Short paper

Explaining Missing Query Results in Natural Language

Daniel Deutch
Tel Aviv University
danielde@post.tau.ac.il

Nave Frost
Tel Aviv University
navefr@mail.tau.ac.il

Amir Gilad
Tel Aviv University
amirgilad@mail.tau.ac.il

Tomer Haimovich
Tel Aviv University
tomerh@mail.tau.ac.il

ABSTRACT

We propose in this paper a novel approach for explaining query non-answers in Natural Language within the context of Natural Language Interfaces to Databases (NLIDBs). Such interfaces allow non-expert users to pose queries over an underlying database; our goal is to further allow users to ask why some results that they have expected to see, are missing from the output. In a nutshell, our approach is to "marry" NLIDBs with an existing model for explaining missing query results by pinpointing the last query operator that is "responsible" for the missing result. We observe that one can often trace the parts of the original NL question that correspond to these operators. This paves the way for intuitive explanations of the non-answers, that are based on highlighting the relevant parts of the question. Our architecture is generic and is not coupled with a specific NLIDB, and our solution yields clear explanations in interactive speed.

1 INTRODUCTION

Natural Language (NL) interfaces to database systems are often used as easy-to-understand gateways for accessing complex databases [1, 8, 13]. The rise of NL interfaces allows non-experts to access and query complex databases, without writing formal queries or understanding execution plans.

Yet often, the answers returned by such queries do not quite match the expectations of the users who formed them. Users are then faced with the problem of understanding the gap between their expectations and the result. Previous work has dealt with presenting the users the reason for the presence of a certain tuple in the result set in a manner that does not require technical proficiency [5, 6]. When users ask about a *missing* tuple, however, a different form of explanation is required. Explaining missing tuples, or non-answers, is termed why-not provenance and has been the focus of multiple previous works [2, 3, 10]. Such information is crucial for understanding the result, debug and improve the query and/or the input database. However, all of these works have provided this information in the form of *internal representation*, not suitable for non-experts.

Rel. *author*

aid	aname	oid
1	Marge	1
4	Bart	1

Rel. *pub*

pid	cid	jid	ptitle	pyear
5	11	-	"Paper x"	2001

Rel. *conf*

cid	cname	dname
11	DBDonut	databases

Rel. *writes*

aid	pid
1	5

Figure 1: MAS database instance

Example 1.1. Assume the NL question depicted in Figure 2a which was correctly translated by an NL interface to a formal query (shown as SQL in Figure 2b) and executed on the database instance depicted in Figure 1. The user is presented with a result set, but is surprised to see that Marge is missing. A why-not explanation for this missing tuple could be the predicate `pub.pyear > 2005` filtering Marge out, or a modified formal query without this predicate. However, understanding such explanations requires, at the very least, SQL knowledge.

```
return authors who published papers in database
conferences after 2005
```

(a) NL Question

```
SELECT DISTINCT author.aname
FROM author, writes, pub, conf,
WHERE conf.domain = `databases'
    AND pub.pyear > 2005
    AND author.aid = writes.aid
    AND writes.pid = pub.pid
    AND pub.cid = conf.cid
```

(b) SQL Query

```
return authors who published papers in database
conferences after 2005
```

(c) Word highlight explanation for "Why not Marge?"

Figure 2: NL query, SQL translation, and our why-not explanations

In this paper, we aim to provide explanations for non-answers through natural language. The setting is unique in the sense that the query is given in NL and the user is not familiar with the technical details of the query execution, and the explanation should be tailored to bridge this knowledge gap. We rely on the frontier picky model [3] to provide explanations to non-expert users. Our main observation is that, if we use this model, when the original query was given in NL, we can in many cases trace back the responsible query operator to the part of the NL query that corresponds to it.

Example 1.2. For the NL query in Figure 2a, the SQL in Figure 2b, and the why-not query "why not Marge?". If we employ the mapping from the words in the NL query to the SQL one we can find that the words "after 2005" are connected to the operator `pub.pyear > 2005`, and reveal that these words in the NL query caused the removal of the result Marge.

To the best of our knowledge, presenting why-not provenance to non-experts was not previously studied. This form of explanations is fundamentally different from existing models that show SQL operators or other technical representations, as it allows users without technical knowledge to understand the gap between their expected result and the one they received. We claim that such explanations are of even greater importance in the context of NLIDBs, because of the cumulative errors that arise in such systems. In this setting, users have to specify their intent in a form of an NL sentence; a failure to specify certain conditions or a too specific sentence might result in tuple loss, simply because the user performed an error in the NL formulation. Since the user has no means of viewing or understanding the SQL query, this error may go unnoticed.

The solution is based on word highlighting in the original sentence form: we highlight the reason for the missing tuples, manifested as one or more words in the original NL query.

Example 1.3. In our running example, an operator-based why-not model would return the selection operator filtering tuples before 2005 as the explanation, i.e. `pub.pyear > 2005`. Using our system, the user will be shown her original NL query, with an emphasis on "after 2005" as the relevant words that caused this absence (see Figure 2c). This enables the user to better understand the query, validate the translation process and the credibility of the database, and reformulate her NL query accordingly.

We have implemented our solution and demonstrated it [7]. We now provide an experimental evaluation, showing multiple use-cases where the system provides useful explanations and showing that generating such explanations does not incur substantial time cost compared to the other steps in the computation.

Related Work: NLIDBs are aimed at bridging the gap between database systems that use formal query languages such as SQL for interaction, and users who are not experts in forming formal queries, yet posses domain knowledge [1, 8, 13, 16]. These interfaces allow users to form questions in English, and be presented with a set of results which satisfy the query. The whole process of converting the NL question to a formal SQL query can be treated by the users as a black box. Explaining query results that were returned in NL has been the focus of previous work [4].

There are multiple approaches and models for explaining why a certain piece of information that is expected to be returned from a query was actually discarded. We can broadly divide the approaches by their type.

(1) *Operator-based explanation* models (e.g. [2, 3]) aim to provide one or more query operators that are responsible for omitting a tuple in the query execution process.
(2) *Query modification* models (e.g. [9, 17]) try to broaden the given query to include the missing tuple. Depending on the model and the query, the change to the query may be as minor as replacing a constant, or even modifying the query completely by joining other tables etc.
(3) *Tuple modification / generation* models (e.g. [10, 11]) create or modify factoids in a way that will ensure that the required tuple will be returned from the query evaluation.

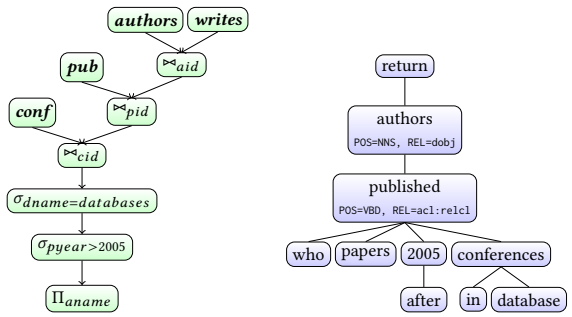
We focus here on explaining non-answers using the frontier picky model showing the "last" responsible operator [3]. To the best of our knowledge, no previous work in this field has dealt with the challenge of explaining missing tuples to non-expert users.

2 MODEL

In this section we review necessary notions in Natural Language Processing, formal queries and relevant provenance models.

From Natural Language to Formal Queries: This work relies on an NL interface named NaLIR [13] for translating English questions to SQL queries. The translation utilizes a data structure called *query dependency tree*, designed for conveying the relations between words in a sentence and their syntactic roles. A dependency tree $T = (V, E, L)$ is a node-labeled tree where labels consist of two components, as follows: (1) Part of Speech (*POS*): the syntactic role of the word [12] ; (2) Relationship (*REL*): the grammatical relationship between the word and its parent in the dependency tree [14].

After the translation phase, an SQL query Q is being generated along with a *query plan*. The query plan is a directed tree $T_Q = (V_Q, E_Q)$, where V_Q is the set of query operators in Q and the database table names which Q takes as input, and E_Q is a set of directed pairs (u, v). Table names form the leaves of T_Q and an edge (u, v) indicates that during the evaluation of Q over the database, all tuples outputted by the operator u are inputted to the operator v. Finally, the output implied by T_Q is the output of Q. Importantly, NaLIR maps the words in the NL query to their respective operators in the generated query plan. Reversing this mapping and augmenting it allows us to map why-not provenance back to NL.



(a) Query Plan **(b) NL query dependency tree**

Figure 3: Query plan and dependency tree

Definition 2.1. Given an NLIDB, a dependency tree $T_d = (V_d, E_d, L_d)$ and a query plan $T_Q = (V_Q, E_Q)$, a *operator-to-word mapping* is a partial function $M_{ops} : V_Q \rightarrow V_d$, where M_{ops} is the reverse of the mapping created by the NLIDB during the mapping from NL query to SQL.

Example 2.2. Consider the MAS database instance in Figure 1, and the NL query in Figure 2a. NaLIR parses this sentence to create the dependency tree shown in Figure 3b and returns the SQL query shown in Figure 2b; its query evaluation plan depicted in Figure 3a. The operator-to-word mapping, M_{ops}, includes the mappings: $M_{ops}[\sigma_{pyear>2005}]$ = "2005", $M_{ops}[\sigma_{dname=databases}]$ = "$database$", and $M_{ops}[\Pi_{aname}]$ = "$authors$".

Why-not provenance: We start by defining a *why-not question* WNQ. Intuitively each hypothetical output tuple of Q that matches the criteria of WNQ is a tuple whose non-existence in the answer we wish to explain.

Definition 2.3. Given an database instance D and an SQL query Q, a why-not question WNQ is an SQL query such that (1) Q and WNQ have the same result schema, and (2) $Q(D) \cap WNQ(D) = \emptyset$.

The why-not question might be more complex than the initial query, as long as the query it represents has the same result schema. This allows the user to pose constraints on other attributes as well.

Example 2.4. Reconsider the database instance in Figure 1 and the SQL query in Figure 2b. Assume the user expects to see the author Marge in the result set, and surprised when she is not a part of the answer. The user may form a why-not question, "Why not Marge?", which would be translated by NaLIR to the query:

```
SELECT DISTINCT author.aname FROM author
WHERE author.aname = 'Marge'
```

The query is over the projected attribute of the original query (author names), and that the author Marge is returned by WNQ.

Some why-not provenance models allow users to define questions over attributes not in the result schema. We focus here on

the subset of queries over the projected attributes, as we think that it is the most natural approach for posing why-not questions.

We will focus on *frontier picky* model [3]. This model for why-not provenance focuses on explaining non-answers using a single query operator. Intuitively, the last operator to contain in its input a tuple matching the why-not predicate. The motivation for explaining tuple loss with the frontier picky operator is that the answer is compact, easy-to-understand and provides a real value in the sense that it is necessary to modify the returned operator in order to include the tuple in the result list.

Definition 2.5 (adapted from [3]). Given a database D, a query Q with its query plan $T_Q = (V_Q, E_Q)$, a WNQ, and a tuple $t \in WNQ(D)$, a *picky operator* w.r.t. t is a node $v \in V_Q$ that gets t (or a predecessor of t in the evaluation process) through one of its incoming edges, as its input, and does not output it through its outgoing edge. An operator $v \in V_Q$ is *frontier picky* if:

(1) v is a picky operator for at least one tuple in $WNQ(D)$.
(2) There is no tuple $t \in WNQ(D)$ in the input of any operator $u \in V_Q$ such that u is a successor of v in T_Q.

According to this model, the answer to a why-not question is the frontier picky operator. Notice that the frontier picky operator might be different even if for the same query, evaluated with different query plans, as it depends on the structure of the query plan, i.e., the ordering of the operators in the plan.

Example 2.6. Consider the SQL query in Figure 2b and its query plan in Figure 3a, with the WNQ from Example 2.4. If Marge has not published papers after 2005, the operator node $\sigma_{pyear>2005}$ is picky w.r.t. the tuple that contains *Marge*. As all successors of this operator do not contain a tuple with $aname = Marge$, the node $\sigma_{pyear>2005}$ is the frontier picky operator w.r.t. this tuple.

3 HIGHLIGHT ALGORITHM

Our approach is composed of two stages: find the frontier picky operator for every removed tuple, and, given a why-not questions, find the relevant words that correspond to the frontier picky operator of the tuple in question.

Provenance-Aware Query Evaluation: As a first step, we evaluate the query while storing why-not provenance. We start by translating the NL query to a formal one, via NaLIR [13] augmented so that we keep track of which word in the original NL query has been mapped to which operator of the formal query, as done in [4]. The reverse of this mapping is stored in the data structure M_{ops}. Then, the query is evaluated. During evaluation, whenever a tuple is removed (due to a selection operator or as part of a filtering join), we update the mapping M_{filter}, which maintains the relation between the query operators and the tuples that were removed by them.

Example 3.1. Reconsider the NL query translated by NaLIR to the query in Figure 2b. First, we store the operator-to-word mapping between the query operators and their respective words in the original NL query, shown in Example 2.6. During evaluation, M_{filter} includes intermediate tuples such as (Marge, Paper x, 2001) and its respective picky operators $\sigma_{pyear>2005}$ (additional attributes, such as *pid*, are omitted for brevity).

Finding Relevant Words to Answer Why-Not: After viewing the evaluation results, the user now formulates a why-not query in NL. Algorithm 1 gets as input the results of the evaluation, i.e., the result, $Q(D)$, the mapping M_{filter}, the mapping M_{ops}, the why-not query formulated in NL Q_{WN}, the dependency tree of the NL query T_d, and the database D. Its output is the set of word indices that correspond to the reason for excluding the tuple of interest. Algorithm 1 operates as follows. It uses a sub-mechanism of NaLIR to convert the NL why-not query Q_{WN} into a formal why-not selection query WNQ (line 1). In lines 2–3 it checks whether WNQ is valid, i.e. if there are indeed no output tuples satisfying both the why-not query and the original query (this is a sort of sanity check). If this is not the case, it finds the frontier picky operator (line 4) for each of the tuples satisfying the formal why-not query in WNQ and returns the last of them which is the frontier picky operator w.r.t WNQ. We may get NULL as the operator in the case where there is no match to WNQ in the input; in this case we consider the last projection operation that took place to be the "reason" (lines 5–6). For instance, if someone asked about "why not Krusty?", who is not an author in the example database, the reason would be Π_{aname} which is mapped to the *author* table.

We then use the mapping outputted by the evaluation process to trace back the words corresponding to the operator. The *map*() function exploits the data structure M_{ops}, and gets as input variable names, values or table names that can be mapped back into words in the original query. Lines 13-18 are used to help link back the join operators which could not be directly mapped to words. The main idea is to exploit the typical scenario where unmapped joins are used as means for the query generation engine to link two other relations, which are directly referenced in the NL query. The algorithm traverses the ancestor and successor join operations of the given join operator (by repeatedly calling *GetJoinedRelations*()), until two operators, one ancestor and one successor of the join, which can be mapped into words $(left, right)$, are found. The returned indices are not of these two words, but rather of the words between $left$ and $right$ in T_d (by calling *GetPath*()), corresponding to the intuition that the answer to the why-not query is an unmapped operator between the two mapped operators.

Example 3.2. Consider the NL query in Figure 2a, its SQL query plan in Figure 3a, and the database in Figure 1. The author Bart exists in the *author* table, but has no published papers. Therefore, the frontier picky operator for the NL why-not query "why not Bart?" is the join operator $\bowtie_{aid}$ (see Figure 3a), connecting the *author* table with the *writes* table, which is a join table, containing author ids and publication ids. Algorithm 1 will first convert Q_{WN} to its SQL form and check that Bart is indeed not in the result set of Q in Figure 2b (lines 1–3). It will then get the frontier picky operator $\bowtie_{aid}$, depicted in Figure 3a, which took the tuple (4, *Bart*, 1) as input and did not output it (line 4). Since the output of this line is not NULL, it will continue to line 9. The *writes* table cannot be mapped into a word in the NL query in Figure 2a because the word *writes* does not even appear in it. The reason for the join operation being included is that NaLIR has used this table as a link between the authors table and the papers table. Lines 13-18 are then used to trace back the joined relations. Left and right would be "authors" and "papers" respectively, and the returned path would include only the word "published" as this is the word connecting "authors" and "papers" as seen in the dependency tree of the NL query in Figure 3b, thus our word highlight answer is "return authors who **published** papers in database conferences after 2005".

Algorithm 1: Highlight

input : $Q(D), M_{filter}, M_{ops}, Q_{WN}, T_d$, and D
output: Word highlight set

```
WNQ = NLToFormal(Q_WN);
if Q(D) ∩ WNQ(D) ≠ ∅ then
    return ∅;
op ← FrontierPicky(M_filter.find(WNQ(D)));
if op = NULL then
    op ← GetProjectionOperator(M_filter);
if op is σ_{A=x} then
    return map(A, M_ops) ∪ map(x, M_ops);
if op is T ⋈ R where R is a new input table then
    if map(R, M_ops) ≠ ∅ then
        return map(R, M_ops);
    else
        (left, right) ← GetJoinedRelations(R);
        while map(left, M_ops) = ∅ do
            (left, _) ← GetJoinedRelations(left);
        while map(right, M_ops) = ∅ do
            (_, right) ← GetJoinedRelations(right);
        return GetPath(left, right, T_d);
```

4 IMPLEMENTATION AND EXPERIMENTS

We describe various use cases and our scalability evaluation.

Use cases:

Table 1: Sample of Natural Language queries along with Why-Not questions and "Why-Not answers"

ID	NL Query With Highlight Explanation	Why-Not Question
	Selection Frontier Picky Operator	
1	Return authors who published in database conferences after **2015**	Catriel Beeri
2	Return authors who published in **database** conferences after 2015	Yishay Mansour
3	Return authors from **"Tel Aviv University"** who published in VLDB	Benny Kimelfeld
4	Return authors from "Tel Aviv University" who published in **VLDB**	Yishay Mansour
5	Return organizations of authors who wrote in **database** journals	AI University (org. without DB authors)
6	Return papers of authors in **"Artificial Intelligence"** after 2005 and before 2007	Active Views for Electronic Commerce
7	Return papers of authors in "Artificial Intelligence" after **2005** and before 2007	Stochastic Link and Group Detection
8	Return authors from **"North America"** who presented in VLDB in 2000	Tova Milo
9	Return authors from "North America" who presented in **VLDB** in 2000	Geoffrey E. Hinton
10	Return authors from "North America" who presented in VLDB in **2000**	Christopher Ré
11	Return publications about **graphics** after 2005	Overflow Controled SIMD Arithmetic
	Join Frontier Picky Operator	
12	Return organizations of **authors** who wrote in database journals	MadeUp College (org. without authors)
13	Return authors who **published** in database conferences	John Doe (author without publications)

Table 1 depicts representative examples of NL queries along with relevant why-not questions that were executed on the MAS database [15]. The first 11 examples demonstrate cases in which the reason for the tuple absence was a selection operator. Queries 12 and 13 demonstrate the operation of Algorithm 1 when the frontier picky operator is a join operation, this is often an indication of missing tuples in the dataset. Overall we can see that for the vast majority of NL queries and why-not questions the explanations supply valuable information that justify in concise manner the absence of the tuples in question.

Scalability: Here again we have used the MAS database whose total size is 4.7 GB, and queries 1–11 from Table 1, running the algorithm to generate word highlight and NL explanation. The computation steps execution times, for each query, are depicted in Figure 4. The computation times are given in nanoseconds and the y axis is log-scaled. As evident from the graph, most of the time is spent on NL to SQL conversion, query evaluation and identifying the relevant tuples for the why-not queries. Generating the why-not explanations incurs a negligible performance cost (less than a millisecond on average for selection frontier picky operators), and thus provides an interactive experience for the user.

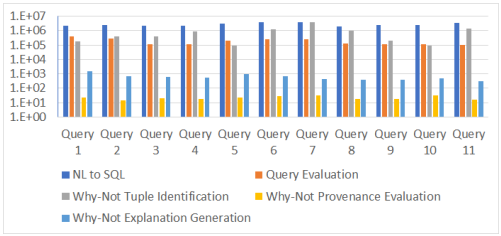


Figure 4: Average computation time by step (log scale, values in nanoseconds)

Acknowledgements. This research has been funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 804302), the Israeli Science Foundation (ISF) Grant No. 978/17, and the Google Ph.D. Fellowship. The contribution of Amir Gilad is part of a Ph.D. thesis research conducted at Tel Aviv University.

REFERENCES

[1] 2002. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE*.
[2] Nicole Bidoit, Melanie Herschel, and Aikaterini Tzompanaki. 2015. Efficient Computation of Polynomial Explanations of Why-Not Questions. In *CIKM*.
[3] Adriane Chapman and H. V. Jagadish. 2009. Why not?. In *SIGMOD*. 523–534.
[4] Daniel Deutch, Nave Frost, and Amir Gilad. 2016. NLProv: Natural Language Provenance. *PVLDB* 9, 13 (2016), 1537–1540.
[5] Daniel Deutch, Nave Frost, and Amir Gilad. 2017. Provenance for Natural Language Queries. *PVLDB* 10, 5 (2017), 577–588.
[6] Daniel Deutch, Nave Frost, and Amir Gilad. 2019. Explaining Natural Language query results. *The VLDB Journal* (2019).
[7] Daniel Deutch, Nave Frost, Amir Gilad, and Tomer Haimovich. 2018. NLProveNAns: Natural Language Provenance for Non-Answers. *PVLDB* 11, 12 (2018), 1986–1989.
[8] Enrico Franconi, Claire Gardent, Ximena I Juarez-Castro, and Laura Perez-Beltrachini. 2014. Quelo Natural Language Interface: Generating queries and answer descriptions. In *NLIWOD*.
[9] Z. He and E. Lo. 2014. Answering Why-Not Questions on Top-K Queries. *IEEE Transactions on Knowledge and Data Engineering* 26, 6 (2014), 1300–1315.
[10] Melanie Herschel and Mauricio A. Hernández. 2010. Explaining Missing Answers to SPJUA Queries. *Proc. VLDB Endow.* 3, 1-2 (2010), 185–196.
[11] Jiansheng Huang, Ting Chen, AnHai Doan, and Jeffrey F. Naughton. 2008. On the Provenance of Non-answers to Queries over Extracted Data. *Proc. VLDB Endow.* 1, 1 (2008), 736–747.
[12] Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. In *Annual Meeting on Association for Computational Linguistics*.
[13] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *PVLDB* 8, 1 (2014), 73–84.
[14] M. Marneffe, B. Maccartney, and C. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In *LREC*.
[15] MAS. 2016. http://academic.research.microsoft.com/.
[16] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a Theory of Natural Language Interfaces to Databases. In *IUI*. 149–157.
[17] Quoc Trung Tran and Chee-Yong Chan. 2010. How to ConQueR Why-not Questions. In *SIGMOD*. 15–26.