Iterative query-seeking through partial query specification

Doris Jung-Lin Lee

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 [2, 10, 14, 17, 22, 26], 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 focused 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 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 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.

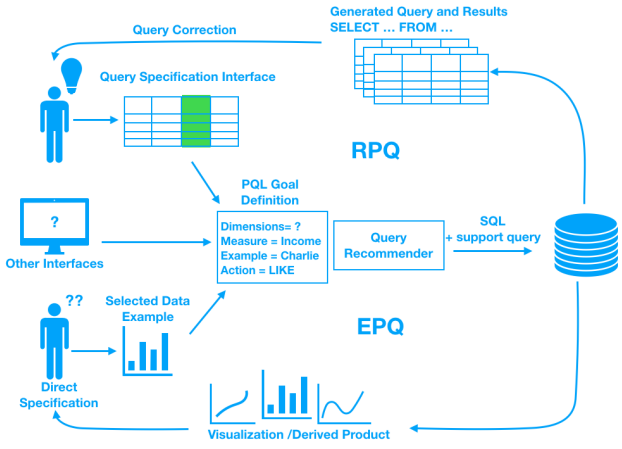

Figure 1: Schematic of PQL system overview and use cases. Example-based partial queries (EPQs) take in data examples based on what they have already seen and use this to jumpstart their query. Relational partial queries (RPQs) follow the conventional approach to querying a database where a user starts with a pre-existing idea of what he is looking for based on what he has already seen or know. The users can move seamlessly across different query specification mechanisms to iteratively improve their query.

2. RELATED WORKS

As illustrated in Fig.2.2, despite extensive work in database usability, there is an inevitable design trade-off between the query expressivity and interface usability [9, 16]. Here, we review three areas of related works in order of increasing usability.

2.1 General-Purposed Analytic Tools

General-purposed analytic tools are systems that are designed for transforming, browsing, or visualizing data mainly via direct manipulation and specification. Manual specification is the conventional approach to querying a database where a user starts with a pre-existing idea of what she is looking for based on what she has already seen or know. However, formulating SQL queries that maps user's high-level intentions to specific query statements is challenging. Users often do not know what they want to query for without context. The extensibility of these systems or querying language also comes with the cost of potentially overloading the users with too many potential options to chose from. Therefore, recommendation and visual query builders have been developed to address these issues to help novices issue common database queries, such as SPJ or basic aggregation.

2.2 Data Recommendation

Recommendation systems used for supporting exploratory analysis largely falls under three main categories: 1) relational recommendations: what data slices, attributes, tables may be of interest to the user given prior history logs [3, 14, 18], 2) visualization recommendation: recommending interesting visualizations, intended to summarize and focus the user's attention onto particular portions of data [22, 25], and 3) explanation recommendation: recommending possible explanations (queries that generated the results) from a given transformed tuple results (e.g. aggregation, top-k) [4, 21, 26]. Our proposed work is more similar to relational recommendation in spirit. However, these systems are often limited by the availability of a query workload to define a supervised metric of interestingness

for query recommendations [12, 14, 16]. For example, SnipSuggest addresses this problem by recommending relevant snippets of SQL queries based on partial queries (user's partially typed input) via auto-complete [14]. Likewise, [18] make use of the assumption that an attribute is interesting if there is a derived table that make use of this attribute as constraint. Visual recommendation system such as Voyager make use of partial specification through "wildcards" to show appropriate visual encodings for all possible x,y based on the data type. While this works well for datasets with small number of attributes, since this approach does not rank visualizations, large numbers of plots will be generated for datasets with large numbers of attributes and would be difficult to interpret.

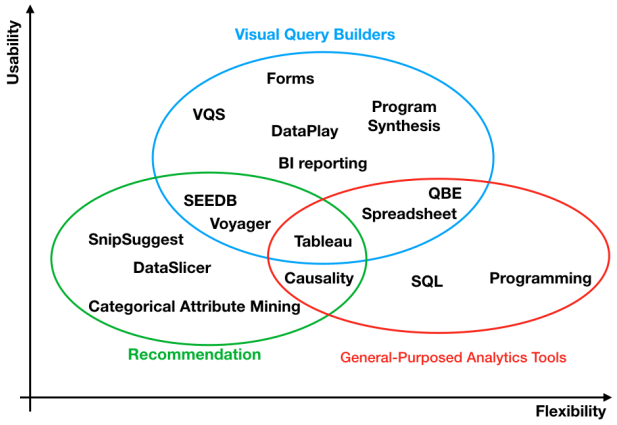

Figure 2: Mapping out related works according to the usability and expressivity dimensions.

2.3 Visual Query Builders

Visual query builders often consist of highly-usable interfaces that asks users for a specific set of information mapped onto a predefined query. 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 [7]. Other related works have asks users to provide I/O examples of the query to be synthesized [11, 23, 24] or enable users to graphically construct quantified queries [2]. However, similar to form-based 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.

Given that there is no one size fit all interface for query specification for users of different expertise levels and workload, PQL is envisioned 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. As illustrated in Fig.1, these can range from cold-start (no supervision) to input examples, input relations to complete specification.

3. FORMATIVE STUDY

In order to motivate the language design of PQL, we were interested in the space of queries that users would be interested in and the portion of data that they want use to achieve their analysis goals. The study made use of a real-world dataset that contains information about the home-rental site Airbnb's rental listings[1], consisting of 100088 rows and 16 columns. Participants were provided the study instructions, four example scenarios, and questionnaire via a Google Form[2]. We asked participants to describe three envisioned scenarios of how people might analyze the given dataset. For each envisioned scenario, participants were required to describe the analysis goals and the analysis task for accomplishing the goal. Optionally, participants were also asked to specify whether there are any columns, rows/examples and other values of interest for the analysis.

4. TASK CATEGORIZATION AND DISCUSSION

We collected 27 example scenarios from 9 participants with prior experience with working with data. We performed a thematic analysis on the textual descriptions of the scenario to obtain the results discussed in this section. Key features of interest such as task categorization and abstracted actions/goals were extracted from the anonymized raw responses as part of the analysis[3]. Fig. 4 show the occurrences of tasks decomposed from the participants description. The focus of this discussion is on advanced multi-step tasks that emerged as a common pattern in study scenarios and its implications on the PQL language design.

Simple Tasks

Task	Count
Filter	13
Aggregation	8
Groupby	6
Visualization	6
Sort	4
Advanced Modelling	3
Top-k	2

Advanced Tasks

Task	Related Approaches	Count
Find/Explain/Compare feature differences across groups		12
Ambiguous Constraint Specification		7
Visual Inspection (Preference Picking)	Crowdsourced top-k	7
Infer new column via heuristic mapping		6
Implicit Join		5
Text Feature	NLP, Mining	4
Understanding relationship between attributes	Correlation statistics	4
Visual Inspection (Understanding)	Crowd summarization, taxonomy creation	3
Find similar record		3
Ranking	Zenvisage	2
Show me interesting	Viz Recommendation	2

Figure 3: Left: Tasks that can be represented as simple one-step queries. The common usage of these operators in a variety of scenarios explains their ubiquity in SQL, visual analytics, and other tools. Right: Advanced query operations that require a composition of operators or are not found in existing systems.

4.1 Find/Explain/Compare feature differences across groups

Comparisons across groups are one of the most common tasks for this dataset. Comparison is a multi-step process often involving: 1) creation of groups, 2) creation of a corresponding "others" group to compare against, and 3) compare/computing statistics across created groups. Step one and two were done via filters, group-bys, or manual selection of tuples in the set. The third step often involves some type of aggregation over the created groups, such as computing averages or correlation among groups and visualizing the results. Scenario D2, H2 also showcased examples of how analysts seek explanations and features that could distinguish one subgroup from another.

[1] https://goo.gl/EUMQy6

[2] https://goo.gl/forms/Y3lvxQsst3VIvoBs1

[3] https://goo.gl/GNtR7y

PQL can be designed to support both steps of the group comparison task. First, PQL can better support manual creation of tuples selection sets by 1) making the group explicit, 2) summarizing selected tuples, and 3) suggesting similar tuples to add to the group. Similar to the idea in Data Tweening [13], group creation is a intermediary step that is in many cases not explicitly shown to the user. Even when group creation was performed through a top-down approach, we want to make the groups explicit by showing summaries or previews of the tuples in the group and "others", so that users can better understand (or even modify) groups that they have created.

Suggestions 2 and 3 deals with the cases where group creation is done through manual selection of tuples. Summarization of selected tuples enables users to translate their bottom-up tuple inputs into top-down specification. For example, the analysts can chose several listings that seem appealing to them, the summarization of the selected group can inform them that all of their selected tuples satisfies a constraint or match a preference. The analysts have the choice to either examine similar suggested tuples to add to the group or if they are confident that the proposed rule is captures what they are looking for, they can translate the tuple selection into a top-down formulation, which ensures that all tuples that satisfies this constraint are added to the created group. Similar formulation of mapping tuple examples to higher-level query specification have been proposed in DataPlay [2] and other works on query synthesis via I/O examples [23, 24]. However, we believe that with the more flexible preference and constraint model of filter specification (discussed in the next section) we will be able to make more realistic query filter recommendations.

Finally, the actual comparison step can be formulated as two operators in PQL:

- DISTINGUISH R_i: Find the feature(s) of record-sets R_i where the records in the selected record-set is most deviating from 'others', where 'others'= all tuples - R_i.
- COMPARE $\{R_1 ... R_n\}$: Finding feature(s) that explain why multiple record-sets are different.

One naive approach is to think of COMPARE as running DISTINGUISH on every record-set and finding the commonalities. The details of how DISTINGUISH can be done is dependent on the how similarity is defined between records, described in later sections.

4.2 Ambiguous Constraint Specification

Another common trend we found in the way participants described filtering constraints was how ambiguous their specification of the constraints were. When considering their descriptions in the form of SQL WHERE clauses, the ambiguity can come in both sides of the inequality constraints.

First, what metrics or column should one use capturing the constraint of interest? This is often done through heuristic mapping, described in the next paragraph. Second, the user needs to figure out the cutoff criterion to set for defining inclusion/exclusion in the constrained group. For example, in scenario F1,

> John is a traveler looking for a house with free parking [during](sic) his holiday in Orlando, if there's not a good match within certain limit, he would like to find one with [paid](sic) parking lot.

the limit on what is defined as "close" is unspecified.

Amongst the 13 filter specification tasks, 4 proposed using sort to ease the process of visually inspecting the records to determine the filter criterion by eye. Another way that participants determined constraints was through visually filtering through similar entries. For example, scenario I1 describes a potential homeowner trying to determine "how much money [he can]make per month given [his] current living/hosting situation". He does this by "filter[ing] out entries according to what's closest to [his] own hosting situation and look at rental prices." Additionally, an analyst can use the result of this manual filter to estimate the values of the desired criterion in the constraint. For example, the analyst in scenario H1 "wonders how much more expensive it would be to stay in a place that are cited as 'downtown' as opposed to those that do not. ". He examines "all rentals that have 'downtown' or similar in their description" and checks "if rentals [described] (sic) as 'downtown' are marked up" by "correlating it with their price and average rating/review".

Given the ubiquity of the visual filter-sort-estimate behavior in the earlier scenarios, we postulate that analysts distinguish between *constraints* and *preferences* when performing filter specifications. Constraints are strict inequalities found in SQL WHERE clauses that specifies membership of a tuple. Preferences are "nice-to-haves", but they do not necessary always need to be satisfied. In addition, preferences may not be strict inequalities, but general specifications such as "the cheaper/nicer/safer the better"(A3, I2) or "would be nice to have parking, but not too expensive"(F1). In total, we found 6 scenarios in the form of " finding 'best' or top-k record while satisfying multiple constraints and preferences". One potential direction for PQL is adding querying functionalities for specifying preference queries that can be used in conjunction with regular constraint clauses.

4.3 Visual Inspection

The role of visual inspection of data records is often understated in exploratory analysis and in some situations can not be replaced by summarized insights (visualization or aggregate statistics). Visual inspection helps user gain an overview of the data and makes the underlying inference model more transparent to the users. This is important because in order for users to make corrections to the query, the users need to be able to trust and interpret the query results and understand why a particular query was suggested and executed. Several of the scenarios from the study have motivated the idea of using human intuition gained from visually inspecting a carefully selected subset of the data (which can be results of a query) as feedback to facilitate ambiguity resolution.

One way to incorporate visual inspection is requesting the summarized insights from user after they performed visual inspections as feedback to help with the next iteration of query recommendation. For example, as an extension to preference picking, we may also consider how data types of an attribute impacts whether a user would be interested in a constraint or a preference. In the filter-sort scenarios, we find that people are more interested in first performing constraint filtering on quantitative attributes, then preference picking via visual inspection on text attribute columns. This process involves reading through listing names or reviews to make the top-k decision. In this case, the human analyst will be performing the visual top-k preference picking and returning these results to the system to continue the next step of the analysis.

The idea of using humans to perform visual inspection is similar in spirit to the work on crowdsourced operators used in databases, such as CrowdDB [6] and Deco [19]. Visual inspection is also analogous to crowdsourced algorithms for taxonomy creation and text summarization [5]. However, there are two key differences: 1) most crowdsourced databases were designed for an OLTP workload in mind where the final product of human computation workflow is data rather than analysis insights and 2) crowdsourcing algorithms often work in the high-noise, large number of responses paradigm, rather than the high-quality, sparse input that analyst would give as feedback to the system.

4.4 Infer new column via heuristic mapping

A common theme that we found across many scenarios is that analysts often have a measure of interest in mind (e.g. page views on a particular website [C2]), but the data that they are interested in is not available in the data table. In this case, participants either work with the assumption that this information exist somewhere and can be used alongside the existing information (described in the next section) or try to derive it based on existing relations. Analysts often introduce assumptions and heuristics to derive desired information from existing columns.

For example, B1 describes a market analyst studying how hosts can change rental prices to generate more revenue. She computes a new 'revenue' column by multiplying 'price per night' with the 'total number of days the listing is available per year'. This inference introduces two unchecked assumptions: 1) the listing has full occupancy on the days that it lists as available and 2) the price per night does not vary across the year. Sometimes, the inference is the best that the analysts could do based on existing data, nonetheless, inferring new data column via heuristic mapping using existing column introduces modeling assumptions that needs to be thoroughly checked for validity.

A common heuristic mapping that showed up in four scenarios is the need of extracting an additional feature column based on the textual based columns. For example, D3 describes "a social scientist who wants to analyze the behavior of couples who own [Airbnb] (sic) houses". He divides the dataset by selecting "host names that include 'and' or '&'. verify that the names in these names are male and female using binary classification." Since these existing approaches are well-supported in text-mining and NLP software packages, supporting these interactions will not be a core contribution of PQL

4.5 Implicit Join

When a measure of interest is not available to an analyst, they could also bring in data from an external table. During the study, some participants worked with the assumption that such a table was available. However, when specifying information from the hypothetical external table, none of the participants specified join as a step in the subsequent analysis gathered from both the provided and external data. This phenomena of "implicit join" performed by the analyst supports existing work that discussed how joins specification is unnatural and difficult for novice users [9, 14, 16, 17]. Since join is something analysts don't think about or assume should be done automatically, this finding points to the need for auto-join based on heuristics, such as joining on the same column names, similar/identical data columns, or based on existing primary foreign key constraints. Such features have been implemented in some DBMS management tools such as Pandas[4] and Microsoft's SQL Server Management Studio[5].

4.6 Other low-frequency tasks

We summarize other low-frequency tasks that came up in the scenarios, but will not be the focus of PQL.

Find similar records.

We started the PQL design motivated by the potential application that users may find it useful to perform bottom-up querying based on example records (i.e. find me record LIKE listing #232). However, we find that this behavior is uncommon in a standalone query situation. Out of 27 scenario, only 7 scenario had listed rows of interest. Out of those, only 3 scenarios explicitly stated that they were interested in records like the ones chosen. The rest had simply listed example records that either fully or partially satisfies the constraints they have listed, without specifying what actions to be performed on these selected tuples. In contrast, all but 2 scenarios included explicit mentioning of at least one column of interest. Four of such scenarios had incorrect selection of columns, mainly due to overselection (i.e. selecting columns that were not necessary for the scenario described). From this analysis, it is clear that for standalone querying, it is more natural for users to specify explicitly columns of interest than specifying rows of interest.

Attribute relationship/influence.

Four scenarios involved computing correlation or understanding the relationship between multiple variables for feature discovery. For simplicity, we will not consider complex relationships between variables in PQL and simply use correlation statistics to infer relationships between columns.

Ranking.

The ranking task involves enumerating over record-sets or combination of record-sets according to an objective to select the best possible choices. These objective can be similarity or other statistics. Depending on the formulation, this functionality may be covered by the suggested features for group creation, where the group is the complete record-set that the analyst is iterating over.

Show me interesting (Explore).

Only 2 out of the 27 scenarios had examples where participants did not specify concrete analysis task to support their goals. While this may be an artifact of choosing a highly-familiar dataset for the study, this result still invalidates our initial motivation for this work based on the notion that users often do not have a good idea on what they want to do when they first examine the data. Instead, we find that users generally have a reasonable high-level idea on what they want to do and the challenge is to support them in finding out how to do what they want to do. Since many proposed actions are half-baked or exploratory in nature, PQL's main goal should be to make it easier to test exploratory hypothesis and actions, even if such queries are ambiguous or underspecified, to minimize the barrier it takes to go from high-level ideas to actually testing them out.

5. ENVISIONED USE CASES

5.1 Application #1: Spreadsheet Guidance

A team of analyst is a 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 5.1 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

[4] https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.merge.html

[5] https://technet.microsoft.com/en-us/library/ms191279(v=sql.110).aspx

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.

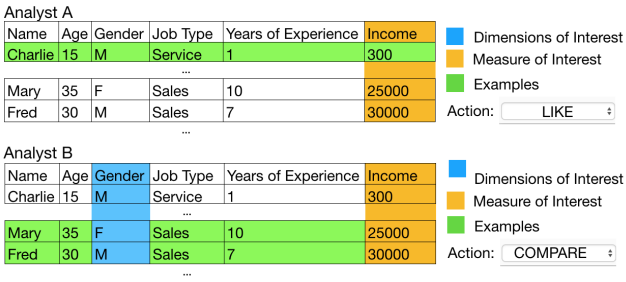

Figure 4: One example of a potential interface that could be used for submitting PQL queries. Users can select examples or columns of interest.

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 5.1 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.

5.2 Application #2: NLP Data Exploration

There has been recent interest in NLP for data exploration systems that facilitates a more natural conversational approach version of business reporting for decision makers who may not have the expertise to formulate or time to explore the data comprehensively to gain quick insights [1]. However, ambiguity is inherent to natural language queries [7, 8]. Given that PQL can tolerate ambiguity and partial specification, it can directly execute the ambiguous query formulations. Support for more ambiguity tolerant operators such as preferences, as well as incorporating the human analyst in an mixed-imitative workflow will enable PQL to address some of the ambiguous scenarios as shown in the study.

6. CONCLUSION

For the course project, I surveyed related works, provided concrete examples from real-world datasets, and conducted a formative study to motivate the design choices of the PQL language. While the high-level tasks found in the study may be an artifact of the chosen dataset, the study nevertheless serves as a proof-of-concept to the potential applicability of PQL. Given the results from the formative study and existing systems in this space, we summarize a set of desiderata for the PQL language, in order of most to least importance.

1. **Tolerant to ambiguity:** PQL requires 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 and corrected as user gain more information about the data to update the model.

2. **Feedback model based on human visual inspection:** During the study, we observed the ubiquity of the filter-sort-estimate paradigm and the importance of visual inspection during many different analysis task. To harness the power of visual inspection of record to built trust and transparency in the query recommendation, we need to design PQL operators that takes in the result of human visual inspection (such as a preference picked top-k list) as inputs. As an extension, we need to consider whether the model for feedback can enable query modification and correction for generating better query recommendation and ensure that the model of inference is easily interpretable by the users.

3. **Supporting records and columns as inputs of interest:** Given the prior work of how analyst make sense of their data [15,20], PQL must support both bottom-up example records and top-down columnar specification of interest. Given the same data output, different analysts may have different interpretation and approach the same dataset and problem. We found in the user study evidence of both top-down and bottom-up querying. However, we were surprised to learn that our initial support for the explicit "Find records LIKE <r1>" query was not a common use case. Instead, we see bottom-up querying being done implicitly in the compare-estimate step in visual inspection as well as during filter steps during group creation. We also advocate ways to bridge together these two approaches through three suggestions on how to improve group creation.

4. **Workload-independent model:** The existence of query 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.

5. **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.

The forseeable future work in this project includes designing a complete set of functions, operators, and rules for the PQL language that fits the desiderata and formulating the problem of query execution with ambiguous operators potentially used alongside conventional SQL operators.

7. REFERENCES

[1] 2018 top 10 business intelligence trends, 2017. Accessed: December 7, 2017.

[2] 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.

[3] F. Alborzi, R. Chirkova, P. Deo, C. Healey, and J. Reutter. DataSlicer : Task-Based Data Selection For Visual Data Exploration is unlikely to be helpful to those users who are not familiar.

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

[5] L. B. Chilton, G. Little, D. Edge, D. S. Weld, and J. A. Landay. Cascade: Crowdsourcing taxonomy creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1999–2008, New York, NY, USA, 2013. ACM.

[6] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: Answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 61–72, New York, NY, USA, 2011. ACM.

[7] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. Karahalios. Datatone: Managing ambiguity in natural language interfaces for data visualization. *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*, pages 489–500, 2015.

[8] E. Hoque, V. Setlur, M. Tory, and I. Dykeman. Applying Pragmatics Principles for Interaction with Visual Analytics. *IEEE Transactions on Visualization and Computer Graphics*, (c), 2017.

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

[10] 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.

[11] Z. Jin, M. R. Anderson, M. Cafarella, and H. V. Jagadish. Foofah : Transforming Data By Example. *Proceedings of the 2017 international conference on Management of Data - SIGMOD '17*, pages 683–698, 2017.

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

[13] M. Khan, L. Xu, A. Nandi, and J. M. Hellerstein. Data Tweening: Incremental Vizualizations of Data Transforms. *Proceedings of the VLDB Endowment*, 10(6):661–672, 2017.

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

[15] 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.

[16] 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.

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

[18] K. Pal and S. Michel. Learning Interesting Categorical Attributes for Refined Data Exploration. 2017.

[19] A. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: Declarative crowdsourcing. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 1203–1212, New York, NY, USA, 2012. ACM.

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

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

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

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

[24] 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.

[25] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2 : Augmenting Visual Analysis with Partial View Specifications. 2017.

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