# Erica: Query Refinement for Diversity Constraint Satisfaction


Jinyang Li
University of Michigan
jinyli@umich.edu

Alon Silberstein
Ben Gurion University of the Negev
alonzilb@post.bgu.ac.il

Yuval Moskovitch
Ben Gurion University of the Negev
yuvalmos@bgu.ac.il

Julia Stoyanovich
New York University
stoyanovich@nyu.edu

H. V. Jagadish
University of Michigan
jag@umich.edu



## ABSTRACT

Relational queries are commonly used to support decision making in critical domains like hiring and college admissions. For example, a college admissions officer may need to select a subset of the applicants for in-person interviews, who individually meet the qualification requirements (e.g., have a sufficiently high GPA) and are collectively demographically diverse (e.g., include a sufficient number of candidates of each gender and of each race). However, traditional relational queries only support selection conditions checked against each input tuple, and they do not support diversity conditions checked against multiple, possibly overlapping, groups of output tuples. To address this shortcoming, we present Erica, an interactive system that proposes minimal modifications for selection queries to have them satisfy constraints on the cardinalities of multiple groups in the result. We demonstrate the effectiveness of Erica using several real-life datasets and diversity requirements.



**PVLDB Reference Format:**
Jinyang Li, Alon Silberstein, Yuval Moskovitch, Julia Stoyanovich, and H. V. Jagadish. Erica: Query Refinement for Diversity Constraint Satisfaction. PVLDB, 16(12): 4070 - 4073, 2023.
doi:10.14778/3611540.3611623

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/alons9911/Query_refinement.


## 1 INTRODUCTION

With the increasing awareness of the importance of diversity and representation, many companies, educational institutions, professional societies, and other organizations around the world are focusing on developing their diversity recruiting strategy. Query processing in relational databases is often used to select and prioritize candidates in such settings. Traditional relational queries specify conditions as part of the query predicates, and produce tuples that satisfy these predicates. If a query is used as part of some high-stakes selection process, then it would be natural to also state diversity requirements — as cardinality constraints over some demographic groups in the query result — as we demonstrate next.



Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611623


| ID | Gender | Race | Major | GPA | Q1 | Q2 |
|---|---|---|---|---|---|---|
| 1 | F | White | ME | 3.65 | | |
| 2 | F | White | CS | 3.95 | ✓ | ✓ |
| 3 | F | Black | CS | 3.40 | | |
| 4 | F | White | ME | 3.60 | | |
| 5 | F | White | EE | 3.85 | | |
| 6 | F | Black | EE | 3.90 | | ✓ |
| 7 | F | Asian | EE | 3.85 | | |
| 8 | M | White | CS | 3.65 | | |
| 9 | M | White | CS | 3.90 | ✓ | ✓ |
| 10 | M | Black | CS | 3.85 | ✓ | |
| 11 | M | White | CS | 3.40 | | |
| 12 | M | White | EE | 3.65 | | |
| 13 | M | Asian | EE | 3.95 | | ✓ |
| 14 | M | Black | ME | 3.60 | | |

**Figure 1: Job applicant dataset used in Example 1.1. Applicants selected by queries Q1 and Q2 are marked with ✓.**

Example 1.1. *Figure 1 shows a dataset* D *of 14 job candidates applying to a tech company, with four attributes: gender, race, (college) major, and GPA. Major and GPA are the* qualification attributes *that may be used directly as part of the selection process. Gender and race are the* sensitive attributes *that denote membership in demographic groups, and to avoid direct discrimination in decision-making, they are not directly specified in the selection conditions of a query. However, to counteract the consequences of historical discrimination, sensitive attributes may be used to state cardinality constraints for the result set. To invite applicants with technical majors and high GPAs for an interview, the company employs the following query:*

```
Q1: SELECT *
    FROM Applicants AS a
      WHERE a.Major in ('CS') AND a.GPA >= 3.85
```

*Candidates #2, #9, and #10 are selected by query* Q1*, as marked in the corresponding column in Figure 1. However, this result does not meet the diversity requirements of this company: only one female applicant is selected, although half of the applicants are female, and no Asian is selected, although Whites, Blacks and Asians are all present in the applicant pool. To improve diversity, the company would like to interview at least two female applicant, and at least one applicant of each race. Additionally, to avoid overwhelming the human resources department, the total number of selected candidates should not exceed five. These requirements can be expressed as a set of* cardinality constraints *over some groups in the query result set.*

Diversity is a compelling need when distributing access to resources and opportunities in a society, as we see in our running example. Furthermore, it is also desirable in many other settings. For example, we usually want search and recommendation systems to produce diverse results [4]. The techniques we present here are applicable to all these contexts.

One way to satisfy the diversity constraints is to modify the result set directly, adding or removing some tuples: Adding candidate #7, who is both female and Asian, to the result of Q1 would meet the requirements. However, this method may be illegal in some jurisdictions and application contexts (e.g., it would be illegal in the US in the context of employment, housing, and lending), because it uses demographic group membership explicitly as part of decision making, effectively subjecting applicants from different demographic groups to different processes. Even where legal, this method may have undesirable side effects, such as tarring all members of a group, including its best-qualified members, as "weaker" on account of there being a different standard applied to the group.

Unlike the explicit use of demographic attributes, modifying the query predicates is usually legal as they serve as qualification attributes for all candidates, irrespective of their demographic group membership. For instance, the University of Texas faced a notable case where explicit preference for Black applicants was disallowed, but they were permitted to use rank in school as an admission criterion, thereby admitting top students from poor-performing segregated schools in preference to second-tier White students from top schools, even if they have higher standardized test scores.

Returning to our running example, the company may slightly adjust some selection criteria, such as adding Electrical Engineering (EE) as an accepted major, to select more female applicants. However, this would expand the result set size beyond 5 candidates to interview, so refining the query again by *restricting* the GPA to at least 3.90 helps. The following query Q2 is a refinement that satisfies per-group and total cardinality constraints.

```
Q2: SELECT *
    FROM Applicants AS a
      WHERE a.Major in ('CS', 'EE') AND a.GPA >= 3.90
```

To assist the decision maker in refining their queries, we present Erica (for quEry Refinement for dIversity Constraint sAtisfaction), a system that generates minimal query refinements to satisfy cardinality constraints on groups in the result. There are two main challenges. First, executing candidate query modifications to assess constraint satisfaction can be time-consuming, especially with large datasets or remote databases. Second, the abundance of predicate variations and the large number of predicates within a query make exhaustive analysis computationally expensive. As proved in [7], there is no polynomial time algorithm to solve this problem.

To address the first challenge, Erica uses provenance annotations [5] and translates constraints into algebraic expressions [2] that can be used to test whether the queries satisfy the constraints. To address the second challenge, Erica implements a search algorithm that allows for efficient traversal over possible refinements using a dedicated data structure.

## 2 TECHNICAL BACKGROUND

We next informally introduce the model and algorithms underlying Erica. Please see [7] for more details.

### 2.1 Model

*Query refinement.* We support the class of queries considered in [8], conjunctive Select-Project-Join (SPJ) queries with selection predicates over *numerical* or *categorical* attributes. Selection predicates over numerical attributes include range ($<, \leq, \geq, >$) and equality (=). Categorical predicates are of the form `attribute in (constant_1,...,constant_n)`[1]. For ease of presentation, in the rest of the paper, we assume numerical predicates are of the form `attribute <op> constant`.[2]

We use the notion of query refinement defined in [8] to formally state our problem. For a numerical predicate, refinements are changes to the value of the constant; for categorical predicates, a refinement is done by adding or removing predicates from the original constant list. We say that a query $Q'$ is a *refinement* of query $Q$ if $Q'$ is obtained from $Q$ by refining some predicates of $Q$.

*Cardinality constraints.* Let $Q(D)$ be the result of executing query $Q$ over dataset $D$, and $\mathcal{G}$ be a group defined by specifying the value of some attributes. A cardinality constraint $Cr$ over a group $\mathcal{G}$ in $Q(D)$ is a conjunction of expressions of the form $|Q(D)_{\mathcal{G}}|$ $op$ $x$, where $op \in \{=, \leq, <, >, \geq\}$ and $x$ is a constant.

*Minimal refinements.* Given a dataset $D$, a query $Q$, and a set of cardinality constraints $Cr$ such that $Q(D)$ does not satisfy $Cr$, there may be multiple ways to refine $Q$ in order to satisfy the constraints, as we demonstrate next.

Example 2.1. *Revisiting our running example, query* Q2 *is a refinement of* Q1 *obtained by applying refinements to the predicates over GPA and Major.* Q2 *satisfies all cardinality constraints, e.g., it selects at least two females:* $|Q(D)_{Gender=F}| \geq 2$. *Another possible refinement of* Q1 *that satisfies all constraints is* Q3*: adjusting the Major predicate to be* `a.Major in ('EE')`.

The above example suggests that the company can achieve its diversity goal through various query modifications. Minimal modifications to the original query are preferred, prioritizing `a.GPA >= 3.60` over `a.GPA >= 3.55`. However, refinements modifying different attributes might be incomparable without additional preference information provided by the end user (as long as they all satisfy the constraints). Thus, we define the set of minimal refinements.

For query $Q$ and its refinements $Q'$ and $Q''$, $Q'$ *dominates* $Q''$ if $Q'$ is "closer" to $Q$ than $Q''$ for every refined attribute. A *minimal refinement* of $Q$ w.r.t. $Cr$ satisfies three conditions: (i) it is a refinement of $Q$, (ii) $Q'(D)$ satisfies $Cr$, and (iii) there is no other refinement $Q''$ that satisfies conditions (i) and (ii) while dominating $Q'$. Multiple minimal refinements may exist, and the aim is to report all of them. However, there may be cases where no refinement satisfies the constraints, resulting in an empty set of refinements.

Example 2.2. *For our running example in Figure 1, query* Q1*, and five cardinality constraints,* Q2 *and* Q3 *(from Example 2.1) are both minimal refinements. Although* Q4 *with* `a.Major in ('EE')` *and* `a.GPA >= 3.65` *also satisfies all the constraints, it is not minimal since it is dominated by both* Q2 *and* Q3.

### 2.2 Generating minimal refinements

Given a query, the number of possible refinement queries can be extremely large, especially when dealing with queries involving multiple tables and numerous attributes. Evaluating their satisfaction of cardinality constraints can be expensive, particularly with large or remote databases, and/or a large number of constraints. However, our solution eliminates the need for costly query evaluation by utilizing data annotations based on provenance theory. Our

[1]Equivalent to `attribute = constant_1 OR...OR attribute = constant_n`
[2]Extending our solution to support other forms of numerical predicates, such as `attribute1 <op> constant·attribute2` is straightforward, and we do not discuss them in this paper.

solution utilizes provenance to find the set of minimal refinements, and leverages hypothetical reasoning [2] to examine the effect of possible relaxations on the outcome of the query.

*Provenance model.* To capture possible refinements, we employ conditional tables (c-tables) [6], to annotate tuples in the data with the query selection conditions. Specifically, we use a set of variables $A_{[t.A]}$ for each selection predicate $A$ in the query and each value $t.A$ in the attribute $A$, for some tuple $t$ in the data. The annotation of a tuple $t$, $prov(t)$, is the product $\prod A_{[t.A]}$ of the variables that correspond to the attributes in the query and their values in $t$. Using these provenance annotations, we can express cardinality constraints with inequalities. The provenance inequality of the constraint $Q(D)_{\mathcal{G}}\ op\ x$ is $\sum_{t \in Q_{\mathcal{G}}} prov(t)\ op\ x$.

Example 2.3. *The following is the provenance inequality corresponding to the condition* $|Q(D)_{Gender=F}| \geq 2$

$$M_{ME} \cdot G_{3.65} + M_{CS} \cdot G_{3.95} + M_{CS} \cdot G_{3.40} + M_{ME} \cdot G_{3.60}$$
$$+ M_{EE} \cdot G_{3.85} + M_{EE} \cdot G_{3.90} + M_{EE} \cdot G_{3.85} \geq 2$$

For a query $Q$ over the database $D$, each possible refinement $Q'$ corresponds to a *valuation* — an assignment of values to the variables $\mathcal{V}_{Q(D)}$ in the corresponding provenance expression. Variables that satisfy the query are assigned the value 1, while those that do not are assigned the value 0. This approach effectively represents the query's conditions and adapts to queries with varying conditions or constants as we explore refinements. To account for cardinality constraints, these values are then aggregated, enabling us to assess the impact of query refinements on their satisfaction.

Example 2.4. *In Example 2.3, the valuation of query* Q1 *assigns* 1 *to* $M_{CS}$ *and* 0 *to* $M_{EE}$ *and* $M_{ME}$*, as only tuples with Major CS satisfy* Q1*. Similarly,* 1 *is assigned to* $G_{3.95}$*,* $G_{3.90}$*, and* $G_{3.85}$*, and* 0 *to other GPA variables. With this assignment, only* $M_{ME} \cdot G_{3.65} = 1$*, making the left-hand side sum* 1*, and so the inequality does not hold.*

A query satisfies a set of cardinality constraints if and only if the corresponding assignment satisfies the associated provenance inequalities. By analyzing these inequalities, we can efficiently assess the impact of refinements on constraint satisfaction without accessing data or executing potential refinements.

*Search algorithm.* With a provenance model to test potential refinements, the next question is how to find all the minimal refinements efficiently. Our algorithm employs Possible Value Lists (PVL) to represent potential refinements for each attribute. Numerical predicates ($A\ op\ c$) are stored in a single list ($l_A$) of possible values for $A$, sorted by their absolute distance from $c$ ($|x - c|, x \in l_A$). Categorical predicates ($A \in C$) are represented using multiple lists, including $l_{A_v}$ for each possible value $v$ of $A$, with values of 1 (existence) or 0 (absence) to indicate the presence of $v$ in $C$. The order of 1 and 0 is determined based on whether $v$ is already part of $p_c$.

Each minimal refinement can be represented as a list of indices, one for each list in the PVL. The algorithm uses the PVL to locate minimal refinements. Starting with the initial index in each list (corresponding to the original query), the algorithm incrementally increases the indices until the query satisfies the cardinality constraints. Based on the definition of minimality, any other minimal refinement must have a smaller index in at least one list than in the discovered query. Thus, the algorithm fixes the index in one list with a smaller index and recursively searches for more refinements. We show in [7] that our algorithm for searching minimal refinements

| | $l_{M_{EE}}$ | $l_{M_{ME}}$ | $l_{M_{CS}}$ | $l_G$ |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 3.85 |
| 2 | 1 | 1 | 0 | 3.90 |
| 3 | | | | 3.95 |
| 4 | | | | 4.00 |
| 5 | | | | 3.65 |
| 6 | | | | 3.60 |
| 7 | | | | 3.40 |

**Figure 2: Possible value lists (PVL) of the running example.**

using the PVL is sound, complete, and can run over 100 times faster than naively iterating over all possible refinements. Furthermore, when all constraints are relaxation or contraction constraints (i.e., all inequalities are of the form $C(D)_{\mathcal{G}}\ op\ x$, where $x$ is a constant and $op \in \{>, \geq\}$ (or $op \in \{<, \leq\}$) for *all* the constraints), we can optimize the search further by leveraging a monotonicity property.

Example 2.5. *Figure 2 shows the PVL of our running example, containing one list for the GPA predicate and three for the Major predicate. To find the minimal refinements satisfying all constraints (Example 2.1), the algorithm uses a list of four indices to denote the index in each of the four lists, initialized with the smallest indices* $[1, 1, 1, 1]$ *for the original query* Q1*. The algorithm increases the indices until the corresponding query satisfies the constraints. The minimal refinement with indices* $[2, 1, 1, 2]$ *(highlighted in blue) is found. Based on the definition of minimality, other minimal refinements must have index* 1 *in* $l_{M_{EE}}$ *and/or index* 1 *in* $l_G$*. The algorithm then fixes the index in* $l_{M_{EE}}$ *to be* 1 *and recursively searches for minimal refinements in the other three lists, followed by doing the same for* $l_G$*.*

## 2.3 Related work

Prior research on query refinement [1, 8] focuses on introducing slight query modifications to ensure the overall output satisfies specified cardinality constraints. However, cardinality constraints over *specific groups* in the output are not supported by these works.

Our work shares motivation with [9], aiming to satisfy group cardinality constraints over a query result. However, their work differs from ours in several important ways: (1) They only handle constraints over a single binary sensitive attribute (e.g., either gender or race), while we handle multiple sensitive attributes (e.g., both gender and race) and do not limit them to binary; (2) We support both query relaxation (i.e., generating more result tuples) and query contraction (i.e., generating fewer result tuples) while [9] only supports query relaxation; (3) Their objective is to minimize the distance between the *result sets* produced by the original and the rewritten query, while we aim to minimize the distance between the *queries themselves*; (4) They support queries over a single relation, while we support SPJ queries with predicates and constraints on attributes across multiple tables (see [7] for details).

Our problem shares some similarities with work on why-not questions [10, 11], where the goal is to explain missing tuples in the output by refining the query. The main difference is that why-not questions aim to include user-specified missing tuples, while we aim to modify the cardinality of groups in the output.

# 3 SYSTEM OVERVIEW

Erica is implemented in Python 3 and ReactJS, designed to be deployed on personal computers. Figure 3 shows the input and output screens through which the user interacts with the system and explores query refinements.

*Input Screen (Figure 3a).* The user provides Erica with data, a selection query, and a set of cardinality constraints. They can upload their dataset or choose from pre-loaded ones [1]. Erica offers an easy-to-use interface to build selection queries by defining conditions over the attributes [2]. The generated query [3] and its output [4] are presented to the user. Users can also specify cardinality constraints [5]. The query is executed, and the cardinality of each group in the output is displayed [6]. If constraints are not met, users can ask Erica to refine the query by clicking the execution button [7].

*Output Screen (Figure 3b).* Once the minimal refinements are computed, Erica displays all of them to the user, along with group cardinalities in the result [8]. Modified selection conditions are highlighted in blue, and selected tuples are displayed next to each refinement [9]. New tuples are highlighted in green, and removed tuples are shown in red with a strike-through. Furthermore, users can also sort refinements based on different properties [10], such as how much a particular selection condition, the cardinality of a group, or the result set is changed. For example, in Figure 3b, refinements are sorted based on changes in `grade1` selection condition [11].

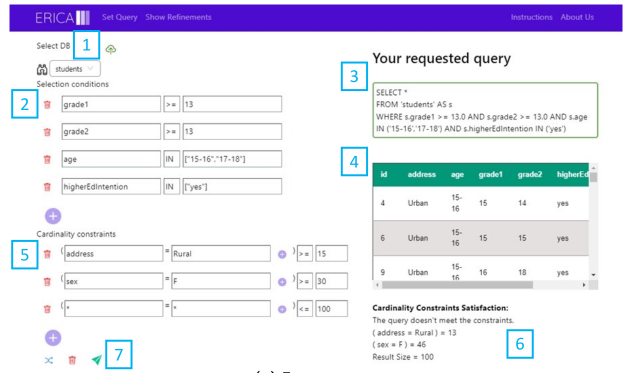
## 4 DEMONSTRATION PLAN

*Overview.* We will demonstrate the functionality of Erica using three real-life datasets: (1) **Student**[3], containing demographic and grade data of 395 students from two Portuguese high schools; (2) **Healthcare**[4], with demographics and clinical history of 887 patients; and (3) **ACSIncome**, with demographics and socioeconomic information of 1,664,500 US Census respondents [3]. Attendees will specify queries and cardinality constraints, and observe refinements produced by the system. The running time of Erica can vary based on dataset size and characteristics, and on whether the monotonicity optimization can be used (i.e., whether all constraints are relaxations or contractions). For a query with 2-5 predicates and 2-5 cardinality constraints, Erica runs in interactive time for all three datasets: within 1-2 seconds when the optimization can be used, and within 5-60 seconds otherwise.

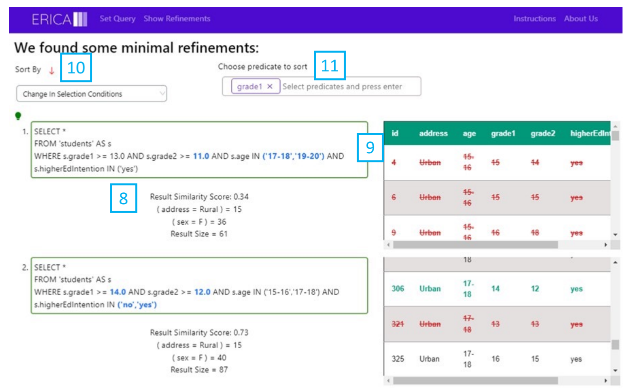
*Demonstration scenario.* Let us consider the **Student** dataset and play the role of a policymaker selecting students to receive college application guidance. First, as shown in Figure 3a, we enter selection conditions: good grades (`grade1` and `grade2` are at least 13 out of 20), age between 15 and 18, and an expressed interest in attending college. Next, we add cardinality constraints to the query results to ensure adequate representation for underprivileged groups. We aim to select a minimum of 30 female students and at least 15 students from rural areas, making sure that students from these groups receive information about colleges, and are encouraged to apply. We also limit the total number of students to 100 due to resource constraints. Upon executing the query, we obtain 100 students, with 46 being female, satisfying the first constraint. However, only 13 students come from rural areas, falling short of the desired 15. Therefore, we ask Erica to refine the query. We navigate to the output view, where Erica has identified five minimal refinements that satisfy all constraints. Changes in the selection conditions and result sets are highlighted for us to review. Sorting the refinements based on how much the condition `grade1 >= 13` is changed (as in Figure 3b), we observe variations from no change to a 2-point adjustment. This helps the user find a refinement that best suits their preference. Other properties, such as changes in group cardinalities and overall results, can also be used to sort refinements.

[3]https://archive.ics.uci.edu/ml/datasets/student+performance

[4]https://github.com/stefan-grafberger/mlinspect/tree/master/example_pipelines/healthcare




**(a) Input screen**




**(b) Output screen**

**Figure 3:** UI of Erica.

## ACKNOWLEDGMENTS

This work was supported in part by NSF under grants 1741022, 1916505, 1922658, 1934565, and 2106176, and by ISF grant 2121/22.

## REFERENCES


[1] Wesley W. Chu and Qiming Chen. 1994. A structured approach for cooperative query answering. *ACM TKDE* 6, 5 (1994), 738–749.

[2] Daniel Deutch, Zachary G. Ives, Tova Milo, and Val Tannen. 2013. Caravan: Provisioning for What-If Analysis. In *CIDR*. www.cidrdb.org.

[3] Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. 2021. Retiring Adult: New Datasets for Fair Machine Learning. In *Proceedings of NeurIPS*.

[4] Marina Drosou, HV Jagadish, Evaggelia Pitoura, and Julia Stoyanovich. 2017. Diversity in big data: A review. *Big data* 5, 2 (2017), 73–84.

[5] Todd J. Green, Gregory Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of PODS*.

[6] Tomasz Imielinski and Witold Lipski Jr. 1984. Incomplete Information in Relational Databases. *J. ACM* 31, 4 (1984).

[7] Jinyang Li, Yuval Moskovitch, Julia Stoyanovich, and H. V. Jagadish. 2023. Query Refinement for Diversity Constraint Satisfaction. https://github.com/JinyangLi01/Query_refinement/blob/master/FullPaper/Query_Refinement.pdf.

[8] Chaitanya Mishra and Nick Koudas. 2009. Interactive query refinement. In *EDBT*.

[9] Suraj Shetiya, Ian P Swift, Abolfazl Asudeh, and Gautam Das. 2022. Fairness-aware range queries for selecting unbiased data. In *Proceedings of IEEE ICDE*.

[10] Quoc Trung Tran and Chee-Yong Chan. 2010. How to conquer why-not questions. In *Proceedings of ACM SIGMOD*. 15–26.

[11] Quoc Trung Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. 2009. Query by output. In *Proceedings of ACM SIGMOD*. 535–548.