that "all changes or all versions of the database must be fully documented so that scientific results remain verifiable" [50]. As we have pointed out, PROV-IDEA also has other specific properties (*P2–P5*) that differentiate it from other proposals, which we will detail in the following sections.

In addressing how we propose to fulfill property (*P1*), we must consider that the ultimate goal should be the consumption of provenance (Figure 3, Phase 3). In this sense, it is important to consider the *relevance* property, defined as "the degree to which provenance is relevant and useful to the consumer's needs" [10]. These needs will be determined by each scenario or use case in which it is relevant to consider database provenance. Examples of such scenarios are database audit and data recovery [17], explainability [1, 17], or reproducibility [29].

The templates defined in the first phase of the PROV-IDEA approach capture fine-grained provenance to record both schema and data elements (so that property (*P1*) is satisfied). This fact makes it feasible to answer provenance queries with different granularity, both at schema and instance levels. Focusing, for example, on a database audit scenario, to justify compliance with

company policies or legal regulations (related to data privacy or others), it may be necessary to track all information about the existence (or deletion) of data at a very fine level of granularity. In particular, it may be required to answer questions about why, how, where, and when [1, 11, 17] a given piece of data exists. The PROV templates we define allow on-demand generation of PROV documents that will, in turn, enable the following types of questions to be answered.

— *Why-Provenance.* For each element of the database (schema, table, column, value), the SMOs/DMOs operations that generated it, and the users who executed them, must be known. It can be distinguished between whether only the main SMOs executed by the user are known (*Why Shallow*), or if also the SMOs/DMOs into which they are decomposed are available (*Why Deep*). This information can contribute, among other things, to the explainability of the performed database evolution and to audit the responsibility of the users.
— *How-Provenance.* For each element of the database (schema, table, column, value), it must be known how the inputs used to produce it were manipulated, that is, the modifications carried out on the inputs to obtain an output element. The answer to this type of questions facilitates the reproducibility of the performed operations.
— *Where-Provenance.* It describes the schema each table, column, or value belongs to. This information can help to achieve schema and data auditing.
— *When-Provenance.* For each element of the database (schema, table, column, value), the creation, modification, or deletion time must be known. Registering the timestamp can facilitate data recovery, which is especially important when non-information preserving operations are performed.

As a tracing personal data case, given the surname of a developer stored in the final `developer_sub_A` table of the *running example*, the designed PROV templates must allow us to know the last SMO used to generate it and the user who executed this operation (Why), the modifications performed in the database elements to obtain it (How), the schema it belongs to (Where), and its creation time (When).

3.2 Evolution Provenance Design

Building on the analysis carried out, PROV-IDEA proposes to design templates to capture the provenance information coming from database evolution (see Figure 3, Phase 1). Specifically, by using PROV-Templates, PROV-compliant provenance is generated, which facilitates the *interoperability (P2)* of the resulting evolution information.

In order to explain our proposal of PROV templates for schema evolution, we begin detailing the strategy that can be followed for combining them in a modular way. Next, we go on to detail how they are specified by means of patterns.

3.2.1 PROV Templates Composition. The main idea behind the PROV-Template approach concerning an SMO/DMO is that it represents, by means of PROV elements, the artifacts involved in the transformation. In particular, each template includes the SMO/DMO taking place (PROV *activity*), the user who performs it (PROV *agent*) and the affected elements of the old schema, the new schema or their underlying data (PROV *entities*). These SMOs/DMOs PROV templates can be combined like pieces of a puzzle to adapt different evolution implementation approaches (Figure 4). For example, if we focus on the first SMO performed in the *running example*, that is, in the `COPY TABLE` operator, it may be implemented as the consecutive execution of the `CREATE TABLE` SMO, needed to create the new table, followed by the corresponding `INSERT` DMO, needed to populate the new table with the rows in the source table. This characteristic allows capturing not only the evolution of provenance data related to an SMO itself but also specific information

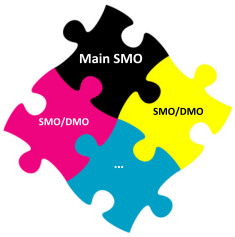


Fig. 4. *Adaptable and expandable* characteristic of the PROV-IDEA approach.

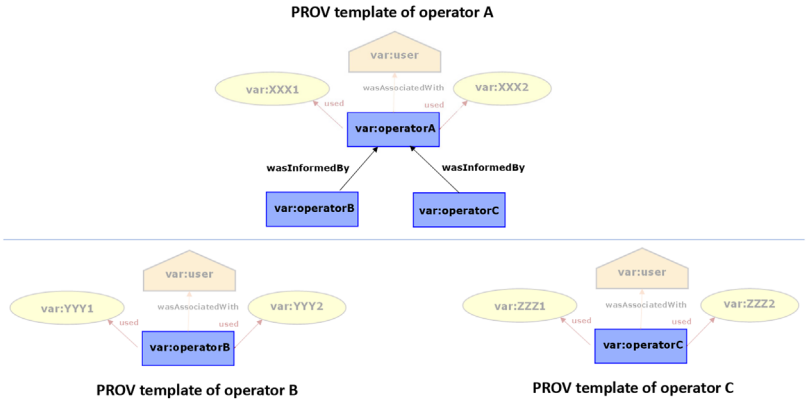


Fig. 5. *PROV templates' associations.*

regarding the different SMO/DMO it is made up of. According to this, PROV templates are defined as atomic elements that can be combined in order to represent more complex evolution operations. This fact, together with that the set of PROV templates is conceived as an open collection that can be extended to cover other variants of evolution operations, makes our proposal *adaptable and expandable (P3)*.

This puzzle-like strategy is given by using the *PROV relation* `wasInformedBy`, which allows us to link provenance data related to the associated operation executions. More specifically, let us assume that an operator `A` is implemented in a concrete evolution implementation approach as the consecutive execution of another two operators `B` and `C`. In this case, we say that each activity representing the execution of `B` and `C` was informed by (*wasInformedBy*) the activity representing the execution of `A` (see Figure 5). As a consequence, the PROV template representing the operator `A` will contain as many *PROV relations wasInformedBy* with the corresponding activity elements as necessary. When executing the operator `A` (and, as consequence, operators `B` and `C`) the corresponding bindings will be generated and, when expanded with the corresponding PROV templates (the concerned to operator `A`, with the *PROV relations wasInformedBy*, and the PROV templates of `B` and `C`), the resulting PROV documents will include the information of the chained operations.

3.2.2 Schema Evolution Patterns. With the goal of specifying the PROV templates in a systematic and rigorous way, we propose to describe them by means of a pattern-based structure. This structure consists of six blocks: *Identifier* assigning a unique acronym to each SMO/DMO; *Syntax*, that states the SQL-like syntax of the SMO/DMO pattern (in the case of SMOs, we have mainly based on the syntax given in [19]); *Semantics* expressing the effect of the application of the SMO/DMO in the

database, including a description of the key elements; *SMO/DMO's involved elements*, that contains the elements that model the key elements of the SMO/DMO semantics; *Mapping to PROV*, that corresponds to the PROV template proposed from the previous SMO/DMO's involved elements; and *Discussion*, that presents issues related to the representation of the SMOs/DMOs with PROV.

The proposed specification structure facilitates their understandability and maintenance, enabling other researchers to replicate the presented proposal. Concretely, we have specified six SMO patterns and two DMO patterns which correspond to those represented in bold-italic text in Figure 1, that is, the `CREATE TABLE`, `DROP TABLE`, `ADD COLUMN`, `COPY TABLE`, `MERGE TABLE` and `DECOMPOSE TABLE` SMOs, and the `INSERT` and `DELETE` DMOs. We note that our intention is not to define patterns for all the schema evolution operators, but focusing on those we have considered of more interest for provenance purposes and for evaluating the feasibility of the proposal. The complete specification of these eight database evolution patterns can be consulted in the Supplementary Material [36].

In addition, the PROV templates included in the *Mapping to PROV* section of the patterns are publicly available through the *ProvStore* utility on the *openprovenance.org* Web site (just searching for the keyword "PROV-IDEA") [39]. Each of these PROV templates includes the elements that both represent the schemas and data involved in the corresponding SMO/DMO (for which we particularly take into account their formal semantics) and allow questions to be answered during provenance consumption. Next, we explain the general process for defining the template of an SMO based on its formal definition. Let us note that, as some SMOs can be defined as a composition of simpler ones (see Figure 1), the templates for the simplest SMOs must be defined first so that these templates will be taken into account when defining the template for the most complex ones. Regarding the process for specifying a DMO template, it is similar and even simpler because it does not produce a schema change. The process for specifying a PROV template for an SMO has five steps, that we illustrate using as example the first SMO executed in the *running example*, concerning a `COPY TABLE` operator. In the Supplementary Material [36] we have included a more algorithmic-style description of this process, as well as a textual specification of the template generated for the `COPY TABLE` SMO.

Step 1. *Formal Semantics and, When Appropriate, Decomposition of the SMO.* In this preliminary step, before specifying the PROV template, the formal semantics of the SMO must be considered. Specifically, we have taken into account the proposal, based on the relational algebra, provided in [19]. For example, the formal definition of the `COPY TABLE T INTO R` operator is: $Add(R, \sigma_{true}(T))$. This semantics determines that a new table `R` must be created and populated by selecting all the values of the source table `T`.

Taking into account the formal definition, it is necessary to analyze whether it is an SMO that can be implemented as a composition of other simpler SMOs/DMOs. For example, from the definition of the `COPY TABLE` operator, it is clear that, as we have mentioned previously, this SMO can be implemented as the composition of the `CREATE TABLE` SMO and the `INSERT` DMO. In these cases, the PROV templates of the simplest SMOs/DMOs (that must have been previously defined) will be considered during the following steps.

Step 2. *Schema Change.* The PROV template of an SMO, regardless of the performed operation, always includes the activity `var:smo` which *was associated with* the agent `var:user`. Furthermore, the execution of an SMO (i.e., an instance of the SMO) produces a change in the database schema, so that a certain schema `S` goes from state S_i to state S_{i+1}. For this reason, every PROV template also includes a modified version of the schema (`var:targetSchema`) which *was derived from* the input schema (`var:sourceSchema`). The new schema *had as members* the same tables (`var:previousTable`) as the original schema. As a consequence,

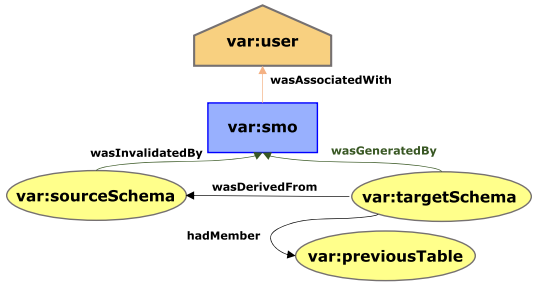


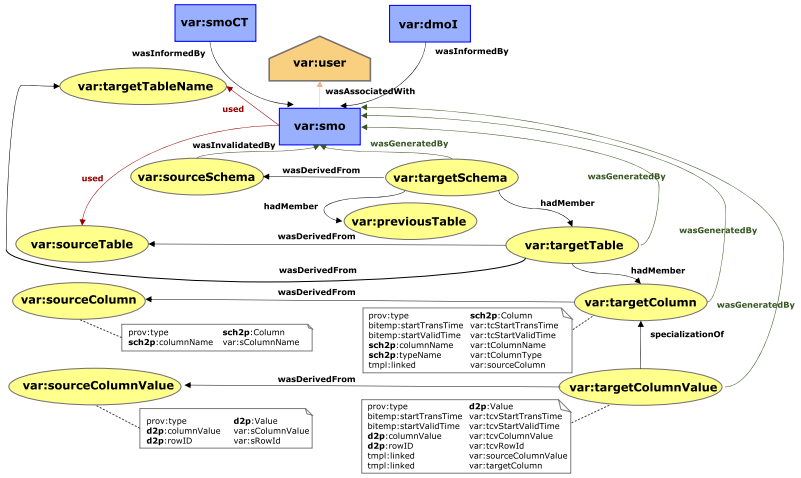
Fig. 6. Basic PROV template elements involved in a schema change.

every template contains the structure shown in Figure 6. For example, this structure appears in the PROV template of the `COPY TABLE` schema operation depicted in Figure 7(a).

By means of the elements contained in the structure shown in Figure 6, the new schema version, together with information of its tables, is registered, allowing *Where-provenance* questions to be answered during provenance consumption. In the same way, since the executed SMO and the user who executed it are recorded, *Why shallow-provenance* questions can also be processed.

Step 3. *Input and Output Parameters.* Every SMO has some input values that will be related in the template with the activity `var:smo` by means of *used* relations. Regarding the output values, they will be related in the template with the SMO activity by means of *wasGeneratedBy* relations. When the SMO can be decomposed into other SMOs/DMOS, the analysis of the input and output values is performed in conformance with the input and output values of these other SMOs/DMOs. In the `COPY TABLE` SMO (see Table 2 and Figure 7(a)), the input values are the name of the new table `R` (`var:targetTableName`) and the original table `T` (`var:sourceTable`). The input target table name and the input source table provide the input values necessary for the `CREATE TABLE` SMO and the `INSERT` DMO. On the other hand, the output value of the `COPY TABLE` operation is the *generated* table `R` (`var:targetTable`), which *had as members* columns (`var:targetColumn`) populated with values (`var:targetColumnValue`). These output values conform with the result of the `CREATE TABLE` and `INSERT` operations. The registration of the input and output parameters is necessary to be able to respond *How-provenance* questions.

Step 4. *Evolution Process.* Furthermore, the template includes the elements that represent the modifications performed in the original schema giving rise to the new schema. These new elements are derived from the formal definition of the SMO. If the semantics implies the execution of an *Add* table sentence, then the *created table* will be related in the template with the *new schema* (`var:targetSchema`) by means of a *hadMember* relation, and the *created elements* (table, columns, values) will be related with the *source elements* by means of *wasDerivedFrom* relations. In case the semantics involves a *Del* table sentence, then the template will include the corresponding *wasInvalidatedBy* relations. In the particular case of the `COPY TABLE` SMO (see Figure 7(a)), the formal semantics establishes that the new schema (`var:targetSchema`) *had as member* the new table `R` (`var:targetTable`) which *was derived from* the original table `T` (`var:sourceTable`). Besides, table `R` is populated by selecting all the values (`var:sourceColumnValue`) of the columns of `T` (`var:sourceColumn`). Let us note that table `T` plays two roles at the same time. On the one hand, it is the *source table* that will be used for deriving the *target table* `R` and, on the other hand, it is a *previous table* that will be a member of the *target schema*.

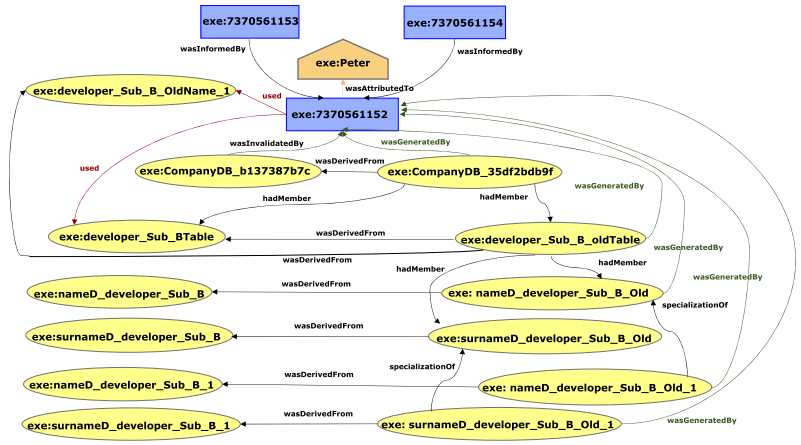


(a) *Copy table* PROV template, including properties only on the lower entities

```
document
  prefix tmpl <http://openprovenance.org/tmpl#>
  prefix var <http://openprovenance.org/var#>
  prefix exe <http://example.org/>

  entity(var:user, [tmpl:value_0='exe:Peter'])
  entity(var:smo, [tmpl:value_0='exe:7370561152'])
  entity(var:smoCT, [tmpl:value_0='exe:7370561153'])
  entity(var:dmoI, [tmpl:value_0='exe:7370561154'])
  entity(var:targetTableName, [tmpl:value_0='exe:developer_Sub_B_OldName_1'])
  entity(var:sourceTable, [tmpl:value_0='exe:developer_Sub_BTable'])
  entity(var:sourceSchema, [tmpl:value_0='exe:CompanyDB_b137387b7c'])
  entity(var:targetSchema, [tmpl:value_0='exe:CompanyDB_35df2bdb9f'])
  entity(var:previousTable, [tmpl:value_1='exe:developer_Sub_BTable'])
  entity(var:targetTable, [tmpl:value_0='exe:developer_Sub_B_OldTable'])
  entity(var:sourceColumn, [tmpl:value_1='exe:surnameD_developer_Sub_B', tmpl:value_2='exe:nameD_developer_Sub_B'])
  entity(var:sourceColumnValue, [tmpl:value_1='exe:nameD_developer_Sub_B_1',tmpl:value_0='exe:surnameD_developer_Sub_B_1'])
  ...
```

(b) Possible set of bindings obtained from the execution of the `COPY TABLE` operator



(c) PROV document (without properties), resulting after expanding the PROV template with the binding

Fig. 7. Process: (a) *Copy table* PROV template (with some properties); (b) set of bindings; (c) the expanded PROV document (without properties).

Table 2. Examples of the Main Elements Involved in the PROV Template Specification Process

Syntax	**COPY TABLE** T INTO R	**MERGE TABLE** S, T INTO R	**DECOMPOSE TABLE** R INTO S(g_i, ..., g_n)[,T(g_j, ..., g_m)]
Semantics	$Add(R, \sigma_{true}(T))$	$Add(R, \pi_{S.C}(S) \cup \pi_{T.C}(T))$; $Del(S)$; $Del(T)$	$Add(S, \pi_{g_i,...,g_n}(R))$; [$Add(T, \pi_{g_j,...,g_m}(R))$;]$Del(R)$
Related SMOs/DMOs	CREATE TABLE SMO INSERT DMO	CREATE TABLE SMO INSERT DMO DROP TABLE SMO	CREATE TABLE SMO INSERT DMO DROP TABLE SMO
Input parameters	Source table: T Target table name: R	Source tables: S and T Target table name: R	Source table: R Target table name[s]: S [and T] Target column names: g_i, ..., g_n[, g_j, ..., g_m]
Output parameters	Target table: R Target columns Target column values	Target table: R Target columns Target column values	Target table[s]: S [and T] Target columns Target column values

The modifications carried out on the inputs to obtain the output database elements are registered by means of the derivations, memberships, or invalidations performed by the SMO, making it feasible to answer *How-provenance* questions.

Step 5. *Related DMO/SMOs.* Finally, following our puzzle-like strategy, when the SMO is a composition of other SMOs/DMOs, the template includes activities representing those SMOs/DMOs involved in its implementation and which *were informed by* that SMO. For example, the template of the COPY TABLE SMO (see Figure 7(a)) includes two activities representing the CREATE TABLE SMO (var:smoCT) and INSERT DMO (var:dmoI) which *were informed by* the activity representing the COPY TABLE SMO (var:smo). As a consequence, provenance will capture SMOs/DMOs decomposition relations, allowing *Why deep-provenance* questions to be answered.

Each element of a PROV template can have properties associated with it, represented as a set of attribute-variable pairs capturing additional information about the element. For example, the properties of the entities presented in the lower part of Figure 7(a) are depicted (we have not included all the properties to make the figure readable). In particular, the properties registering the transactional and valid time (bitemp prefix) of the different database elements can be used to answer *When-provenance* questions during provenance consumption. Additionally, what is especially remarkable is the fact that we have defined two prefixes to differentiate between *schema level provenance* information (sch2p prefix) and *data level provenance* information (d2p prefix). For example, the var:sourceColumn entity contains the attribute sch2p:columnName associated with the variable var:sColumnName to denote the name of a column of var:sourceTable (schema level). Whereas the var:sourceColumnValue entity contains the attribute d2p:columnValue associated with the variable var:sColumnValue to represent a value of a column (data level). The inclusion of these prefixes in the PROV templates makes it feasible to achieve the *join management of data and schema provenance information (P1)* property, and as a consequence, the provenance consumption of the generated PROV documents will be able to differentiate between schema and data level information.

Table 2 contains a summary of the main elements involved in the process for specifying the PROV template of the COPY TABLE SMO, together with the elements of the other two SMOs that conform the *running example*, that is, the MERGE TABLE and the DECOMPOSE TABLE SMOs. This table shows that the key elements of these three SMOs are very similar, and as a consequence, the PROV templates of the MERGE TABLE and DECOMPOSE TABLE SMOs are of the same complexity than the template of the COPY TABLE SMO. As we have mentioned previously, the corresponding

patterns and templates of these three SMOs can be consulted in the Supplementary Material [36] and in *ProvStore* [39], respectively. We would like to note that these aspects could be extrapolated to the PROV templates of the remainder SMOs.

3.3 Evolution Provenance Capture

The evolution provenance information has to be captured during the execution of a schema evolution application. Our proposal implies that, in order to carry out this task, it is necessary to build (for each application) a module that is responsible for capturing the provenance according to the PROV templates defined in the previous phase. In this way, PROV-IDEA improves schema evolution applications, making them provenance-aware (see Figure 3, Phase 2). The capture of provenance data during the SMOs/DMOs execution involves providing the application with additional instructions for the generation of bindings. Bindings are variable-value pairs that associate concrete values collected from the schema and/or data changes (provenance information) with variables in a PROV template.

Our approach proposes a *non-intrusive and seamless integration (P4)* of the PROV-IDEA module into the chosen schema evolution application. More specifically, in order to achieve this property, the database application must be enriched with an additional module, without modifying the application code. To generate the bindings, this module not only has to contain the instructions for bindings generation, but it also has to identify when SMOs/DMOs operations are executed by the schema evolution application. For this reason, within the PROV-IDEA module, an *application-independent* component and an *application-dependent* component, respectively, are differentiated.

Furthermore, following a similar approach as that presented in [48, 49], we propose, for the *PROV-IDEA module*, an event-based design that identifies two main elements: *events*, which are notable occurrences that happen while the application is running, and *listeners*, which specify the behavior for processing the events. This distinction allows decoupling provenance capture from the actual generation of provenance data, so that the same schema evolution application can have simultaneously several listeners, each managing provenance in a different way.

Thanks to decoupling provenance capture from provenance generation, our proposal is *flexible with regard to provenance computation (P5)*, being agnostic regarding *where*, *how*, and *when* to compute provenance. More specifically, bindings, generated by the *application-independent* component, can be stored in different storage systems (e.g., file systems, relational databases or NoSQL systems), serialized in a more or less verbose format (e.g., JSON, TURTLE), or being persisted in different ways (individually or as sets of bindings, or even as expanded PROV documents). Besides, considering the two main strategies generally used by provenance systems to decide when to compute the final provenance data [33] (either when it is required, usually referred to as *lazy*, or immediately, *eager*), our proposal gives freedom to follow a *lazy* or an *eager* approach when implementing listeners. Thus, the event-based design facilitates decision making on all these aspects.

Regarding our *running example*, let us suppose that the schema operation `COPY TABLE developer_sub_B INTO developer_sub_B_old` is executed on the evolution application. Some of the possible bindings that will be generated by the PROV-IDEA module due to this operation execution are depicted in Figure 7(b); for example, the variable `var:targetTable` of the *copy table* template is associated with the value `exe:developer_sub_B_old Table` and the variable `var:sourceColumnValue` is related to `exe:nameD_developer_sub_B_1` and `exe:surnameD_developer_sub_B_1`. Let us note again that the set of bindings contains information at both schema and data level, making it feasible the *join management of schema and data provenance (P1)*.

Finally, the expansion algorithm [30] takes as input the PROV templates and the bindings, and generates the PROV documents for the executed SMOs/DMOs. The resulting PROV documents

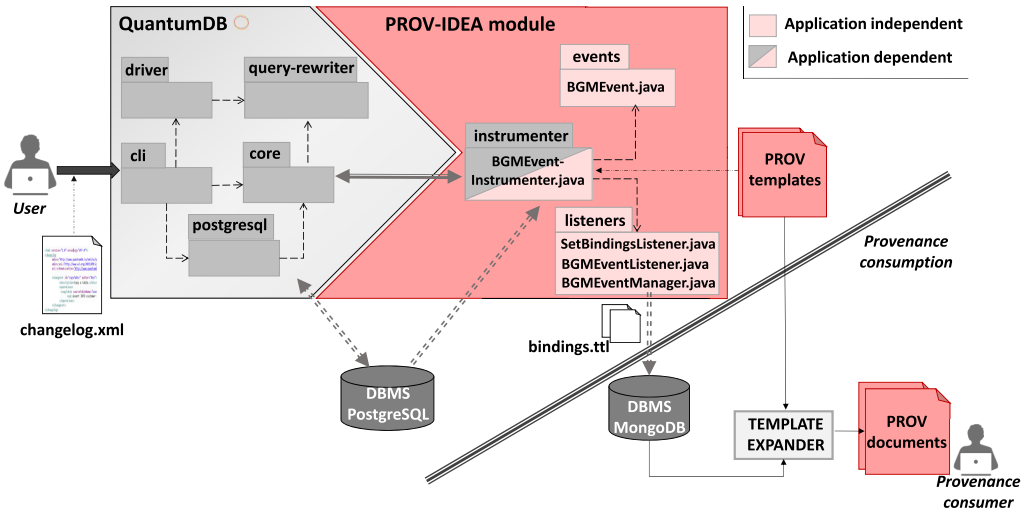


Fig. 8. The PROV-IDEA approach applied to QuantumDB tool.

include high-quality schema and data provenance ready to be exploited by the provenance consumer. With regard to our *running example*, Figure 7(c) shows a PROV document (without properties) obtained from the template in Figure 7(a) and the bindings in Figure 7(b).

4 Development of PROV-IDEA Modules: A Proof of Concept

In this section, we introduce general aspects regarding the development of a PROV-IDEA module. More specifically, this module comprises the *application-independent* components, which are agnostic about the concrete schema evolution application being used, and the *application-dependent* component, whose implementation depends on the specific application at hand. Aimed at providing a proof of concept for our proposal, we have applied it to a concrete schema evolution application called QuantumDB. Thus, in this section, we also provide additional insights into the development of the PROV-IDEA module to be integrated into this application, and more specifically, of the *application-dependent* component.

QuantumDB's Way of Working and Architecture. QuantumDB [15, 41] is an open source Java tool designed to execute database schema migrations on SQL databases with zero-downtime, that is, without exhibiting blocking behavior during the evolution of databases (e.g., table locks are set when performing changes). QuantumDB was chosen for several reasons, among which the following two stand out. First, QuantumDB tracks finer-grained data than other schema evolution tools (such as Liquibase [28] or Flyway [16]), so that we can compare and place value on the provenance generated by PROV-IDEA. Second, this tool has been compared extensively with other 11 schema migration tools against a list of four criteria concerning *functional*, *correctness*, *performance,* and *tolerance of errors* aspects, concluding that QuantumDB stood out from the rest, scoring highest in the list of criteria [43].

QuantumDB, like other schema evolution tools, supports schema changes to multiple tables using *changesets*, which are comprised of a logical sequence of operations which alter or refactor the database schema. These changesets, at the same time, are grouped in *changelogs*, which allow making several non-trivial changes to the database schema in one go. In this proof of concept, we provide such changesets in a changelog XML file (named `changelog.xml`, see Figure 8), for being the recommended and easiest option to manage databases with QuantumDB [41]. In particular,

QuantumDB is not strictly driven by SMOs and DMOs, but implements a variety of schema change operators (see [41] for more details). Among the different components QuantumDB is made of (see Figure 8), here we focus on those more relevant to the implementation of the PROV-IDEA module, that is, the *cli* component, which mainly contains specific classes needed to process the XML code in the changesets, and the *core* component, that contains the main logic and ontology classes that represent the database structure.

PROV-IDEA Module Development. Our event-design proposal has been materialized in the *application-independent* components (which correspond to the `events` package, with the `BGMEvent` class, and the `listeners` package, mainly with the `BGMEventListener` interface and the `BGMEventManager` class) and the *application-dependent* component (which correspond to the `instrumenter` package, mainly with the element `BGMEventInstrumenter` class).

Application-Independent Components. Regarding the *application-independent* components (see Figure 8), the objects of the `BGMEvent` class, in the *events* package, carry information about the occurrence of events related to the execution of SMOs/DMOs and encapsulate the data provenance needed for constructing the corresponding bindings (mainly, variable-value pairs). These events are captured by *listeners*, that is, classes that implement the `BGMEventListener` interface. This interface can be implemented according to the preferred strategy for computing provenance (i.e., for generating, managing, and storing the bindings data carried by the `BGMEvents`), thus allowing us to reach the *flexible provenance computation* (*P5*) property of our approach. More specifically, in this proof of concept, we have developed a single listener, named `SetBindingsListener`, which performs an asynchronous recording, in which bindings are accumulated in memory before being shipped, as sets of bindings (bulk submission), serialized in TURTLE format, to a MongoDB database after finishing the execution of each operation. Although we have defined a single listener, the proposal allows the use of several listeners, each considering a different strategy for computing provenance. At this respect, the helper class `BGMEventManager` is used to manage the list of subscribed listeners and to disseminate the `BGMEvent` objects among them as events occur.

It is worth noting that both *application-independent* components are agnostic about the concrete schema evolution application being used. Since they are made up of those elements that do not depend on the concrete application, they are the same in all the PROV-IDEA modules (depending on the programming language used to develop the application).

Application-Dependent Component. As for the *application-dependent* component, the `BGMEvent-Instrumenter` class is in charge of recognizing the execution of SMOs/DMOs in the instrumented application and, consequently, triggering events by generating `BGMEvent` instances with the provenance data, and disseminating them among the `BGMEventListeners`, by means of the `BGMEventManager`. This task requires identifying specific application classes, and those places within them, where `BGMEvent` objects have to be fired. For this, as done in [48, 49], we rely on the **Aspect-Oriented Programming (AOP)** paradigm [24] which allows us to implement the `BGMEventInstrumenter` by placing provenance capture instructions apart from the application's source code, thus avoiding to scatter such instructions across this code. This strategy provides non-intrusive instrumentation of schema evolution applications which, together with the large number of languages that implement AOP (e.g., Java, PHP, Python or C#) [27], allows us to fulfill the *non-intrusive and seamless integration* property (*P4*). For example, in the particular case of QuantumDB, given that it is implemented in Java, we have developed the `BGMEventInstrumenter` using AspectJ [26], an AOP extension created for Java.

Although the development of the `BGMEventInstrumenter` depends on the concrete schema evolution application, we have established general ideas developers of the PROV-IDEA module can follow to create this component. In order to support our explanations, we describe them considering

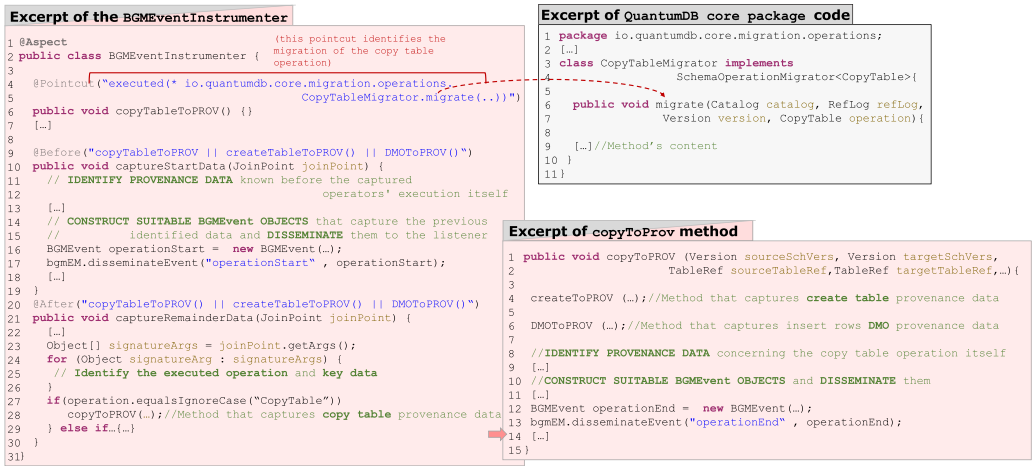


Fig. 9. Structure overview of the BGMEventInstrumenter in AspectJ.

the particular case of QuantumDB, using as an example the implementation of the COPY TABLE SMO, for being one of the most complex SMOs supported by this application.

We propose to develop the BGMEventInstrumenter by means of an AOP *aspect* (see in Figure 9 an excerpt of the AspectJ *aspect* defined for QuantumDB). The aspect will be made up of specific *pointcut*s and two *advice*s of type *before* and *after*, respectively, according to the following considerations.

— *Pointcuts*. First, we propose to define pointcuts to capture the execution of SMOs/DMOs, so that we can be aware of the execution of the code operations concerning schema evolution. The intention is that the execution of the operations captured by the pointcuts allows us to gather the provenance data to be captured concerning the schema evolution operation at hand, either directly or after performing some data preprocessing. Both the *pointcuts*' characteristics and their number would depend on the strategy followed by the schema evolution application to implement the SMOs/DMOs.

For example, in the particular case of QuantumDB, the defined *pointcuts* are built upon the *core* component, since the logic classes it contains provide us with a basis to identify when SMOs/DMOs operations are executed, as well as to gather the corresponding provenance data. However, the implementation strategy of SMOs and DMOs in QuantumDB differs. Each SMO is encapsulated by an independent Java class in charge of implementing the core logic of the SMO, thus making it easier for us to gather the provenance data to be captured. Whereas, DMOs are executed by a general Java method that directly sends the SQL statement to the database to be executed, thus requiring an extra effort to extract the desired provenance data (such as involved tables, columns, or rows) involved in the execution of the SQL statement. In the particular case of the COPY TABLE SMO, since it implicitly includes the CREATE TABLE SMO and the INSERT DMO, its implementation also considers these two operators, thus giving support to a complete set of related SMOs/DMOs. Taking this into account, a *pointcut* has been defined per each considered SMO to capture its execution while, given the previous explanation, we have created an only *pointcut* to capture the execution of any DMO operation (the INSERT operation, in this case). As a way of example, in the *pointcut* named copyTableToPROV (see lines 4 and 5 on the left hand side of Figure 9), the operation whose execution is captured corresponds to the method migrate of class CopyTableMigrator. This method is in charge

of migrating the database to the evolved state, which ensures us that the SMO has been performed.

— *Before and After Advices.* Associated to the defined *pointcut*s, a `before` and an `after` advice are created to construct (and disseminate) suitable instances of the `BGMEvent` class which encapsulate provenance data known before or after the operators' execution itself, respectively.

Regarding the `before` advice, it is mainly in charge of creating and disseminating an instance of the `BGMEvent` class with the information regarding the start of the SMO/DMO operator's execution captured by the corresponding *pointcut*. The provenance data captured are built upon the parameters passed into the captured operation (the data would be obtained either directly or after performing some preprocessing, depending on the strategy followed for developing the application). See, for example, the `before` advice in lines from 9 to 19 on the left hand side of Figure 9 and, in particular, the creation and dissemination of the `BGMEvent` instance (see lines 16 and 17, respectively).

Concerning the `after` advice, it mainly identifies the type of the captured operation (i.e., a concrete SMO or a DMO) and gathers preliminary key data from the parameters passed into such an operation (see lines from 24 to 26 on the left hand side of Figure 9) so that they can be passed into concrete auxiliary methods that finally capture provenance data concerning the executed SMO/DMO. For example, in the particular case of the `COPY TABLE` SMO, we have implemented three auxiliary methods, one per each considered operator, where the method implementing the `COPY TABLE` SMO (see the `copyToPROV` excerpt on the right hand side of Figure 9) not only captures provenance data related to the execution of the operator itself, but it also invokes the auxiliary methods implementing the `CREATE TABLE` SMO and the `INSERT` DMO (see lines 4 and 6) in order to finally capture the overall provenance data, concerning the `COPY TABLE` SMO.

As a result, the strategy we propose to develop the `BGMEventInstrumenter` by means of the *aspect* allows the collection of both schema and data provenance information considering the structure of the defined PROV templates and the variables included in the bindings corresponding to those in the templates (see Section 3.2). Thus, it allows us a *join management of both data and schema provenance information* (*P1*), enabling *interoperability* of data *using provenance standards* when finally obtaining the corresponding PROV documents (*P2*). More specifically, QuantumDB, which gives support to schema evolution considering a coarser level of granularity than PROV-IDEA and does not support data versioning (inserting, updating, and deleting), is enriched with a finer level of granularity of schema evolution and with data versioning support. Additionally, the strategy followed to develop the AOP aspect by identifying such auxiliary methods according to the strategy followed by QuantumDB to implement SMOs and DMOs guarantees the *adaptable and expandable* property of the PROV-IDEA approach (*P3*). For more information, see the Supplementary Material [36] and the QuantumDB code enriched with the PROV-IDEA module [42].

5 Evaluation

We have evaluated our proposal using qualitative and quantitative arguments to show the benefits and tradeoffs of applying PROV-IDEA for both provenance consumers and software engineers seeking to make their schema evolution applications provenance-aware. In particular, in this evaluation we show the usefulness and potential of the generated provenance, as well as PROV-IDEA impact and costs.

The evaluation has been performed using the QuantumDB tool together with the PROV-IDEA module, based on the schema evolution operations performed in our *running example*. In particular, the evaluation was run on a personal computer, Intel(R) Core(TM) i5 CPU, 2.4 GHz, with Oracle

JDK-17 and a Windows 10 Enterprise OS. We used MongoDB database version 7.0.5 to store the sets of bindings resulting from the schema and data evolution, while PostgreSQL 16 was used as persistence system of the QuantumDB tool.

5.1 Qualitative Evaluation

In order to qualitatively assess the provenance information generated by our approach, we must validate that we can answer *Why, How, Where, and When-provenance* questions from the PROV information obtained. In particular, we must analyze the relevance of PROV-IDEA's capability to record fine-grained provenance. Our qualitative assessment setup consists of the generation of PROV documents and their consumption by performing the concrete list of provenance questions to be asked (in this case, these questions have been defined for the *running example*).

The generation of PROV documents has been carried out using our tool called *PROV-IDEA Expander* (for details see the Supplementary Material at [36] and the corresponding GitHub repository at [37]). This tool allows us to obtain PROV documents from bindings with schema evolution provenance data. More specifically, the tool automatically goes through the binding documents associated with the different schema evolution operations (both SMOs and DMOs), finds the PROV template related to each binding document (i.e., to each operation), and, finally, uses the *expansion algorithm* [30] to generate the corresponding PROV document ready for provenance consumption.

To perform provenance consumption, we have developed a tool called *PROV-IDEA Provenance Inspector* (again, the details can be found in the Supplementary Material [36], while the corresponding GitHub repository is available at [38]). More specifically, this tool allows us to query the generated provenance using different SPARQL [47] query forms, that is, `SELECT`, `DESCRIBE`, `CONSTRUCT`, and `ASK` (see the top panel of Figure 10). With these different variants of SPARQL, we can obtain different types of information, either in tabular format (`SELECT` queries) or in PROV graphic format (`DESCRIBE`, `CONSTRUCT`, and `ASK`). Figure 10 shows the graphic PROV representation of the result obtained from the execution of a `CONSTRUCT` query that provides *Why-provenance* information, by obtaining the last SMO used to generate a concrete value (the example is explained in detail below).

Regarding the specific experiment setup, we have started from the tables `developer_sub_A` and `developer_sub_B` of our *running example*, each one with several developers simulating workers of the two subsidiaries of the main company. Next, we have used QuantumDB, together with the PROV-IDEA module, to perform the schema operations included in the *running example* (i.e., a `COPY TABLE` SMO, a `MERGE TABLE` SMO, and a `DECOMPOSE TABLE` SMO, as described in Section 3). The bindings, generated as TURTLE format files while performing the evolution changes, have been transformed into PROV documents (also in TURTLE format) using the *PROV-IDEA Expander*.

Finally, we have defined, in the context of the *running example*, the specific questions that will be answered by consulting the PROV documents using the *PROV-IDEA Inspector* tool. These questions are related to the registered provenance information (*Why, How, Where, When*), and, therefore, can only be answered from the information generated by the PROV-IDEA module. It is worth noting that, although we have applied these questions to the particular context of the *running example*, here we present their general form to show their interest and usefulness in any other context. Among the many questions that could be asked, we detail the following:

— *What Was the Last SMO Used to Generate a Concrete Column Value in a Concrete Table and the User Who Executed Such an Operation?—Why*. In this case, we can find out why a certain value ultimately appeared in a table row and by whom, facilitating the explainability of the column value creation/modification. We are able to answer this question thanks to different key elements included in the PROV templates. More specifically, we know which was the last

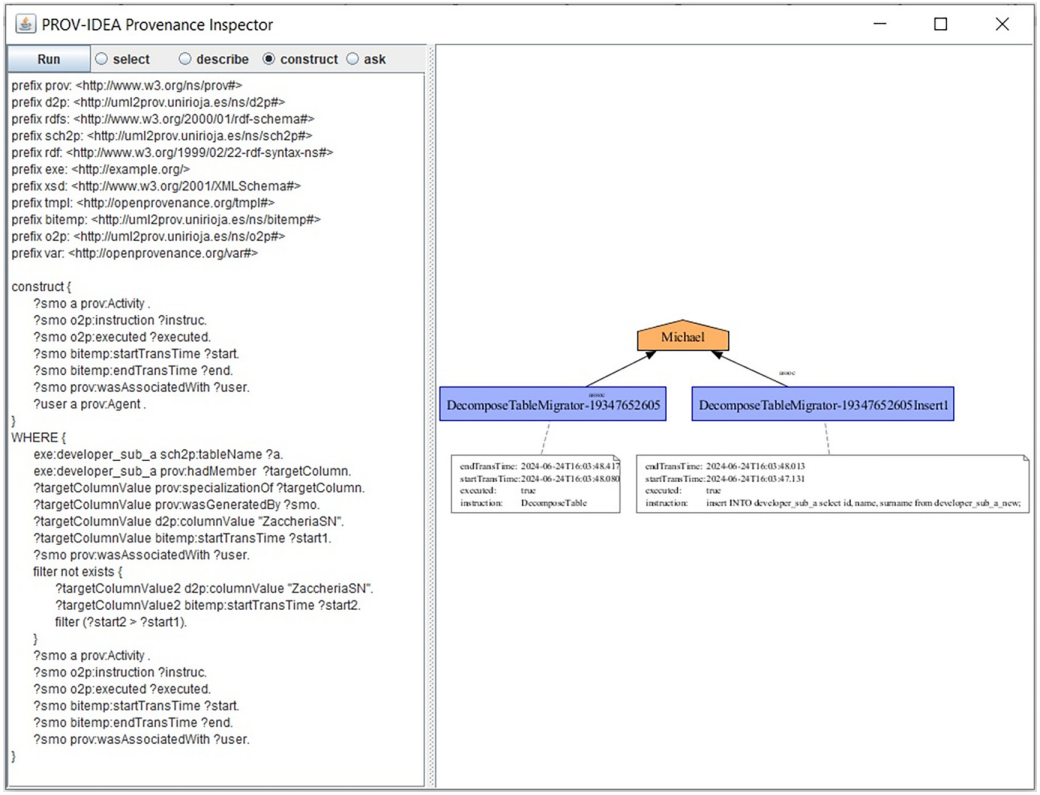


Fig. 10. Interface of the PROV-IDEA Provenance Inspector.

SMO thanks to the timestamp information registered related to each SMOs/DMOs execution. The executor of the operation is known by means of the *wasAssociatedWith* relation between an SMO and the agent who executes it. Additionally, we can identify the generated column value thanks to the *wasGeneratedBy* relation, while the concrete table to which it belongs is known thanks to the *hadMember* relation. In the case of a *Why deep-provenance* question, the information regarding the executed SMOs/DMOs together with the relations among them (complex operations and their decomposition into simpler ones) is available thanks to the *wasInformedBy* relation. In particular, Figure 10 shows this question and its answer, particularized for a concrete surname value of a developer of table `developer_sub_A`. The query has been defined by a `CONSTRUCT` query form using a *Why deep-provenance* strategy. The result, depicted by a graphic PROV representation, shows the SMO (`DECOMPOSE TABLE`) and one of the concrete operators into which it is decomposed (`INSERT`) which have generated such a value, together with the user who executed them.

— *What Have Been the Schema/Tables/Columns or Column Values Invalidated by the Execution of the SMOs Performed by a Concrete User?—Why and How.* In this case, as part of the answer we can obtain not only the name or identification of the database element itself but also the type of schema evolution operator the user performed. In addition to other key elements mentioned previously, the key aspect used to answer this question is mainly the *wasInvalidatedBy* relation among the executed SMOs and the corresponding database element. This query could be especially useful for recovery purposes (e.g., to know invalidated tables).

Table 3. Runtime Overhead and Storage Costs

Num. rows	Execution time without PROV-IDEA (ms)	Execution time with PROV-IDEA (ms)	Time overhead	Total size of bindings (KB)
10	673.8	1,659.2	146%	89.16
100	692.4	1,732.6	150%	754.8
500	716.8	1,777.8	148%	3,737
900	743	1,896.8	155%	6,748

— *What SMOs Have Been Carried Out between Two Dates?—Why and When.* In this case, we can audit not only the schema evolution operations that were performed during that period but also the relations among them (complex operations and their decomposition into simpler ones), and how long they took to execute. As in a previous question, the key aspects used to answer this one are mainly the timestamp information registered related to the SMOs/DMOs execution and the *wasInformedBy* relation among the executed SMOs/DMOs.
— *Which Were the Concrete Modifications Performed in the Database Elements to Obtain a Concrete Column Value—How.* In this case, we can obtain, for reproducibility purposes, the concrete sequence of executed operations that give rise to such a value (even including the concrete insert SQL instruction), together with the input entities of such operations that gave rise to the concrete value. Here the key aspects used to answer this question are the temporal information registered, together with the *wasInformedBy*, the *wasGeneratedBy*, and the *hadMember* relations.
— *What Is the Schema a Database Element (Table, Column, Column Value) Belongs to?—Where.* Since we register the schema version each database element belongs to at any time, this question is easily answered by the provenance registered by our proposal. In particular, we have performed this question for a concrete surname value of a developer of table `developer_sub_A`, useful for tracing this personal data.

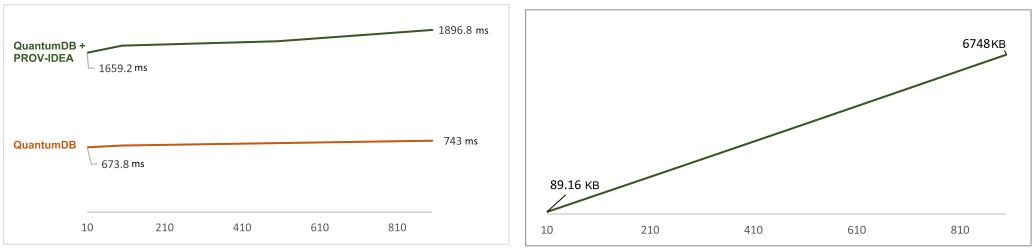
The details of the questions we have defined for this evaluation, together with the corresponding answers, are available in the Supplementary Material [36].

In conclusion, we have shown that the fine-grained provenance registered by our proposal makes it feasible to answer useful provenance queries with different granularity, both at schema and instance levels.

5.2 Quantitative Evaluation

This evaluation, on its part, has been performed in terms of runtime overhead and storage cost. Regarding the experiment setup, we have based on the execution of the `COPY TABLE` SMO of our *running example*. In particular, we have considered the copy of table `developer_sub_B` with four different number of rows (see Table 3). We have used the average of five runs to report both execution times and provenance storage. It is worth noting that both runtime overhead and storage cost are mainly caused by data rows involved in the schema evolution, since the provenance associated to schema changes themselves remains constant no matter the number of rows in the table to be copied.

Runtime Overhead. We have evaluated the effect of PROV-IDEA compared to the average execution time when it is not used. As shown in Table 3, the runtime overhead compared with



(a) Runtime overhead.

(b) Storage cost.

Fig. 11. Evaluation.

QuantumDB without PROV-IDEA is on average around 150% (see Figure 11(a)). That is, the runtime overhead remains constant even as the number of rows increases.

Although these results are higher than others obtained in previous works [32, 48, 49] where provenance is captured at a fine-grained level of detail with a runtime overhead of around 50–75%, we are aware that the use of QuantumDB with PROV-IDEA entails extra runtime. More specifically, QuantumDB has an overload of calls to some of the methods our AOP proposal needs to use to gather the data provenance (some methods are invoked several times on the same schema evolution operation), which means that the AOP strategy acts more times than necessary. This aspect is, however, independent of our strategy, and it is difficult to exactly quantify its impact on the time required by PROV-IDEA.

Storage Cost. We have measured the size of the sets of bindings stored as TURTLE format files as evolution changes are performed, affecting both schema and data. As commented previously, the provenance stored related to the copy of the source table itself (including the creation of the new table) keeps constant no matter the number of rows in the table to be copied. Thus, the results collected in Table 3 and depicted in Figure 11(b) show that the size of the set of bindings is strictly proportional to the number of rows in the source table.

All in all, the results show that runtime overhead remains constant even as the number of rows increases, while the size of the stored provenance is strictly proportional to the number of rows involved in the schema evolution.

6 Related Work and Discussion

The interest of joint research on database evolution and data provenance has been recognized in several studies. For example, in their recent work on trends and future of data modelling, Jaakkola and Thalheim [23] point out that "data modelling must change along with changes in applications and technology evolution," and that this is one of the future paradigms in this area. More specifically, they point out that "data models reflecting provenance and quality" are needed, that classical approaches "will be replaced by the co-evolution of models and their data," and that adequate documentation of "data models that have evolved over time" is required.

Schema evolution and schema versioning are two different techniques that support database schema changes without loss of existing data. As Roddick [44] points out, both terms have sometimes been used interchangeably in the literature. The nuance lies in the fact that schema versioning implies that both new and old schema versions are preserved in the database, while schema evolution implies that only the current schema version is kept [45]. Brahmia et al. [6] note that there is no work on data provenance in the framework of schema versioning. We believe that the PROV-IDEA proposal can, at least partially, fill this gap. We have framed our approach within schema evolution, since in our proposal the old schemas are not preserved as-is (as required by

the definition of schema versioning just discussed), but the provenance information we record is rich enough to allow recovery of the old schemas (as well as their data), for which we could apply similar strategies to the used by Curino et al. [12] (invertible data migration, query rewriting).

The remainder of this section is structured according to the five outstanding properties of the PROV-IDEA proposal, to make it easier to follow the comparison with other related work.

Join Management of Data and Schema Provenance Information (P1). A tool for representing schema evolution by using provenance queries is presented in [1]. This tool focuses on visually showing the evolution at the schema level and does not address the data level. In fact, there are few proposals in the literature that have jointly considered the provenance of both schema and data evolution. Among these scarce proposals, it is worth mentioning the AM&PM system [17]. This proposal stores provenance of schema and data updates (inserts, updates, etc.) in a transaction-time database, but the level of detail of the registered provenance is not as fine-grained as the one considered in PROV-IDEA. More specifically, it does not give attention to archive explicit information regarding tables' or columns' versions, or information regarding database elements invalidated by or derived from other elements, and even, it does not register finer-grained data such as when data are taken as source to populate other tables (insert-select operations). The works of Auge and Heuer [2, 3] also recognize the need to consider the evolution of data and schema together as well as the benefits of using provenance techniques to address evolution issues. However, being restricted to the field of research data management, these works are limited to solving the problem of determining the minimum information required for the reconstruction of a sub-database. On the other hand, although neither strictly provenance nor SMO/DMO driven, it is worth noting database migration tools such as Flyway, Liquibase, or QuantumDB itself. These tools handle database changes based on, namely, Migration Based Database Delivery [4], so that, to transition a database from one version to the next, they base on migration steps, which are Data Definition Language and Data Manipulation Language changes. Similarly to QuantumDB, in Flyway and Liquibase migrations are structured in changesets which are the basic unit of change and which, at the same time, are grouped in changelog files. Flyway and Liquibase use these files to track every version of the database. Focusing on provenance aspects, these tools register scarce or no information regarding the trace of database objects' versions concerning both schema and data. More specifically, while Liquibase and Flyway just track executed changesets in a special metadata table in the same database, QuantumDB goes one step further registering also how every table evolves over time through evolution schema operations, as well as what transformations they undergo. PROV-IDEA makes a clear difference among these tools providing a finer level of granularity considering both schema evolution and data versioning. Thus, our scope is much wider, as our solution allows us to generate as much provenance information as the designer/developer needs, being able to answer *Why, How, Where, and When-provenance* questions. In any case, whatever proposal, the challenge is to strike a balance between the desired tracked data and the storage overhead.

Interoperability by Means of Provenance Standards (P2). A recent study [33] has concluded that a large number of provenance systems (considering both general and specific purpose systems) have their own model for managing provenance. In the schema evolution context, works such as [1, 2, 12], or [17] manage provenance information to some extent, but none of them do so based on the use of a standard. The absence of the use of standards for collecting, representing, storing, or querying provenance constitutes a hindrance to promoting the sharing and use of provenance information [14, 46]. Against this background, PROV and PROV-Template have emerged as a cross-domain standard for provenance and a templating system to generate PROV, respectively. Since PROV is indeed a W3C standard, which in particular is a key factor in facilitating interoperability, it is striking that there is hardly any works using PROV in relation to data evolution management. In fact, in this respect, we can only refer to the proposal of Pimentel et al. [35], who define an

extension of PROV to support mutable data entities. It is therefore a proposal that relates PROV to the idea of entities that change in a general sense, so it is not specifically linked to the problem of the evolution of database schemas. PROV-IDEA benefits, on the one hand, from the maturity and interoperability of the PROV standard, and on the other hand, from the PROV-Template approach regarding the reduction of both development and maintenance effort, thanks to the separation of responsibilities between software and provenance designers. Thus, in this respect, our proposal provides clear benefits over other systems managing provenance in the schema evolution context. In the provenance context in general, our approach keeps some similarity with the *UML2PROV* proposal [48, 49], which bases on the PROV-Template approach to representing the provenance related to operations' executions in general purpose applications, to make them provenance-aware. However, PROV-IDEA manages two levels of provenance information (database schema, and data), while UML2PROV just captures one level (data generated from the execution of applications' operations).

As a possible threat to validity, in order to create provenance correctly, it is required that the sets of bindings generated by the PROV-IDEA module must be compatible with the template they are meant to be used with. At this respect, the PROV-Template approach allows a series of static and dynamic checks to be supported, which help the application log the necessary information to create provenance correctly [30]. In particular, the PROV templates defined in our approach (1) have been checked against the *ProvValidator* tool [40] to prove that they are valid (static checks) and (2) include a number of safety checks which help determine whether bindings are compatible with templates, such as type-compatibility or arity (dynamic checks).

Adaptable and Expandable (P3). Our approach is based on the PROV-Template approach to represent SMOs and DMOs, so that templates can be adapted to represent different evolution implementation approaches, also registering possible dependencies among SMOs and DMOs. Additionally, it also allows the expansion of templates in order to consider different evolution variants within the same evolution approach. For example, in [19] the semantics of the `MERGE TABLE` SMO (`MERGE TABLE S, T INTO R`) includes the creation of a new table *R* populated with tuples of the source tables *S* and *T*, followed by the removal of the original tables (thus, the implementation of the PROV template for the `MERGE TABLE` SMO would include the PROV templates corresponding to the `CREATE TABLE` SMO, the `INSERT` DMO, and the `DROP TABLE` SMO). However, in [12] this operator involves merging the tuples of one table *S* into another target one *R*, being table *S* optionally removed (thus, this proposal would include the PROV template corresponding to the `INSERT` DMO and, optionally, the `DROP TABLE` SMO). Our proposal can be adapted to different evolution approaches (such as those in [19] and [12]), allowing the representation of alternative variants of the same approach by means of template expansion (e.g., in the [12], expanding the PROV template to consider the `DROP TABLE` SMO, if this operator is required). The *UML2PROV* approach [48, 49] comes closer to our proposal, using PROV templates to represent the provenance related to operation's executions. However, while UML2PROV registers provenance information concerning the execution of applications themselves (generated objects, methods' execution, etc.) without considering dependencies among operations' executions, PROV-IDEA focuses on registering the effect that the execution of SMOs/DMOs in evolution applications have on schemas and data, also registering the influence among operations' executions.

As a possible threat to validity, our proposal works on the assumption that the schema evolution process is driven by SMOs and DMOs. However, our proposal, based on the PROV-Template approach, is easily adaptable to different evolution formats. In particular, in the proof of concept we have applied our proposal to QuantumDB, a tool not strictly driven by SMOs and DMOs, showing that PROV-IDEA could also give support to these situations.

Non-Intrusive and Seamless Integration (P4). According to Brahmia et al. [6], all commercial DBMSs lack explicit support for schema versioning. In addition, the existing proposals that consider provenance for managing schema evolution provide their own stand-alone tools or research prototypes [1, 2, 12, 17], not giving the possibility to be integrated with third-party database evolution tools. Our proposal goes one step further, not only aiming at enriching database evolution tools with provenance capabilities but also avoiding interleaving provenance generation code into the application's source code. While PROV-IDEA does not establish a concrete method or technology to be used to achieve non-intrusive and seamless integration, we can compare existing approaches with the strategy followed in our proof of concept. More specifically, we have relied on AOP which has been widely used for monitoring purposes [5, 9]. However, to our knowledge, only *UML2PROV* [48, 49] and the *CAPS* framework [7] use AOP to capture provenance apart from PROV-IDEA. Other examples are *PASS* [21] or *noWorkflow* [34] but, contrary to these approaches, PROV-IDEA provides a general solution which does not require a kernel modified for automatic collecting provenance (like PASS) and it is not restricted to a specific programming language (such as *noWorkflow* or *CAPS*).

Taking this into account, a possible threat to validity of our proposal could be related to the effort needed to develop the *PROV-IDEA module* and, more specifically, the *application-dependent* component. In this respect, one of the positive aspects derived from the use of PROV templates lies in the fact that the provenance-related coding of this component is reduced to bindings generation, which can be embraced by developers without PROV expertise. Taking this into account, the greatest effort attributable to the development of the *application-dependent* component would correspond to the identification of the methods in charge of executing the SMOs/DMOs and the extraction of the data needed to create the bindings (either directly from their parameters or after performing some preprocessing). Thus, this task would depend on the development strategy followed by the application; the more specified the information is in the evolution operations, the easier it is the development of the module. At this respect, aspects such as the degree of familiarization the developer has with the application's code, as well as having direct contact with the application's developers would contribute to make this step easier. Finally, it is worth noting that our approach based on PROV templates facilitates the maintenance of the provenance since, as stated in [30], it allows minor revisions of provenance to be supported, without having to modify the application (the *application-dependent* component in our case), as long as the templates still rely on the same logged values.

Flexible Provenance Computation (P5). PROV-IDEA facilitates decision making on *where*, *how*, and *when* to compute provenance, so that software developers can program their preferred persistence systems, storage or distribution approaches, format, and so on. The chosen option can have a direct impact on aspects such as runtime overhead, storage needs, or provenance consumption effort. Some insights and recommendations could be given based on the systematic evaluation made in [48, 49] within a similar provenance-capture context. In such an evaluation the following three strategies are analyzed (*when*), all of which using MongoDB as storage system (*where*) and JSON to serialize bindings (*how*): (1) bulk submission of bindings (lazy approach), (2) storing the bindings one by one (lazy approach), and (3) storing the expanded PROV document (eager approach). The evaluation concludes that, while the bulk submission strategy has the lowest performance penalty and the more compact storage approach compared with the other strategies, the direct storage of PROV documents requires less effort to consume provenance (e.g., SPARQL queries can be executed over PROV documents, as we have made in the qualitative evaluation). Additionally, as stated in [30], using a bindings-based strategy provides extra storage advantages over using PROV documents, since the size of bindings turns out to be on average 40 percent of the size of expanded templates. Thus, and focusing on PROV-IDEA, users can choose the strategy that better

fits with their needs and interests considering the pros and cons of these strategies in terms of computing/storage overheads.

As a final remark, the adoption of our proposal may have different implications for practitioners and researchers. In particular, it may be useful to address different challenges around database evolution, such as query rewriting (a similar challenge is addressed by Kondylakis and Plexousakis [25] in the different context of ontology evolution), or issues related to explainability or reproducibility [1, 17, 29]. Since our proposal is based on the PROV-Template approach, a certain level of knowledge of this technology is needed. This is clearly a minor drawback in view of the advantages provided by PROV-IDEA thanks to the properties we have discussed.

7 Conclusions and Future Work

Schema databases evolve to meet the ever-changing environment requirements. As a consequence, the management of the evolution provenance is relevant for dealing with the version history of databases. In this article, we propose PROV-IDEA for managing provenance information of relational database schema evolution, using PROV as provenance framework. As we have shown, PROV-IDEA has five noteworthy properties (P1–P5) that make our approach stand out from other proposals. On the one hand, in terms of provenance *design*, schema and data provenance are managed simultaneously, using the PROV standard as a way to ensure interoperability, and allowing the adaptation and extension of the proposed PROV templates. On the other hand, in terms of provenance *capture*, the proposal advocates for the non-intrusive and seamless integration with existing applications, being agnostic in terms of where, how, and when to compute provenance data. Furthermore, the development of the PROV-IDEA module for the QuantumDB tool, together with its evaluation, has demonstrated the feasibility, the usefulness and potential of the generated provenance, as well as the implications and costs, of our approach.

There are several lines of further work. First, SMOs/DMOs can be complemented with **Integrity Constraints Modification Operators (ICMOs)**. The provenance of ICMOs within relational schema evolution must also be addressed by our proposal. Second, we plan to continue analyzing the applicability of our proposal to tackle aspects such as query rewriting, explainability, or reproducibility. Finally, broadening the scope of our proposal, the management of provenance information for multi-model databases evolution deserves to be investigated.

Acknowledgments

We acknowledge Michael de Jong, QuantumDB's developer, for the information provided on the tool, and Carlos Sáenz-Adán, for his support regarding PROV and the PROV-Template approach.

References

[1] Christos Athinaiou and Haridimos Kondylakis. 2019. VESEL: VisuaL Exploration of Schema Evolution Using Provenance Queries. In *Proceedings of EDBT/ICDT Workshops*, Vol. 2322.

[2] Tanja Auge and Andreas Heuer. 2018. Combining Provenance Management and Schema Evolution. In *Proceedings of the 7th International Provenance and Annotation Workshop*. K. Belhajjame, Ashish Gehani, and Pinar Alper (Eds.), Lecture Notes in Computer Science, Vol. 11017, 222–225. DOI: https://doi.org/10.1007/978-3-319-98379-0_24

[3] Tanja Auge and Andreas Heuer. 2021. Tracing the History of the Baltic Sea Oxygen Level. Evolution and Provenance for Research Data Management. In *Datenbanksysteme für Business, Technologie und Web 2021*. Kai-Uwe Sattler, Melanie Herschel, and Wolfgang Lehner (Eds.), 337–348. DOI: https://doi.org/10.18420/btw2021-18

[4] Bartłomiej Żyliński. 2022. *Database Migration Tools: Flyway vs Liquibase*. Retrieved 4 September 2024 from https://dzone.com/articles/flyway-vs-liquibase

[5] Marco Boskovic, Timo Warns, and Wilhelm Hasselbring. 2006. Model Driven Instrumentation for Relational Event Traces. *Radioelektronic and Computer Systems* 6, 18 (2006), 124–129.

[6] Zouhaier Brahmia, Fabio Grandi, Barbara Oliboni, and Rafik Bouaziz. 2018. Schema Versioning in Conventional and Emerging Databases. In *Encyclopedia of Information Science and Technology (4th ed.)*. Mehdi Khosrow-Pour (Ed.), IGI Global, 2054–2063.

[7] Peer C. Brauer, Florian Fittkau, and Wilhelm Hasselbring. 2014. The Aspect-Oriented Architecture of the Caps Framework for Capturing, Analyzing and Archiving Provenance Data. In *Proceedings of the International Provenance and Annotation Workshop (IPAW '14)*, 223–225.

[8] Peter Buneman and Wang Chiew Tan. 2007. Provenance in Databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (MOD '07)*. ACM, New York, NY, 1171–1173.

[9] Ian Cassar, Adrian Francalanza, Luca Aceto, and Anna Ingólfsdóttir. 2017. A Survey of Runtime Monitoring Instrumentation Techniques. In *Proceedings of the 2nd International Workshop on Pre- and Post-Deployment Verification Techniques*, 15–28.

[10] You-Wei Cheah and Beth Plale. 2014. Provenance Quality Assessment Methodology and Framework. *Journal of Data and Information Quality* 5, 3 (2014), 1–20.

[11] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Foundations and Trends® in Databases* 1, 4 (2009), 379–474.

[12] Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. 2013. Automating the Database Schema Evolution Process. *The VLDB Journal* 22 (2013), 73–98.

[13] Carlo A. Curino, Hyun J. Moon, and Carlo Zaniolo. 2008. Graceful Database Schema Evolution: The PRISM Workbench. In *Proceedings of VLDB Endowment*, 761–772. DOI: https://doi.org/10.14778/1453856.1453939

[14] Susan B. Davidson and Juliana Freire. 2008. Provenance and Scientific Workflows: Challenges and Opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (MOD '08)*. ACM, New York, NY, 1345–1350.

[15] Michael de Jong. 2015. *Zero-Downtime SQL Database Schema Evolution for Continuous Deployment*. Ph.D. Dissertation. Delft University of Technology.

[16] Flyway. 2024. Retrieved 4 September 2024 from https://flywaydb.org/

[17] Shi Gao and Carlo Zaniolo. 2012. Supporting Database Provenance under Schema Evolution. In *Proceedings of the International Conference on Advances in Conceptual Modeling*, Vol. 7518, 67–77. DOI: https://doi.org/10.1007/978-3-642-33999-8_9

[18] Paul Groth and Luc Moreau (Eds.). 2013. *PROV-Overview. An Overview of the PROV Family of Documents*. W3C Working Group Note NOTE-prov-overview-20130430. World Wide Web Consortium. Retrieved 4 September 2024 from www.w3.org/TR/2013/NOTE-prov-overview-20130430/

[19] Kai Herrmann, Hannes Voigt, Andreas Behrend, and Wolfgang Lehner. 2015. CoDEL - A Relationally Complete Language for Database Evolution. In *Proceedings of the 19th East European Conference of Advances in Databases and Information Systems (ADBIS '15)*. Lecture Notes in Computer Science, Vol. 9282, Springer, 63–76.

[20] Andrea Hillenbrand, Uta Störl, Shamil Nabiyev, and Stefanie Scherzinger. 2021. MigCast in Monte Carlo: The Impact of Data Model Evolution in NoSQL Databases.Retrieved 4 September 2024 from https://arxiv.org/abs/2104.11787

[21] David A. Holland, Margo I. Seltzer, Uri Braun, and Kiran-Kumar Muniswamy-Reddy. 2008. PASSing the Provenance Challenge. *Concurrency and Computation: Practice and Experience* 20, 5 (2008), 531–540.

[22] Irena Holubová, Michal Vavrek, and Stefanie Scherzinger. 2021. Evolution Management in Multi-Model Databases. *Data & Knowledge Engineering* 136 (2021), 101932. DOI: https://doi.org/10.1016/j.datak.2021.101932

[23] Hannu Jaakkola and Bernhard Thalheim. 2020. Trends and Future of Data Modelling. In *Proceedings of the 30th International Conference on Information Modelling and Knowledge Bases*, 57–76.

[24] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP '97)*, 220–242.

[25] Haridimos Kondylakis and Dimitris Plexousakis. 2012. Ontology Evolution: Assisting Query Migration. In *Conceptual Modeling*. Paolo Atzeni, David Cheung, and Sudha Ram (Eds.), Springer, Berlin, 331–344.

[26] Ramnivas Laddad. 2009. *Aspectj in Action: Enterprise AOP with Spring Applications*. Manning Publications Co.

[27] Yannis Lilis, and Anthony Savidis. 2020. A Survey of Metaprogramming Languages. ACM Comput. Surv.. 52, 6 (Nov 2020), 1–39. DOI: https://doi.org/10.1145/3354584

[28] Liquibase. 2024. Retrieved 4 September 2024 from https://liquibase.org/

[29] Wolfgang Mauerer and Stefanie Scherzinger. 2021. Nullius in Verba: Reproducibility for Database Systems Research, Revisited. In *Proceedings of the 2021 IEEE 37th International Conference on Data Engineering (ICDE '21)*, 2377–2380. DOI: https://doi.org/10.1109/ICDE51399.2021.00270

[30] Luc Moreau, Belfrit Victor Batlajery, Trung Dong Huynh, Danius Michaelides, and Heather Packer. 2018. A Templating System to Generate Provenance. *IEEE Transactions on Software Engineering* 44, 2 (2018), 103–121.

[31] Luc Moreau and Paolo Missier (Eds.). 2013. *PROV-DM: The PROV Data Model*. W3C Recommendation REC-prov-dm-20130430. World Wide Web Consortium. Retrieved 4 September 2024 from https://www.w3.org/TR/2013/REC-prov-dm-20130430/

[32] Hyunjung Park, Robert Ikeda, and Jennifer Widom. 2011. RAMP: A System for Capturing and Tracing Provenance in Mapreduce Workflows. In *Proceedings of the VLDB Endowment*, Vol. 4, 1351–1354.
[33] Beatriz Pérez, Carlos Sáenz-Adán, and Julio Rubio. 2018. A Systematic Review of Provenance Systems. *Knowledge and Information Systems* 57, 3 (2018), 495–543. DOI: https://doi.org/10.1007/s10115-018-1164-3
[34] Joao Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2017. noWorkflow: A Tool for Collecting, Analyzing, and Managing Provenance from Python Scripts. In *Proceedings of the International Conference on Very Large Data Bases (VLDB '17)*, Vol. 10, 1841–1844. DOI: https://doi.org/10.14778/3137765.3137789
[35] João Felipe N. Pimentel, Paolo Missier, Leonardo Murta, and Vanessa Braganholo. 2018. Versioned-PROV: A PROV Extension to Support Mutable Data Entities. In *Provenance and Annotation of Data and Processes-Proceedings of 7th International Provenance and Annotation Workshop (IPAW '18)*. Lecture Notes in Computer Science, Vol. 11017, Springer, 87–100.
[36] PROV-IDEA. 2024. Supplementary Material. Retrieved 4 September 2024 from https://zenodo.org/doi/10.5281/zenodo.6380742
[37] PROV-IDEA Expander. 2024. Retrieved 4 September 2024 from https://github.com/PROV-IDEA/PROV-IDEA-Expander/
[38] PROV-IDEA Provenance Inspector. 2024. Retrieved 4 September 2024 from https://github.com/PROV-IDEA/PROV-IDEA-Provenance-Inspector/
[39] ProvStore. 2024. Provenance Storage and Distribution. Retrieved 4 September 2024 from https://openprovenance.org/store/public/
[40] ProvValidator. 2024. Retrieved 4 September 2024 from https://openprovenance.org/service/validator.html
[41] QuantumDB GitHub Repository. 2022. Retrieved 4 September 2024 from https://github.com/quantumdb/quantumdb
[42] QuantumDB with PROV-IDEA. 2024. GitHub Repository. Retrieved 4 September 2024 from https://github.com/PROV-IDEA/QuantumDBWithPROV-IDEA
[43] Nick Geral Richter. 2021. Zero-Downtime PostgreSQL Database Schema Migrations in a Continuous Deployment Environment at ING. Business Information Technology MSc (60025). Retrieved 4 September 2024 from http://purl.utwente.nl/essays/88687
[44] John F. Roddick. 2009. Schema Evolution. In *Encyclopedia of Database Systems*. Ling Liu and M. Tamer Özsu (Eds.), Springer, 2479–2481.
[45] John F. Roddick. 2009. Schema Versioning. In *Encyclopedia of Database Systems*. Ling Liu and M. Tamer Özsu (Eds.), Springer, 2499–2502.
[46] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. 2005. A Survey of Data Provenance Techniques. Extended version of SIGMOD Record 2005. Retrieved from https://www.cs.indiana.edu/pub/techreports/TR618.pdf
[47] SPARQL 1.1 Query Language. 2013. Retrieved 4 September 2024 from https://www.w3.org/TR/sparql11-query/
[48] Carlos Sáenz-Adán, FranciscoJ García-Izquierdo, Beatriz Pérez, TrungDong Huynh, and Luc Moreau. 2022. Automated and non-intrusive provenance capture with UML2PROV. *Computing* 104, 4 (Apr 2022), 767–788. DOI: https://doi.org/10.1007/s00607-021-01012-x
[49] Carlos Sáenz-Adán, Beatriz Pérez, Francisco J. García-Izquierdo, and Luc Moreau. 2022. Integrating Provenance Capture and UML with UML2PROV: Principles and Experience. *IEEE Transactions on Software Engineering* 48, 1 (2022), 53–68.
[50] Wang Chiew Tan. 2007. Provenance in Databases: Past, Current, and Future. *IEEE Data Engineering Bulletin* 30, 4 (2007), 3–12.
[51] Tilmann Zäschke, Stefania Leone, and Moira C. Norrie. 2012. Optimising Schema Evolution Operation Sequences in Object Databases for Data Evolution. In *Proceedings of the 31st International Conference on Conceptual Modeling (ER '12)*, Springer, 369–382.
[52] Yu Zhu. 2017. *Towards Automated Online Schema Evolution*. Ph.D. Dissertation. University of California, Berkeley.

Received 12 September 2023; revised 16 July 2024; accepted 4 September 2024