On Modeling Query Refinement by Capturing User Intent through Feedback

Md. Saiful Islam, Chengfei Liu and Rui Zhou
Faculty of Information and Communication Technologies
Swinburne University of Technology
VIC 3122, Australia
{mdsaifulislam,cliu,rzhou}@swin.edu.au

Abstract

SQL queries in relational data model implement the binary satisfaction of tuples. Tuples are generally filtered out from the result set if they miss the constraints expressed in the predicates of the given query. For naïve or inexperienced users posing precise queries in the first place is very difficult as they lack of knowledge of the underlying dataset. Therefore, imprecise queries are commonplace for them. In connection with it, users are interested to have explanation of the missing answers. Even for unexpected tuples present in the result set advanced users may also want to know why a particular piece of information is present in the result set. This paper presents a simple model for generating explanations for both unexpected and missing answers. Further, we show how these explanations can be used to capture the user intent via feedback specifically for refining initial imprecise queries. The presented framework can also be thought as a natural extension for the existing SQL queries where support of explanation of expected and unexpected results are required to enhance the usability of relational database management systems. Finally, we summarize future research directions and challenges that need to be addressed in this endeavour.

Keywords: Imprecise Query, Explanation, Point Domination Theory, User Feedback, Query Refinement.

1 Introduction

Relational Data Model is one of the widely used data model for storing and retrieving information. The underlying query engine accepts request from users through SQL by which the non-answers are filtered out from the answers. To filter the non-answers users express their constraints in the form of predicates. Predicates are generally grouped in conjunctive or disjunctive normal form or a combination of them. For naïve or inexperienced users, posing appropriate constraints in the predicates without having a complete knowledge of the underlying dataset is a tedious job. In most of the cases, users go for a number of trials before getting the ultimate or precise one. In the worst case, they reach to an unsatisfactory one. A system that can generate explanations for the unexpected and the expected information is advantageous in this case as the user can directly concentrate on the appropriate causes to alleviate their problems. Particularly, this could give them an opportunity to know *why* a particular piece of information is present in the result set and why the result set *misses something* that is expected, i.e., *why not.* At present, relational database systems does not support these kinds of explorative data analysis facilities to answer *why* and *why not* questions.

In cooperative database system a dialog is established between the user and the system to understand the intent of user (Nandi and Jagadish 2007, Sultana et al. 2009) by returning more information to the user in response to a query than just the query's answer set itself to help users in this regard (Motro 1994). Recently, explanation of query results and their provenance information such as *why* and *how* a particular piece of information arrived (Green et al. 2007, Cheney et al. 2009 and Glavic et al. 2009) and even missed (Huang et al. 2008, Herschel et al. 2010, Chapman et al. 2009 and Tran et al. 2010) are suggested to gain the confidence of users in this endeavour.

User feedback is extensively studied in information retrieval areas specifically for searching similar pages in the web and is generally considered as the basis for improving the sensitivity of the system (Moon et al. 2010). Feedback based query refinement is a newer concept in relational data model. Some researchers proposed keyword search in relational databases in this direction where feedback is an important step in ranking the resultant tuples by some non-linear ranking functions (Baid et al. 2010). In our model, feedback are collected in the form of *what part of information* in the result set a user doesn't want and *what part of non-answers* the user expects to see. Taking into consideration this feedback we then try to capture the user intents. This kind of cooperative behavior of DB is discussed by Amer-Yahia et al. (2005) to bridge the gap between DB and IR techniques to increase the usability of databases (Jagadish et al. 2007). In our model, explanations of unexpected and expected information are analysed to refine the initial query. This kind of query refinement approach can greatly minimize the distances between the original and the refined queries.

Our main contributions are as follows:

- In our model, we provide a unified framework in which we can explain why result set includes something we don't expect and why it misses something we expect. That is, our model is equally applicable to explain unexpected as well as missing answers.
- We show how the explanations can be used to capture user intent and exploited by our query refinement model. In other words, we show how incomplete user

feedback can be automatically made complete by our model.
- Finally, we also show how explanation based query refinement could be used as a helping hand to support imprecise queries in relational data model.

The remainder of the paper is organized as follows: Section 2 provides the preliminaries, Section 3 describes our explanation model, Section 4 presents how we can capture user intent, Section 5 describes the explanation based query refinement and future challenges, Section 6 presents related work and finally, Section 7 concludes the paper.

2 Preliminaries

This section describes briefly about the types of SQL queries that are targeted in this paper, their forms and transformations, taxonomy of database tuples with respect to the submitted query, provenance query and attributes, and finally presents the query refinement problem.

2.1 Types of SQL Queries

In our model, we consider simple queries in which each projected attribute is relation's attribute and the selection condition is a conjunction of predicates or disjunction of predicates or an arbitrary combination of them. For simplicity and without any loss of generality, we assume that the user queries can be mapped to either Disjunctive Query (DQ) or Conjunctive Query (CQ). The definitions of these queries are given below:

Definition 1 (Disjunctive Query): A disjunctive query (DQ) is a query (Q), where the conditions are specified in disjunctive normal form as follows:

$$DQ = C_1 \vee C_2 \vee ... \vee C_n \text{ and } C_i = C_{i1} \wedge C_{i2} \wedge ... \wedge C_{ik}$$

Definition 2 (Conjunctive Query): A conjunctive query (CQ) is a query (Q), where the conditions are specified in conjunctive normal form as follows:

$$CQ = C_1 \wedge C_2 \wedge ... \wedge C_n \text{ and } C_i = C_{i1} \vee C_{i2} \vee ... \vee C_{ik}$$

where C_{ij}'s are selection or join predicates in both DQ and CQ.

Example 1: Consider a database consisting of two tables: Publication (PubID, Pages, Pubyear, Book, CitationCnt) and Book (ERAID, Acronym, Rank, Areanum, Area).
Now, a disjunctive SQL query (DQ) is given in Fig. 1 to retrieve publications from the above mentioned database.

```
SELECT PubID
  FROM publication, book
    WHERE (PubYear >= 1990 AND CitationCnt >= 80
      AND Rank <= 2 AND Book = Acronym) OR
       (CitationCnt >= 60 AND Book = 'vldb' AND
         Book =  Acronym);
```

Figure 1: An example of disjunctive SQL query (DQ)

2.2 Conversion between CQ and DQ

The conversion between CNF and DNF is a well-known problem in the history of computational science. The prospect of our explanation model depends on the successful conversion between CQ and DQ. We assume that there would be no missing data in our database, at least virtually. For example, if there are missing values for which RDM maintains *null*, we assume comparing ($<$, $\leq$, $=$, $\geq$, $>$,) anything with *null*– even another *null* values – results in *false* truth state. This assumption simplifies SQL's three-valued logic to two-valued logic. Therefore, we can treat our query Q as a Boolean expression. That is, evaluation of a tuple with respect to Q will be *true* or *false*. Hence, any DQ can be converted to the corresponding CQ by the *distributive law* from elementary algebra (distributing $\vee$ over $\wedge$) and vice versa, e.g., CQ of DQ (Fig. 1) is given Fig.2. The equivalent CQ (Fig. 2) has seven disjunctions whereas the original DQ had two conjunctions. This suggests that the resulting CQ query could be large compared to the size of original DQ query. But in practice users are willing to issue fewer predicates in their queries. Therefore, we assume we can handle the complexity of the converted CQ and DQ in reasonable time. This is further motivated by users' interest on the explanation of expected and unexpected tuples that are very few.

```
SELECT PubID
  FROM Publication, Book
    WHERE (CitationCnt >= 80 OR CitationCnt >= 60) AND
      (PubYear >= 1990 OR CitationCnt >= 60) AND
       (Rank <= 2 OR CitationCnt >= 60) AND
        (CitationCnt >= 80 OR Book = 'vldb') AND
         (PubYear >= 1990 OR Book = 'vldb') AND
          (Rank <= 2 OR Book = 'vldb') AND (Book = acronym);
```

Figure 2: Equivalent CQ of the given DQ (Fig.1) obtained by applying the distributed law.

2.3 Taxonomy of Database Tuples

We treat user query (Q) in our model as a binary classifier by which the entire database tuples will be divided into two broad categories: Resultant Tuples (RT) and Non-Resultant Tuples (NRT). If submitted query is the imprecise or vague one (predicates are not specified correctly), the resultant tuples can further be partitioned into two groups: True Positives (TP) and False Positives (FP) or unexpected tuples. Similarly, non-resultant tuples are of two types: True Negatives (TN) and False Negatives (FN) or expected tuples or missing tuples. The taxonomy of database tuples regarding the user's imprecise query is given below:

(a). **Resultant Tuple (RT).** A *resultant tuple* is a tuple in the result set for which at least one C_i in DQ evaluates to true. In other words, an RT tuple satisfies at least one C_i in DQ.
  (i). **Truly-positive Tuple (TP)**: *A truly-positive* tuple is a *resultant* tuple that satisfies at least one C_i in DQ with desirability 'yes' from user's point of view.
  (ii). **Unexpected Tuple (FP)**: An *unexpected tuple* is a *resultant* tuple that satisfies at least one C_i in DQ with desirability 'no' from user's point of view.

(b). **Non-resultant Tuple (NRT)**: A *non-resultant* tuple is a tuple for which at least one C_i in CQ evaluates to false. In other words, an NRT tuple dissatisfies at least one C_i in CQ.
  (i). **Expected Tuple (FN)**: An expected tuple is a non-resultant tuple that fails to satisfy at least one C_i in CQ with desirability 'yes' from user's point of view.

PubID	PubYear	CitationCnt	Rank	Book	Acronym
P1	1993	90	1	sigmod	sigmod
P2	1990	148	1	sigmod	sigmod
P3	1996	81	1	sigmod	sigmod
P4	1994	82	1	vldb	vldb
P5	1983	72	1	vldb	vldb
P6	1980	66	1	vldb	vldb
P7	1987	117	1	vldb	vldb

Table 1: Set of resultant (RT) tuples of the query given in Fig. 4. The shaded region represents the provenance information

(ii). **Truly-negative Tuple (TN)**: *A truly-negative* tuple is a *non-resultant* tuple that dissatisfies at least one C_i in CQ with desirability 'no' from user's point of view.

Let R is the set of resultant tuples, R^p is the set of truly positive tuples, R^u is the set unexpected tuples, R_N is the set of non-resultant tuples, R_N^n is the set of truly negative tuples and R_N^e is the set of expected tuples. Then the following equations hold:

(i). $R^p \subseteq R$, $R^u \subseteq R$
(ii). $R = R^p \cup R^u$
(iii). $R^p \cap R^u = \emptyset$
(iv). $R_N^n \subseteq R_N$, $R_N^e \subseteq R_N$
(v). $R_N = R_N^n \cup R_N^e$
(vi). $R_N^n \cap R_N^e = \emptyset$
(vii). $R \cap R_N = \emptyset$
(viii). $Q \vDash R^p \cup R^u$
(ix). $Q \nvDash R_N^n \cup R_N^e$

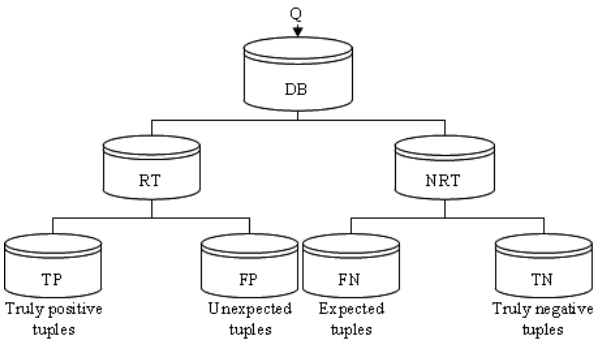

Figure 3: Taxonomy of DB tuples wrt to user query

2.4 Provenance Query and Attributes

Provenance describes the linage or pedigree of data and finds paramount of importance in scientific databases, data warehouses, and workflow management systems. W*hy provenance* gives information about the source tuples that have direct contribution to the resultant tuples and w*here provenance* tells about the location of source tuple values from where the output tuple values are copied from (Tan 2004, Cheney et al. 2009). Provenance Query (Q^p) produces the same result as like as Q but adds additional provenance attributes through which information about the input tuples that contributed to the creation of output tuple is propagated. Our provenance query is largely based on the query rewrite rules developed by Glavic and Alonso (2009). Fig. 4 shows the equivalent provenance query of the query given in Fig. 1. We apply the eager approach (Tan, 2004) for computing the provenance information of our resultant tuples. That is, provenance information is computed along with our query outputs. Table 1 shows the resultant tuples together with their corresponding provenance information.

In our model, we exploit provenance information to find the implicit predicates, eliminate the need of accessing the database again, classify the tuples and foster the decision process for query refinement.

```
SELECT PubID, PubYear, CitationCnt,
  Rank, Book, Acronym
   FROM publication, book
     WHERE (PubYear >= 1990 AND CitationCnt >= 80
       AND Rank <= 2 AND Book = Acronym) OR
        (CitationCnt >= 60 AND Book = 'vldb'
          AND Book = Acronym);
```

Figure 4: Equivalent provenance query Q^p for the query given in Fig. 1

2.5 Query Refinement Problem

We refer to queries that require the conditions to be slightly adjusted as imprecise queries while those requiring no adjustment to as precise queries. Supporting imprecise queries over databases necessitates a system that collects feedback from the user to understand the submitted query intent and returns a new query to encounter both unexpected and expected tuples. Therefore, we define the query refinement problem as follows:

Given Q, R^u and R_N^e find a refined query Q^* such that,

$$Q^* \vDash R^p \cup R_N^e \text{ and } Q^* \nvDash R^u \cup R_N^n$$

But in reality, the distance between the optimum refined query Q^* and the initial query Q could be large. Therefore, our target is to find an approximate refined query Q^f that will minimize the number unexpected tuples and maximize the number of expected tuples as much as possible. But the approximate query Q^f will be highly similar to the initial query Q. That is,

$$Q^f \vDash \mathbb{R}^p \cup \mathbb{R}_N^e,\ Q^f \nvDash \mathbb{R}^u \cup \mathbb{R}_N^n$$
$$\text{and } sim(Q, Q^f) \gg sim(Q, Q^*)$$

where $\mathbb{R}^p \subseteq R^p$, $\mathbb{R}_N^e \subseteq R_N^e$, $R^u \subseteq \mathbb{R}^u$, $R_N^n \subseteq \mathbb{R}_N^n$ and $sim(Q, Q^f)$ is the similarity score between Q and Q^f. The similarity calculation of the refined query to the original query is given in section 5.

In refining initial query, we give emphasis to both the quality of query results as well as similarity metric. That is, we model refined query as the one that can return better quality results and at the same time will also be highly similar to the initial query.

3 Explanation Model

In our explanation model, we generate appropriate explanations for why the result set includes some information that is unexpected as well as why it misses something that is expected. For these, we consider the user query divides the entire *DB* into resultant (*RT*) and non-resultant (*NRT*) tuples (as given in Fig. 3). But because of user's imprecise specification of constraints in the predicates of the query *RT* also includes some unexpected tuples that we wish to exclude. Similarly, *RT* may also miss some expected tuples that we want to get back in the result set. In our explanation model we use clauses in *DQ* and *CQ* to explain expected and unexpected tuples, respectively. The advantage of using clauses to explain expected and unexpected tuples is that the generated explanations are exact and complete. Therefore, we know which causes we need to alleviate to encounter them in the refined queries.

3.1 Explanation of Unexpected Tuple

To model why a particular tuple (including the unexpected one) presents in the result set we rely on the C_i in *DQ* that are evaluated to true. This is because in *DQ* a C_i will be evaluated to true if each individual predicate C_{ij} in C_i is satisfied. Therefore, we can say that our explanations are exact for unexpected tuples. The following defines the explanation of unexpected tuple (t^u) more formally:

Definition 3: The explanation of an unexpected tuple, t^u, consists of those conjunctions C_i in *DQ* that are evaluated to true. That is,

$$Expl(t^u): \{ C_i | C_i \in DQ \Vdash t^u \}$$

We assume for unexpected tuple, only few C_i in *DQ* will be evaluated to true. This is because, unexpected tuples lies on the boundary of user defined predicates. Therefore, to refine the initial query we can concentrate only on the explanation of the unexpected tuple, i.e., $C_i \in Expl(t^u)$. It should be noted that to exclude an unexpected tuple t^u from the result set we need to modify at least one C_{ij} in C_i appearing in $Expl(t^u)$ so that $C_i \in DQ \nVdash t^u$.

Example 2: Consider the resultant tuples given in Table 1 and the *DQ* given in Fig 1. Now, user tells that the tuple P6 is the unexpected one and seeks explanation for it. If we analyse tuple P6 we can see that it satisfies the second conjunction of *DQ*. That is, the explanation of why P6 is in the result set consists of second conjunction. Hence, we can say:

> P6 is in the result set as CitationCnt (66) >= 60 AND Book ('vldb') = 'vldb'

Therefore, if we would like to exclude P6 from the result set we can set either CitationCnt>66 or Book!='vldb'.

3.2 Explanation of Expected Tuple

To model why result set misses a particular tuple (including the expected one) we rely on the C_i in *CQ* that are evaluated to false. This is because in *CQ* a C_i will be evaluated to false if each individual predicate C_{ij} in C_i is dissatisfied. Therefore, we can say that our explanations are exact for expected tuples. The following defines the explanation of expected tuple (t^e) more formally:

Definition 4: The explanation of an expected tuple, t^e, consists of those disjunctions C_i in *CQ* that are evaluated to false. That is,

$$Expl(t^e): \{ C_i | C_i \in CQ \nVdash t^e \}$$

We assume for expected tuple only few C_i in *CQ* will be evaluated to false. This is because expected tuples lies on the boundary of user defined predicates. Therefore, to include the expected tuple in the result set we can modify only those C_{ij} in C_i that appears in $Expl(t^e)$ so that $C_i \in CQ \Vdash t^e$.

Example 3: Consider the resultant tuples given in Table 1 and the *CQ* given in Fig 2. Now, user wants to know why result set misses tuple: ('P8', 78, 1986, 1, 'sigmod', 'sigmod') and seeks explanation for it. If we analyse tuple P8 we can see that it dissatisfies the fourth and fifth disjunctions of *CQ*. That is, the explanation of why P8 is not in the result set consists of the fourth and fifth disjunctions. Hence, we can say:

> P8 is not in the result set as (CitationCnt (78) < 80 or Book ('sigmod')! = 'vldb') and (PubYear (1986) <1990 or Book ('sigmod')! = 'vldb').

Now, if we would like to include it in the result set then we can set (CitationCnt>=78) and PubYear>=1986) in the first conjunction or Book in ('vldb, 'sigmod') in the second conjunction of *DQ*.

4 Capturing User Intent

The primary goal of any information retrieval (IR) system is to retrieve documents, which are relevant to the user query while retrieving as few non-relevant as possible. In IR, user feedback is one of the extensively studied and widely accepted techniques to refine the initial result set (Mishra and Koudas 2009, Belhajjame et al. 2011). Feedback is generally gathered to improve the precision and sensitivity of the retrieval system. We bring the idea of IR system into RDM to capture the user intent and refine the initial imprecise SQL query.

In our model, we propose the system to present the result set of the initial imprecise query to the user for her judgement. Then, she can select which tuples she considers unexpected as well as what other tuples from non-answers she expects to see in the result set. This could also complicate the collection of feedback as the user may feel reluctant to provide the complete set of unexpected and expected tuples. Therefore, we want our users to input only a subset of expected and unexpected tuples. We consider these tuples as the hints for the system. In our model, we then complement the feedback through point domination theory.

4.1 Point Domination Theory

Let $G = \{G_1, G_{12}, \ldots G_n\}$ be a set of the predicate preferences. We denote by $t \succ_{G_i} t'$ and $t \succcurlyeq_{G_i} t'$ the statements "tuple t satisfies preference G_i better than t' and "tuple t satisfies preference G_i as least as t' respectively. If we plot the preference predicates G in an n-dimensional space, each tuple will be an n-dimensional point. Then we say a point t is as good as another point t'

iff $\forall i \in [1,n], t \succcurlyeq_{G_i} t'$, and point t dominates point t' iff $\forall i \in [1,n], t \succcurlyeq_{G_i} t'$ and $\exists k \in [1,n], t \succ_{G_k} t'$ (Pareto-order: Voorneveld 2003). We denote by $t \succcurlyeq t'$ and $t \succ t'$ the statements "tuple t is as good as tuple t'" and "tuple t dominates tuple t'" respectively.

To make the above clear, consider there are two predicates we wish to consider and place these two predicates in *X* and *Y* directions in a two-dimensional space. Suppose there are five points 'a', 'b', 'p', 'c' and 'd' in this space as shown in Fig. 5. Now, both 'a' and 'b' dominates 'p' as both 'a' and 'b' is as good as 'p' and satisfies at least one predicate better than 'p'. Similarly, 'p' dominates both 'c' and 'd'. If 'p' is our reference point, then 'p' divides the entire space into four regions as shown in Fig. 5. Now in general, any point from region *R1* is as good as or dominates 'p'. Similarly, 'p' is as good as or dominates any point in region *R3*.

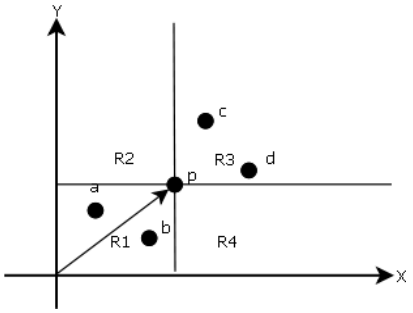

Figure 5: Point domination theory for discovering implicit 'yes' and 'no' tuples

Definition 5 (Explicit 'yes'): Tuples given explicitly by the user as expected are called explicit 'yes' tuples. Explicit 'yes' tuples are part of the non-answers (NRT tuples).

Definition 6 (Explicit '*no*'): Tuples given explicitly by the user as unexpected are called explicit 'no' tuples. Explicit 'no' tuples are part of the answers (RT tuples).

Definition 7 (Implicit 'yes'): Any tuple from non-answers (NRT tuples) that is not dominated by explicit 'yes' tuples is called implicit 'yes' tuple. That is, implicit 'yes' tuples are as good as explicit 'yes' tuples.

Definition 8 (Implicit 'no'): Any tuple from answers (RT tuples) that is dominated by explicit 'no' tuples is called implicit 'no' tuple. That is, implicit 'no' tuples are no better than explicit 'no' tuples.

Example 4 (Implicit 'no'): Suppose that the user gives the following tuple as the unexpected tuple: ('P5', 1983, 72, 1, 'vldb', 'vldb').

The explanation of the above tuple is: CitationCnt (72) >= 60 AND Book ('vldb') = 'vldb'. Then according to the point domination theory, the following tuple is also unexpected as the above tuple dominates the following tuple in terms of both *CitationCnt* and *Book* attribute. Therefore, the following tuple is implicit 'no' tuple.

('P6', 1980, 66, 1, 'vldb', 'vldb')

Example 5 (Implicit 'yes'): Consider the user inputs the following tuple as the expected tuple: ('P8', 1986, 78, 1, 'sigmod', 'sigmod').

Now if we analyse the explanation of the above tuple we see that the above tuple is dominated by the following two tuples in terms of both *PubYear* and *CitationCnt* attributes. Then, according to the point domination theory, the following two tuples are implicit 'yes' tuples as they are expected too.

('P9', 1986, 79, 1, 'oopsla', 'oopsla')
('P10', 1986, 79, 1, 'sigmod', 'sigmod')

Definition 9 (Explicit Predicates): The predicates that are given explicitly by the user in the initial query are called explicit predicates.

Example 6: The explicit predicates for our example query (Fig. 1) are:

(i). PubYear >= 1990	(i). CitationCnt >= 60
(ii). CitationCnt >= 80	(ii). Book = 'vldb'
(iii). Rank <= 2	(iii). Book = Acronym
(iv). Book = Acronym	

The predicates on the left side are the explicit predicates for the first conjunction and the predicates on the right side are the explicit predicates for second conjunction of the *DQ* given in Fig. 1.

Definition 10 (Implicit Predicates): Implicit predicates are the predicates that are inferred from the (RT) resultant tuples and explicit feedback.

Example 7: The explanation of the first four tuples (as shown in Table 2) consists of the first conjunctions of *DQ* of Fig.1. That is, these four tuples satisfy the explicit predicates of the first conjunction. Assume user inputs the fourth tuple as the unexpected one. We can easily observe that for all tuples including the explicit 'yes': Book= 'sigmod' and Rank≤1. Hence, we infer the following predicates as the *implicit predicates* for the first conjunction.

(i). Book = 'sigmod'
(ii). Rank <= 1

4.2 Reduction of Implicit Feedback

After getting feedback from the user (both unexpected and expected tuples) we find the implicit 'yes' tuples. But these implicit 'yes' could be far from user's expectation. Sometimes this implicit feedback could be difficult to encounter by the system and system response could be overwhelming for the users too. To alleviate this problem, we reduce them by evaluating the implicit predicates. That is, if the implicit 'yes' tuples fail to satisfy implicit predicates we delete them from the feedback list as shown in Fig. 6.

Example 8: Implicit 'yes' tuples ('P10', 1986, 79, 1, 'sigmod', 'sigmod') satisfies the implicit predicates for the first conjunction of *DQ*. So, we accept this tuple as user implicit feedback. But tuple ('P9', 79, 1986, 1, 'oopsla', 'oopsla') is rejected as it fails to satisfy the implicit predicates.

As we see from example 5 and 8 that the reduction rate is quite promising (50% in this case). This suggests that implicit feedback should be treated carefully and must be corrected by analysing the implicit predicates. Implicit predicates could be advantageous in the case where users are willing to provide only the values in few dimensions (e.g., virtual tuples) rather than a complete set of tuples.

PubID	PubYear	CitationCnt	Rank	Book	Acronym	Desirability
P1	1993	90	1	sigmod	Sigmod	yes
P2	1990	148	1	sigmod	Sigmod	yes
P3	1996	81	1	sigmod	Sigmod	yes
P4	1994	82	1	vldb	vldb	no
P5	1983	72	1	vldb	vldb	no
P6	1980	66	1	vldb	vldb	no
P7	1987	117	1	vldb	vldb	yes

Table 2: Set of resultant (*RT*) tuples of the user query, *Q*. The last columns are the user feedback. Desirability with 'no' are the unexpected tuples

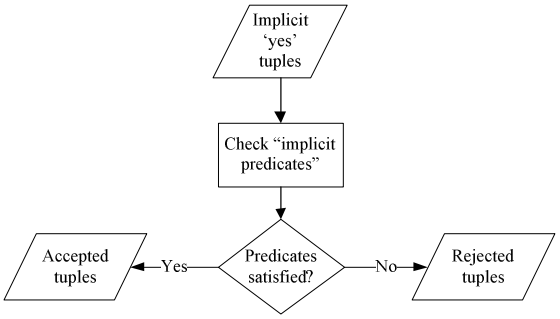

Figure 6: Reduction of implicit 'yes' tuples

5 Query Refinement Model

One particular problem related to the imprecise query is how to refine it to encounter both unexpected and expected tuples. Our purpose is to exclude the unexpected as well as include the expected tuples as much as possible after getting feedback from the user. In our model, we don't require our user to provide the complete set of unexpected and expected tuples. Only a subset of them can be sufficient. These tuples are then analysed to complement the feedback as we explain in section 4.

We consider One Clause At a Time (OCAT) in the given query and propose structural changes to minimize the number of unexpected and maximize the number of expected tuples. In our model, we assume that the following assumptions hold regarding the initial imprecise query:

- Users are more confident and less flexible about the equality ('=') operator.
- Users are more flexible about the inequality (>, ≥, <, ≤) operators.

5.1 Exclusion of Unexpected Tuples

Unexpected tuples are part of the resultant (RT) tuples. Though these tuples satisfy the user defined initial imprecise constraints, they generally stretch out near the boundaries of the predicates. In our query refinement model, to exclude these unexpected tuples from the result set we propose OCAT approach. In this technique, we group all resultant tuples based on their satisfying conjunctions (explanations). Then, we propose structural changes for each conjunction if they include any unexpected tuples. Otherwise, we leave them as they were before.

For each group *D* of resultant tuples, we tighten some predicates that appear in the conjunction to separate the unexpected tuples from the truly positive tuples. We rely on the information gain theory and Decision Tree (Mitchell 1997) to find the best separating attributes. To do so, we calculate the information gain of each attribute (*A*) that appears in the predicates. The formulas for calculating the information gain are given below:

$$Info(D) = -\sum_{i}^{2} p_i \log p_i$$

where $Info(D)$ is the expected information needed to classify a tuple in D and $p_i$is the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$. In our model, there are two categories of resultant tuples: C_1 (desirability 'yes') and C_2 (desirability 'no'). Tuples with desirability 'yes' are the truly positive tuples and tuples with desirability 'no' are the unexpected tuples as given in Table 2. Information needed (after using A to split D into v partitions) to classify D is:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

Information gained by branching on attribute *A* is:

$$Gain(A) = Info(D) - Info_A(D).$$

We use the best-known and most widely-used C4.5 decision tree learner (Mitchell 1997) and WEKA implementation of it (Hall et al. 2009). All paths *P* from root to a leaf ('yes') node of the tree forms the CNF expressions that are ORed together to form a DNF expression. The DNF expression is then ANDed to the original query and finally simplified to get the refined query.

PubID	PubYear	CitationCnt	Desirability
P4	1994	82	no
P5	1983	72	no
P6	1980	66	no
P7	1987	117	yes

Table 3: Tuples identified by the second conjunction of *DQ* (Fig. 1)

Example 9: Consider the *DQ* given in Fig. 1 and the resultant tuples given in Table 2. The last four tuples (shown in Table 3) are identified by the second conjunction in *DQ*. The separation of the unexpected tuples (P4, P5 and P6) from truly positive tuples (P7) is done by the best separating attribute *CitationCnt* (among Pubyear and CitationCnt) as shown in Fig. 7.

Therefore, we propose the following structural changes for the second conjunction: CitationCnt >= 82 AND

Book = 'vldb' AND Book = Acronym and we have the following refined *DQ* to exclude the unexpected tuples in the result set:

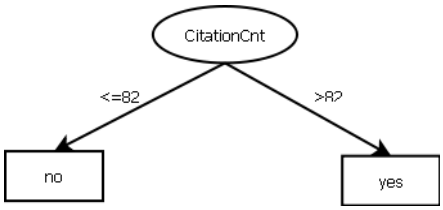

Figure 7: Separation of unexpected tuples from truly positive tuples

```
SELECT PubID
  FROM publication, book
  WHERE (PubYear >= 1990 AND CitationCnt >= 80 AND
     Rank <= 2 AND Book = Acronym) OR
      (CitationCnt >= 82 AND Book = 'vldb'
        AND Book = Acronym);
```

Figure 8: Refined query of DQ given in Fig. 1

The query given in Fig. 8 excludes the expected tuple by restricting the predicates CitationCnt >= 60 to CitationCnt > 82 in the second conjunction.

5.2 Inclusion of Expected Tuples

To include the expected tuples in the result set we usually need to relax some constraints for some predicates in the query. In our model, we summarize the explanation of all expected tuples and suggest structural changes for the user's imprecise query. To do so, we group the expected tuples based on what changes we need to include them in the result set Then, we propose structural changes to the closest conjunctions in *DQ* for each group.

Example 10: Consider the *CQ* given in Fig. 2 and expected feedback {('P8', 1986, 78, 1, 'sigmod', 'sigmod'), ('P9', 1986, 79, 1, 'oopsla', 'oopsla'), ('P10', 1986, 79, 1, 'sigmod', 'sigmod')}. All these tuples fail to satisfy the following disjunctions:

(i). CitationCnt >= 80 or Book = 'vldb'
(ii). PubYear >= 1990 or Book = 'vldb'

To include these tuples in the result set, we can make following changes in the first conjunction in *DQ*:

(i). CitationCnt >= 78 instead of CitationCnt >= 80
(ii). PubYear >= 1986 instead of PubYear >= 1990

For the second conjunction in *DQ*, we can make the following change:

(i). Book in ('sigmod', 'oopsla') instead of Book ='vldb'

So we have two options to encounter these expected tuples. As per our assumptions (assumption 2), we have the following refined *DQ* to include the expected tuples in the result set:

```
SELECT PubID
  FROM publication, book
  WHERE (PubYear >= 1986 AND CitationCnt >= 78 AND
     Rank <= 2 AND Book = Acronym) OR
      (CitationCnt >= 82 AND Book = 'vldb'
        AND Book = Acronym);
```

Figure 9: Refined query of DQ given in Fig. 1

The query given in Fig. 9 includes the expected tuple by relaxing the predicates PubYear >= 1990 to PubYear >= 1986 and CitationCnt >= 80 to CitationCnt >= 78 in the first conjunction. The target predicates are identified by the explanation of expected tuples.

5.3 Evaluation of Refined Queries

It is hard to formulate the automatic evaluation of the refined queries without incorporating user preferences as the effectiveness of query refinement is tied to the convergence of the results to the users information need. There are two obvious desiderata for refined queries that can be used for this purpose. One is quality of the results of the refined query and another one is the similarity of the refined query to the original input query.

5.3.1 Quality of Refined Query Results

The quality of the refined query results can be measured by the standard evaluation metric (Mitchell 1997, Binderberger et al. 2002, Stasiu et al. 2005, Mishra and Koudas 2009). Let n_{tp} defines the number the number of truly positive tuples and n_{fp} defines the number the number of false positive tuples (unexpected) retained by Q^f, and n_{tn} is the number of truly negative tuples and n_{fn} is the number of false negative tuples (expected) filtered out by Q^f. Now, we define the standard evaluation metric sensitivity and specificity in our framework as follows:

$$sensitivity = \frac{n_{tp}}{n_{tp} + n_{fn}} \text{ and } specificity = \frac{n_{tn}}{n_{tn} + n_{fp}}$$

Sensitivity measures the proportion of actual positives tuples which are correctly retained by Q^f. On the other hand, specificity measures the proportion of negatives (unexpected) tuples which are correctly excluded by Q^f. An alternate measure that offers a trade-off between sensitivity and specificity is balanced accuracy. Balanced accuracy is defined as follows:

$$Accuracy = \frac{\alpha * n_{tp}}{n_{tp} + n_{fn}} + \frac{\beta * n_{tn}}{n_{tn} + n_{fp}}$$

The values of α and β are generally set to 0.5 to treat both sensitivity and specificity with equal importance. But these values can be set to different values in the range [0, 1] to have different preferences for sensitivity and specificity. The imprecision metric defined by Tran and Chan (2010) covers only *why-not* (expected) questions and therefore does not serve our purposes.

Example 11: The sensitivity, specificity and balanced accuracy for the query given in Fig. 9 are 100%, 40% and 70% respectively when we don't consider the implicit feedback. We set α and β to 0.5 for the calculation of the balanced accuracy.

Implicit Feedback	Sensitivity	Specificity	Balanced Accuracy
Not Considered	100%	40%	70%
Considered	100%	100%	100%

Table 4: Quality of the refined query (shown in Fig. 9) results with and without implicit feedback

Now consider the implicit 'yes' tuples P4 and P9 in our example tuple set Tuple P4 dominates tuple P3 and tuple P9 dominates P8 in terms of CitationCnt and Rank. Therefore, these tuples are retained in the result set by the

refined query given in Fig. 9. But if we add the implicit predicate (e.g., Book = 'sigmod') to the first conjunction of *DQ* in the refined query, tuples P4 and P9 are filtered out. Therefore, the corresponding metric is boosted as we see from Table 4. That is, implicit feedback greatly reduces the false positives rate and results in increased specificity.

5.3.2 Similarity of Refined Query to the Original Query

Any query refinement framework should emphasize on producing refined queries which are as similar as possible to the original input query. This has intuitive appeal since a refined query that is minimally modified from the original query is likely to retain as much of the intention of the original input query. Tran and Chan (2010) propose *edit distance* as the dissimilarity metric for this purpose. The edit operators considered in their work are as follows: modify the constant value of a selection predicate in the where-clause, add a selection predicate in the where-clause, add/remove a join predicate in the where-clause, and add/remove a relation in the from-clause. In our framework, we define imprecise queries that require the conditions to be slightly adjusted. That is, we allow only relaxation and/or restriction of query conditions in the refined queries. But there could be some predicates that need to be unaltered from users' point of view as mentioned by Kießling and Köstler (2002).

We define the following distance metric between the initial imprecise query (Q) and the refined query (Q^f) to take into account the user preferences:

$$dis(Q,Q^f) = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{\sum_{j=1}^{k} w_{ij} * d\left(C_{ij}, C_{ij}^f\right)}{\sum_{j=1}^{k} w_{ij}}\right)$$

where w_{ij}'s are the predicate preferences of user and are set to between 0 and 1. If w_{ij} is set to 1, it means user prefers to maintain this predicate strongly (*hard constraints* as defined by Kießling and Köstler 2002). If it is 0, then the user does not care about the replacement of the corresponding predicates in the refined query. If it is between 0 and 1, then the user is flexible to slightly modify the corresponding constraints (*soft constraints* as defined by Kießling and Köstler 2002). The distance function $d(C_{ij}, C_{ij}^f)$ is currently open in our framework. As an example, we define them as follows depending on whether a_{ij} is numeric or categorical attribute:

(a). $d(C_{ij}, C_{ij}^f) = \dfrac{\left|v_{ij} - v_{ij}^f\right|}{\max(v_{ij}, v_{ij}^f)}$ if a_{ij} is numeric.

(b). $d(C_{ij}, C_{ij}^f) = \begin{cases} 1 & if \quad v_{ij} != v_{ij}^f \\ 0 & if \quad v_{ij} = v_{ij}^f \end{cases}$ if a_{ij} is categorical.

The corresponding similarity metric is defined as follows:

$$sim(Q,Q^f) = 1 - dis(Q,Q^f)$$

Example 12: The refined query given in Fig. 9 is 7.2% dissimilar to the original query given in Fig. 1. That is, the refined query is 92.8% similar to the original query. The individual weight for each predicate is set to 0.5 for the calculation of similarity score. If we consider the implicit predicates (e.g., Book = 'sigmod' to the first conjunction of *DQ* in the refined query) as given in Example 7, then the similarity score is decreased as we see from Table 5. That is, the more new predicates we add the more dissimilar the refined query becomes to the original query.

Implicit Predicates	dis(Q, Q^f)	sim(Q, Q^f)
Not Considered	7.2%	92.8%
Considered	38.2%	61.8%

Table 5: Similarity score of the refined query (shown in Fig. 9) with and without implicit predicates

5.3.3 Combined Score on Quality and Similarity Metric

To emphasize on both quality of refined query results and similarity of the refined query to the original user submitted query we define a combined metric as follows:

$$CombinedScore = \delta * quality + (1-\delta) * sim(Q,Q^f)$$

The value of δ can be set to 0.5 to treat both quality and similarity metric with equal importance. To have different preferences on quality and similarity metric δ can be set to any value in the range [0, 1]. We can set δ to 0 to ignore the quality metric and 1.0 to ignore the similarity score. The combined score for the refined query given in Fig. 9 together with implicit feedback considered or not is given in Table 6.

Implicit Feedback	Combined Score
Not Considered	81.4%
Considered	80.9%

Table 6: Combined score together with implicit feedback and predicates considered or not

We set δ to 0.5 for the calculation of the combined score. Table 6 manifests that the combined score provides a balance between quality and similarity metric for the refined query, though the relationship between the quality and similarity metric is reciprocal (if we add implicit predicates to the refined query, it improves the quality and as a side effect it makes the refined query more dissimilar to the original query).

5.4 Future Challenges

Though the explanation based query refinement model is promising in producing highly similar refined queries to the user submitted query, there are several challenges that need to be addressed before its successful application.

Firstly, there could be more than one ways to exclude an unexpected tuple if the corresponding query is *DQ*. This is because if the predicates are ANDed together then to exclude an unexpected tuple we need to dissatisfy at least one of them as we explain it before in our explanation model (Section 3). Thus we need to select a subset of predicates from a conjunction in *DQ* to exclude a set of unexpected tuples which is an NP-hard problem. An approximation algorithm for selecting the best subset is needed that can run in polynomial time. By the best

subset we mean the set that incurs minimal changes in the original query. This is also true for *CQ* and expected tuples. This could be further complicated if we need to both include expected tuples as well as exclude the unexpected tuples through the same conjunction.

Secondly, a user defined quality metric aware query refinement algorithms need to be designed so that the parameters (sensitivity and specificity thresholds, predefined soft and hard constraints) can be specified beforehand. This is because there are few applications including medical databases where it is crucial to maintain the user defined quality metric.

Thirdly, the implicit feedback could be treated carefully, otherwise it could be explosive. In this paper, we show how we can reduce this feedback by identifying the implicit predicates. Besides of that, pre-calculated approximate functional dependencies and concept learning (Nambiar et al. 2004) can also be coupled for this purpose.

Lastly, we consider only the adjustment of the constraints in our query refinement model. But insertion/deletion of conditions in the where-clause, add/drop relations in the from clause and even relaxation of join conditions (Tran and Chan 2010) could be considered to provide more flexibility in the explanation based query refinement model.

6 Related Work and Discussion

Huang et al. (2008) explain missing tuple by allowing modifications to the database such that the expected tuple appears in the query result wrt the modified database. But the number of returned alternatives can be huge and difficult for users to comprehend the appropriate explanation for the missing tuple. Herschel et al. (2010) extend the idea of (Huang et al. 2010) to take into account a set of missing tuples and select-project-join-union (SPJU) queries. The intuition of this model is to explain in terms of how to modify some of the untrusted data in order to produce the missing tuple. However, this model may not be applicable in applications where all the data stored are trusted. Champan and Jagadish [4] model explanation of missing tuple by identifying the operator(s) that filters it out from the result set.

Tran and Chan (2010) model explanation of expected tuples by refining the original query to include them in the result set. Authors exploit the idea of skyline queries to report the closest refined query wrt original one to minimize the distance between refined and original queries. It may happen that the number of returned refined queries is overwhelming for the naïve user and frustrating as the user looks for exact explanations. Moreover, refined queries may introduce more false positives (unexpected tuples) in the result set. A more helpful explanation should be one that can model both why and why-not questions. We propose query refinement exploiting the explanations generated by our framework to minimize the number of unexpected tuples as well as maximize the number of expected tuples.

Liu et al. (2010) collect false positives identified by the users as feedback to modify the initial rules in information extraction settings to exclude unexpected results. However, they don't consider the expected results to refine the imprecise rules. In our model, we collect both false positives and false negatives to refine the initial imprecise query. We don't require our user to provide the complete set of unexpected and expected tuples but only a subset of them. We automatically suggest other members based on point domination theory.

Meliou et al. (2010) propose causality as a unified framework to explain both resultant and non-resultant tuples. This approach generalizes and extends previously proposed definitions of provenance by leveraging the lineage of query answers (positive provenance) and non-answers (negative provenance) under the same framework. But they don't separate unexpected tuples from answers (also expected tuples from non-answers) as we do in our model. We also show how one can modify the initial imprecise query exploiting the explanations of expected and unexpected tuples generated by our framework.

The need of supporting exploratory or imprecise queries in RDM and Web databases is studied with great importance in (Nambiar et al. 2003, Kadlag et al. 2004, Nambiar and Kambhampati 2005, Ma et al. 2006, Koudas et al. 2006, Mishra and Koudas 2009). This has particular application to the many/few answers problem often faced by database users. In (Mishra and Koudas 2009) to address this problem user feedback is incorporated to best capture user preferences whereas in (Koudas et al. 2006) a lattice based framework is proposed for minimum amount of query conditions relaxation. We show how one can capture user intent by collecting feedback (a sample set of unexpected and expected tuples) and modify the initial imprecise query exploiting the explanations generated by our model.

7 Conclusion and Future Work

This paper presents a unified framework to explain both the presence of unexpected and the absence of expected tuples. We show how to capture user intent exploiting the explanations generated by our model. For this, we require the user to provide a sample set of expected and unexpected tuples. Then, we show how one can complement it by automatically suggesting the complete set. Finally, we show how we can refine the initial imprecise query to exclude the unexpected tuples exploiting the explanations of them. Similarly, to include the missing information we relax certain predicates that appear in the explanation of expected tuples. In other words, both *why* and *why not* causes are exploited by our model to refine the initial imprecise query. We also present future challenges of explanation based query refined model.

Future work of our paper includes addressing the future challenges of explanation based query refinement model and the detailed evaluation of the proposed framework for demonstrating its performance and effectiveness in real world data set. We also plan to adapt our framework to solve many/few answers problem incorporating user feedback and attribute/predicate preferences.

8 References

Amer-Yahia, S., Case, P., Rolleke, T., Shanmugasundaram, J. and Weikum, G. (2005): Report

on the db/ir panel at sigmod 2005. *SIGMOD Record*, **34(4)**: 71-74.

Baid, A., Rae, I., Li, J., Doan, A. and Naughton, J. F. (2010): Toward scalable keyword search over relational data. *Proc. VLDB Endowment*, **3(1)**: 140-149.

Belhajjame, K., Paton, N.W. and Fernandes, A.A.A. (2011): User feedback as a first class citizen in information integration systems. *Proc. Biennial Conference on Innovative Data Systems Research*, **5**: 175-183.

Binderberger, M. O., Chakrabarti, K. and Mehrotra, S. (2002): An approach to integrating query refinement in SQL. *Proc. International Conference on Extending Database Technology*, **8**: 15-33.

Chapman, A. and Jagadish, H. V. (2009): Why not?. *Proc. ACM SIGMOD Conference*, 523-534.

Cheney, J., Chiticariu, L. and Tan, W. C. (2009): Provenance in databases: why, how, and where. *Foundations and Trends in Databases*, **1(4)**: 379-474.

Glavic, B. and Alonso, G. (2009): The perm provenance management system in action, *Proc. ACM SIGMOD Conference*, 1055-1058.

Green, T. J., Karvounarakis, G. and Tannen, V. (2007): Provenance semirings. *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database System*, **26**: 31-40.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009): The WEKA data mining software: an update. *SIGKDD Explorations*, **11(1)**: 10-18.

Herschel, M. and Hernández, M.A. (2010): Explaining missing answers to SPJUA queries. *Proc. VLDB Endowment*, **3(1)**: 185-196.

Huang, J., Chen, T., Doan, A.H. and Naughton, J.F. (2008): On the provenance of non-answers to queries over extracted data. *Proc. VLDB Endowment*, **1(1)**: 736-747.

Jagadish, H.V., Chapman, A., Elkiss, A., Jayapandian, M., Li, Y., Nandi, A. and Yu, C. (2007): Making database systems usable. *Proc. ACM SIGMOD Conference*, 13-24.

Kadlag, A., Wanjari, A. V., Freire, J. and Haritsa, J. R. (2004): Supporting exploratory queries in databases. *Proc. Database Systems for Advanced Applications*, **9**: 594-605.

Kießling, W. and Köstler, G. (2002): Preference SQL - design, implementation, experiences. *Proc. International Conference on Very Large Data Bases*, **28**: 990-1001.

Koudas, N., Li, C., Tung, A.K.H and Vernica, R. (2006): Relaxing Join and Selection Queries. *Proc. International Conference on Very Large Data Bases*, **32**: 199-210.

Liu, B., Chiticariu, L., Chu, V., Jagadish, H.V. and Reiss, F. R. (2010): Automatic rule refinement for information extraction. *Proc. VLDB Endowment*, **3(1)**: 588-597.

Ma, Y., Mehrotra, S., Seid, D.Y. and Zhong, Q. (2006): RAF: An activation framework for refining similarity queries using learning techniques. DASFAA, 587-601.

Meliou, A., Gatterbauer, W., Moore, K.F. and Suciu, D. (2010): Why so? or why no? functional causality for explaining query answers. *Workshop on Management of Uncertain Data*, 3-17.

Mishra, C. and Koudas, N. (2009): Interactive query refinement *Proc. International Conference on Extending Database Technology*, **12**: 862-873.

Mitchell, T.M. (1997): *Machine Learning*. McGraw Hill.

Moon, T., Li, L., Chu, W., Liao, C., Zheng, Z. and Chang, Y. (2010): Online learning for recency search ranking using real-time user feedback. *Proc. ACM Conference on Information and Knowledge Management*, **19**:1501-1504.

Motro, A. (1988): Vague: A user interface to relational databases that permits vague queries. *ACM Trans. Inf. Syst.,* **6(3)**: 187-214.

Motro, A. (1994): Cooperative database systems. *Proc. International Conference on Flexible Query Answering Systems*, 1-16.

Nambiar, U. and Kambhampati, S. (2003): Answering imprecise database queries: a novel approach. *Proc. ACM CIKM International Workshop on Web Information and Data Management*, **5**: 126-133.

Nambiar, U. and Kambhampati, S. (2004): Mining approximate functional dependencies and concept similarities to answer imprecise queries. *Proc. International Workshop on the Web and Databases*, **7**: 73-78.

Nambiar, U. and Kambhampati, S. (2005): Answering imprecise queries over web databases. *Proc. International Conference on Very Large Data Bases*, **31**: 1350-1353.

Nandi, A., and Jagadish, H. V. (2007): Assisted querying using instant-response interfaces. *Proc. ACM SIGMOD Conference*, 1156-1158.

Stasiu, R.K., Heuser, C. A. and Silva, R. D. (2005): Estimating recall and precision for vague queries in databases. *Proc. International Conference on Advanced Information Systems Engineering*, **17**: 187-200.

Sultana, K.Z., Bhattacharjee, A., Amin, M.S. and Jamil, H.M. (2009): A model for contextual cooperative query answering in e-commerce applications. *Proc. International Conference on Flexible Query Answering Systems*, **8**: 25-36.

Tan, W. C. (2004): Research problems in data provenance. *IEEE Data Eng. Bull.*, **27(4)**: 45-52.

Tran, Q. T. and Chan, C. Y. (2010): How to conquer why-not questions. Proc. *ACM SIGMOD Conference*, 15-26.

Voorneveld, M. (2003): Characterization of Pareto dominance. *Oper. Res. Lett.*, **31(1)**: 7-11.