Towards an Efficient and Effective Framework for the Evolution of Scientific Databases

Robert E. Schuler
USC Information Sciences Institute
Marina del Rey, California
schuler@isi.edu

Carl Kesselman
USC Information Sciences Institute
Marina del Rey, California
carl@isi.edu

ABSTRACT

Database systems are well suited to scientific data management and analysis workloads, however, a database must evolve to keep pace with changing requirements and adjust to changes in the domain conceptualization as applications mature. Evolving a database (i.e., updating its schema and instance data) is one of the greatest challenges in database maintenance and the difficulties are compounded by the lack of sufficient tools to support scientists. This paper presents a schema evolution framework based on an algebraic approach that introduces extended and higher-level composite relational operators tailored to the task of schema evolution. These higher-level operators simplify the task of evolving a database for non-expert users, while enabling efficient evaluation of schema evolution expressions.

CCS CONCEPTS

• **Information systems → Database utilities and tools**; **Extraction, transformation and loading**; *Entity relationship models*; Deduplication; Data cleaning;

KEYWORDS

database evolution, schema evolution, relational algebra, scientific data management

ACM Reference Format:
Robert E. Schuler and Carl Kesselman. 2018. Towards an Efficient and Effective Framework for the Evolution of Scientific Databases. In *SSDBM '18: 30th International Conference on Scientific and Statistical Database Management, July 9–11, 2018, Bozen-Bolzano, Italy.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3221269.3221300

1 INTRODUCTION

An important use of databases in science is for the organization, sharing and reuse of data and results. Researchers may start with a data model to represent their initial understanding of the domain and that model may evolve frequently (daily) as the course of a scientific investigation evolves. The difficulty of evolving the model to keep pace with the changing requirements and conceptualizations becomes a limiting factor in the usefulness of scientific databases. Studies of database use in general [11] and scientific databases in particular [9] identify numerous problems encountered with schema definitions. *Evolving the database schema and its instance data remains a difficult, ad-hoc, and labor intensive activity.*

Motivating Example: Suppose that a laboratory conducting research on developmental dysmorphology generates imaging and genetics data on bioreplicate mutants and controls at multiple age stages (embryonic days) to determine when specific phenotypes are present and their underlying genetic evidence. To expedite the setup of their database, they begin with a minimal design of a single table of RNA-seq, ChIP-seq, and micro-CT assays, which embeds biosample details. Rather than committing to specific bio-ontologies, they allow free form data entry for most fields (e.g., age, genes, etc.). They expect to improve the database continually as their protocols take shape (see Table 1).

Table 1: Dataset with bioassay records.

id	type	...	sid	age	genes
A1	RNA-seq	...	V14	E14.5	
A2	ChIP-seq	...	V14	E14.5	
A3	ChIP	...	V14	E14.5	BET1, COL1A2
A4	ChIP	...	V14	14.5	Mitf, Sox5
A5	ChIP	...	V14	E14.5	Leprel1, Leprel1
A6	microCT	...	V14	14.5	
A7	RNA-seq	...	V11	E11.5	
A8	Chip	...	V11	E11.5	ETNK1, SOX5
A9	micro-CT	...	V11	E11.5	

While this schema was good for an expedient start, the research team continually re-evaluates it and seeks to make improvements to address issues of data quality and maturing conceptualization of the research. Since multiple assays are typically performed on each biosample the dataset contains redundant entries (A1–A6 and A7–A9). Attribute fields contain inconsistencies due to differences in terminology use or data entry errors (A6, A8, A9). Free form lists of values (e.g., genes) compound the above problems (A4, A5, A8).

Typical Approach: To remedy the issues, several transformations would be desired: decomposing the assays and samples into separate related tables, defining domain tables for each set of terms (i.e., type, age, genes), factoring out genes into a normalized table related to bioassays. Fixing these issues usually involves the following steps to *transform the schema and its data*: (1) define new tables, (2) query data from source tables, (3) apply transformation (and possibly cleaning) rules to data values, (4) migrate data to the target tables, and (5) drop unused source tables or columns. In the best cases,

SSDBM '18, July 9–11, 2018, Bozen-Bolzano, Italy

ACM ISBN 978-1-4503-6505-5/18/07…$15.00
https://doi.org/10.1145/3221269.3221300

all transformations (2–4) can be expressed in relational terms (e.g., using SQL), however, often they must instead be written in ad hoc procedural programs. Even then, the transformations require a trained DBA and are not accessible to the scientist.

Proposed Framework: Scientists need a semantically higher-level framework for schema evolution (i.e., transformation of schema and instance data [13]) to evolve scientific databases more effectively and efficiently than with conventional tools or related approaches. We present the *Compositional Higher-level Schema Evolution Language* (CHiSEL), a framework based on a unified algebraic approach consisting of extended and higher-level relational operators that raise the level of abstraction for schema evolution operations. Thus, the framework offers users a simpler language in which to express complex schema evolution operations, making the user more effective in the task. By formalizing the *composite operators* algebraically, unlike User Defined Functions (UDFs) or monolithic operators ("black boxes" to the query planner), the framework is able to identify work sharing opportunities to increase the efficiency of evaluating user expressions.

We make the following contributions: (1) definitions of a concise set of basic and extended algebraic operators for schema evolution; (2) definitions of an extensible set of composite, high-level operators that raise the level of abstraction for schema evolution; (3) a user story that exemplifies the usage of the algebra on a realistic scenario of evolving a scientific dataset; and (4) a description of the system design of the framework.

2 RELATED WORK

Schema evolution languages have been an area of active research with a long history [7]. Recently, PRISM++ [3] proposed a language of schema and integrity constraint modification operators intended to ease schema evolution by preserving backward compatibility, and CoDEL [8] proposed a relationally-complete language for evolving schema and data. These languages, however, are relatively low-level and intended for DBAs with specialized skills not accessible to scientists.

Model management approaches [1] address issues of evolving the "mappings" – views, wrappers, object-relational mappings (ORMs) – between applications and databases. A programming platform and complex "executable" model management operators have been proposed [12], while MoDEF [14] proposed an IDE for generating schema versioning scripts based on software model changes. Unlike model management approaches, our approach is concerned with migration of instance data while evolving the schema.

Data cleaning is often done within the context of database evolution and offers complementary approaches that support schema evolution. Wrangler [10] explores an interactive approach to data cleaning, while CleanM [5] presents a query language for data cleaning. While these approaches address concerns for data repair, they do not encompass the evolution of database schema itself.

3 AN ALGEBRA FOR SCHEMA EVOLUTION

Our algebra aims to address requirements derived from recent evaluations of real-world database evolution [4, 6] and our experiences with several active large-scale science databases [2]. These include: table and column (re)definition [6]; merging, splitting, partitioning, and joining tables [4]; normalizing nested values, defining domains, aligning values with dictionaries, and generating "tags" from nested values [2].

3.1 Basic and Extended Operators

The basic logical operators share the conventional definitions of their synonymous operators from the relational algebra; *Select* (σ_φ), *Project* ($\pi_{a_1,\ldots,a_n}$), *Equijoin* ($\bowtie_=$), *Rename* ($\rho_{a_1,\ldots,a_n}$), *Distinct* (δ), and *Union* ($\cup$), while *Assign* ($\leftarrow$) binds a computed relation to a database instance. The framework extends conventional relational algebra with additional primitive operators (see Table 2).

Table 2: Summary of extended logical operators.

Name	Nest	Unnest	Similarity Join	Deduplicate
Symbol	$\Gamma^{e/f}_{path}$	$\mu^{e/f}_{path}$	$\bowtie^{g/s}_{\equiv}$	$\delta^{g/s}_{\equiv}$

Nest and *unnest* extend from nested relational algebra (a.k.a., NF^2 algebra), however, those algebras assume *relation-valued attributes*. Instead, we must allow for *polymorphic-valued attributes*; i.e., attributes that may look like collections (e.g., comma-separated values) but are not well-defined nested relations. The operators accept collection nesting/unnesting expressions (e) and item nesting/unnesting expressions (f) to be applied to the *path*.

Similarity join is a binary operator that performs a fuzzy comparison over the elements of two input sets joining the tuples that meet the matching condition. *Deduplicate* is a unary operator that compares pairs of elements of a set and eliminates tuples that are similar. The operators accept grouping (g) and similarity (s) expressions.

3.2 Composite Operators

In order to improve the effectiveness and reduce the difficulty of schema evolution tasks for the science user, we define higher-level operators that are more semantically aligned with the schema evolution goals of the user. By defining these operators as a functional composition of other (usually primitive) operators, the framework may rewrite user expressions into more efficient equivalent expressions. Figure 1 provides definitions of the composite operators.

$$\Delta_{a_1,\ldots,a_n;b_1,\ldots,b_m} r \Rightarrow \delta(\pi_{a_1,\ldots,a_n,b_1,\ldots,b_m} r) \quad (1)$$

$$\Pi_{a_1,\ldots,a_n} r \Rightarrow \pi_{key[r],a_1,\ldots,a_n} r \quad (2)$$

$$\alpha^e_{a_i} r \Rightarrow \mu^{e/a_i}_{a_i}(\Pi_{a_i} r) \quad (3)$$

$$\omega^{g/s}_{a_i} r \Rightarrow \rho_{a_i/term}(\delta^{g/s}_{\equiv}(\pi_{a_i} r)) \quad (4)$$

$$\xi^{g/s}_{a_i} r \Rightarrow \rho_{term,synonyms}(\Gamma_{syn}(\delta^{g/s}_{\equiv}(\rho_{term,syn}(\pi_{a_i,a_i} r)))) \quad (5)$$

$$d\ \Phi^{g/s}_{a_i}\ r \Rightarrow \rho_{term/a_i}(\pi_{-a_i}(r \bowtie^{g/s}_{\equiv} d)) \quad (6)$$

$$d\ \tau^{e/(g/s)}_{a_i} r \Rightarrow d\ \Phi^{g/s}_{a_i}(\alpha^e_{a_i} r) \quad (7)$$

Figure 1: Summary of composite operator definitions.

Decompose (eq. 1) provides a simple formula to break a functional dependency between non-prime attributes $F(A \rightarrow B) \subset R$

by projecting the distinct non-prime attributes. **Partition** (eq. 2) produces a new relation based on a projection of an introspected or inferred key ($key[r]$) and a set of user-specified non-prime attributes of the input relation. **Atomize** (eq. 3) takes a relation with a non-atomic attribute and produces a new relation by first partitioning the non-atomic attribute and then unnesting its atomic values. **Domainify** (eq. 4) produces a new domain of values (or distinct term set) from an attribute of a relation by projecting the target attribute (a_i), deduplicating its values, and renaming the attribute. **Canonicalize** (eq. 5) extends from the domainify operator by producing a simple vocabulary defined as (term, [synonym...]) pairs by deduplicating the original term values yet retaining them as synonyms of the canonical term. **Align** (eq. 6) replaces the values of a (possibly, loosely-constrained) attribute of a relation with values from a controlled domain (i.e., a dictionary). **Tagify** (eq. 7) transforms a relation r containing a (potentially, non-atomic) attribute of values into a normalized set of values aligned to a dictionary d.

4 USER STORY: BIOASSAY DATABASE

Here we return to the motivating example introduced in Section 1. We use it to illustrate the utility of the framework which embodies lessons from real experiences with multiple laboratories facing similar issues working with scientific databases [2].

Decomposing biosamples from the bioassay table. Recalling the issues from Section 1, combining the biosample details in the bioassay table led to redundant data entry, thus wasted human effort, and potential for data entry errors. They use the *decompose* operation to create a distinct relation out of the unique biosamples originally embedded (and duplicated) in the bioassay table with the following statements.

```
# let 'c' be a data catalog
ba = c.bioassay # get table in catalog c
c.biosample = ba.decompose(ba.sid, ba.age)
ba.age.drop()
c.commit() # evaluate expressions
```

The partitioned biosample relation is shown in Table 3. The restructuring improves the cohesiveness of the database, reduces data entry and therefore potential for human error, and allows them to find all of the assays conducted on each biosample.

Table 3: Decomposed `biosample` table.

sid	age	...
V14	E14.5	...
V11	E11.5	...
...	...	...

Normalizing lists of genes. Next, they realize that they also have difficulty searching for bioassays based on a particular gene name. The "nearest genes" relative to a transcription factor binding site are stored as a list in the `genes` column. This makes it difficult to find, manipulate and to remedy inconsistencies. They decide to *atomize* the `genes` column into its own table to make explicit the multi-valued cardinality of the field as it relates to bioassays.

```
ba = c.bioassay # get table in catalog c
c.nearest_genes = ba.genes.atomize()
ba.genes.drop()
c.commit() # evaluate expressions
```

The new relation is illustrated in Table 4. Transforming the nested values into their own distinct table also reveals errors in data entry (i.e., misspellings, duplicate entries), which can now be addressed through subsequent *domainify* and *align* operations (not illustrated).

Table 4: Atomized `nearest_genes` table.

bioassay_id	gene
A3	BET1
A3	COL1A
A4	Mitf
...	...

Define domain tables. Later, they notice that searching for "micro-CT" and other terms in the dataset returns less results than they believe should be in the database and discover that the term has been spelled five different ways due to misspellings and commonly accepted variations on the terminology (e.g., "micro-CT", "microCT", "Micro computed tomography") because the bioassay `type` column allowed free text entry. To remedy this, they create a table of bioassay type terms (commonly called a "domain table"), populate it with the distinct terms found in the `type` column of the bioassay table and de-duplicate the terms. To define new domain tables from existing column values the user invokes the following commands.

```
c.genes = c.nearest_genes.gene.domain()
c.age = c.biosample.age.domain()
c.type = c.bioassay.type.domain()
c.commit() # evaluate expressions
```

The new "domain table" of assay types is illustrated in Table 5. Using this new domain of terms for assay types, they then *align* the bioassay `type` column with the cleansed terms (not illustrated).

Table 5: Bioassay `type` domain table.

term
total RNA-seq
ChIP-seq
micro-CT
...

As the project progresses, new requirements stipulate that they submit data and metadata to community repositories (e.g., dbGaP, GEO, etc.) and must map these locally coded terms to standard ontologies, such as those maintained by the National Center for Biomedical Ontologies (NCBO). They further transform the domain tables of local terms into tables that reflect the structure of controlled vocabularies, both improving the expressiveness of their database and streamlining the exchange of information.

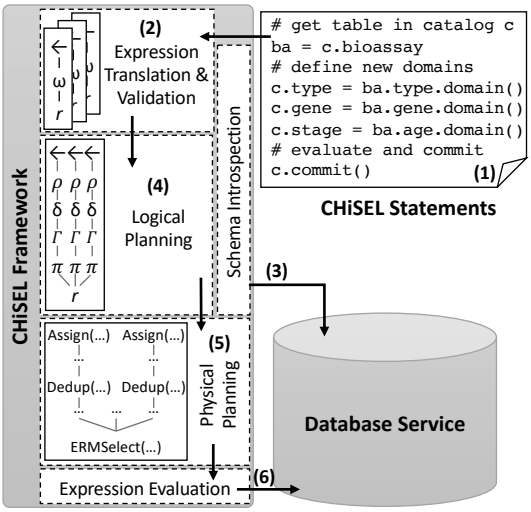

Figure 2: CHiSEL architecture.

5 SYSTEM DESIGN

Figure 2 presents the system design of the CHiSEL framework. **(1)** Users' schema evolution expressions are written using bindings in the Python programming language, which are **(2)** translated into an algebraic expression tree and validated against the source data schema by **(3)** introspecting the database. **(4)** The *logical planner* first *rewrites composite operators* according to their definitions (Section 3) then *consolidates redundant expressions* for work sharing. **(5)** The *physical planner* then rewrites the logical plan into a physical plan, where possible *pushing down* operations to the underlying database service rather than evaluating locally. **(6)** Finally, the *expression evaluator* executes the physical plan. The primary target of CHiSEL is the ERMrest metadata catalog of the Deriva asset management system [2], which tracks historical changes to the database so that users can view previous versions of data and schema.

6 FUTURE WORK

CHiSEL is in an early stage of active development and ongoing evaluation. We are conducting a rigorous review of schema evolution in scientific databases, both from our experiences and from public datasets, which we intend for future publications. While considerable evaluation has gone into the design of the algebra, we expect further refinement as additional requirements are surfaced. We are developing CHiSEL in the context of Deriva [2] and are working on a toolkit for schema evolution in the Python language. To scale our approach to large datasets, we intend to implement some of the physical operators (nest, unnest, similarity search) in map-reduce frameworks, which are well suited to the operations. Finally, we plan to validate the approach through performance evaluation on real [9] and synthetic datasets, and through usability studies involving science users [2] to evaluate the claims of improving the efficacy of evolving databases in real science applications.

7 CONCLUSION

Schema evolution remains one of the most difficult tasks in maintaining scientific databases. Here, we introduced CHiSEL, which offers higher-level operations to simplify complex schema evolution tasks for science users. Its design has been driven by recent surveys and experience with scientific applications. We described a user story that illustrates the utility of the framework for common scenarios as scientific datasets evolve through the course of active investigation. We described the system design, which benefits from the algebraic approach to providing higher-level operations, allowing it to evaluate and optimize user expressions. Finally, we outlined our plans for future work to take this early proposal to a fully functional system.

REFERENCES

[1] Philip A. Bernstein. 2003. Applying Model Management to Classical Meta Data Problems.. In *Proceedings of the 2003 CIDR Conference*. Asilomar, CA, USA, 209–220.

[2] Alejandro Bugacov, Karl Czajkowski, Carl Kesselman, Anoop Kumar, Robert Schuler, and Hongsuda Tangmunarunkit. 2017. Experiences with Deriva: An Asset Management Platform for Accelerating eScience. In *The IEEE 13th International Conference on eScience*. Auckland, New Zealand.

[3] Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. 2013. Automating the database schema evolution process. *VLDB Journal* 22, 1 (2013), 73–98. https://doi.org/10.1007/s00778-012-0302-x

[4] Carlo A Curino, Hyun J Moon, and Carlo Zaniolo. 2008. Graceful Database Schema Evolution: The PRISM Workbench. *Proc. VLDB Endow.* 1, 1 (aug 2008), 761–772. https://doi.org/10.14778/1453856.1453939

[5] Stella Giannakopoulou, Manos Karpathiotakis, Benjamin Gaidioz, and Anastasia Ailamaki. 2017. CleanM: An Optimizable Query Language for Unified Scale-out Data Cleaning. *Proc. VLDB Endow.* 10, 11 (Aug. 2017), 1466–1477. https://doi.org/10.14778/3137628.3137654

[6] M. Gobert, J. Maes, A. Cleve, and J. Weber. 2013. Understanding Schema Evolution as a Basis for Database Reengineering. In *2013 IEEE International Conference on Software Maintenance*. 472–475. https://doi.org/10.1109/ICSM.2013.75

[7] Michael Hartung, James Terwilliger, and Erhard Rahm. 2011. Recent Advances in Schema and Ontology Evolution. In *Schema Matching and Mapping*, Zohra Bellahsene, Angela Bonifati, and Erhard Rahm (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter Recent Adv, 149–190. https://doi.org/10.1007/978-3-642-16518-4_6

[8] Kai Herrmann, Hannes Voigt, Andreas Behrend, and Wolfgang Lehner. 2015. CoDEL – A Relationally Complete Language for Database Evolution. In *Advances in Databases and Information Systems*, Morzy Tadeusz, Patrick Valduriez, and Ladjel Bellatreche (Eds.). Springer International Publishing, Cham, 63–76.

[9] Shrainik Jain, Dominik Moritz, Daniel Halperin, Bill Howe, and Ed Lazowska. 2016. SQLShare: Results from a Multi-Year SQL-as-a-Service Experiment. In *SIGMOD'16*. ACM, San Francisco, CA, USA. https://doi.org/10.1145/2882903.2882957

[10] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive Visual Specification of Data Transofmration Scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '11*. ACM Press, New York, New York, USA, 3363–3372. https://doi.org/10.1145/1978942.1979444

[11] Sean Kandel, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. 2012. Enterprise data analysis and visualization: An interview study. *Visualization and Computer Graphics, IEEE Transactions on* 18 (2012), 2917–2926.

[12] Sergey Melnik, Philip A Bernstein, Alon Halevy, and Erhard Rahm. 2005. Supporting Executable Mappings in Model Management. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD '05)*. ACM, New York, NY, USA, 167–178. https://doi.org/10.1145/1066157.1066177

[13] John F Roddick. 1995. A survey of schema versioning issues for database systems. *Information and Software Technology* 37, 7 (jan 1995), 383–393. https://doi.org/10.1016/0950-5849(95)91494-K

[14] James F Terwilliger, Philip A Bernstein, and Adi Unnithan. 2010. Worry-free Database Upgrades: Automated Model-driven Evolution of Schemas and Complex Mappings. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*. ACM, New York, NY, USA, 1191–1194. https://doi.org/10.1145/1807167.1807316