# REQUIRED: A Tool to Relax Queries through Relaxed Functional Dependencies

Loredana Caruccio, Stefano Cirillo, Vincenzo Deufemia, Giuseppe Polese, Roberto Stanzione
Department of Computer Science
University of Salerno
Fisciano, Salerno, Italy
{lcaruccio,scirillo,deufemia,gpolese,rstanzione}@unisa.it

## ABSTRACT

Query relaxation aims to relax the query constraints in order to derive some approximate results when the answer set is small. In this demo paper, we present REQUIRED, an automatized, portable, and scalable query relaxation tool leveraging metadata learned from an input dataset. The intuition is to use relationships underlying attribute values to derive a new query whose approximate results still meet the user's expectations. In particular, REQUIRED exploits relaxed functional dependencies to modify the original query in two different ways: (*i*) relaxing some query conditions by replacing the equality constraints with ranges and/or collections of admissible values, and (*ii*) rewriting the original query by replacing some or all the attributes involved in the conditions of the query with attributes related to them. Our demonstration scenarios show that REQUIRED is effective in properly relaxing queries according to the considered strategy.

## 1 INTRODUCTION

With the proliferation of devices capable of connecting to the Internet, the Web has become populated with numerous platforms that integrate data from different sources and enable users to conduct even complex searches in a few steps. For instance, it is not unusual for a user to buy products on e-commerce platforms among thousands of available products, after performing several queries based on his/her needs.

*Motivation.* Considering the scenario provided above, it is worth to highlight that the capability of gathering proper results from a large amount of available data is not a trivial task, even for expert users (users, in the following), such as DBA and/or researchers, which would use such platforms for different kinds of analysis processes. In this scenario, since the user might not know the entire data collection, it is not uncommon to receive none or few results upon entering a query on a dataset. In such cases, the user must manually modify the query parameters until the desired results are obtained. This kind of operation is time-consuming, also providing an uncomfortable user experience. To address these issues, Query Relaxation techniques aim to automatically relax the parameters of a query in order to obtain additional results related to the ones of the original query [5].

*Contribution.* In this paper, we present REQUIRED (RElaxing QUerIes through RElaxed Dependencies), a query relaxation tool leveraging metadata extracted from the dataset to extend and relax the user's query and derive a larger answer set, which is still compliant with the intent expressed by the user. Through its simple graphical user interface, REQUIRED allows to execute queries on a dataset and visualize the obtained results. Moreover, it provides the possibility to enlarge the query answer set by considering a set of Relaxed Functional Dependencies (RFDs), which can be directly loaded from the tool interface. Notice that, REQUIRED does not prescribe to use a specific RFD discovery algorithm, but it is able to parse collections of automatically discovered RFDs and rank them based on a proper *utility strategy* (see Section 2.2). Thus, the user can select an RFD that will then be used for automatically deriving the relaxed query. As a consequence, REQUIRED is able to (*i*) extend the user's original query by replacing the equality constraint in accordance with the thresholds defined by the selected RFD, and/or (*ii*) completely rewrite the original query by replacing attributes recalled in the query conditions with those determined by the considered RFD. With respect to the first version of the methodology underlying REQUIRED [5], which has proven effective on several real-world datasets, in this work we enable the possibility to simply extend the query conditions, and define a proper RFD ranking strategy to allow users to first focus on RFDs that are more compliant with the conditions of the original query.

*Related Work.* In recent years, many proposals have explored the problem of query relaxation from a methodological point of view. Some of these works are based on the generation of special statistical summaries to support the rewriting of the user's query [13], whereas other approaches rely on the use of machine learning models. In particular, in [10] the authors use a sample of the dataset to learn decision rules, which are then considered by a nearest-neighbor model to generate an alternative query if the original one did not yield results. This method has been subsequently revisited in [11], as the previous version tended to generate queries that were too simple when the algorithm was trained on small datasets. A first proposal entailing the exploitation of data dependencies, i.e., Approximate Functional Dependencies (AFDs) [7], to capture relationships holding among the attributes and to pick a heuristic for the relaxation process, has been defined in [12]. In particular, the authors' intuition was that the attributes with the least influence on the other attributes should be relaxed first aiming to preserve the semantics of the original query as much as possible. On the other hand, other approaches involve the user in an interactive relaxation process, leading to the definition of semi-automatic tools. Among these, Albarrak et al. [1] proposed a Web-based prototype for refining range queries, while [9] introduces a Web application based on a probabilistic framework. More recent works explore the relaxation of queries on spatiotemporal data [2], and study the application of query relaxation for similarity search of RDF nodes [6]. Furthermore, Li *et al.* [8] applied query relaxation for semantic association search, which returns an entity-relation subgraph. With respect to these approaches, REQUIRED provides, to the best of our knowledge, both a high level of automation and a simple user interface, which better motivates users to engage in query relaxation processes.

## 2 RELAXING QUERIES THROUGH METADATA

In this section, we introduce the concepts of Functional Dependency (FD) and Relaxed Functional Dependency (RFD). Then, we present the query relaxation problem and the proposed solution.

### 2.1 Preliminaries

Consider $R$ to be a relation schema defined over a set of attributes $attr(R)$, where each attribute $A \in attr(R)$ has a domain $dom(A)$. Given an instance $r$ of $R$, a *Functional Dependency* (FD) $X \longrightarrow Y$ ($X$ *implies* $Y$) is an integrity constraint among two sets of attributes $X, Y \in attr(R)$. $X$ is referred to as left-hand side (LHS), while $Y$ is the right-hand side (RHS). Thus, by denoting with $t_i[X]$ the projection of the tuple $t_i$, the functional dependency $X \longrightarrow Y$ is said to be satisfied if and only if for every pair of tuples $(t_1, t_2)$, whenever $t_1[X] = t_2[X]$, then $t_1[Y] = t_2[Y]$. Over the years, the concept of FD has been extended since the equality constraint is too strict for real-world data, and, then, generalized under the concept of *Relaxed Functional Dependencies* (RFD), which can be classified in two categories: RFD$_c$ relaxing the equality constraint by means of a similarity/distance operator, and RFD$_e$s allowing for the tolerance of a limited number of tuples that violate the dependency. Notice that, since the methodology underlying the proposed tool only uses RFD$_c$s, in what follows we provide only the definition of this type of RFDs. For a more general definition of RFD see [4]. Formally, an RFD$_c$ is defined as follows:

$$\varphi : X_{\Phi_1} \longrightarrow Y_{\Phi_2}$$

where $X = \{X_1, \ldots, X_k\}$ and $Y = \{Y_1, \ldots, Y_h\}$ are attribute sets, and $\Phi_1 = \{\phi_{X_1}, \ldots, \phi_{X_k}\}$ and $\Phi_2 = \{\phi_{Y_1}, \ldots, \phi_{Y_h}\}$ are conjunctions of predicates, each defining a similarity/distance constraint to be satisfied over the attribute value comparison process.

### 2.2 Exploiting RFD$_c$s in query relaxation

A query relaxation process requires the modification of some of the constraints of a query with the likelihood that the database will contain more tuples satisfying them than those resulting from the original query.

More formally, given a relation instance $r$ of a relation $R$, and a query $Q$ whose answer set $Q(r)$, $|Q(r)| \leq |r|$ is derived through a set of query conditions $\Gamma = \{\gamma_{B_1}, \ldots, \gamma_{B_k}\}$ such that $\gamma_{B_i}$ has the form $(B_i = b_i)$, a query relaxation process aims to find a query $Q'$ whose answer set $Q'(r)$, $|Q(r)| \leq |Q'(r)| \leq |r|$ is derived through a novel set of query conditions $\Gamma'$, such that each tuple $t_j \in Q'(r)$ either satisfies conditions in $\Gamma$ (i.e., $t_j \in Q(r)$), or satisfies conditions *related* to those in $\Gamma$. To this end, we propose a query relaxation methodology that exploits the concept of RFD$_c$ in order to derive the set of *related* conditions $\Gamma'$.

The proposed methodology provides users with the possibility to select an RFD$_c$ $\varphi$ involving at least one attribute included in the conditions defined by $Q$, and exploits $\varphi$ to automatically define $Q'$ according to correlations among data contained in $\varphi$. In what follows, we describe the two implemented query relaxation strategies, which permit to (*i*) extend query conditions by including similar values, and (*ii*) change query conditions by considering some others defined on alternative attributes, but related to those involved in $Q$. Finally, we present the RFD$_c$ ranking strategy enabling the user to properly select the RFD$_c$ that will drive the query relaxation process.

*Extending query conditions through RFD$_c$s.* Given a query $Q$ and an RFD$_c$ $\varphi : X_{\Phi_1} \longrightarrow A_{\phi_2}$, for each $B \in X$ if $\gamma_B = (B = b) \in \Gamma$ (i.e., $\gamma_B$ is one of the conditions in $Q$), then we might rewrite $Q$ in $Q'$ by replacing $\gamma_B$ with $\gamma'_B = (b' \ like\ b)$, where *like* means that either $b - \varepsilon \leq b' \leq b + \varepsilon$ (if $B$ is a numerical attribute), with $\varepsilon$ derived from the RFD$_c$ threshold on $B$, or $b'$ IN $\{b_1, \ldots, b_l\}$ (if $B$ is a textual attribute), where $\{b_1, \ldots, b_l\}$ represent a collection of values similar to $b$ according to the similarity/distance constraint associated to $B$ in $\varphi$.

*Changing query conditions through RFDs.* Given an RFD$_c$ $\varphi : X_{\Phi_1} \longrightarrow A_{\phi_2}$ and the answer set $Q'(r)$ obtained from the previous query extension strategy, then we might rewrite $Q'$ in $Q''$ by replacing the condition $\gamma'_B$ from $\Gamma'$ according to the following formula:

$$\gamma''_B = \bigvee_{t \in Q'(r)} \left( \bigwedge_{C \in X \cup A \ \wedge \ \gamma_C \notin \Gamma} c'' \ like\ t[C] \right)$$

In other words, given the selected RFD$_c$ $\varphi$, $\gamma''_B$ rewrites $\gamma'_B$ by considering each tuple $t$ in $Q'(r)$ (i.e., $t$ satisfies $\gamma'_B$), and by considering each other attribute $C$ involved in $\varphi$ from which it is possible to define a new condition according to the similarity/distance constraint associated to $C$ and the value $t[C]$. Notice that, no conditions are included for the attributes $C$ involved in the original query (i.e., $\gamma_C \in \Gamma$), since these attributes are already involved in the query relaxation process started from $\gamma_C$. Thus, $\gamma''_B$ represents a disjunction of new conditions (i.e., one for each tuple in $Q'(r)$), each rewriting $\gamma'_B$ with a conjunction of new conditions over alternative attributes included in $\varphi$ but not in $Q$.

*Ranking and filtering RFD$_c$s.* The number of RFD$_c$s holding on a given dataset can be high. For this reason, it is important to have a strategy for identifying the most useful ones holding on a relation $r$. In the proposed methodology, we rank the collection of RFD$_c$s according to the following criteria that maximize the original semantics of the query: (*i*) minimizing the LHS cardinality, (*ii*) maximizing thresholds for the LHS attributes, (*iii*) minimizing the threshold for the RHS attribute [3], and (*iv*) maximizing the number of attributes involved in the original query conditions. Moreover, a filtering strategy imposes the removal of all the RFD$_c$s that do not include at least one attribute involved in the conditions defined over the original query.

## 3 THE REQUIRED TOOL

The REQUIRED tool implements the methodology defined in the previous section, and enables the user to perform a query relaxation process driven by a set of RFD$_c$s holding on the considered dataset. The system architecture and functionalities of the proposed tool are described in the following subsections.

### 3.1 System Architecture

REQUIRED relies on a microservice architecture in which data is exchanged through REST APIs by means of the JSON standard format in order to guarantee high portability, scalability, and maintainability of components. Although in this demo paper we present a WebApp, the Back-end component of REQUIRED can be integrated with a plugin into other systems, such as DBMS clients. Figure 1 shows an overview of the system architecture. As we can see, REQUIRED is separated into two containers: one including the user interfaces (*REQUIRED UI*) and the other containing the services required for executing the system functionalities (*REQUIRED* Back-end). The components of the user interfaces access the REQUIRED services in the back-end through the *API Client* component. The latter interacts with the REQUIRED services in the other container by invoking the *Server* component,

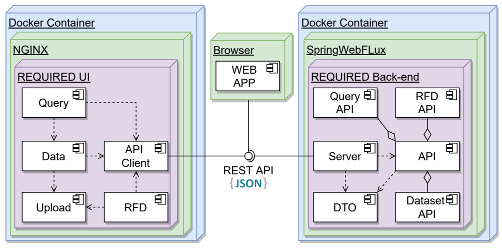


**Figure 1: Overview of the REQUIRED Architecture.**

which is responsible for communicating with the other services in the container and for executing their functionalities.

The user interacts with REQUIRED through a web browser through which s/he can upload a dataset and submit queries. Then, the user can upload the $\text{RFD}_c$s holding on the dataset, so that s/he can select the one best suited for relaxing the query.

*Implementation details.* The tool has been developed as an application divided into multiple Docker containers. The Client components have been designed as a monorepo architecture using Angular and NX frameworks, which permit to define reactive interfaces and facilitate future updating of the functionalities behind REQUIRED. The Server is a reactive RESTFul web service that has been built as a Maven multi-module project using Java and SpringWebFlux. Through Docker, both the SpringWebFLux microservice and the Angular UI application have been containerized in two Docker Images freely accessible on Docker Hub[1].

## 3.2 Functionalities

REQUIRED is able to provide users with several functionalities, requiring a minimum number of configurations for their execution, as shown in Figure 2. After uploading a dataset, REQUIRED automatically identifies its attributes and their types, i.e., numerical or textual, and it generates input fields with which the user can interact (Figure 2a). Starting from this, REQUIRED provides three main functionalities:

- **Executing a query**: We can execute a query by instantiating the attributes recalled in its conditions. REQUIRED automatically composes and enquires the dataset, and shows the answer set within the table at the bottom of the interface. Moreover, the percentage value on the top of the table shows the rate by which occurrences contained in the dataset match the executed query. Figure 2a shows an example of the query performed on the *Songs*[2] dataset. As we can see, we have set the values "*Ed Sheeran*" and "*80*" for the attributes Artistname and Energy, respectively. Consequently, REQUIRED enquires the dataset by executing the following query: "SELECT * FROM Songs WHERE Artistname = 'Ed Sheeran' AND Energy = 80".
- **Extending a query**: After executing a query, we can upload a set of $\text{RFD}_c$s holding on the considered dataset. Then, REQUIRED shows in a table, those automatically ranks and, and shows the $\text{RFD}_c$s in a table, after applying the ranking and filtering strategy (Figure 2b). After that, we can select an $\text{RFD}_c$, and REQUIRED automatically extends the query results according to the methodology described in Section 2.2. Figure 2c shows an example of the extended query after selecting the $\text{RFD}_c$ $\varphi$ : $\mathsf{Energy}_{(\leq 2)}, \mathsf{Artistname}_{(\leq 3)} \rightarrow \mathsf{Danceability}_{(\leq 3)}$, leading to the considerations of songs with similar Energy i.e., in the range $[78, 82]$, and with similar Artistname, i.e., the ones that have a Levenshtein distance $\leq 3$ with respect to 'Ed Sheeran'. As we can see, REQUIRED updates the previous query by extending its conditions and rewrites it in the following query: "SELECT * FROM Songs WHERE Energy BETWEEN 78 AND 82 AND Artistname IN ('Ed Sheeran')". The extended query led to an increase in the matching occurrences, reaching a rate of 4% with respect to the whole dataset.
- **Relaxing a query**: After extending a query, we can relax some constraints according to the approach shown in Section 2.2. In particular, after selecting the "*Relaxed Query*" button, REQUIRED automatically re-defines some constraints of the extended query by considering the $\text{RFD}_c$ selected during the previous step. Figure 2d shows an example of the relaxed query considering the selected $\text{RFD}_c$ $\varphi$, which has led to the definition of the following new relaxed query: "SELECT * FROM Songs WHERE Danceability BETWEEN 69 AND 78". As we can see, the resulting occurrences have significantly increased with respect to those achieved with the original query, reaching a matching rate of 42% with respect to the whole dataset. Notice that, if the query in the previous step already provides more than 10 tuples in the results, REQUIRED triggers a prompt warning to notify the user with a possible over-relaxation of the query's results in the application of this relaxation step.

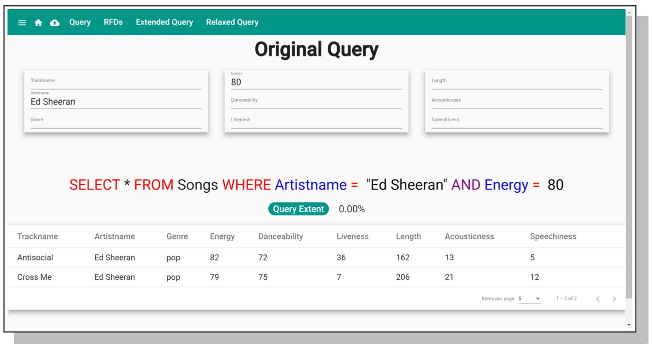
## 4 DEMONSTRATION PLAN

Our demonstration aims to show how REQUIRED can help users in enriching the query answer set by exploiting $\text{RFD}_c$s. In particular, REQUIRED permits to upload .csv datasets and $\text{RFD}_c$s in JSON format resulting from an $\text{RFD}_c$ discovery process. We use 5 datasets and the $\text{RFD}_c$s extracted from them by using the Domino algorithm [3]. In particular, starting from providing a demonstration on the basic functionalities of REQUIRED, such as loading a dataset and executing a query, we demonstrate two scenarios: relaxing queries without and with REQUIRED.

*Scenario 1. Relaxing query conditions without REQUIRED.* Let us suppose that we would like to execute a query in order to collect and analyze several results concerning products, songs, movies, and so forth. Thus, the user can set one or more selection conditions, which should encounter desirable characteristics of collected items. By focusing on queries that obtain no or few results, it would be desirable to enable users to relax query conditions in order to enlarge the output answer set. In this scenario, the user can only consider the interface of REQUIRED shown in Figure 2a to load a dataset and provide a query, then s/he can only try to manually change query conditions in order to increase the output answer set.

*Scenario 2. Relaxing query conditions with REQUIRED.* Starting from the previous scenario, let us suppose that the user can use REQUIRED in order to find good relaxation criteria based on data correlations inferred from data, and/or consider alternative items related to the ones characterized by the relaxed query conditions, but by changing attributes involved in the query. This means that, even though the provided query conditions possibly disappear from the relaxed query, the user is sure that the relaxed result set will contain items related to those under consideration. In this scenario, the user can load a dataset and provide a query, and then upload a set of $\text{RFD}_c$s discovered through the Domino

[1]https://hub.docker.com/u/rfdtoolbox/
[2]Spotify's Top 50 Songs in 2019 available on Kaggle



Original Query

| | | |
|---|---|---|
| Trackname | Energy: 80 | Length |
| Artistname: Ed Sheeran | Danceability | Acousticness |
| Genre | Liveness | Speechiness |

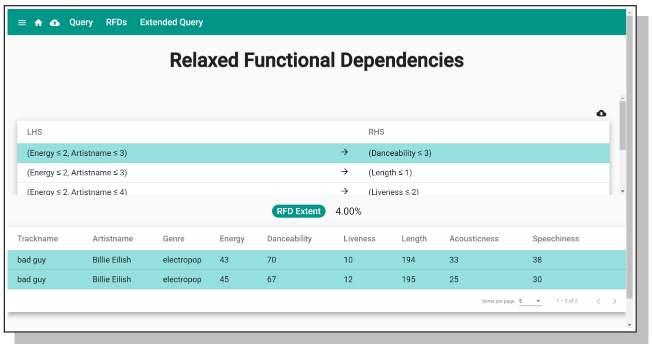
SELECT * FROM Songs WHERE Artistname = "Ed Sheeran" AND Energy = 80

Query Extent 0.00%

| Trackname | Artistname | Genre | Energy | Danceability | Liveness | Length | Acousticness | Speechiness |
|---|---|---|---|---|---|---|---|---|
| Antisocial | Ed Sheeran | pop | 82 | 72 | 36 | 162 | 13 | 5 |
| Cross Me | Ed Sheeran | pop | 79 | 75 | 7 | 206 | 21 | 12 |

Items per page: 5 1 – 2 of 2

(a) Interface for performing a query on a dataset.



Relaxed Functional Dependencies

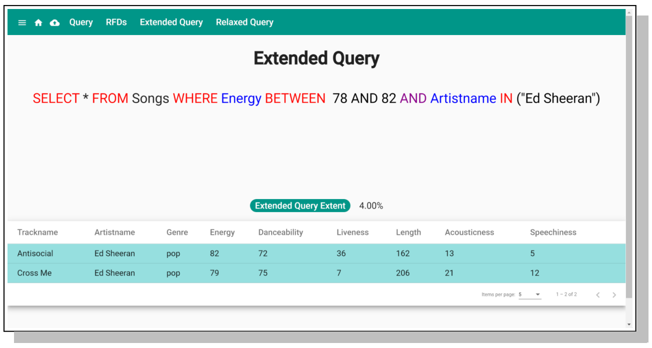
| LHS | | RHS |
|---|---|---|
| (Energy ≤ 2, Artistname ≤ 3) | → | (Danceability ≤ 3) |
| (Energy ≤ 2, Artistname ≤ 3) | → | (Length ≤ 1) |
| (Energy ≤ 2, Artistname ≤ 4) | → | (Liveness ≤ 2) |

RFD Extent 4.00%

| Trackname | Artistname | Genre | Energy | Danceability | Liveness | Length | Acousticness | Speechiness |
|---|---|---|---|---|---|---|---|---|
| bad guy | Billie Eilish | electropop | 43 | 70 | 10 | 194 | 33 | 38 |
| bad guy | Billie Eilish | electropop | 45 | 67 | 12 | 195 | 25 | 30 |

Items per page: 5 1 – 2 of 2

(b) Interface for the selection of an RFD.



Extended Query

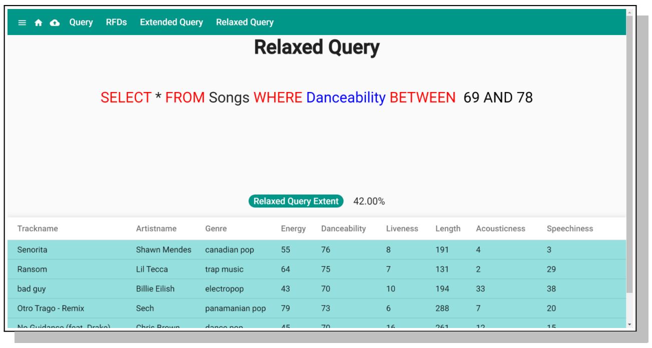
SELECT * FROM Songs WHERE Energy BETWEEN 78 AND 82 AND Artistname IN ("Ed Sheeran")

Extended Query Extent 4.00%

| Trackname | Artistname | Genre | Energy | Danceability | Liveness | Length | Acousticness | Speechiness |
|---|---|---|---|---|---|---|---|---|
| Antisocial | Ed Sheeran | pop | 82 | 72 | 36 | 162 | 13 | 5 |
| Cross Me | Ed Sheeran | pop | 79 | 75 | 7 | 206 | 21 | 12 |

Items per page: 5 1 – 2 of 2

(c) Query extension interface.



Relaxed Query

SELECT * FROM Songs WHERE Danceability BETWEEN 69 AND 78

Relaxed Query Extent 42.00%

| Trackname | Artistname | Genre | Energy | Danceability | Liveness | Length | Acousticness | Speechiness |
|---|---|---|---|---|---|---|---|---|
| Senorita | Shawn Mendes | canadian pop | 55 | 76 | 8 | 191 | 4 | 3 |
| Ransom | Lil Tecca | trap music | 64 | 75 | 7 | 131 | 2 | 29 |
| bad guy | Billie Eilish | electropop | 43 | 70 | 10 | 194 | 33 | 38 |
| Otro Trago - Remix | Sech | panamanian pop | 79 | 73 | 6 | 288 | 7 | 20 |
| No Guidance (feat. Drake) | Chris Brown | dance pop | 45 | 70 | 16 | 261 | 12 | 15 |

(d) Query relaxation interface.

Figure 2: Overview of REQUIRED functionalities.

algorithm [3]. Then, REQUIRED shows the RFD$_c$s involving at least one attribute included in query conditions, which are ranked according to their utility (see Section 2.2). The user can select an RFD$_c$, which characterizes the relaxation criteria that will be applied over the original queries, by analyzing occurrences that satisfy the selected RFD$_c$. Thus, REQUIRED redefines the original query according to the selected RFD$_c$, and shows the relaxed result set. Thus, the user can obtain results related to the ones of the original queries, by also considering correlations over other parameters that have not been considered by the user. This permits to increase the answer set and supports the user in finding novel items hardly reachable from refinement processes over conditions of the original query.

## 5 CONCLUSION

In this demo paper, we presented REQUIRED, a query relaxation tool that leverages Relaxed Functional Dependencies. Through its user interface, REQUIRED offers the possibility to upload a dataset and execute queries. To enlarge the result set, the user can load a set of RFD$_c$s, which are automatically ranked by the tool according to a utility strategy. After selecting an RFD$_c$, REQUIRED can extend the query conditions or even completely rewrite some conditions of the user's query. We showcased a demonstration to illustrate how REQUIRED can represent an effective tool for enriching the query results. In the future, we would like to extend REQUIRED in order to admit the consideration of multiple tables, join queries, and the integrity constraint compliance.

## 6 ACKNOWLEDGMENTS

This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

We thank Maurizio Casciano for his contribution to the development of the first version of the REQUIRED tool.

## REFERENCES

[1] Abdullah Albarrak, Tatiana Noboa, Hina A Khan, Mohamed A Sharaf, Xiaofang Zhou, and Shazia Sadiq. 2014. Orange: Objective-aware range query refinement. In *Proceedings of IEEE 15th International Conference on Mobile Data Management*, Vol. 1. IEEE, 333–336.

[2] Luyi Bai, Xiaofeng Di, and Lin Zhu. 2022. Query relaxation of fuzzy spatiotemporal RDF data. *Applied Intelligence* (2022), 1–19.

[3] Loredana Caruccio, Vincenzo Deufemia, Felix Naumann, and Giuseppe Polese. 2021. Discovering relaxed functional dependencies based on multi-attribute dominance. *IEEE Transactions on Knowledge and Data Engineering* 33, 9 (2021), 3212–3228.

[4] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2016. Relaxed functional dependencies—A survey of approaches. *IEEE Transactions on Knowledge and Data Engineering* 28, 1 (2016), 147–165.

[5] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2017. Learning Effective Query Management Strategies from Big Data. In *Proceedings of 16th IEEE International Conference on Machine Learning and Applications*. IEEE, 643–648.

[6] Sébastien Ferré. 2018. Answers partitioning and lazy joins for efficient query relaxation and application to similarity search. In *European Semantic Web Conference*. Springer, 209–224.

[7] Ykä Huhtala, Juha Karkkainen, Pasi Porkka, and Hannu Toivonen. 1998. Efficient discovery of functional and approximate dependencies using partitions. In *Proceedings 14th International Conference on Data Engineering*. IEEE, 392–401.

[8] Shuxin Li, Gong Cheng, and Chengkai Li. 2020. Relaxing relationship queries on graph data. *Journal of Web Semantics* 61 (2020), 100557.

[9] Davide Mottin, Alice Marascu, Senjuti Basu Roy, Gautam Das, Themis Palpanas, and Yannis Velegrakis. 2014. IQR: an interactive query relaxation system for the empty-answer problem. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of Data*. 1095–1098.

[10] Ion Muslea. 2004. Machine learning for online query relaxation. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 246–255.

[11] Ion Muslea and Thomas J Lee. 2005. Online query relaxation via bayesian causal structures discovery. In *AAAI*. 831–836.

[12] Ullas Nambiar and Subbarao Kambhampati. 2004. Mining approximate functional dependencies and concept similarities to answer imprecise queries. In *Proceedings of the 7th International Workshop on the Web and Databases: Colocated with ACM SIGMOD/PODS 2004*. 73–78.

[13] Viswanath Poosala and Venkatesh Ganti. 1999. Fast approximate query answering using precomputed statistics. In *Proceedings of the 15th International Conference on Data Engineering (Cat. No. 99CB36337)*. IEEE, 252.