Demonstration of a Multiresolution Schema Mapping System

Zhongjun Jin Christopher Baik Michael Cafarella H. V. Jagadish Yuze Lou
University of Michigan, Ann Arbor
{markjin,cjbaik,michjc,jag,lyzlyz}@umich.edu

ABSTRACT

Enterprise databases usually contain large and complex schemas. Authoring complete schema mapping queries in this case requires deep knowledge about the source database schemas and the target schema and is thereby very challenging to programmers. Sample-driven schema mapping allows the user to describe the schema mapping using data records. However, real data records are still harder to specify than other useful insights about the desired schema mapping the user might have. In this project, we develop a schema mapping system, Beaver, that allows *multiresolution schema mapping*: the end user is not limited to providing high-resolution constraints like exact and complete target schema data examples but may also provide other constraints of various resolutions like incomplete or ambiguous target schema data examples, data types or value ranges of columns in the target schema. This new interaction paradigm gives the user more flexibility in describing the desired schema mapping. This demonstration showcases how to use Beaver for schema mapping in a real database.

KEYWORDS

Schema mapping, exploratory search, non-expert users, program synthesis, SQL query

ACM Reference Format:
Zhongjun Jin Christopher Baik Michael Cafarella H. V. Jagadish Yuze Lou. 2019. Demonstration of a Multiresolution Schema Mapping System. In *Proceedings of 8th Biennial Conference on Innovative Data Systems Research (CIDR'19).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3209900.3209902

1 INTRODUCTION

Schema mapping is the problem of discovering queries that convert data from source databases with different schemas to a *target schema*, i.e., the schema that is expected by the end user. In real-world complex databases, composing schema mapping queries is challenging because it requires deep understanding of the database schema and the target schema. Previous works [1, 3, 5–8] have adopted a *sample-driven* approach to lower the requirement for the end user in schema mapping: the user can demonstrate the desired schema mapping process by providing a few data records in the target schema without being familiar with the source database schemas. However, we argue that the sample-driven schema mapping approach has two practical issues:

(1) *Hi-resolution issue.* The user is required to provide *high-resolution constraints*, i.e., complete data records with exact values in the target schema. Providing exact values can be challenging for a user unfamiliar with the database content.
(2) *Low-expressivity issue.* The user often has insights on the desired database schema other than target schema data examples. Existing methods lack mechanisms for capturing these kinds of knowledge.

State	Lake Name	Area (km^2)
California	Lake Tahoe	497
Oregon	Crater Lake	53.2
Florida	Fort Peck Lake	981
…		

Table 1: Desired target schema

```
SELECT
  geo_Lake.Province,
  Lake.Name,
  Lake.Area
FROM
  geo_Lake,
  Lake
WHERE
  geo_Lake.Lake = Lake.Name
```

Figure 1: Desired schema mapping query

Take Mondial—a relational geography dataset integrated from a number of data sources—as an example. Suppose the goal is to list all lakes, their area and the states they belong to as Table 1 from the Mondial database. The expected SQL query to obtain such a table is shown in Figure 1.

A sample-based schema mapping system, such as MWeaver [6], takes complete target schema data samples from the user and synthesizes schema mapping queries in the form of SPJ SQL queries. A person without the precise geographical knowledge might not be able to use the system because it is hard to specify complete data records in the desired schema, especially the area (*Hi-resolution issue*). However, this does not necessarily mean that the person has no insights to help discover the desired schema mapping query. For example, the person may know that "Lake Tahoe" is close to California and Nevada, so one of them must be part of the example. Also, even if the exact lake area of Lake Tahoe is beyond the user's knowledge, she may know that these values must be at least numeric and positive. Although such marginal knowledge should still be useful in narrowing down the search space of possible queries, existing systems cannot use them (*Low-expressivity issue*).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIDR'19, Jan 13-16, 2019, Asilomar, California, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-5827-9/18/06…$15.00
https://doi.org/10.1145/3209900.3209902

$$
\begin{aligned}
\text{Value Constraint } c_k &:= p_v \mid p_v \vee p_v \mid \epsilon \\
\text{Metadata Constraint } c_m &:= p_m \mid p_m \; logicalop \; p_m \mid \epsilon \\
logicalop &:= \; \wedge \; \mid \vee \\
\text{Value Predicate } p_v &:= binop \; const \\
\text{Metadata Predicate } p_m &:= type \; binop \; const \\
\text{Metadata Type } type &:= \text{DataType} \mid \text{ColumnName} \mid \\
&\quad \text{MaxValue} \mid \text{MinValue} \\
binop &:= \; > \mid \geq \mid < \mid \leq \mid = \mid \neq
\end{aligned}
$$

Figure 2: Multiresolution schema mapping language

Our Approach — To address the above limitations of sample-driven schema mapping, we developed Beaver[1], a *multiresolution schema mapping* system that can discover schema mapping queries employing user insights of various resolutions.

Multiresolution schema mapping is a schema mapping process using a novel interaction model which increases the scope of descriptions the end user can provide. The model is empowered by a *multiresolution schema mapping description language* enriched to support constraints of various resolutions: (1) high resolution: complete sample constraints with precise data values, (2) medium resolution: incomplete sample constraints with approximate data values (a set of possible data values, value ranges), (3) low resolution: column-level descriptions like data type, value range or even user-defined functions.

Once the user provides a *multiresolution query* in the proposed language, we synthesize the desired schema mapping query matching this query. A major technical challenge is to ensure the program search process is efficient enough to be interactive. The search space of all schema mappings is inherently massive; it is exponential in the complexity of the desired schema mapping and the source database schema. Moreover, the number of satisfying solutions can be relatively large because the types of constraints we support are more relaxed than those ingested by the sample-driven approach. As a result, performing a fast search for a complete solution set in our case is difficult. Another challenge is that displaying SQL queries as the output result of a schema mapping system may be difficult to understand for many non-expert users. We propose an interactive approach using visualizations to make the synthesized queries more explainable.

In Section 2, we present the design of Beaver. We demonstrate how to use Beaver in Section 3.

2 SYSTEM OVERVIEW

2.1 Multiresolution Schema Mapping

User Input — As enterprise databases today are usually large and complex, users might not have deep understanding about the database schema or precise knowledge about the database content. Therefore, it is difficult for a user to author a schema mapping query or provide even a few data examples in the target schema. However, the user might still have some insights that are useful in narrowing down the search space of possible desired schema mapping queries.

[1]the url to the project will be released upon publication

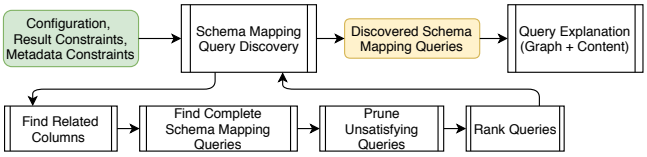

Figure 3: Beaver architecture

Figure 4: Specify constraints for the desired schema mapping

To allow users to comfortably express their insights, we propose a **Multiresolution Schema Mapping Language** (Figure 2), composed of two kinds of constraints the users can specify: at the row level, *target schema result constraints* ("result constraints" for short) and, at the column level, *target schema metadata constraints* ("metadata constraints" for short). We also allow the user to add logical operators "AND" and "OR" between constraint values.

A *result constraint* is a set of *sample constraints*, which are composed of a sequence of value constraints.

(1) ***Value Constraints***. A value constraint means that a tuple in the target schema must contain a given keyword. Unlike the value constraints handled by traditional schema mapping systems, the user may also specify a disjunction of possible values or a value range.
(2) ***Sample Constraints***. Multiple value constraints listed in the same row together form a sample constraint. A schema mapping query satisfies a sample constraint if the result set of the query contains this sample. Such constraints are also handled by other sample-driven schema mapping systems [6, 7].

A *metadata constraint* represents factual knowledge about individual columns in the source database. Currently, the kinds of metadata we support in Beaver are data type (including *decimal*, *int*, *text*, *date*, *time*), maximum text length, and value range. In the future, we plan to support more metadata constraints, and even user-defined functions. Metadata constraints are allowed to be "ambiguous" too: the user could specify a conjunction or disjunction of multiple metadata constraints for one column.

Problem Definition — Given a multiresolution schema mapping query Q (or "multiresolution query" for short) in the proposed language, and a database $\mathcal{D}$, the multiresolution schema mapping problem is synthesizing the schema mapping specification, $\mathcal{M}$, such that $\mathcal{M}$ and the resulting target schema $\mathcal{M}(\mathcal{D})$ satisfy all the constraints in Q.

System Output — To focus on the problem without loss of generality, we restrict the space of synthesized schema mapping queries to support Select-Project-Join (SPJ) queries.

2.2 System Architecture

Figure 3 shows Beaver's architecture and user interaction workflow. Beaver provides users with a web-based graphical interface with three major sections: **Configuration**, **Description** and **Result**.

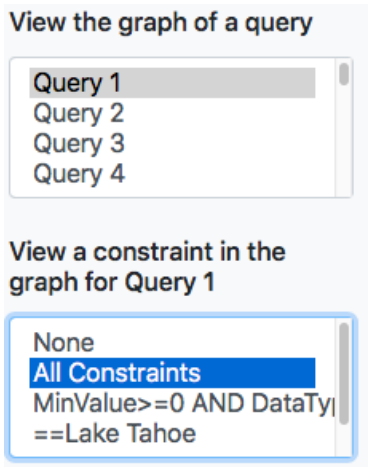

Figure 5: Choose to view a synthesized SQL query and its graph

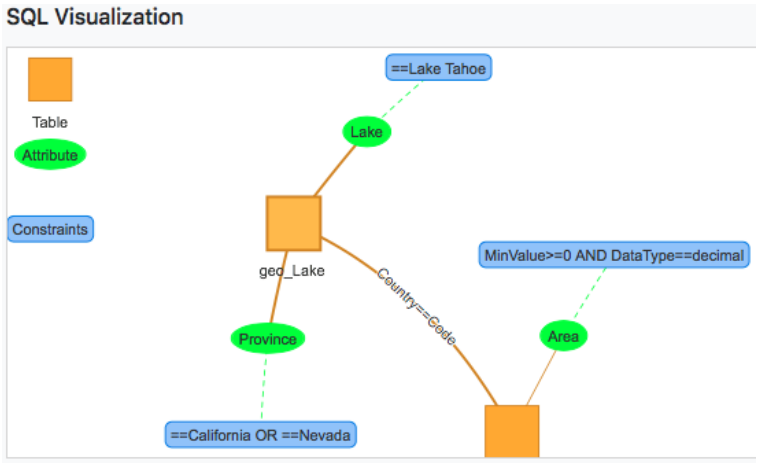

Figure 6: The chosen SQL query graph and all constraints

Initially, in the **Configuration** section, the user tunes the system for the schema mapping task. Current configurations include the source database, number of columns in the target schema, number of sample constraints, and whether metadata constraints are specified.

Next, the user specifies a multiresolution query, including result constraints and metadata constraints, to describe the desired schema mapping. The **Description** section (Figure 4) has two regions to take in these constraints.

Once our system obtains the multiresolution query, it executes the algorithm introduced in Section 2.3 to initiate a discovery for the desired schema mapping query. In our system, we set a 60-second time limit for each round of query discovery. If Beaver successfully finds a set of schema mapping queries, they are displayed in the **Result** section. If Beaver encounters a timeout, it reports a failure. A synthesized schema mapping query and its results are guaranteed to match the constraints the user initially provided.

If multiple satisfying schema mapping queries are discovered, the user needs to understand each query and pick the one that is desired. To help the user comprehend each query, in the Results section, Beaver creates a visualization to explain any discovered schema mapping query the user selects (as Figure 6). We discuss this in more detail in Section 2.3.

2.3 Our Approach

Query Discovery — Inspired by [6, 7], we designed an algorithm discovering the satisfying schema mapping queries for the proposed multiresolution schema mapping, which consists of four phases: (1) find related columns, (2) discover complete schema mapping queries, and (3) pruning, (4) ranking.

Initially, we identify *related columns*—columns in the database potentially used in the schema mapping—so that the search scope for potential schema mapping queries is limited within this small set of columns and tables, and hence, the search space can be significantly reduced. In multiresolution schema mapping, finding related columns is essentially finding columns in the database matching at least one constraint in the multiresolution query the user provides. Checking a metadata constraint against a column is simple and fast because all column metadata information can be calculated prior to any user interaction. Verifying a result constraint is less straightforward because yielding a user-specified sample usually requires joining multiple relations in the database which can be very expensive given the size and complexity of the database. Instead, we only check value constraints on individual columns here, which is much cheaper when we leverage the inverted index in the database.

With related columns discovered, we build up *candidate* schema mapping queries using a search-based approach, which gradually finds longer join paths (SPJ queries) covering more constraints. A candidate schema mapping query is *complete* when all value constraints and metadata constraints are satisfied.

The set of candidate schema mapping queries is a superset of the *final* schema mapping queries, i.e., schema mapping queries matching all user constraints; some of them only match value constraints and metadata constraints but not sample constraints. We must prune those that fail to produce any sample constraint in the user query. A naïve approach is to sequentially check each candidate schema mapping query against all sample constraints. However, as this requires running the entire join query on the database, this naïve approach can be very expensive. Inspired by [7], we divide such an expensive verification task into smaller chunks, noted as *filters*. Verifying a filter is cheaper than validating the entire candidate schema mapping specification. More importantly, if the verification of a filter fails, the entire candidate schema mapping specification fails. This gives us an opportunity to quickly prune invalidate candidate schema mapping queries. For now, we follow a practice of filter selection and evaluation similar to [7]. Developing a new filter selection strategy to improve the efficiency is in our future work. In the end, we rank the remaining schema mapping queries by *simplicity* using the *Occam's razor principle* (the simplest explanation is usually correct). Designing better ranking strategies is also in our future work.

Query Explanation — During schema mapping, if multiple schema mapping queries are found satisfying the user's constraints, the user needs to understand each query and choose one that is desired.

Previous schema mapping systems made little effort to help the user understand the query, and hence, making it difficult to choose the right query. In addition to showing the actual generated SQL queries, we explain the discovered schema mapping queries using visualizations in Beaver.

Whenever the user points to a schema mapping SQL query (top of Figure 5), we draw a corresponding query graph representation for this query (Figure 6). Orange squares represent relations, green ellipses are the attributes to project, and edges represent join conditions. To help the user understand why a given query matches all the constraints she provides, the user could pick one or more constraints (bottom of Figure 5), and Beaver draws these constraints (as blue boxes) in the previous graph to show the locations in the database where these constraints are satisfied.

2.4 System Evaluation

We evaluated the system efficiency of Beaver on both a sample-driven schema mapping workload and a multiresolution schema mapping workload on a public relational database Mondial [4]. Multiresolution queries are likely to be more computationally challenging than sample-driven queries, but should require less work from the user. Our goal is for multiresolution queries to execute as fast as sample-driven queries on Beaver.

In our experiment, we simulated an end user who provided queries we synthesized. The sample-driven queries for testing were randomly synthesized using the Mondial database in the following two steps: (1) we randomly generated target schema mapping queries of different join lengths using the database schema of Mondial, (2) then, we randomly generated sample-driven queries by querying the database using the target schema mapping queries generated in the last step. We observed that, when the join length is one to three, Beaver is able to complete within 1 second on average. The results suggest that Beaver can handle traditional sample-driven schema mapping workloads in interactive time.

To make the multiresolution query tests comparable to sample-driven query tests, we synthesized the multiresolution workload using the sample-driven queries generated previously: given a sample-driven query, we randomly chose various numbers of cell values of constant strings, and replaced the values in the chosen cells with random disjunctive values, value ranges, or metadata constraints.

Beaverś efficiency in handling multiresolution queries with disjunctive values or value ranges was comparable to that of executing sample-driven queries. For one specific case (metadata constraints), queries were around 5 seconds slower, which we still consider within an acceptable threshold for our interface.

Our observation suggests that the end user can provide multiresolution queries, which we believe are easier to input, and retain a similar system efficiency as for traditional sample-driven queries. More details about the experiments are presented in [2]. In the future, we plan to conduct user studies to evaluate the usability of Beaver compared to other schema mapping systems.

3 DEMONSTRATION

Our demonstration aims to show the conference attendees that our proposed system, Beaver, is able to help a naïve user synthesize schema mapping queries with multiresolution queries.

We use the Mondial relational database mentioned in Section 1 and two other databases, IMDB and DBLP as the source databases the user can interact with. The user can choose from a set of suggested target schemas or come up with her own. Then, the user is free to provide any multiresolution queries she can come up with to constrain the desired schema mapping process. In the end, Beaver will show all satisfying schema mapping queries discovered and present visualizations to explain the query the user selects, as discussed in Section 2.3.

To best illustrate how to use our tool, we will use the motivating example from Section 1, where a user would do the following[2]:

(1) In the **Configuration** section, 1) choose "**Mondial**" as the source database from the supported databases, 2) set the number of columns in the target schema as "3", 3) set the number of sample constraints as "1", 4) confirm to specify metadata constraints.
(2) Specify the multiresolution query to describe the desired schema mapping in the **Description** section.
  2.1. Type "California || Nevada" in the first cell in the Sample Constraint field.
  2.2. Type "Lake Tahoe" in the second cell in the Result Constraints field.
  2.3. Type "DataType=='decimal"' AND MinValue>='0"' in the third cell in the Metadata Constraints field.
(3) Hit the "Start Searching!" button.
(4) In the **Result** section, Beaver shows a list of satisfying schema mapping queries. View the queries and pick the one that is correct.
  4.1. Select the first synthesized query in the top field in Figure 5. The SQL query is shown as Figure 1.
  4.2. The system draws a graph.
  4.3. Interaction: choose the constraints in the bottom field in Figure 5 to show in the visualization (Figure 6 shows a SQL graph with all constraints user provided). This helps the user understand why the selected query matches the constraints she provided.
  4.4. If the selected query is not desired, repeat the above process for the second query.

4 CONCLUSION

Our demonstration shows that the proposed multiresolution schema mapping system, Beaver, makes schema mapping in a large complex database easy for non-expert users. To build a schema mapping SQL query generating the target schema in a large and complex database, the user may specify constraints of various resolutions, such as disjunctions of values, value range, and even column-level metadata constraints which presumably requires less domain expertise than the high-resolution constraints, i.e., exact data samples. We will continue to develop this system to achieve our vision for a schema mapping system for non-expert users.

REFERENCES

[1] Angela Bonifati, Ugo Comignani, Emmanuel Coquery, and Romuald Thion. Interactive mapping specification with exemplar tuples. In *SIGMOD*, 2017.

[2] https://markjin1990.github.io/assets/doc/beaver-demo.mp4

[2] Zhongjun Jin, Christopher Baik, Michael Cafarella, and H. V. Jagadish. Beaver: Towards a declarative schema mapping. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA'18, pages 10:1–10:4, New York, NY, USA, 2018. ACM.
[3] Dmitri V Kalashnikov, Laks VS Lakshmanan, and Divesh Srivastava. Fastqre: Fast query reverse engineering. In *SIGMOD*, 2018.
[4] Wolfgang May. Information extraction and integration with Florid: The Mondial case study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999. Available from http://dbis.informatik.uni-goettingen.de/Mondial.
[5] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. Exemplar queries: Give me an example of what you need. In *PVLDB*, 2014.
[6] Li Qian, Michael J Cafarella, and HV Jagadish. Sample-driven schema mapping. In *SIGMOD*, 2012.
[7] Yanyan Shen, Kaushik Chakrabarti, Surajit Chaudhuri, Bolin Ding, and Lev Novik. Discovering queries based on example tuples. In *SIGMOD*, 2014.
[8] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. Synthesizing highly expressive sql queries from input-output examples. In *PLDI*, 2017.