QPlain: Query By Explanation

Daniel Deutch and Amir Gilad
Computer Science Department, Tel Aviv University

***Abstract*—To assist non-specialists in formulating database queries, multiple frameworks that automatically infer queries from a set of input and output examples have been proposed. While highly useful, a shortcoming of the approach is that if users can only provide a small set of examples, many inherently different queries may qualify. We observe that additional information about the examples, in the form of their *explanations*, is useful in significantly focusing the set of qualifying queries. We propose to demonstrate `QPlain`, a system that learns conjunctive queries from examples and their explanations. We capture explanations of different levels of granularity and detail, by leveraging recently developed models for data provenance. Explanations are fed through an intuitive interface, are compiled to the appropriate provenance model, and are then used to derive proposed queries. We will demonstrate that it is feasible for non-specialists to provide examples with meaningful explanations, and that the presence of such explanations result in a much more focused set of queries which better match user intentions.**

I. Introduction

It has long been acknowledged that writing database queries in a formal language may be a cumbersome task for the non-specialist. Different approaches have been proposed to assist users in this task; a prominent approach (see e.g. [1], [2], [3]) allows users to provide examples of input and output databases, based on which the intended query is automatically inferred. This approach can be highly effective if the examples provided by the user are plenty and representative. But coming up with such a set of examples is non-trivial, and unless this is the case, the system may be unable to distinguish the true intention of the user from other qualifying queries.

As a simple illustration, consider a user planning to purchase plane tickets for a trip. She has rather specific requirements of this trip: it should include five countries in South America, visiting each for a week and staying in Bolivia in the third week, in time for a carnival taking place there. After viewing a list of border crossings such as Table I, she concludes that Argentina and Brazil would serve as good end-points for the trip, and so would Peru and Paraguay. Since airfare from the US to these particular destinations at her dates of interest are quite expensive, she is interested in viewing additional recommendations. However, based only on these two examples of output tuples, there are many inherently different queries that yield them, and there is no reasonable way to distinguish between them. In particular the trivial query copying the content of Table I clearly does not capture the intention of the user, but is a reasonable query that may be proposed given the examples set.

Intuitively, if users would provide some form of "explanations" attached to their examples, it could guide the system in identifying the actual intended query. Of course, the most useful form of such explanation is the query itself, but the premise is that users are unable to write formal queries. Still, they have provided examples with some underlying logic in mind, and may be able to describe it. Ideally, we would like to allow users to provide explanations of varying level of detail (to be made formal), and to have the system compute and present an increasingly focused candidate queries in response to an increasing precision of the explanation. Continuing our running example, an explanation for a pair of end-points involves a full or partial description of actual trips that she has in mind, and are compatible with the example end-points. This would in turn limit the queries of interest to those that not only result in the example output, but rather do so based on criteria that are compatible with the explanation.

To achieve this goal, we first need a formal notion of explanations, one that supports different levels of granularity. To this end, we note that multiple lines of work have focused on the somewhat "reverse" problem of *explaining query results*. Multiple forms for such explanations, often termed lineage or provenance, have been proposed in recent years, and the tracking, storing and use of such explanations have been extensively studied. The work of [4] has put multiple such models on common grounds, namely the provenance semiring model [5]. Importantly, it has further formalized and characterized the level of detail supported by each model.

Building upon these foundations, we propose here a novel framework of *query-by-explanation*. The input to the framework is a set of examples, each consists of an input and output relation. Importantly, output tuples are further associated with explanations, which may take a form of one of multiple provenance models (we support here the "provenance semiring" $\mathbb{N}[X]$ [5], as well as Why(X) [6] and Lineage [7] in their polynomial interpretation of [4]; see Section II). The framework then computes and presents candidate queries, namely queries that, when evaluated with respect to the example input database, return the example output tuples, *and the provenance that is associated with them is consistent with the provided explanation*. We make a quite standard choice in this context (see e.g. [3], [2]) of further restricting our attention to *minimal* Conjunctive Queries with disequalities ($CQ^{\neq}$).

We note that explanations essentially detail the course of derivation of output tuples by the intended query. The various explanation models differ in the level of details they provide for each derivation. The most informative model is $\mathbb{N}[X]$ (called the "provenance semiring" in [5]), detailing precisely the tuples used in different derivations, including the number of times each tuple was used. In the context of our example, this means that the user explains a desired pair of countries by detailing all border crossings in some (but not necessarily all) trips that match her criteria and start and end in these countries. A less informative explanation, but one that is typically easier to formulate, would detail all tuples participating in some derivation, without distinguishing joint from alternative uses (this corresponds to the *lineage* model). E.g., the user may

detail all countries (or all border crossings) that she would like to visit in her trips between the two end-points, without mentioning whether two countries should appear together in a single trip. Naturally, the proposed set of queries would be more focused for the more informative explanation.

We have designed a suite of algorithms for learning queries from examples and their explanations, based on the different provenance models. We have implemented the algorithms in a system called `QPlain`. The system allows users to input examples and explanations through a user-friendly GUI, which does not assume any knowledge on provenance or its particular models. Instead, they simply drag-and-drop database tuples to form explanations, and the system automatically compiles an instance of the "simplest" model that can still accommodate the complexity of the provided explanation (see Section III).

We propose to demonstrate `QPlain` using a diverse dataset derived from Wikipedia. We will guide participants in choosing a "topic" out of the dataset, and will allow them to provide examples of output tuples as well as explanations, through the GUI. The system will compute a candidate consistent query, and will run it to compute more output tuples which will in turn be presented to the participants. They will then be able to provide feedback on the presented output, adding or removing examples as they see fit, and re-compute the results. We will demonstrate that it is feasible for non-experts to provide explanations, and that using such explanations, `QPlain` proposes queries of superior quality.

II. Technical Background

Provenance Polynomials: We focus here on *conjunctive queries* with disequalities ($CQ^{\neq}$) [8] and consider various provenance models proposed in the literature, varying in the amount of information that is maintained. The work of [4] has led to a unified perspective of these models via the notion of *provenance polynomials*, which we next intuitively explain. The idea is to associate an annotation with every tuple in the input database, and to extend the operations of relational algebra so that they will work on these annotated tuples, in an algebraic structure is based on two main components: *operations* that match the transformations that the data undergoes and *equivalence relations* over obtained expressions, that match equivalences in the underlying transformation language. It was shown in [5] that the most general such structure, i.e. the one storing the most information while obeying to the equivalence laws of CQs is that of *semiring of polynomials over the domain of basic annotations, with natural numbers as coefficients*, denoted $\mathbb{N}[X]$. Intuitively, in this structure, multiplication (denoted by $\cdot$) corresponds to joint use of tuples in a derivation, and summation (denoted by $+$) corresponds to alternative derivations. Multiple other proposed models for provenance, such as Why(X) [6], lineage [7] and others may then be expressed in this model by introducing further congruences (e.g. "forgetting" coefficients or exponents). By that, they encode less provenance information, but may do so in a much more concise manner.

Example 2.1: Reconsider our example from the Introduction of a user planning a trip with specific requirements. Her logic may be captured by the following query ($route$ is the relation depicted in Table I):

TABLE I: Relation $route$

prov.	A	B
a	Argentina	Brazil
b	Brazil	Bolivia
c	Bolivia	Argentina
d	Peru	Colombia
e	Colombia	Bolivia
f	Argentina	Colombia
g	Peru	Paraguay
h	Argentina	Paraguay

prov.	A	B
i	Germany	Belgium
j	France	Belgium
k	Belgium	Germany
l	Hungary	Slovakia
m	Poland	Slovakia
n	Slovakia	Hungary

TABLE II: Relation $trip$

A	B	$\mathbb{N}[X]$	Why(X)	Lin(X)
Argentina	Brazil	$f \cdot e \cdot c \cdot a + a^2 \cdot c \cdot b$	$f \cdot e \cdot c \cdot a + a \cdot c \cdot b$	$\{a, b, c, e, f\}$
Peru	Paraguay	$d \cdot e \cdot c \cdot h$	$d \cdot e \cdot c \cdot h$	$\{c, d, e, h\}$

TABLE III: Relation $trip_2$

A	$\mathbb{N}[X]$	Why(X)	Lin(X)
Germany	$i \cdot j \cdot k$	$i \cdot j \cdot k$	$\{i, j, k\}$
Hungary	$l \cdot m \cdot n$	$l \cdot m \cdot n$	$\{l, m, n\}$

$$q = trip(x, w) :- \quad route(x, y), route(y, Bolivia), \\ route(Bolivia, z), route(z, w)$$

The provenance-aware result of evaluating q over the relation $route$ is shown in Table II. Different columns include provenance based on different models. Consider for example the tuple $trip(Argentina, Brazil)$. It is obtained through two different derivations: one that maps four distinct tuples (annotated a, c, e, f) to the four atoms, and one that maps the tuple annotated a to two atoms and b, c to the rest. Consequently, the tuple "exact" ($\mathbb{N}[X]$) provenance is $f \cdot e \cdot c \cdot a + a^2 \cdot c \cdot b$. Note that each summand stands for an alternative suitable trip that starts at Argentina and ends at Brazil. If we alternatively store *Why* [6] provenance, we still have summands standing for alternative derivations (trips), but we lose track of exponents and coefficients, i.e. the number of times each tuple and summand were used (e.g., we do not know that a border was crossed twice). Finally, one may be interested in the tuple *lineage*, in which case we store the set of tuples that have been used in *some* derivation, but we cannot distinguish between alternatively and jointly used tuples.

Query-By-Explanation: We consider the following problem. The user provides as input multiple examples for input databases and tuples in the desired output relation, and further associates *an explanation* with each of these output tuples. Explanations are captured by either one of the provenance models we have illustrated above, with the more experienced users being able to provide a more detailed form of provenance. The system output is then a conjunctive query (possibly with disequalities) that meets the following requirements: (1) it "matches" the specification, i.e. when evaluated with respect to the input database it would generate the output tuples (and possibly more), with their provenance matching the user specification, and (2) it is *minimal* in the standard sense: no conjunctive query with less atoms is equivalent to it. There may be multiple queries that qualify, in which case we present multiple options and prioritize them, preferring short and more specific queries (see Section III).

Example 2.2: Reconsider our running example. Assume first that the user gives no provenance at all. Of course, the "real" underlying user query is still a valid output, but there are many other simpler queries that fit the specification. In

particular, the simple query:

$$q_2 = trip(x, y) :- \quad route(x, y)$$

would have been a reasonable output. However, it is quite far from matching the user's intention, which is only reflected in the provenance. As a more detailed provenance model is used, more queries become unsuitable as answers, which means that the set of possibly derived queries is more focused. Indeed, given each of the provenance expressions we have exemplified, q_2 is not a valid output. In particular, an $\mathbb{N}[X]$ provenance expression *exactly dictates the number of query atoms* (4 in this case). It may still allow multiple options, such as (in our case):

$$q_3 = trip(x, w) :- \quad route(x, y), route(y, Bolivia), \\ route(Bolivia, Argentina), route(Argentina, w)$$

which is more specific than q (and would be "disqualified" if we had further examples not involving Argentina in the trip) but much closer to the user's true intention than q_2. In this particular case, the Why(X) provenance still shows that there are at least 4 atoms; furthermore, one may verify that introducing an exponent to either c or b in the first monomial does not result in a qualifying query. Consequently, the "shortest" qualifying query is obtained by assuming an exponent 2 associated with a, and we again get q_3 (but note if we do not care about length, many more queries now quality, including one with 16 atoms). However, if e.g. we were only given the output tuple $trip(Argentina, Brazil)$ with only the derivation $a^2 \cdot c \cdot b$ (which is represented by $a \cdot c \cdot b$ in Why(X)), then there exist three-atom queries such as $trip(x, w) :- \quad route(x, w), route(w, z), route(z, x)$ that fit the Why(X) but not the $\mathbb{N}[X]$ provenance.

We next use a simpler example to illustrate the difference between learning from Lin(X) and Why(X) expressions.

Example 2.3: Consider Table III detailing examples of different constraints for a trip in Europe. In this case, the shortest query based on the $\mathbb{N}[X]$ and Why(X) provenance is $trip_2(x) :- \quad route(x, y), route(z, y), route(y, x), z \neq x$. In contrast, when given the Lin(X) expressions, they may in principle correspond to any combination of the annotations into monomials of any length (only some of which actually fit the examples). In this case, the shortest one based on the Lin(X) expression is $trip_2(x) :- \quad route(y, z), route(z, x)$.

We have developed a suite of algorithms for learning queries from expressions corresponding to the different provenance models. We next briefly explain each algorithm, omitting details for lack of space.

Algorithm for $\mathbb{N}[X]$: With $\mathbb{N}[X]$ provenance, the exact number of query atoms and the input relations they pertain to, is immediately given by the provenance expression. Thus, what needs to be decided is (1) which body variables appear in the head and (2) which equalities/disequalities are imposed. For the former, we generate a separate output row for each monomial and pick two output rows with maximally different set of constants. For these, we generate a full bipartite graph whose nodes are the annotations, each side corresponds to the monomial of one of the chosen tuples, and each edge is labeled with a set of output attributes that are "covered" by the edge end-nodes. Covering of an attribute A means here that the vector of values appearing in A for the output tuples equals the vector of values appearing in some attribute for the two input tuples corresponding to the end-nodes. Next, we find a matching in this graph whose edges cover all attributes of the output tuples. We can show that it is sufficient to look for such matchings of size bounded by the number of output attributes. This matching dictates the location of head variables in the body, to be placed at the appropriate locations of "covering" atoms. Now, the algorithm checks if this query is consistent with the rest of the examples, by simply matching it to their provenance (this is much faster than evaluation). If so, the algorithm attempts to minimize the query by gradually imposing further equalities and disequalities between variables (and replacing variables by constants where possible), in a greedy manner. If not, we try a different matching in the graph and repeat (if no matching works, we declare that there is no qualifying query).

Algorithm for Why(X): In contrast, Why(X) provenance does not reveal the exact number of query atoms, since arbitrary exponents may have appeared in the $\mathbb{N}[X]$ provenance of the intended query, and are not shown by the Why(X) expression. In principle, the number of queries to consider could have thus been infinite. However, we can show a "small world" property, namely that if a qualifying query exists, then there exists such query whose number of atoms is less than the product of all monomial sizes (perhaps surprisingly, the *maximum* monomial size is an insufficient bound). We can use this result to generate a "sufficiently long" $\mathbb{N}[X]$ provenance, and re-run the previous algorithm on it. However, in practice one expects shorter queries to qualify; consequently, we greedily consider greater exponents and attempting a solution for them, using the bound to decide when to terminate the algorithm if a query was not found.

Algorithm for Lin(X): With Lin(X), we face the challenge of not knowing the number of query atoms, and can again derive a "small world" property with the bound being the product of annotation sets sizes. In addition, for each query size there are multiple combinations of annotations into monomials. Again we employ a greedy approach in this two axes, generating combinations based on locally maximizing the number of "covered" (as explained above) attributes.

Related work: Previous work has focused on learning queries from examples (see e.g. [2], [1], [3], [9], [10] and the somewhat different problem in [11]). Here we propose, for the first time to our knowledge, to further leverage provenance information as explanations in this process. This requires more work from the user, but one that is feasible in many cases (see further discussion in Section III) and provides otherwise unattainable insights regarding the intended query. An additional, computational advantage, is that only tuples whose annotation occur in the provenance need to be considered. Multiple models for data provenance have been proposed (see e.g. [5], [6], [12], [13], [14]); we have focused here on three particular models, but others may fit the framework as well.

III. Implementation

`QPlain` is implemented in JAVA with JAVAFX GUI using SceneBuilder, and runs on Windows 8. It uses MS SQL server as its underlying database management system. The system architecture is depicted in Figure 1. Users can load an input database, and provide examples of output tuples (see Fig. 2). For each output tuple they may press the *Explain!* button,

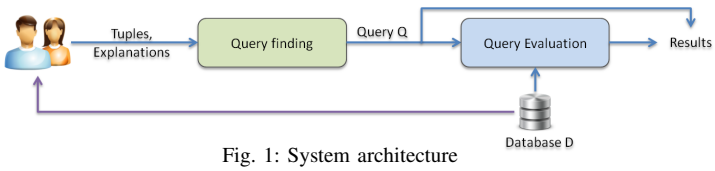

Fig. 1: System architecture

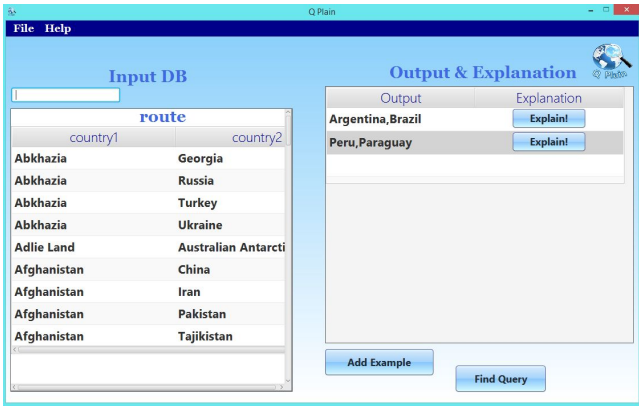

Fig. 2: Input screen

which leads to the explanation pop-up window (see Fig. 3). Users drag-and-drop tuples from the input database to form an explanation. They may further press the "Add Explanation" button to start a new explanation, so that each row represents a single alternative explanation. The "type" of provenance ($\mathbb{N}[X]$, Why(X), Lin(X)) is not specified by the user, but rather is automatically identified (see next). Then, the system generates a minimal-length qualifying query, evaluates it and shows the query along with its full result in a separate output screen. The user may further evaluate the query with respect to other databases, may ask for other qualifying queries (recall our greedy approach of generating candidate queries), or may refine the example and re-run the computation.

Recognizing Provenance Types: Before an appropriate query can be generated, `QPlain` needs to know the type of provenance that has been fed. Since the intended users are non-experts, they cannot be expected to be able to know and distinguish between the different provenance models. To this end, we assume the "simplest" model of provenance that can accommodate the user-provided explanations. That is, if only a single explanation was fed for every tuple, then Lin(X) is assumed to be used. If multiple explanations have been used but every tuple was drag-and-dropped at most once to a single explanation, then Why(X) provenance is assumed. Otherwise, $\mathbb{N}[X]$ provenance is assumed.

IV. Demonstration Scenario

We will demonstrate the usefulness of `QPlain`, and in particular that (1) users are able to provide explanations for their examples, (2) with such explanations, `QPlain` can infer more focused queries, and (3) queries are inferred at interactive speed. The system will be demonstrated using real-life data from Wikipedia (stored as relational database) and will interactively engage the audience, demonstrating the different facets of the system. The demonstration will start by reproducing the scenario in Example 2.1, showing the system performance and output for different types of provenance. During this demonstration we will explain the system architecture and the different options that it provides. We will show that a more informative provenance results in a more focused set of

Fig. 3: Explanation screen

queries that better fits the user intention. In the second part of the demonstration, we will ask participants to choose a "topic", out of a variety of Wikipedia datasets: geographic data (as in our running example), academic and influence relationships, movies and trading data. We will allow the audience to experiment with the system, giving examples and explanations of their choice. If the results are not satisfactory, they could change the tuples or the explanations and iteratively run the system while improving the example until the results are satisfactory. The output tuples and explanations will be fed by the participants to `QPlain` through its user-friendly GUI, and we will again show and discuss the analysis results. Finally, we will allow the audience to look "under the hood", discussing details of the algorithms and showing alternative queries that they have considered (and discarded).

Acknowledgment

This research was partially supported by the Israeli Science Foundation (ISF, grant No. 1636/13), by the Broadcom Foundation and by ICRC — The Blavatnik Interdisciplinary Cyber Research Center.

References

[1] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy, "Query by output," in *SIGMOD '09*.

[2] M. Zhang, H. Elmeleegy, C. M. Procopiuc, and D. Srivastava, "Reverse engineering complex join queries," in *SIGMOD '13*.

[3] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik, "Discovering queries based on example tuples," in *SIGMOD '14*.

[4] T. J. Green, "Containment of conjunctive queries on annotated relations," in *ICDT*, 2009.

[5] T. J. Green, G. Karvounarakis, and V. Tannen, "Provenance semirings," in *PODS*, 2007.

[6] P. Buneman, S. Khanna, and W. C. Tan, "Why and where: A characterization of data provenance," in *ICDT*, 2001.

[7] Y. Cui, J. Widom, and J. L. Wiener, "Tracing the lineage of view data in a warehousing environment," *ACM Trans. Database Syst.*, 2000.

[8] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[9] Y. Ishikawa, R. Subramanya, , and C. Faloutsos, "Mindreader: Querying databases through multiple examples," in *VLDB 1998*.

[10] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas, "Exemplar queries: Give me an example of what you need," *VLDB 2014*.

[11] M. M. Zloof, "Query by example," in *AFIPS NCC*, 1975.

[12] D. Gawlick and V. Radhakrishnan, "Fine grain provenance using temporal databases," in *TaPP*, 2011.

[13] U. A. Acar, A. Ahmed, J. Cheney, and R. Perera, "A core calculus for provenance," *Journal of Computer Security*, vol. 21, no. 6, 2013.

[14] B. Glavic and G. Alonso, "Perm: Processing provenance and data on the same data model through query rewriting," in *ICDE*, 2009.