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 and target schemas 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, PRISM, that enables *multiresolution schema mapping*. The end user is not limited to providing high-resolution constraints like exact data records but may also provide constraints of various resolutions, like incomplete data records, value ranges, and data types. This new interaction paradigm gives the user more flexibility in describing the desired schema mapping. This demonstration showcases how to use PRISM for schema mapping in a real database.

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 expected by the end user. In real-world complex databases, composing schema mapping queries is challenging because it requires a deep understanding of the source database schema and the target schema. Previous works [7, 8, 9, 6, 1, 4] have adopted a *sample-driven* approach to simplify this process for end users: 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 schema. However, we argue that the sample-driven schema mapping approach has two practical challenges:

1. *High-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. For example, the user might know the area of Take Tahoe roughly but cannot provide an exact value.
2. *Low-expressivity issue.* The user may have insights on the target schema other than data examples, like column names, data types. 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

Take MONDIAL—a relational geography data set—as an example. Suppose the goal is to list all lakes, their area and the states they belong to from the MONDIAL database as in Table 1. The desired SQL query to obtain such a table is "`SELECT geo_lake.Province, Lake.Name, Lake.Area FROM Lake, geo_lake WHERE Lake.Name = geo_lake.Lake`".

A sample-based schema mapping system, such as MWEAVER [7], takes complete target schema data samples from the user and synthesizes schema mapping queries in the form of Project-Join (PJ) 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 (*High-resolution issue*). However, this does not necessarily mean that the person has no insight 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*).

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

Multiresolution schema mapping is a schema mapping process with a novel interaction model which increases the scope of descriptions the end user can provide. The model is empowered by a 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.

[1]Available at https://github.com/umich-dbgroup/prism

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

Figure 1: Multiresolution schema mapping language

Once the user provides *multiresolution constraints* in the proposed language, we synthesize the desired schema mapping query matching these constraints. A major technical challenge is to ensure that 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, for many non-expert users, displaying SQL queries as the output result of a schema mapping system may be difficult to understand. We propose an interactive approach using visualizations to make the synthesized queries more explainable.

In Section 2, we present the design of PRISM. We demonstrate how to use PRISM 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 source database schema or precise knowledge about the database content. In this case, 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 space of possible desired schema mapping queries.

To allow users to comfortably express their insights, extending our previous work [3], we propose a **Multiresolution Schema Mapping Language** (Figure 1), composed of two classes 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 requires a tuple in the target schema to 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 [7, 8].

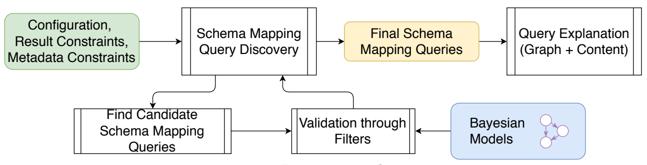
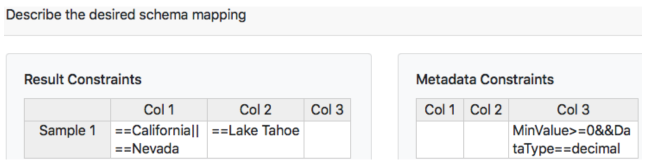

Configuration, Result Constraints, Metadata Constraints
Schema Mapping Query Discovery
Final Schema Mapping Queries
Query Explanation (Graph + Content)
Find Candidate Schema Mapping Queries
Validation through Filters
Bayesian Models

Figure 2: PRISM architecture



Describe the desired schema mapping

Result Constraints

	Col 1	Col 2	Col 3
Sample 1	==California|| ==Nevada	==Lake Tahoe	

Metadata Constraints

Col 1	Col 2	Col 3
		MinValue>=0&&DataType==decimal

Figure 3: Specify constraints for the desired schema mapping (Description Sec.)

A *metadata constraint* represents factual knowledge about individual columns in the source database. Currently, the kinds of metadata we support in PRISM 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 set of multiresolution schema mapping constraints $\mathcal{Q}$ (or "multiresolution constraints" for short) in the proposed language and a database $\mathcal{D}$, the problem is to synthesize a schema mapping query, $\mathcal{M}$, so that $\mathcal{M}$ and the query result $\mathcal{M}(\mathcal{D})$ satisfy all constraints in $\mathcal{Q}$.

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

2.2 System Architecture

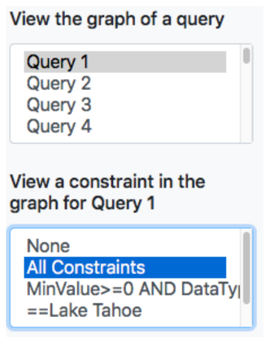
Figure 2 shows PRISM's architecture and user interaction workflow. PRISM provides users with a web-based graphical interface with three major sections: **Configuration**, **Description** and **Result**.

Initially, in the **Configuration** section, the user sets up 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 set of multiresolution constraints, including result constraints and metadata constraints, to describe the desired schema mapping. The **Description** section (Figure 3) has two regions to take in these constraints.

Once our system obtains the multiresolution constraints, it executes the algorithm introduced in Section 2.3 to initiate a search for the desired schema mapping query. In our system, we set a 60-second time limit for each round of query discovery. If PRISM successfully finds a set of schema mapping queries, they are displayed in the **Result** section (Figure 4). If PRISM 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,



View the graph of a query

Query 1
Query 2
Query 3
Query 4

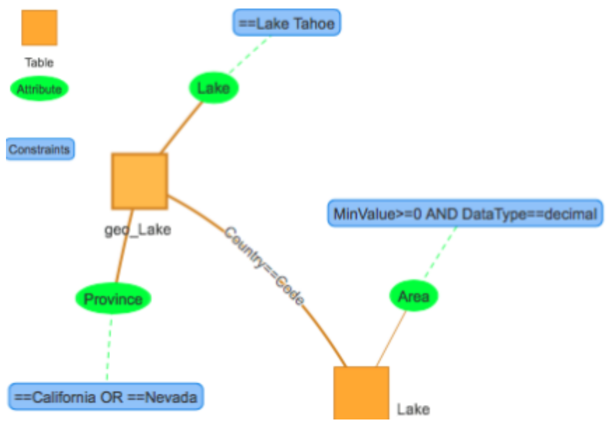
View a constraint in the graph for Query 1

None
All Constraints
MinValue>=0 AND DataTy|
==Lake Tahoe

(a) Choose to view a synthesized SQL query and its graph

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

(b) Schema mapping query content



(c) The chosen SQL query graph and all constraints

Figure 4: Result Section: show the returned set of schema mapping queries and their graphs

in the Result section, PRISM creates a visualization to explain any discovered schema mapping query the user selects (as Figure 4c). We discuss this in more detail in Section 2.3.

2.3 Our Approach

Query Discovery — Like [7, 8], we split our schema mapping query discovery into of two steps: (#1) discovering candidate complete schema mapping queries, and (#2) validating candidate schema mapping queries.

The first step, discovering the candidate complete schema mapping queries, is relatively straightforward. First, 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 our setting, finding related columns is essentially finding columns in the database matching at least a value constraint or metadata constraint. Validating sample constraints requires executing expensive join queries on the database and is done in Step 2.

The way we validate a value constraint on a column is same as that in [7, 8]: leveraging the inverted index provided in most DBMS systems. To check a metadata constraint, we use metadata information, e.g., min/max values, collected during preprocessing. With related columns found, we exhaustively search through the source database schema graph and find all possible join paths, each connecting a set of related columns that altogether can be mapped to all columns in the target schema. Every join path along with the set of related columns it connects becomes a *candidate* schema mapping query (in form of a PJ query). Note that these candidate schema mapping queries are not final; we have never executed these queries and checked if their query results match the sample constraints.

In Step 2, a naïve solution to validating all candidate queries is to execute them one by one on the source database and check their query results, which can be very expensive. In our project, we divide such an expensive verification task into a set of cheap validations of *filters*, i.e. sub(join)trees along with projected attributes (shorter PJ queries), inspired by [8]. If a filter fails, its parent filters and entire candidate schema mapping query, from which the filter is derived, automatically fail, and thereby pruned. This gives us an opportunity to replace expensive validations of complex schema mapping queries with cheaper validations of filters.

Although validating filters instead of schema mapping queries saves time, it is still a relatively expensive process. A new important issue becomes the filter validation scheduling: in what order the filters are validated so that the most number of filters are pruned, as well as overall filter validation time is minimized.

A filter scheduling algorithm should naturally consider two important aspects of a filter: pruning power and cost. Estimating the cost of a filter is essentially estimating the cost of executing a SQL query on a database, which is known to be very challenging because the actual cost can be affected by many database tuning parameters, and is beyond the scope of our project. We focus on improving the estimation of pruning power of the filter.

The pruning power of a filter depends on two things: filter dependency and filter failure (success) probability. While dependency relationships among a set of filters is fixed and can be easily captured, estimating the failure probability of a filter is a bit more tricky. Instead of setting the failure probability proportional to the join path length of a filter [8], we take a machine learning approach: we estimate the filter probability using Bayesian models trained a priori for the source database. A Bayesian model is able to give an estimated probability of a certain record matching the sample constraint exists. With this probability and the relation size information, we can obtain a rough estimation of the failure probability good enough to boost our filter scheduling. While learning a Bayesian model in a single relation is no different from learning a model for a data set, learning a model capturing the correlations among multiple relations is more difficult. This problem is solved by using the join indicator introduced by Getoor et al. in [2]. Details about this idea will be discussed in our future paper.

In the end, we return all final schema mapping queries and let the user choose the desired one.

Query Explanation — In PRISM, we go beyond simply showing the actual generated SQL queries; we explain the discovered schema mapping queries using visualizations.

Whenever the user points to a schema mapping SQL query (top of Figure 4a), we draw a corresponding query graph representation for this query (Figure 4c). 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 4a), and PRISM 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 compared PRISM with FILTER on a set of synthesized test cases created from a public relational database MONDIAL [5]. In summary, we observed that the overall execution time of user constraints did not grow significantly as user constraints became loose (containing constraints with disjunctions, value ranges, etc.). Meanwhile, the number of satisfying schema mapping queries discovered did not increase much (unless when there were too many missing values). All these evidences suggest that, PRISM not only requires less user knowledge, it does not increase the user interaction effort in schema mapping. Also, our approach significantly reduced the gap of the required number of filter validations between FILTER and the optimum (up to $\sim 70\%$; on average $\sim 30\%$), which shows our Bayesian-model-based approach can effectively improve the filter scheduling. This section will be discussed in more details in our future paper.

3. DEMONSTRATION

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

We use the MONDIAL data set mentioned in Section 1 and two other data sets, IMDB and NBA 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 constraint she can come up with to constrain the desired schema mapping process. In the end, PRISM 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 constraints 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, PRISM 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 4a. The SQL query is shown as Figure 4b.
   - 4.2. The system draws a graph.
   - 4.3. Interaction: choose the constraints in the bottom field in Figure 4a to show in the visualization (Figure 4c 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.

[2]https://markjin1990.github.io/assets/video/prism.mp4

4. CONCLUSION

Our demonstration shows that the proposed multiresolution schema mapping system, PRISM, 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 ranges, and even column-level metadata constraints which presumably require less domain expertise than 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.

5. ACKNOWLEDGMENTS

This project is supported in part by by NSF grants IIS-1250880, IIS-1054913, NSF IGERT grant 0903629, a Sloan Research Fellowship, a CSE Dept. Fellowship, and a University of Michigan MIDAS grant.

REFERENCES

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

[2] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In *ACM SIGMOD*, 2001.

[3] 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.*

[4] Dmitri V Kalashnikov, Laks VS Lakshmanan, and Divesh Srivastava. Fastqre: Fast query reverse engineering. In *SIGMOD*, 2018.

[5] 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.

[6] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. Exemplar queries: Give me an example of what you need. In *PVLDB*, 2014.

[7] Li Qian, Michael J Cafarella, and HV Jagadish. Sample-driven schema mapping. In *SIGMOD*, 2012.

[8] Yanyan Shen, Kaushik Chakrabarti, Surajit Chaudhuri, Bolin Ding, and Lev Novik. Discovering queries based on example tuples. In *SIGMOD*, 2014.

[9] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. Synthesizing highly expressive sql queries from input-output examples. In *PLDI*, 2017.