Learning through play: Iterative query-seeking through partial query specification

ABSTRACT

Abstract

1. INTRODUCTION

Formulating ad-hoc database queries for exploratory data analysis is a challenging problem for analysts. Prior work in query by example and query synthesis address this by inferring candidate queries based on input and output data examples and visual query construction through a specification interface. âĂĺHowever, the more pressing challenge for both novice and expert analysts is often coming up with the right question to ask. 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 âĂIJlanguage barrierâĂİ to help novices unfamiliar with SQL issue queries to the database. 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 interface 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 units of interest, albeit partial, to the system, and let the system infer what is best to show the user. To this end, we proposed a unifying querying language that captures a wide spectrum of input query specificity (from no user input to complete query specification), called partial query language (PQL). Unlike structured querying language, such as SQL and XQuery, which 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 uses a rule-based template to execute suggested actions, and updates the next set of recommendations based on users feedback on the seen output. 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.

Contribution:

- unifying model for inference
- vizRec that account for feedback (not based on logs)

Our key idea is to make use of feedback to support partial queries. IR relevance feedback idea that queries are hard to come up with, but it should be easy to rate something or say if it is relevant or not. Dataplay has this with "want-in", "want-out example"

2. MOTIVATION & RELATED WORKS

Depsite extensive work on making database usable, there is an inevitable design tradeoff between the query expressivity and interface usability that depends on user's expertise and workload[**?**, **?**]. Given that there is no one size fit all interface for query specification, PQL is designed as a middlelayer 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 these interfaces. Here we list the potential query inputs in order of increasing specificity:

- Cold-start: no supervision, only given the dataset as input (viz-summarization).
- **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 the area of Query-by-Examples (QBE) asks users to provide I/O examples of the query to be synthesized. 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 offerring explainations[**?**, **?**, **?**].
- **Relational partial queries (RPQ):** RPQs follow the conventional approach to querying a database where

ACM ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

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 the users for additional information to resolve the ambiguity through form-based interface if needed. Simmilar to form-based or visual query builders[**?**], 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 (userâĂŹs partially typed input) via autocomplete[**?**]. 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.

these two inter-related modes of data exploration, where data or bottom up approaches are initiated through data examples and framing comes from specifying distribution or relationships in a top-down manner.

3. USAGE SCENARIO

A team of analyst is a given a dataset that consist of the reported income of all the villager living in their county. They wanted to study whether there is any evidence of institutional bias worker salaries. Given this large dataset with hundreds of attributes about each individual, they are not sure where to start. The only thing they know is that their measure attribute of interest the Income column, so they specify this in PQL. In this inital 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.

Given this information, analyst A was still not able to infer anything about any particular column to ask a new question, but he notice that Charlie has an incredibly low income compared to the average income. Using PQL, he can searches for other records simmilar to Charlie. He notices that all of the returned records are all young children who work part-time jobs.

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. Given this hypothesis, he specifies a PQL query to find women who are in Sales and have simmilar years of experience as Mary and Fred. The PQL engine generates two support queries on this selected population to display the overview statistics and histograms of the selected subpopulation and its counterpart (i.e. men who are in Sales with simmilar years of experience).

4. PQL DESIGN

The language design of PQL is motivated by several desiderata, based on the analyst's needs and limitations of existing systems:

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 jumpstarting 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. 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.

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.
- Examples $\{r_1...r_n\}$

All action-operators can be decomposed as a set of goal definitions.

Action-operators takes a record (r_i) or relation (A_i) as inputs and outputs and performs some action.

- EXPLORE r_i: Given a data record, explore
- EXPLORE $\{A_1...A_n\}$:
- EXPLAIN $\{A_1...A_n\}$:
- EXPLAIN r_i:
- LIKE r_i: Find records that are simmilar to r_i.
- COMPARE $\{r_1...r_n\}$: Compare between records that are different
- DISTINGUISH r_i: Find what is the distinguishing feature of r_i

5. PROBLEM FORMULATION

Since the space of operators for a given input is limited, finding the best action to perform is not hard. involves finding the most suitable attribute or relations to perform the action on.

6. SYSTEM ARCHITECTURE

Given the prior work and usage scenarioes of how analyst make sense of their data, PQL language design supports both example-driven and idea-driven queries to offer suggestions and explanations to the users. 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 enables analysts to seamlessly switch between the two modes of inquiry as shown in Figure **??**.

7. PROJECT SCOPE