

QRelX: Generating Meaningful Queries that Provide Cardinality Assurance*

Manasi Vartak
Worcester Polytechnic Institute
mvartak@wpi.edu

Venkatesh Raghavan
Worcester Polytechnic Institute
venky@cs.wpi.edu

Elke A. Rundensteiner
Worcester Polytechnic Institute
rundenst@cs.wpi.edu

ABSTRACT

In many business and consumer applications, queries have cardinality constraints. However, current database systems provide minimal support for cardinality assurance. Consequently, users must adopt a cumbersome trial-and-error approach to find queries that are close to the original query but also attain the desired cardinality. In this demonstration, we present *QRelX* – a novel framework to automatically generate alternate queries that meet the cardinality and closeness criteria. *QRelX* employs an innovative query space transformation strategy, proximity-based search and incremental cardinality estimation to efficiently find alternate queries. Our demonstration is an interactive game that allows the audience to compete with *QRelX* via manual query refinement. We illustrate the importance of cardinality assurance through real-time comparisons between manual refinement and *QRelX*. We also highlight the novelty of our solution by visualizing the core algorithms of *QRelX*.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems - Relational Databases

General Terms

Algorithms, Design, Management

Keywords

Query Refinement, Cardinality Assurance

1. INTRODUCTION

In many application domains ranging from business intelligence and decision support to travel services and web search, user queries often have cardinality constraints. However, in spite of the frequent presence of cardinality constraints, database systems currently provide minimal support for cardinality assurance. Lack of

*This work is supported by the National Science Foundation under Grant No. IIS-0633930, CRI-0551584 and CRA-W CREU Grant for 2009-10.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.
Copyright 2010 ACM 978-1-4503-0032-2/10/06 ...\$10.00.

automatic cardinality assurance has led to many applications having the following problems related to query result size: (1) queries return no results (*empty result-set* [4] or *failing query* [7] problem); (2) queries return too many results (*too many* problem [2]) or (3) queries return too few results (*too few* problem [3]). To illustrate these problems, consider the examples described below.

1.1 Motivating Examples

EXAMPLE 1. *Medical researcher A wants to conduct an obesity study examining the link between obesity and low income. Since A's grant provides him with resources to study 2000 volunteers, A formulates the query Q_1 to select these volunteers. However, Q_1 is able to identify only 1300 candidates in the medical database.*

```
Q1: SELECT * FROM PatientRecords
WHERE (25 < age < 55) AND (BMI > 30)
AND (income < $55000) AND (familyHistory = 0)
```

Without any explanation for the deficiency of results or suggestions for alternate queries producing 2000 results, the burden of refining Q_1 falls on researcher A. However, manual query refinement is tedious because A lacks knowledge about the characteristics of the underlying data set. He must undertake a frustrating trial-and-error process and attempt what can potentially be a very large number of refined queries. For instance, the following modifications form a small subset of the possible refinements of Q_1 : (1) expanding the range to 25 - 60 years, (2) increasing the income threshold to \$65,000, (3) allowing the individual to have a family history of obesity, or (4) any combination of the above. Even for a simple query like Q_1 , which spans a single table and contains four select predicates, a user must try numerous refined queries. An additional constraint on the refinement process is the requirement that the refined query should not grossly alter the original query semantics. For instance, refining the income threshold to \$85,000 may return the required number of results, but such a query will violate the low income selection criteria for volunteers. Such radical alterations to a query can be undesirable, and hence, in this work, we focus not only on cardinality, but also on closeness to the original query submitted by the user.

Manual query refinement is further complicated when a query combines two or more tables, and hence requires refinement of join predicates as shown in Example 2.

EXAMPLE 2. *Consider the data collected by two physiological sensors X and Y that monitor a patient's blood oxygen content and blood pressure respectively. Scientist B wants to study the drop in oxygen content when blood pressure goes below the normal threshold. Further, to have statistically significant findings, she requires at least 10,000 pairs of readings that satisfy specific temporal and*

physiological conditions as defined in Q_2 . However, \mathbf{B} 's query only returns 5,000 pairs of readings.

```
Q2: SELECT * FROM SENSOR X, SENSOR Y
WHERE X.time = Y.time AND X.bloodOxygen < 80
AND Y.systolic < 90 AND Y.diastolic < 60
```

In this scenario, \mathbf{B} must manually refine query Q_2 until it returns 10,000 results. Moreover, unlike query Q_1 , Q_2 can be reformulated by refining only its select predicates, only its join predicate, or a combination of both types of predicates. Join predicate refinement is especially beneficial for Q_2 as it may enable us to meet our cardinality constraint without significantly altering the original query semantics. Further, join refinement for this data set will allow approximate matching of timestamps to accommodate naturally occurring physiological and sensor delays.

1.2 Problem Description

Database systems can eliminate the need for manual query refinement and improve end-user experience if they provide automatic cardinality assurance. Thus, the problem can be stated as follows. Given a user query Q and its expected cardinality C , we seek to automatically generate alternate “meaningful” queries that attain the expected cardinality. Since users prefer alternate queries close to the original query, an alternate query is considered “meaningful” if it doesn’t grossly alter the original query. Formally, an alternate query minimizes the distance to the original query. We refer to the above query evaluation criteria as the *cardinality* criterion and *closeness* criterion respectively. Lastly, we aim to provide the user a set of alternate queries, as opposed to a single one, because the user can then employ domain expertise to pick the query most suited for the given application.

While the automated generation of alternate queries is highly desirable, it poses several challenges. First, exact cardinality assurance is an NP-Hard problem [1]. Second, the number of possible combinations of individual predicate refinements increases exponentially with the number of predicates. Third, the amount of refinement required to meet the cardinality constraint cannot be predicted *a priori*. Fourth, executing each refined query is extremely database resource intensive, especially if the query is long-running. Last, the additional constraint of minimizing distance to the original query further increases the complexity of the problem.

1.3 State-of-the-Art Approaches

Traditionally, *query refinement* has been used to selectively modify subsets of query predicates to alter query cardinality [3, 5, 7]. However, current techniques do not simultaneously address the cardinality and closeness criteria discussed above. For example, techniques proposed in [1, 6] generate queries for testing databases. Since query semantics are irrelevant for testing purposes, these techniques focus on attaining specific query cardinality without regard to the meaningfulness of the query. On the other hand, [3, 7], which address the problem of failing queries, emphasize the closeness of query results rather than the result size and do not produce alternate queries. In [5], the authors propose a semi-automatic approach to query refinement. However, the proposed approach cannot minimize distance to the original query or refine join predicates. This approach also requires a high degree of user involvement through the refinement process. Although techniques based on scoring functions have been proposed for cardinality assurance, these techniques do not provide an overall query characterizing the results produced.

In this demonstration, we present *QRelX* – the first framework that supports the generation of alternate queries meeting the cardinality and closeness criteria.

2. THE QRELX FRAMEWORK

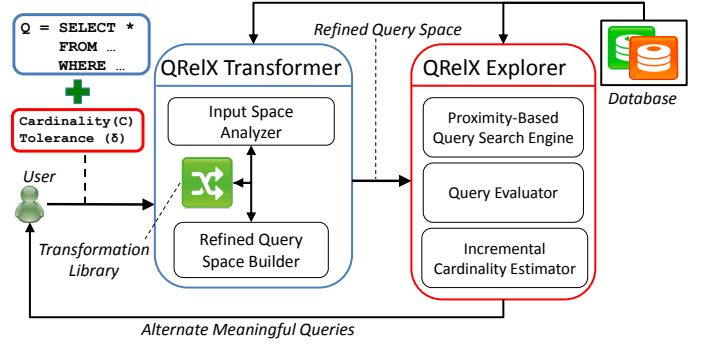


Figure 1: System architecture for QRelX

Figure 1 presents an overview of *QRelX*, our query refinement framework. *QRelX* takes a conjunctive SPJ query Q , its desired cardinality C , and a cardinality tolerance threshold δ , to produce a set of alternate queries meeting the cardinality and closeness criteria. We introduce the cardinality tolerance threshold δ because approximately meeting the expected query cardinality is sufficient in many scenarios. Thus, the queries produced by *QRelX* are guaranteed to have cardinality within $C \pm \delta$ and minimum distance from Q .

The core functioning of the *QRelX* framework is divided between two key components: the *QRelX Transformer* and the *QRelX Explorer*. Given a user query, the *QRelX Transformer* employs an innovative *query space transformation strategy* to generate a *refined query space* that allows the refinement of select-project-join queries. Following this transformation, the *QRelX Explorer* searches the *refined query space* for alternate queries. Once queries that adhere to both the cardinality as well as closeness criteria are found, we report these alternatives to the user. The *QRelX Transformer* is in turn composed of the *Input Space Analyzer* and the *Refined Query Space Builder* which respectively analyze the input tables and construct the *refined query space*. Similarly, *QRelX Explorer* is composed of the *Proximity-Based Query Search Engine* which traverses the refined query space, the *Query Evaluator* which examines whether a query meets both acceptance criteria, and the *Incremental Cardinality Estimator* which calculates the cardinality of each examined query.

3. SALIENT FEATURES OF QRELX

Since the search for alternate meaningful queries poses several challenges, *QRelX* uses the key principles of query space transformation, proximity-based query search, and incremental cardinality estimation to efficiently generate alternate queries as described below.

3.1 Query Space Transformation

A shortcoming of existing query refinement techniques is their inability to relax join predicates. However, join refinement is essential in many applications where exact matches of data are rare. Therefore, *QRelX* adopts an innovative query space transformation strategy that enables the refinement of select-project-join queries using a uniform query search technique. Our space transformation strategy works in two stages: First, the *Input Space Analyzer*

processes the input tables at a coarse granularity to study the data distribution with respect to the original query. Second, the *Refined Query Space Builder* applies a refinement-sensitive transformation function to the input data, and creates a new *refined query space*. The *refined query space* has two interpretations: (1) it represents all possible refined queries ordered by their proximity measure and (2) it represents an abstract view of all input tuples based on their collective proximity. The former interpretation of the refined space helps to guide the alternate query search, while the latter helps to reduce computational costs by facilitating a “need-based” tuple evaluation strategy.

3.2 Proximity-based Query Search

Since a user always prefers alternate queries close to the original query, i.e., queries with low overall refinement, the *QRelX* search strategy is designed to preferentially examine queries close to the original one. The space transformation strategy described above ensures that within the *refined query space*, queries close to the original query lie near the origin, while those vastly different from the original query are far from the origin. The *refined query search engine* of the *QRelX Explorer* then uses this knowledge to examine refined queries in order of closeness to the original query, i.e., it initially examines queries close to the origin and then gradually moves outward. For each refined query, the *Incremental Cardinality Estimator* calculates its cardinality and passes the results to the *Query Evaluator*. The *Query Evaluator* determines whether the query satisfies both cardinality and closeness criteria to decide whether a broader query search is required. Since proximity-based search guarantees that queries close to the original query are examined before those far from it, *QRelX* can terminate the search for alternate queries as soon as refined queries are found to satisfy both acceptance criteria.

3.3 Incremental Cardinality Estimation

One of the chief drawbacks of traditional cardinality assurance techniques is the high computational cost associated with repeated query execution. These techniques have no memory of previously obtained cardinality estimations; they re-execute all queries regardless of similarities between them. The incremental cardinality estimation technique implemented by *QRelX* presents a new model for performing rapid cardinality calculations. Instead of undertaking redundant cardinality computations, incremental estimation reuses previously obtained query cardinality information via specialized linear recurrences based on the refined space configuration. This enables *QRelX* to examine only a small subset of tuples for each query, and compute the total cardinality through a handful of calculations. Incremental estimation therefore drastically reduces computational expenses by ensuring that once a tuple has been found to satisfy a given refined query, it is never re-evaluated for any other query in the *refined query space*.

4. DEMONSTRATION

In our demonstration, we present a real user scenario that illustrates the importance of cardinality assurance and the novelty of our approach. For this purpose, we have created an interactive gaming environment which allows the user to pose ad-hoc queries with cardinality constraints and compete with *QRelX* via manual query refinement to find alternate queries. Throughout the refinement process, we provide real-time qualitative and quantitative comparisons between manual query refinement and the automated *QRelX* approach. In addition, we visualize the core algorithms of *QRelX* to demonstrate the key ideas and innovative strategies used to generate alternate queries.

4.1 Demonstration Setup

For the demonstration, we implemented in Java the *QRelX* refinement framework as well as the interactive game portal that provides manual query refinement capabilities. *QRelX* takes as input a user query, the expected cardinality and the cardinality tolerance threshold to produce alternate queries. On the other hand, the manual query refinement framework allows the user to refine the original query predicates and submit refined queries to a simple cardinality estimator for result size calculations. To facilitate comparison between the manual and *QRelX* approaches, we run both techniques in parallel on duplicate copies of the underlying database.

Our demonstration system consists of three interfaces: the *QRelX Query Definition View*, the *QRelX Game view*, and the *QRelX System View* as depicted in Figures 2, 3, and 4 respectively. We now discuss these views in detail.

4.1.1 QRelX Query Definition View

The query definition view is the interface, as shown in Figure 2, that enables the user to formulate the initial query Q and specify its expected cardinality C . As the first step, the audience can submit query Q for execution to calculate its result cardinality.

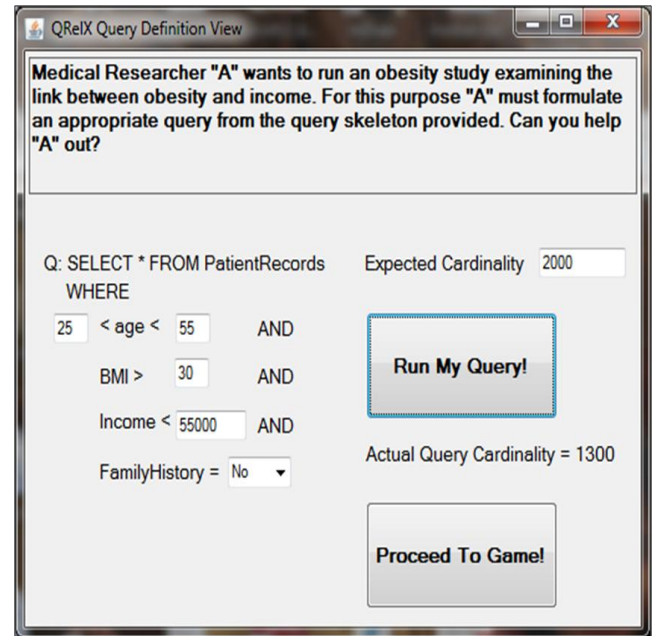


Figure 2: QRelX Query Definition View

4.1.2 QRelX Game View

The game view is the interface that allows the audience to compete with *QRelX* via manual query refinement. As presented in Figure 3, the GUI is divided into two panes: the left pane is dedicated to manual query refinement and the right to our *QRelX* framework. The manual refinement pane enables the user to repeatedly re-formulate the original query and evaluate it in real-time. In contrast, the *QRelX* pane displays real-time status information about *QRelX*. The game view also contains graphs in each pane to depict cardinality and proximity comparisons between the two techniques.

4.1.3 QRelX System View

The system view is the interface presenting run-time progress of *QRelX* and data about past computations conducted by it. As

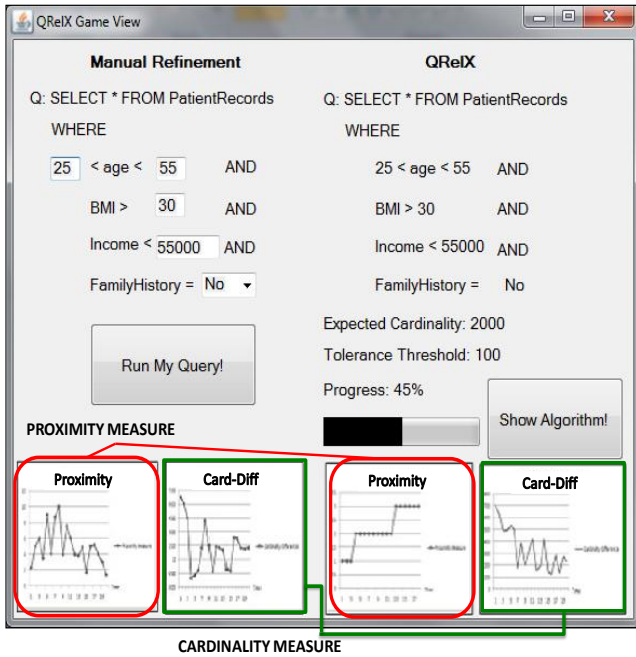


Figure 3: QRelX Game View

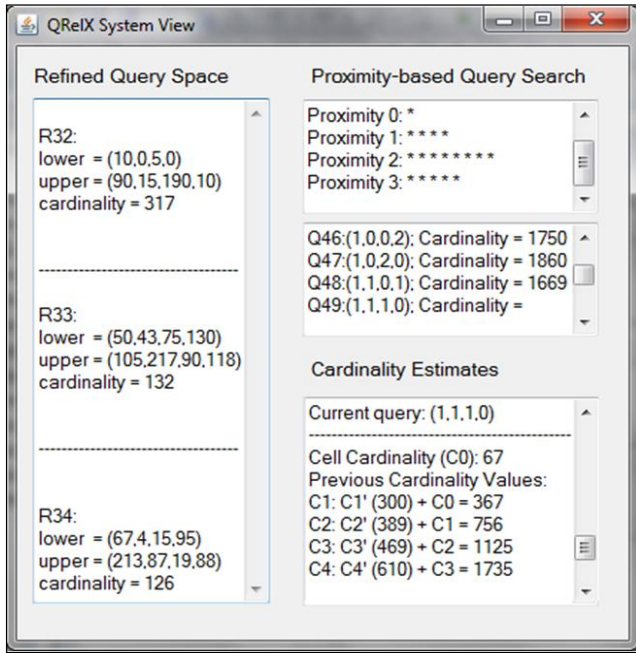


Figure 4: QRelX System View

Figure 4 depicts, the *QRelX System View* shows the key steps performed by the *QRelX* framework: (1) construction of the refined search space, (2) functioning of the proximity-based query search engine, and (3) results from incremental cardinality estimation.

4.2 Walk-through Example

Since our demonstration of *QRelX* is game-based, it is inherently interactive, and consists of the following sequence of steps:

Step 1: The user specifies a query and the associated cardinality

constraint in the *QRelX Query Definition View* as in Figure 2. The query is then executed to calculate its cardinality. If the initial query does not meet the cardinality criteria, the user proceeds to the *QRelX Game View*.

Step 2: Once the user moves to the game view, the system starts *QRelX* execution. Concurrently, the user manually refines the initial query to meet the cardinality constraint. Each refined query formulated by the user is executed to calculate its cardinality. Moreover, for every query execution (through manual or automatic refinement), graphs in the game view are updated with the current query's cardinality and its distance to the original query. Based on our experiments [8], we expect *QRelX* to have a monotone increasing graph for distance to the original query and a decreasing graph for difference between the expected and actual cardinality. In contrast, however, we expect the graphs from manual query refinement to be uneven and contain significant noise.

Step 3: During the query refinement process, the user can view computations performed by *QRelX* in real-time via the *QRelX System View* as shown in Figure 4.

Step 4: For a user to win the game, the user must formulate a refined query which is better than *QRelX* in terms of cardinality difference, proximity to the original query, and total refinement time. The *QRelX* approach wins if it can produce a better answer in a shorter time.

The demonstration of *QRelX* thus presents the audience with a real-world scenario that illustrates the importance of cardinality assurance and provides hands-on interaction with our framework.

5. CONCLUSION

In this demonstration, we present *QRelX* our novel framework to automate the generation of meaningful queries that provide cardinality assurance. *QRelX* employs an innovative query space transformation strategy, proximity-based search and incremental cardinality estimation to efficiently find queries meeting the cardinality and closeness criteria. Our interactive game-based demonstration allows users to compete against *QRelX* via manual query refinement to formulate alternate queries. In addition, we visualize the core algorithms of *QRelX* and present qualitative as well as quantitative comparisons between manual and automated query refinement.

6. REFERENCES

- [1] N. Bruno, S. Chaudhuri, and D. Thomas. Generating queries with cardinality constraints for dbms testing. *IEEE Trans. Knowl. Data Eng.*, 18(12):1721–1725, 2006.
- [2] M. J. Carey and D. Kossmann. On saying "enough already!" in sql. In *SIGMOD*, pages 219–230, 1997.
- [3] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica. Relaxing join and selection queries. In *VLDB*, pages 199–210, 2006.
- [4] G. Luo. Efficient detection of empty-result queries. In *VLDB*, pages 1015–1025, 2006.
- [5] C. Mishra and N. Koudas. Interactive query refinement. In *EDBT*, pages 862–873, 2009.
- [6] C. Mishra, N. Koudas, and C. Zuzarte. Generating targeted queries for database testing. In *SIGMOD*, pages 499–510, 2008.
- [7] I. Muslea. Online query relaxation. In *SIGKDD*, pages 246–255, 2004.
- [8] M. Vartak, V. Raghavan, and E. Rundensteiner. Technical report: Query oriented refinement for cardinality assurance. In *progress*, 2010.