Beaver: Towards a Declarative Schema Mapping

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

ABSTRACT

Schema mapping is used to transform data to a desired schema from data sources with different schemas. Manually writing complete schema mapping specifications requires a deep understanding of the source and target schemas, which can be burdensome for the user. Programming By Example (PBE) schema mapping methods allow the user to describe the schema mapping using data records. However, real data records are still harder to specify compared to other useful insights about the desired schema mapping the user might have. In this project, we develop a new schema mapping technique, Beaver, that enables an interaction model that gives the user more flexibility in describing the desired schema mapping. The end user is not limited to providing exact and complete target schema data examples but may also provide incomplete or ambiguous examples. Moreover, the user can provide other types of descriptions, like data type or value range, about the target schema. We design an explore-and-verify search-based algorithm to efficiently discover all satisfying schema mapping specifications. We implemented a prototype of our schema mapping technique and experimentally evaluated the efficiency of the system in handling traditional PBE schema mapping test cases, as well as our newly-proposed declarative schema mapping test cases. The experiment results show that the declarative queries, which we believe are easier for non-expert user to input, often cost around zero to five seconds more than the traditional PBE queries. This suggests we retain a system efficiency comparable to traditional PBE schema mapping systems.

ACM Reference Format:
Zhongjun Jin   Christopher Baik   Michael Cafarella   H. V. Jagadish. 2018. Beaver: Towards a Declarative Schema Mapping. In *HILDA'18: Workshop on Human-In-the-Loop Data Analytics, June 10, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3209900.3209902

1 INTRODUCTION

Schema mapping is used to transform data to a desired schema from data sources with different schemas. Manually writing schema mapping specifications requires a deep understanding of the source and target schema. Programming By Example (PBE) has been adopted as a human-in-the-loop interaction model to reduce the requirement for the user expertise in schema mapping [2, 7–10]. The user only has to provide example data records in the target schema and does not need familiarity with the source schema. Yet, the user can still be stumped for the following two reasons:

(1) *Assumption of database content knowledge.* The user is required to provide complete records with exact values in the target schema. Providing exact values can be challenging for a user unfamiliar with the database content.
(2) *Limited expressiveness of user constraints.* The user often has insight on the desired database schema other than target schema examples. Existing methods have insufficient mechanism for capturing these descriptions.

Our Approach — In this paper, we present a new schema mapping method to address the above limitations of PBE schema mapping.

We propose a novel interaction model to increase the scope of descriptions the end user can provide to discover schema mapping specifications. The model is empowered by a declarative language enriched to support **1)** sample-based constraints with ambiguous or missing values, **2)** declarative constraints, like data type, value range or even user-defined functions, on individual data columns.

Given a user's declarative query, we synthesize the desired schema mapping specifications matching the query. The main technical challenge is to ensure the program search process is efficient enough for user interaction. The search space of all schema mappings is inherently massive; it is exponential to the complexity of the desired schema mapping and the source database schema. Moreover, the number of satisfying solutions can be relatively large because the type of constraints we support are more relaxed than traditional PBE constraints. Given this, performing a fast search for a complete solution set in our case is difficult.

Organization — In this project, we present a declarative schema mapping interaction model (Section 2). We provide a formal description of the declarative schema mapping problem, along with the user query (Section 3) and an efficient algorithm inferring the schema mapping specifications matching the user description (Section 4). We built a prototype system, called Beaver, and experimentally evaluated it using the Mondial dataset.(Section 5).

2 MOTIVATING EXAMPLE

Mondial is a relational geography dataset integrated from a number of data sources. A freshman student Ashley wants to list all lakes, their area and the states they belong to as Table 1 from the Mondial database.

A generic PBE schema mapping system, such as MWeaver [8], takes complete target schema data samples from the user and synthesizes schema mapping specifications in the form of SPJ SQL queries. To Ashley, such a system is unusable because she finds it hard to specify complete data records in the desired schema, the area in particular, as it is her first time using the dataset.

In contrast, to use Beaver, instead of a complete data record, Ashley provides a partial data record for this desired schema with

HILDA'18, June 10, 2018, Houston, TX, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5827-9/18/06…$15.00
https://doi.org/10.1145/3209900.3209902

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

Column Id	1	2	3
Sample	California OR Nevada	Lake Tahoe	
	...		
Metadata			{decimal} AND >'0'

Table 2: A declarative schema mapping query example.

a lake name "Lake Tahoe" in column 2. Ashley vaguely remembers "Lake Tahoe" is close to California and Nevada, but she is not sure which state it actual belongs to, she specifies a disjunction of "California" and "Nevada" in column 1. Ashley does not know the exact lake area of Lake Tahoe, but she at least knows that every lake area value in column 3 must be numeric and positive. To contribute this knowledge, Ashley creates a metadata constraint "{decimal} AND '> 0"' for column 3.

In a few seconds after Ashley inputs all these hints, BEAVER synthesizes the exact schema mapping specification (as a SQL script in Figure 1) that yields a relation containing state name and total lake area in each state.

```
SELECT geo_lake.Province,
  Lake.Name, Lake.Area
FROM Lake, geo_lake
WHERE Lake.Name = geo_lake.Lake;
```

Figure 1: Desired full schema mapping specification

3 OVERVIEW

Programming By Example (PBE) as a human-in-the-loop interaction model is frequently applied in domains [1–5, 7–10] where the user is required to have high domain expertise. Previous research works [2, 7–10] have proposed PBE-based techniques for schema mapping. These systems usually ask the user for complete data records in the desired schema, which can still be hard to specify if the user is not familiar with the database content.

In practice, while the user might not have accurate knowledge about the database content, it is reasonable to assume that the user intuitively knows some characteristics of the target schema or the relevant information to generate the target schema. To allow users to comfortably express their intuitions, we propose a **Declarative Schema Mapping Query** Q (or "declarative query" for short), composed of two kinds of constraints the users can specify: *sample constraints* and *metadata constraints*. We also let the user add logical operators "AND" and "OR" between constraint values.

Sample Constraint, C_{sample}. Like other PBE schema mapping systems [8, 9], we support *sample constraints*: the end user can provide one or more example data records from the target schema or the temporary schema.

The difference in our project is that we allow some level of "ambiguities" in the samples. The user can suggest several possible values (connected by the logic operator "OR"), or value range, if she has vague knowledge about certain cells in a sample constraint (e.g., column 1 in the sample constraint in Table 2), or simply NULL values if she knows nothing about some values in this sample record (e.g., column 3 in the sample constraint in Table 2).

Metadata Constraint, $C_{metadata}$. A *metadata constraint* denotes the 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. For instance, in Table 2, the user provides a constraint suggesting that the values in column 3 are positive decimals. In the future, we plan to support more metadata constraints, and even user-defined functions. Similarly, the metadata constraints are allowed to be "ambiguous" too: the user could specify multiple metadata constraints for one column as a conjunction or disjunction (e.g., the metadata constraint in column 3 of Table 2).

Problem Definition — We now formalize our problem definition.

Declarative Schema Mapping Problem. Given the declarative query $Q = (C_{sample}, C_{metadata})$ and a database $\mathcal{D}$, synthesize 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.

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

4 SCHEMA MAPPING ALGORITHMS

Inspired by [8, 9], we propose an explore-and-verify algorithm to efficiently discover all satisfying schema mapping specifications.

Find Related Columns — We first identify columns in the database potentially used in the schema mapping described by the user query and prune the ones violating the user's metadata and sample constraints. The space of possible schema mapping specifications is significantly reduced using this small set of related columns.

To evaluate if a column matches a metadata constraint, we query the database using a SQL query translated from the metadata constraint. For example, when the user provides a metadata constraint "column 3 must be positive numbers", we check the database for the minimum value of column 3 to see if it is greater than 0. Sample constraints describe the data in the target schema. Yet, they also imply that the relevant columns must contain certain keywords (i.e., in the motivating example, the sample constraint suggests that column 3 must contain the keywords "Lake" and "Tahoe"). We use such implicit "keyword constraints" derived from the sample constraints to identify the related columns as well.

Explore Candidate Schema Mapping Specifications — The above step discovers the potential columns in the database being projected in the desired target schema. But these columns usually reside in different relations in the database. Discovering how to join these relations (i.e., the join path) is the only uncertainty left to be resolved. Algorithm 1 shows the algorithm finding all *complete* join paths (a complete join path is a join path that yields all columns in the target schema). We first discover all *pairwise* join paths (line 2-4)–join paths covering two columns in the target schema–and use them to discover complete join paths (line 6-12).

To find all pairwise join paths, we enumerate all combinations of related columns in the database, and check if there is a pairwise join path between them. Discovering pairwise join paths between

Algorithm 1: Discover complete join paths

```
Data: Number of columns in the target schema $m$, set of
      related columns $\mathcal{R}[i]$ for each target schema column
      $i \in \{1, \ldots, m\}$
Result: Set of complete paths $C$
1  $\mathcal{P}, C \leftarrow \{\}$;
2  for $(i, j) \in \mathbb{N}^2 : 1 \le i < j \le m$ do
3      for $(c_i, c_j) : c_i \in \mathcal{R}[i], c_j \in \mathcal{R}[j]$ do
4          $\mathcal{P} \leftarrow \mathcal{P} \cup$ pairwise join paths between $c_i$ and $c_j$ ;
5  $Q \leftarrow \mathcal{P}$;
6  while $|Q| > 0$ do
7      $p \leftarrow$ pop $Q$;
8      if $p$ is a complete join path then
9          $C \leftarrow C \cup p$;
10     else
11         for $p' \in \mathcal{P}$ do
12             $Q \leftarrow Q \cup$ join $p$ with $p'$ if possible;
```

two columns in the database (line 4) is a pathfinding problem in the database schema graph, and a BFS algorithm can simply do the job.

With all pairwise join paths discovered, we propose an algorithm finding complete join paths—join paths that cover all columns in the target schema—inspired by path weaving in [8]: repeatedly combining pairwise join paths (p') with existing incomplete join paths p (line 12) until we find complete join paths (line 9).

Each of the discovered complete join paths is essentially a candidate schema mapping specification.

Verification — So far, the discovered candidate schema mapping specifications are guaranteed to satisfy all metadata constraints and "keyword constraints" derived from the sample constraints but not necessarily the sample constraints themselves. We must prune those that fail to produce any sample constraint in the user query. A naïve approach is to sequentially check each candidate specification against all sample constraints. However, as checking a schema mapping specification is essentially running the entire join query on the database, this naïve approach can be very expensive.

As most of the candidate schema mapping specifications are essentially incorrect, quickly pruning incorrect ones is vital to achieving a decent system efficiency. We use *filters* described in [9] to prune unqualified specifications with fewer and cheaper verifications. Given any complete join path and sample constraints C_{sample}, a *filter* is a partial join path along with a partial sample from C_{sample} covered by the path. Each filter is translated into a SQL query based on the sample constraints we define and sent to the database for verification. For example, to check if a filter composed of a join path joining "geo_lake" and "Lake" on "Lake.Name = geo_lake.Lake" can yield a data record "California OR Nevada; Lake Tahoe", we create the query in Figure 2.

Verifying a filter is cheaper than validating the entire 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 specifications. For now, we

```
SELECT geo_lake.Province, Lake.Name
FROM Lake, geo_lake
WHERE Lake.Name = geo_lake.Lake AND
MATCH (Lake.Name) AGAINST ('"Lake␣Tahoe"'
    IN BOOLEAN MODE) AND
MATCH (geo_lake.Province) AGAINST ('"California"
␣␣␣␣"Nevada"' IN BOOLEAN MODE) LIMIT 1;
```

Figure 2: The actual SQL query for verifying a filter

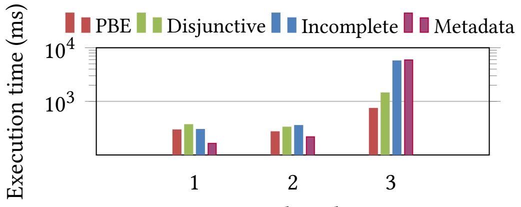

Figure 3: Efficiency experiment results (Dataset: Mondial)

follow a practice of filter selection and evaluation similar to [9]. Improving this practice is in our future work.

5 EARLY EXPERIMENTS

We implemented our proposed schema mapping method in a system called Beaver and performed experiments on a 16-core Intel Xeon server with 128 GB RAM. We first introduce the experimental setup in Section 5.1, and then evaluate Beaver on traditional PBE queries (Section 5.2) and declarative queries (Section 5.3).

5.1 Experiment Setup

Data Set — We examined Beaver on a complex real-world dataset Mondial [6] with 33 relations, 137 columns, and 48 foreign key constraints.

Benchmark Tests Generation — In our experiment, we simulated an end user who provided queries we synthesized using the following method. We created random schema mapping SQL queries of various *join lengths* (i.e., number of joins) using the database schema of the Mondial database. We used the sampled query results of the above queries as the standard PBE user queries provided by the simulated user in our experiment. We also synthesized the declarative schema mapping queries using these PBE queries.

5.2 Traditional PBE Queries

Beaver is designed for declarative schema mapping, but it is able to handle traditional sample-based (i.e. PBE) schema mapping queries. In this experiment, we demonstrate the efficiency of Beaver in answering PBE queries.

Overview — We randomly created three sets of synthesized schema mapping specifications of join length one to three, each containing 8 specifications. We sent each specification to the database and obtained top five random records in the query result and repeat this process twice. We took each query result set as the sample constraints for the "simulated" test query of PBE test cases. Hence, for each specification, we have three simulated PBE test cases (user queries) of 5 sample constraints each. We measured the execution time of these test cases on Beaver, and calculated the mean execution time for each specification.

Results — During the experiments, we compared the returned schema mapping specifications against the ground truth, which is the synthesized schema mapping queries used to generate the user queries. The results show that in all test cases, the correct schema mapping specification is among the returned results, which demonstrates the correctness of our schema mapping technique. The execution time results are shown in Figure 3. The x-axis is the join length of the schema mapping specifications, and the y-axis is the average execution time of all test cases for all schema mapping specifications with the same join length. In this section, we only focus on the red bar (labeled "PBE"), which denotes the execution time of PBE test cases.

We observe that, when testing MONDIAL, when the join length is one or three, BEAVER is able to complete within 1 second on average. The results suggest that BEAVER is efficient in interacting with the end user for traditional PBE schema mapping workloads.

5.3 Declarative Queries

Declarative queries are likely to be more computationally challenging than PBE queries, but should require less work from the user. In this experiment, we want to check whether BEAVER is able to execute the more-difficult declarative queries with a runtime that is comparable to the easier PBE queries.

We found that in most cases, BEAVER was able to find declarative solutions for most test cases with only a modest runtime premium over PBE queries.

Overview — We synthesized the declarative query test cases using the PBE test cases in Section 5.2. For each PBE test case, we randomly generate three different declarative query test cases in each of the following query genres: **1)** disjunctive, **2)** incomplete, **3)** metadata.

Disjunctive. Disjunctive queries contain sample constraints with disjunctions in one or more columns. For example, the column 1 sample constraint in Table 2 is a disjunctive constraint. To generate disjunctive queries using sample constraints, we picked a random column in each sample constraint from a PBE test case, and replaced the column value "x" with disjunctions in form of "x OR y". The value y in the new disjunctive constraints were randomly sampled from data values in the sample column but different rows.

Incomplete. Incomplete queries contain sample constraints with missing values in one or more columns. For example, in Table 2, the sample constraint has no value for the first and fourth column. Similarly, we randomly removed a column value in every sample constraint from a PBE test case to synthesize incomplete queries.

Metadata. Metadata queries are composed of both metadata constraints and sample constraints. To create metadata queries, we removed a random column value (either textual or numeric) from each sample constraint of a PBE test case, and replaced it with a metadata constraint of the corresponding column. If the chosen column was a textual column, the metadata constraint was the maximum data length. If the chosen column was a numeric column, the metadata constraint was the value range of the column.

For each of the PBE test case, we randomly generated three declarative queries in each of the above categories. We executed the generated queries on BEAVER and presented the mean of the execution time results for each target schema.

Results — Similar to Section 5.2, we validated the correctness of all test cases. We also measured the execution time of declarative queries on BEAVER. As all the declarative query test cases were spawned from the PBE query test cases used by Section 5.2, it is interesting to compare the execution time of the declarative queries against their corresponding PBE queries.

The results are also shown in Figure 3. Compared to PBE test cases, disjunctive, incomplete and metadata test cases on average cost 0.07, 0.01, -0.13 seconds more time when the join length is one, 0.06, 0.08, -0.05 seconds more time when the join length is two, and 0.70, 4.93, 5.17 seconds more when the join length is three. This suggests that the end user can provide declarative queries which we believe is easier to input and retain a similar system efficiency as they have for traditional PBE queries.

6 CONCLUSION AND FUTURE WORK

We introduced a declarative interaction model for schema mapping and a search-based solution to discover the satisfying schema mapping specifications. In the declarative schema mapping setting, the user is allowed to provide hints to the system in a more flexible way, which might require less effort than the traditional PBE schema mapping interaction model. The initial experimental study on the real-world relational dataset MONDIAL has shown that our proposed solution is able to find satisfying schema mapping specifications in reasonably short amount of time.

In the future, we plan to extend the declarative language used to describe the target schema. For example, we hope to allow the user to suggest schema mapping components/operations such as aggregations, joins or selection predicates. In that case, the system is usable by people of all levels of familiarity with the database.

7 ACKNOWLEDGMENTS

This project is supported by National Science Foundation grants IIS-1250880, IIS-1054913, NSF IGERT grant 0903629, a Sloan Research Fellowship, and a CSE Dept. Fellowship.

REFERENCES

[1] Daniel W Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. 2015. FlashRelate: extracting relational data from semi-structured spreadsheets using examples. In *PLDI*.

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

[3] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *POPL*.

[4] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. 2017. Foofah: a programming-by-example system for synthesizing data transformation programs. In *SIGMOD*.

[5] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. 2017. Foofah: Transforming data by example. In *SIGMOD*.

[6] Wolfgang May. 1999. *Information extraction and integration: The Mondial case study*. Technical Report. Universität Freiburg, Institut für Informatik.

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

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

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

[10] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing Highly Expressive SQL Queries from Input-output Examples. In *PLDI*.