On Answering Why and Why-not Questions in Databases

Md. Saiful Islam
Supervised By: Prof. Chengfei Liu
Swinburne University of Technology, VIC 3122, Australia
mdsaifulislam@swin.edu.au

***Abstract*—There is a growing interest in allowing users to ask questions on received results in the hope of improving the usability of database systems. This research aims at answering the so called why and why-not questions on received results w.r.t. different query settings in databases. The main goals of this research are: (i) studying the problem of answering the why and the why-not questions in databases; (ii) finding efficient strategies for answering these questions in terms of different query settings and (iii) finally, developing a framework that can take advantage of the existing data indexing and query evaluation techniques available to answer such questions in databases. We believe that the research undertaken by us can contribute towards improving the usability of traditional database systems.**

I. Introduction

After decades of efforts made by the database community, today's systems have become highly efficient in terms of both query execution time and resource usage. However, these systems are not usable for the end users to the same degree as they are proficient in underlying data management and query evaluation [16]. These days users expect systems to be more interactive and cooperative. That is, the users are not satisfied only with receiving the result from the system, but also they want to know why the system returns only the current set of objects in the result set (i.e., the current result set does not match the user's expectation). In particular, users may want to know **why** a certain data object (which is unexpected) appears in the result set and similarly, **why** a certain data object (which is expected) does **not** appear in the result set. As a next step, users may also seek appropriate explanations for these questions. Any system that can provide good explanations for the above questions can be very helpful for users to understand their information needs and also make the system more interactive and transparent to the users [27], [10], [13]. At present, our traditional database systems do not provide any kind of exploratory data analysis facilities to support the above why and why not questions.

There are different aspects of answering why and why-not questions in databases. These are: (*i*) computing provenance (also called "lineage" or "pedigree") information to describe the origins of data output and the process by which it was arrived at [4], [6] ; (*ii*) modifying the original database so that missing objects become part of the query output [12], [11] ; (*iii*) identifying the causes (e.g., operator(s) in the user submitted query) that filters the desired query output [5]; and (*iv*) modifying the query so that missing objects appear in the refined query output [27], [10]. In this research, we aim to answer why and why-not questions in light of the above aspects in different data and query settings that have not been investigated by others.

The rest of the paper is organized as follows: Section 2 describes the related research; Section 3 presents our research agendas; Section 4 outlines our contributions made so far and future research plan; and Section 5 concludes our paper.

II. Related Work

Previous studies [9], [8], [21], [12], [5], [11], [27] and [10] have investigated the problem of answering why and why-not questions in databases from different perspectives (i.e., varied data and query settings). In [9], Green et al. model provenance as semirings of polynomials which describes their direct derivations from other tuples in datalog. In [8], Glavic et al. compute provenance using query rewriting techniques to annotate tuples with lineage information in relational databases. In [21], Meliou et al. propose causality as a unified framework to describe the lineage of query answers (positive provenance) and non-answers (negative provenance). However, the above models do not separate unexpected (why) tuples from answers and also expected (why-not) tuples from non-answers.

While why questions can be addressed by applying the established why, where and how provenance techniques [6] but why-not questions have received very little attention. In [12], Huang et al. and in [11], Herschel et al. propose to modify the original tuple values in the database so that why-not (missing) tuples become part of the (SPJ and SPJUA, respectively) query output. However, this approach is not applicable where the data are trusted and unmodifiable. In [5], Chapman et al. propose to identify the culprit operator(s) that filters out the why-not (missing) tuple(s) from the query output. As a next step, Tran and Chan [27] answer why-not questions for SPJA queries through query refinement where they collect why-not (missing) tuples as feedback from the user. The authors exploit the idea of skyline queries to report the closest refined query with respect to the original one to minimize the distance between refined and original query. In [10], He et al. propose an approach to answer why-not questions on top-k queries through the modification of both k and/or weightings. Yet, before [12], [5], [11], [27] and [10], Motro [23],[24] has discussed the query modification techniques for supporting vague queries and the approaches for explaining empty answer

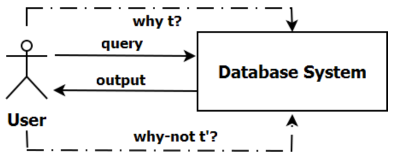


Fig. 1. Why and why-not questions in databases

to a query, respectively.

In user feedback-based query refinement techniques, only false positives (why) feedback have been emphasized in both database and information extraction areas previously [20],[19]. For example, in [20], Ma *et al.* model user feedback query refinement for both learning the structure of the query as well as learning the relative importance of query components, but they collect only false positive feedback from users. In [19], Liu et al. collect false positives (why tuples), again identified by users, to modify the initial rules in information extraction settings. Both *why* and *why-not* questions are treated separately in the above models. We believe that a more helpful explanation should be one that can model both *why* and *why-not* questions asked at the same time for the same query.

The why and why-not questions have been studied not only in databases but also in other areas ranging from software engineering to machine learning and context-aware intelligent systems. The Whyline [17] framework allows Java developers to select "why did" and "why didn't" questions to explore the causal relationships between the output and the program's execution. The Crystal application framework [25] provide explanations that answer why and why-not intelligibility questions for desktop users. Lim et al. [18] propose a model based intelligent system to explain why a system behaved a certain way, and why a system did not behave in a different way. In [26], the authors explore the premise is that, if the machine learning system could explain its reasoning to the user, the user would, in return, specify why the prediction was wrong and provide other, rich forms of feedback that could improve the accuracy of machine learning.

III. Research Problem and Terminology

In this section, we present the research problems targeted by us in our study and explain the tasks which we need to undertake for answering the so called why and why-not questions in databases. First, we define the notion of **why** and **why-not** questions. We consider the user to actively participate in the whole process of retrieving desired information from databases. That is, if the user is not satisfied with the current output returned by the system, he can provide feedback by asking "why" the system returns the data object t in the output which is unexpected to him. Similarly, he can also ask "why-not" the system returns the data object t' which is desirable or expected to him as given in Fig. 1. There are several practical examples where this kind of feedback makes excellent sense [12], [27], [10]. After getting the above why and why-not feedback from the user, we consider it as the system's responsibility to provide appropriate explanations (answers) to the user. Therefore, we define our research problem as follows:

TABLE I
Terminology used in the Paper

Term(s)	Description(s)
database	relational, graph, social, XML, RDF etc.
why-question	unexpected, false positive or undesirable data obj in output
why-not question	expected, false negative or missing data obj from output
query	SPJ, SPJA, SPJUA, NN, RQ, RSQ, graph query, social query etc.
explanation(s)	modification of data, modification of query, or a combination of both
cost-metric(s)	FPR(%), FNR(%), Precision(%), Accuracy(%), No. of subqueries (#NSUB), $\|p - p^*\|$, $\|q - q^*\|$ etc.

Problem Definition. *Given a user query Q and user feedback: why $\{t\}$ and why-not $\{t'\}$, generate an appropriate explanation exp for the user's why and why-not feedback.*

The general problem definition given above can be made specific w.r.t. different data and query settings. The queries may vary from Select-Project-Join-Union-Aggregation (SPJUA) query, nearest neighbour (NN) search, range-query (RQ), reverse skyline query (RSQ) and their variants. The database varies from relational, XML, RDF and graphs.

In our research, we aim to answer (i.e., generate explanation for) the why and why-not questions in light of the following aspects: (*i*) data modification technique; (*ii*) query modification technique and (*iii*) a hybrid (both data and query modification) technique. To judge the appropriateness of the generated explanations, we propose to use information retrieval cost metrics such as false positive rate (FPR), false negative rate (FPR), precision and accuracy improvements [22], complexity of the modified query (no. of subqueries), edit distance of the data and query object (e.g., $|p-p^*|$, $|q-q^*|$) etc. A summary of the above are given in Table I.

IV. Current Status of the Research

In this section, we present the contributions made so far and the future research directions undertaken by us.

A. Contributions

1) ***Explanation Based SPJ Query Refinement:*** In [13], we have shown how we can achieve query refinement by exploiting the explanations of the presence of unexpected tuples (why it appears) and the absence of expected tuples (why it does not appear) in the SPJ query result set. We assume that the user submitted query Q classifies the database tuples into four categories: (i) truly positive tuples $(TP) \subseteq Q(D)$; (ii) false positive tuples $(FP) \subseteq Q(D)$, (iii) truly negative tuples $(TN) \subseteq Q'(D)$ and (iv) false negative tuples $(FN) \subseteq Q'(D)$ as shown in Fig. 2. Then, we form a feedback table by inserting TP, FP and FN tuples and adding a class column as shown in Table II. The class column values of TP and FN, and FP are assigned to 'yes' and 'no', respectively. Then,

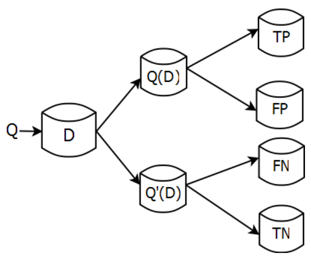


Fig. 2. Taxonomy of database objects w.r.t. user query Q

TABLE II

LABELED DATA FOR DECISION TREE LEARNING

Sname	HD	Npub	Rank	Feedback
John	Bachelor	1	1	yes
Peter	Master	2	4	no
Noah	Master	2	3	yes

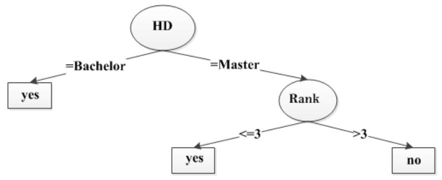


Fig. 3. Decition-Tree (DT) based tuple classifier

we use the information-gain theory based decision-tree (DT) classifier [22] to learn the refined query as shown in Fig. 3.

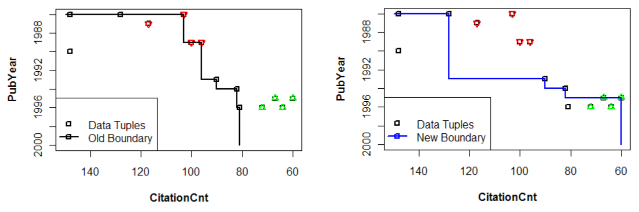
*2) **Query Refinement by Exploiting Skyline Operator**:* In [14], we propose a SPJ query refinement framework, called FlexIQ, by exploiting the skyline operator. In FlexIQ, we use feedback to discover the query intent of the user. We then exploit the skyline operator to confine the search space of the proposed algorithms. The main idea of our approach is given as follows: (i) Firstly, we collect both unexpected and expected data objects as feedback from the user; (ii) Then, we analyze the user feedback to find a minimal representation; (iii) Then, we find the boundary that separates answers (i.e., $Q(D)$) from non-answers (i.e., $Q'(D)$), as shown in Fig. 4(a). Then, we form a new boundary that excludes unexpected data objects and includes expected data objects in the refined query output, $Q^f(D)$, as shown in Fig. 4(b); (iv) Finally, we offer a baseline algorithm (TBA) and a trade-off algorithm (TOA) (which is parametrized by δ) for constructing the refined query Q^f in relation to the new boundary. We have demonstrated the effectiveness of FlexIQ by comparing its performance with the decision-tree (DT) based query refinement for DBLP datasets[1]. We conduct nine different experiments by setting δ to nine different values from 0.65 to 0.99 for TOA. The results are shown in Table III and Table IV.

TABLE III

FLEXIQ: AVERAGE FPR(%) AND FNR(%)

Expr.	TBA		TOA		DT	
	FPR(%)	FNR(%)	FPR(%)	FNR(%)	FPR(%)	FNR(%)
Exp#1 $(\delta{=}0.65)$	0.00	0.00	26.27	06.39	09.72	30.47
Exp#2 $(\delta{=}0.70)$	0.00	0.00	20.94	06.75	09.66	29.20
Exp#3 $(\delta{=}0.75)$	0.00	0.00	15.32	07.71	08.99	25.97
Exp#4 $(\delta{=}0.80)$	0.00	0.00	11.25	06.83	08.79	30.05
Exp#5 $(\delta{=}0.85)$	0.00	0.00	05.74	06.19	14.34	27.99
Exp#6 $(\delta{=}0.90)$	0.00	0.00	02.18	04.54	09.57	30.74
Exp#7 $(\delta{=}0.95)$	0.00	0.00	00.38	01.35	09.79	28.59
Exp#8 $(\delta{=}0.97)$	0.00	0.00	00.11	01.23	09.54	30.47
Exp#9 $(\delta{=}0.99)$	0.00	0.00	00.00	00.04	08.06	28.96

*3) **Answering Why-not Questions in Reverse Skyline Queries**:* A reverse skyline query, RSQ, retrieves all points that contain the query point q in their dynamic skylines [7] and is very important for retrieving information from companies' perspectives. As with top-k and other queries, answering why-not questions in reverse skyline queries also has several practical applications. In this work [15], we study the problem of answering why-not question in reverse skyline queries. We



(a) (b)

Fig. 4. (a) Data Objects and Old boundary (b) Data Objects and New boundary: unexpected and expected data objects are marked with lower and upper triangles, respectively

TABLE IV

FLEXIQ: AVERAGE ACC(%) AND #NSUB

Expr.	TBA		TOA		DT	
	ACC(%)	#NSUB	ACC(%)	#NSUB	ACC(%)	#NSUB
Exp#1 $(\delta{=}0.65)$	100.00	13.04	77.05	1.35	78.19	2.84
Exp#2 $(\delta{=}0.70)$	100.00	12.93	81.33	1.66	78.23	2.86
Exp#3 $(\delta{=}0.75)$	100.00	13.01	86.00	2.05	81.80	2.87
Exp#4 $(\delta{=}0.80)$	100.00	13.04	89.57	2.63	78.09	2.88
Exp#5 $(\delta{=}0.85)$	100.00	12.94	93.85	3.95	78.47	2.92
Exp#6 $(\delta{=}0.90)$	100.00	13.13	97.04	5.02	78.19	2.94
Exp#7 $(\delta{=}0.95)$	100.00	12.97	99.32	6.70	78.41	2.91
Exp#8 $(\delta{=}0.97)$	100.00	12.97	99.67	7.03	78.59	2.82
Exp#9 $(\delta{=}0.99)$	100.00	12.82	99.99	7.22	78.28	2.91

also outline the semantics of answering why-not questions in reverse skyline queries. In connection with this, we show how to modify the why-not point p into p^* and the query point q into q^* to include the why-not point in the reverse skyline of the query point. We then show, how a query point q can be positioned safely anywhere within a region (i.e., called safe region, $SR(q)$) without losing any of the existing reverse skyline (RSL) points. We propose three different techniques to answer why-not question in RSQ: (i) modifying only the why-not point (MWP); (ii) modifying only the query point (MQP); and (iii) modifying both the query point and the why-not point (MWQ) considering $SR(q)$. We have also compared the cost (according to Eqn. 1) of the best solution received by each method to demonstrate their effectiveness in CarDB[2] dataset. The result is shown in Table V[3].

$$\begin{aligned} cost(q^*,p^*) &= cost(q,q^*) + cost(p,p^*) \\ &= \alpha\cdot \mid q - q^* \mid + \beta\cdot \mid p - p^* \mid \\ &= \sum_{i=1}^{d} \alpha_i \times \mid q^i - q^{*i} \mid + \sum_{i=1}^{d} \beta_i \times \mid p^i - p^{*i} \mid \end{aligned} \tag{1}$$

where, the cost of moving q within $SR(q)$ is zero, i.e., $cost(q,q^*) = 0$ if $q^* \in SR(q)$.

B. Future Work

We are currently studying the problem of answering why and why-not questions in other data settings including range queries, location-based services and social networks. In particular, we are working on the following types of queries to answer the why and why-not questions.

*1) **Range Queries**:* A range query retrieves all data points that are within a certain range e from a given query point Q. Due to the high computational cost, an approximate range search are proposed by the researchers as follows: "Given a bounded range Q of diameter w and $\epsilon > 0$, an approximate

[1]downloaded from http://dblp.unitrier.de/xml/

[2]downloaded from autos.yahoo.com

[3]A small difference is significant.

TABLE V
WHY-NOT RSQ: QUALITY OF RESULTS IN CARDB

Queries	MWP	MQP	MWQ
$q_1, \|RSL(q_1)\| = 2$	0.274162197	0.966525902	0.270083225
$q_2, \|RSL(q_2)\| = 3$	0.371958746	1.280325545	0.339225107
$q_3, \|RSL(q_3)\| = 4$	0.230578336	1.10082634	0.211690534
$q_4, \|RSL(q_4)\| = 5$	0.038138603	0.243090389	0.02951842
$q_5, \|RSL(q_5)\| = 6$	0.135468879	1.413703538	0.135450054
$q_6, \|RSL(q_6)\| = 7$	0.089828649	0.642190348	0.089768971
$q_7, \|RSL(q_7)\| = 8$	0.028030879	0.210567316	0.028030879
$q_8, \|RSL(q_8)\| = 9$	0.079989249	0.542973513	0.079284723
$q_9, \|RSL(q_9)\| = 10$	0.044557559	0.400313076	0.033984709
$q_{10}, \|RSL(q_{10})\| = 12$	0.065886539	0.696464647	0.065886539
$q_{11}, \|RSL(q_{11})\| = 13$	0.009078408	0.26180871	0.009078408
$q_{12}, \|RSL(q_{12})\| = 14$	0.07612146	0.959489884	0.07612146
$q_{13}, \|RSL(q_{13})\| = 15$	0.007091629	0.254541311	0.007085752

range query treats the range as a fuzzy object meaning that points lying within distance ϵw of the boundary of Q either may or may not be counted" [1]. However, the user has to mention two parameters here, which could be difficult for a naive user to define. Therefore, inappropriate definition of the user given parameter may result in missing some expected data points. Similarly, it may include some unexpected data points. We believe that why and why-not feedback in this type of query may be found to be effective to correct the initial user given diameter w and parameter ϵ.

*2) **Nearest Neighbor Queries**:* A nearest neighbor (NN) query retrieves neighbor objects to a given point in space [2]. Nearest-neighbor queries are an important query type for application areas such as sensor databases, location based services, GIS etc. A kNN query retrieves k neighbors to a given point. For example, someone is looking for a few good restaurants nearby. In this case, a kNN query is obviously better than an NN query. However, for kNN query a user has to mention the value of k. The query may not return what a user is looking for if the wrong value of k is chosen. Therefore, answering why-not questions in this type of queries makes excellent sense. There are also many NN search based applications where a user might find it beneficial to receive answers for why and why-not questions from systems.

*3) **Social and Graph Queries**:* Due to the emergence of social networking websites and their enormous influence in our day-to-day life, there is an urgency for social queries on such networks. Social networks data are generally represented as graphs in databases [28]. Many social networking websites generate automated recommendation on many items for their users such as suggestions on new friendships, events etc. Therefore, why and why-not feedback on such automated recommendation make excellent sense if the user does not agree with them always. We believe any social networking websites that can answer the so called why and why-not questions will be more interesting to their users. There are also many applications of graph queries in other databses such as PPI or gene-regulatory networks [3], chemical structures [29] etc. Answering why and why-not questions can also be beneficial in these applications to match graphs with distortions.

V. CONCLUSIONS

This paper presents the research agendas for answering why and why-not questions in databases. We have shown why it is worth conducting such research and outlined the different aspects of answering such questions in databases. We have also summarized the related work done in this area and the future research agendas. Finally, we have also presented our contributions made so far and the results. We are currently working on future research problems mentioned in this paper.

REFERENCES

[1] S. Arya and D. M. Mount. Approximate range searching. *Comput. Geom.*, 17(3-4):135–152, 2000.
[2] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *ICDT*, pages 217–235, 1999.
[3] G. Blin, F. Sikora, and S. Vialette. Querying graphs in protein-protein interactions networks using feedback vertex set. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 7(4):628–635, 2010.
[4] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.
[5] A. Chapman and H. V. Jagadish. Why not? In *SIGMOD Conference*, pages 523–534, 2009.
[6] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Found. and Trends in Databases*, 1(4):379–474, 2009.
[7] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, pages 291–302, 2007.
[8] B. Glavic and G. Alonso. The perm provenance management system in action. In *SIGMOD Conference*, pages 1055–1058, 2009.
[9] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.
[10] Z. He and E. Lo. Answering why-not questions on top-k queries. In *ICDE*, pages 750–761, 2012.
[11] M. Herschel and M. A. Hernández. Explaining missing answers to spjua queries. *PVLDB*, 3(1):185–196, 2010.
[12] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.
[13] M. S. Islam, C. Liu, and R. Zhou. On modeling query refinement by capturing user intent through feedback. In *Australasian Database Conference*, volume 124, pages 11–20, 2012.
[14] M. S. Islam, C. Liu, and R. Zhou. User feedback based query refinement by exploiting skyline operator. In *ER*, pages 423–438, 2012.
[15] M. S. Islam, R. Zhou, and C. Liu. On answering why-not questions in reverse skyline queries. In *ICDE*, 2013.
[16] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD Conference*, pages 13–24, 2007.
[17] A. J. Ko and B. A. Myers. Finding causes of program output with the java whyline. In *CHI*, pages 1569–1578, 2009.
[18] B. Y. Lim, A. K. Dey, and D. Avrahami. *Why and why not* explanations improve the intelligibility of context-aware intelligent systems. In *CHI*, pages 2119–2128, 2009.
[19] B. Liu, L. Chiticariu, V. Chu, H. V. Jagadish, and F. Reiss. Automatic rule refinement for information extraction. *PVLDB*, 3(1):588–597, 2010.
[20] Y. Ma, S. Mehrotra, D. Y. Seid, and Q. Zhong. Raf: An activation framework for refining similarity queries using learning techniques. In *DASFAA*, pages 587–601, 2006.
[21] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.
[22] T. M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.
[23] A. Motro. Vague: A user interface to relational databases that permits vague queries. *ACM Trans. Inf. Syst.*, 6(3):187–214, 1988.
[24] A. Motro. Flex: A tolerant and cooperative user interface to databases. *IEEE Trans. Knowl. Data Eng.*, 2(2):231–246, 1990.
[25] B. A. Myers, D. A. Weitzman, A. J. Ko, and D. H. Chau. Answering why and why not questions in user interfaces. In *CHI*, pages 397–406, 2006.
[26] S. Stumpf, V. Rajaram, L. Li, W.-K. Wong, M. M. Burnett, T. G. Dietterich, E. Sullivan, and J. L. Herlocker. Interacting meaningfully with machine learning systems: Three experiments. *Int. J. Hum.-Comput. Stud.*, 67(8):639–662, 2009.
[27] Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *SIGMOD Conference*, pages 15–26, 2010.
[28] D.-N. Yang, Y.-L. Chen, W.-C. Lee, and M.-S. Chen. On social-temporal group query with acquaintance constraint. *PVLDB*, 4(6):397–408, 2011.
[29] X. Zhao, C. Xiao, X. Lin, and W. Wang. Efficient graph similarity joins with edit distance constraints. In *ICDE*, pages 834–845, 2012.