Writeup 9 Problem desc imp rel hard 5 5 5 4 Progress 9 37/40

Iterative query-seeking through partial query specification

Doris Jung-Lin Lee

Good Job!

1. INTRODUCTION

Formulating ad-hoc database queries for exploratory data analysis is a challenging problem for analysts. While many existing systems have been developed to make it easier to perform query specification in exploratory analysis [1, 4, 6, 10, 14, 17], the more pressing challenge of how to help analysts come up with the right questions to ask is relatively unexplored. Most of these prior work have focussed on intent-to-query mapping mechanisms that assume users have a question in mind to begin with. Therefore, these work focuses on resolving the "language barrier" to help novices unfamiliar with SQL issue queries to the database. However, users often only have minimal or partial ideas on what types of data operations they would like to perform or the types of query they are interested in. Moreover, while existing database query interface are capable of synthesizing SQL queries based on high-level specifications, the space of queries that could be issued with these interfaces are often limited by the set of form fields and interactions envisioned by tool-designers. A more natural mode of interaction is perhaps to specify the the known examples and relations of interest, albeit partial, to the system, and let the system infer what is best to show to the user.

To this end, we proposed a unifying query language that captures a wide spectrum of query input specificity, called partial query language (PQL). Unlike structured querying languages, which are monolithic and evaluates only upon exact specification, PQL is tolerant to the inherent ambiguity of user specification. PQL takes in an underspecified query during exploratory data analysis, the system suggests interesting relations or examples to query with or further actions. Based on the user's feedback on the seen output, the system updates the next set of recommendations. The goal of PQL is to simultaneously enable flexible querying as well as gain better understanding of the dataset to facilitate users to discover the interesting questions to ask.

2. MOTIVATION & RELATED WORKS

Despite extensive work in database usability, there is an inevitable design trade-off between the query expressivity and interface usability [3, 9]. Given that there is no one size fit all interface for query specification for users of different expertise levels and workload, PQL is designed as a middle-layer between the interface and querying engine that can take in a wide spectrum of queries of different input types and degrees of specificity that could be potentially generated from different interfaces. Here, we list the potential query inputs in order of increasing specificity:

- **Cold-start**: no supervision, only given dataset as input. (viz-summarization is a more supervised version of this where measure and group-by dimensions of interest are given.)
- **Example-based partial queries (EPQ)**: EPQs takes in data examples based on what they have already seen and use this to jumpstart their query.

  Existing work in query-by-examples (QBE) asks users to provide I/O examples of the query to be synthesized [15, 16]. However, if the user does not know what they are querying for, then they would not be able to come up with such an example. In addition, EPQ accepts a more easy-to-come-up-with input modalities such as visualization distributions or an existing singleton record as examples to query. The types of inputs could be largely classified in two categories: 1) Record querying: querying via non-aggregated set of tuples or visualizations as input (e.g. drag-and-drop in Zenvisage, smart drill down) or 2) Result querying: querying via transformed tuples (aggregation, top-k), often for offering explanations [2, 13, 17].

- **Relational partial queries (RPQ)**: RPQs follow the conventional approach to querying a database where a user starts with a pre-existing idea of what he is looking for based on what he has already seen or know. However, formulating SQL queries that maps user's high-level intentions to specific query statements is challenging.

  Recent work in natural language querying have tried to address this by parsing adjectives and quantifiers and asking

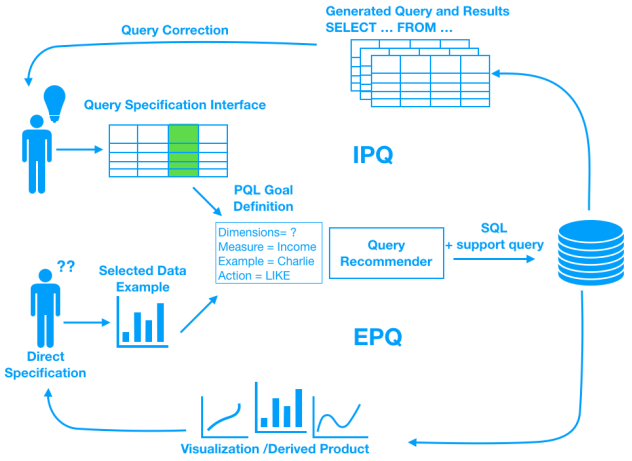

Figure 1: System architecture highlighting the relational and example partial query use cases. The users can move seamlessly across different query specification mechanisms to iteratively improve their query.

You should look up Sumit Gulwani's work on synthesis of programs via examples. Also a related paper by Bjoern (sp?) Hartmann.

the users for additional information to resolve the ambiguity through form-based interface if needed. Similar to form-based or visual query builders [1], these systems are often based on templated queries with limited expressiveness in their linguistic and conceptual coverage, which makes it difficult for expert users to express complex queries. SnipSuggest addresses this problem by recommending relevant snippets of SQL queries based on partial queries (user's partially typed input) via auto-complete [6]. However, their context-aware algorithm crucially depends on the existence of a query workload.

- **Complete specification**: At the highest level of specification, there is SQL and direct manipulation interfaces that performs database operations based on complete specification.

3. USAGE SCENARIO

A team of analysts is given a dataset that consists of the reported income of individuals. They wanted to investigate whether there is any evidence of institutional bias on worker salaries. Given this large dataset with hundreds of attributes and records, they are unsure of where to start. The only thing they know is that they care about the Income column, so they specify this as the measure attribute of interest in PQL by brushing the Income column in yellow (Figure 3 top). In this initial step, PQL tries to provide as much support information as it can by displaying a sample table of data records arranged in the order of increasing income, the attributes are displayed in the order of most to least importance, where importance is defined by how much the feature have an impact on the measure variable of interest. By convention, the primary key is always in the first column. These attribute-importance ranking techniques populates the most important relations that the users has to see in the first few visible columns.

Given this information, analyst A was still not able to infer anything about any particular column to ask a new question, but notices that Charlie has an incredibly low income compared to average. Using PQL specification interface, he uses the green paintbrush to highlight the record for Charlie and specifies the action to look for other records similar to Charlie. He notices that all of the returned records are all young children who work part-time jobs.

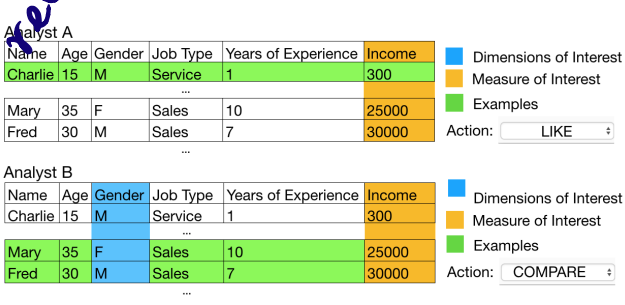

Figure 2: One example of a potential interface that could be used for submitting PQL queries. Users can select examples or columns of interest.

Analyst B skims through the records and notices that despite Mary having more years of working experience than Fred, she is getting paid $5000 less than him. He wants to find out whether employers are marginalizing the pay of certain individuals based on gender for the same job done. He colors the Gender column in yellow and the record of Mary and Fred in green, then specifies the "COMPARE" action (Figure 3 bottom). Given this PQL specification consisting of a mix of records, dimensions and measure attributes, the example finding algorithm in PQL finds records that are similar to Fred AND Mary, while distinguishing these groups of records by gender. The results consist of two tables corresponding to males and females with similar credentials as Fred and Mary. Using the records and supporting overview statistics and histograms for the corresponding result-sets, analyst B finds that Gender is not a factor in determining Income and that the observed example was an anomaly and seeks for additional hypothesis that explains the difference between Mary and Fred.

4. PQL DESIGN

The language design of PQL is motivated by several desiderata:

1. Tolerant to ambiguity : PQL must have a model of inference that can make inference with incomplete specification, since 1) partial queries can be used for jump-starting the exploration of under-developed hypothesis and 2) queries can be iteratively refined by the user as user gain more information about the data to update the model.

2. Model does not depend on a domain-specific query workload: Many existing systems makes use of information from query logs to provide "typical workload" query recommendations [5,6,9]. However, the existence of such workloads is not commonly available outside of large enterprises or scientific collaboration setting. In addition, while PQL can be used for all stages of analysis, it is most beneficial to the users at the early-stages of data analysis. So if there is a pre-existing queries performed, it means that a detailed analysis had already been performed and the insights from that analysis (e.g. through documentations) should have given the analyst a better idea on what questions to ask next. Without the query workload, the model of inference must work with minimal user input and be updatable upon additional user input.

3. Composability: PQL operators must be composable to increase the expressiveness of PQL in supporting a large class of ad-hoc queries, so that analysts are not constrained when exploring the types of questions they might want to ask.

4. Supporting both records and relations as inputs : Given the prior work of how analyst make sense of their data [7, 12], PQL must support both bottom-up example records and top-down relational specification of queries. As shown in the usage scenario, given the same data output, different analysts may have different interpretation and approach the same dataset and problem.

PQL can be written in terms of action-operators and goal definitions. Goal definitions specifies the attributes and records that users are interested in. There are three types of goal definitions supported by PQL:

- Attribute-value of interest (e.g. GENDER= 'Female')
- Attribute of interest (e.g. GENDER=?): can either be a measure or dimension. Measure of interest have a special significance in the algorithm.
- Non-aggregated example records $\{r_1 ... r_n\}$: Point queries where all attribute-values are completely specified (e.g. Mary: GENDER=F, Age=35, ..., Income = 25k)

Any unspecified goal definitions is denoted with the special value "?". Action-operators can take a record (r_i) or relation (A_i) as inputs

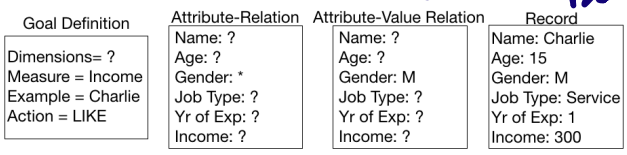

Figure 3: An example PQL specification is shown on the left with ? denoting the underspecified values. The three types of goal definitions are shown in the right.

and outputs and performs some action. Action-operators can be decomposed as a set of goal definitions. Examples of action-operators includes:

- DISTINGUISH r_i: Find the distinguishing feature of r_i.
- EXPLORE $\{A_1 ... A_n\}$: Overview or Drill-down to attribute(s) of interest.
- EXPLAIN r_i: Tracing the lineage of r_i.
- LIKE $\{r_1 ... r_n\}$: Find records that are similar to r_i.
- COMPARE $\{r_1 ... r_n\}$: Compare why two or more records are different.

5. PROBLEM FORMULATION

Described below are two approaches that I am currently exploring for the problem of query recommendation given a PQL specification.

5.1 Recommending the most important relationsexamples for specified actions

Given a limited space of operators, determining the best action to perform for a given input is not a hard problem, as the options could be simply recommended by matching desired inputs and selected by the user. In this case, the problem becomes a recommendation problem that involves finding the most suitable set of attributes or relations to perform the action-operator on.

We define relations and examples as nodes in a graph. As described earlier, examples are simply a special instance of relations where the values for all possible attributes are specified. Edges between the relations and examples can exist if the example contains the relation. Intuitively, a relation is "important" if many of the user's selected examples point to it. Conversely, an example is "important" if is pointed by many selected or high-influencing relations (i.e. measure attribute of interest strongly depends on the feature). Variants to PageRank-like algorithms such as HIT [8] may be suitable for this application in finding the most important node to recommend in a query. Solving this problem is challenging since the number of potential combination of attribute-value combination can be large for datasets with high-cardinality attributes or large numbers of attributes. In addition, how to find the appropriate ranges or bins to create attribute-value combinations for a numerical attribute (such as Age) is unclear.

5.2 User Feedback for iterative steering in data-action space

One alternative problem formulation is to incorporate user feedback to alleviate the problem of lack of user input. User feedback provides a way to collect additional inputs, since it is often easier for a user to make selections or ratings based on seen results, rather than coming up with query specification from scratch. While feedback has been prevalent in databases for the purpose of information retrieval(IR), the concept has not been explored extensively in interactive data exploration system. Most systems have focused on system-level feedback to aid the users in data exploration [4, 10]. However, the human action cycle of interaction requires closed feedback loop between user and the system [11]. DataPlay provide a query correction tool for users to mark 'want out', 'keep in' examples to improve the query [1]. Coming up with a right model for feedback is challenging since users often regard database queries to provide "ground truth"(rather than varying degrees of relevance as in the case of IR), so the feedback information provided by the users will not be as simple as relevant/irrelevant, which makes inference harder.

Feedback also provides a mechanism in which we can collect more information about the user's intent to infer the missing fields in partial query. If we consider query recommendation as a search problem through the space of data and actions, a mixed-initiative interface that enables users to provide feedback as well as update their partial queries can help steer the recommended queries to reach the right balance between the multiple objectives (such as diversity, representative, interestingness, etc.). Contrary to the previous problem formulation, the interactive multi-objective optimization works on top of both the data-action space and seeks the most suitable query to perform on recommended relation/examples, rather than simply a search for the most important relation/examples to perform on a fixed query.

6. PROJECT SCOPE

The project will focus on developing the problem formulation and potential algorithms for solving the query recommendation problem for PQL, as well as expanding on the details of the PQL language design. While I do not plan to build the system prototype for the course project, I will provide concrete examples from real-world datasets to motivate the design choices of the PQL language and serve as a proof-of-concept of the potential applicability.

7. REFERENCES

[1] A. Abouzied, J. Hellerstein, and A. Silberschatz. Dataplay: interactive tweaking and example-driven correction of graphical database queries. *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 207–217, 2012.

[2] A. Chapman and H. Jagadish. Why not? *SIGMOD 2009*, 38(2):711–712, 2009.

[3] H. Jagadish and A. Chapman. Making database systems usable. *Proceedings of the . . .* , D(1):13, 2007.

[4] L. Jiang and A. Nandi. SnapToQuery: Providing Interactive Feedback during Exploratory Query Specification. *Proceedings of the Very Large Data Bases Endowment (PVLDB)*, 8(11):1250–1261, 2015.

[5] N. Kamat and A. Nandi. A Session-Based Approach to Fast-But-Approximate Interactive Data Cube Exploration. 1(1):1–25, 2016.

[6] N. Khoussainova and Y. Kwon. Snipsuggest: Context-aware autocompletion for sql. *Proceedings of the . . .* , pages 22–33, 2010.

[7] G. Klein, B. Moon, R. Hoffman, and K. Associates. A Macrocognitive Model Human-Centered Computing A Macrocognitive Model. *IEEE Intelligent Systems*, 21(5):88–92, 2006.

[8] J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(May 1997):668–677, 1999.

[9] K. Morton, M. Balazinska, D. Grossman, and J. Mackinlay. Support the Data Enthusiast: Challenges for Next-Generation Data-Analysis Systems. *Proceedings of the VLDB Endowment, Volume 7, pp. 453âĂŞ456, 2014*, 7:453–456, 2014.

[10] A. Nandi, L. Jiang, and M. Mandel. Gestural query specification. *Proceedings of the VLDB Endowment*, 7(4):289–300, 2013.

[11] D. Norman. *The Design of Everyday Things*. Basic Books, 2013.

[12] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis, 01 2005.

[13] S. Roy, L. Orr, and D. Suciu. Explaining query answers with explanation-ready databases. *Proceedings of the VLDB Endowment*, 9(4):348–359, 2015.

[14] M. Vartak, S. Madden, and A. N. Parmeswaran. SEEDB : Supporting Visual Analytics with Data-Driven Recommendations. 2015.

[15] C. Wang, A. Cheung, and R. Bodik. Interactive Query Synthesis from Input-Output Examples. *Sigmod*, (1):3–6, 2017.

[16] C. Wang, A. Cheung, and R. Bodik. Synthesizing highly expressive SQL queries from input-output examples. *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2017*, pages 452–466, 2017.

[17] E. Wu and S. Madden. Scorpion: Explaining Away Outliers in Aggregate Queries. *Proceedings of the VLDB Endowment*, 6(8):553–564, 2013.