ExplIQuE: Interactive Databases Exploration with SQL

Marie Le Guilly, Jean-Marc Petit, Vasile-Marian Scuturici, Ihab F Ilyas

ExplIQuE: Interactive Databases Exploration with SQL

Marie Le Guilly
marie.le-guilly@liris.cnrs.fr
Univ Lyon, INSA Lyon, CNRS, LIRIS UMR 5205
Villeurbanne, France

Jean-Marc Petit
jean-marc.petit@liris.cnrs.fr
Univ Lyon, INSA Lyon, CNRS, LIRIS UMR 5205
Villeurbanne, France

Vasile-Marian Scuturici
marian.scuturici@liris.cnrs.fr
Univ Lyon, INSA Lyon, CNRS, LIRIS UMR 5205
Villeurbanne, France

Ihab F. Ilyas
ilyas@uwarteloo.ca
University of Waterloo
Canada

ABSTRACT

To help databases users who have just started learning SQL or are not familiar with their database, we propose ExplIQuE, an exploration interface with query extensions. Its purpose is to assist users to smoothly dive into data exploration, and to be able to express imprecise questions over their data. Indeed, such situations are more and more current with the increasing desire for users to get value out of their data. In this configuration, in addition to classic SQL querying possibilities, ExplIQuE offers the possibility to extend a given SQL query, by suggesting a set of possible selection predicates to add to the query, that aim at dividing the initial answer set to identify interesting exploration zones. In addition, ExplIQuE proposes some indicators to help the user in choosing its desire extension and in understanding her data, as well as interactive visualizations of the result set, in two dimensions revealed by PCA techniques. In this demonstration, we offer the audience the possibility to try the various functionalities of ExplIQuE by trying to express an imprecise question over a scientific database on bacterial colonies, through an iterative process. A video of the proposed demonstration is available at https://youtu.be/oK8xWGCWj_A.

KEYWORDS

SQL, extensions, imprecise queries

ACM Reference Format:
Marie Le Guilly, Jean-Marc Petit, Vasile-Marian Scuturici, and Ihab F. Ilyas. 2019. ExplIQuE: Interactive Databases Exploration with SQL. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3357384.3357847

1 INTRODUCTION

SQL is a language that appeared in the 1960s, and is still widely used nowadays. Over the years, the volume of data that is stored has increased tremendously, creating new needs for users to be able to store, access, and analyze their data. Moreover, with the recent development and popularity of data analysis techniques, combined with an easier access to data collecting and storing devices, more and more users are trying to make sense and get value out of their data.

As a consequence, the structure of databases tends to change, with bigger schemas, larger tables, and of course, more and more tuples. In addition, the use of databases by users is also evolving, because the data they are looking for is less easy to find, because they are less experienced, and because the question they are trying to answer is not always totally clear: we call such questions *imprecise* queries, that are expressed in natural language but not easily translatable into SQL. This is especially true in exploratory contexts, where it is necessary to deeply understand the data, and to try several different SQL queries, before translating the initial question into SQL and reaching the desired data. Indeed, when a user is confronted to a relational database with a question that is not necessarily clear in the first place, she will go through different phases. First, she can explore and navigate the database to understand its content and structure better. Then, once more familiar with the data structure, she can express a first simple and general query, before refining it over and over, until she answers the initial question at hand. As a result, the desired tuple set is not reached with a single query, but with a sequence of iterative refinement that, one by one, that gets the user closer to what she is looking for, and gradually builds her final answer set.

This query refinement process is not always easy for users, especially when they do not know where to start it from, or when for example their initial query returns an answer set that presents too many tuples. Tools to assist users in such situations are therefore necessary, as the one for interactive query refinement presented in [3].

More specifically, there is a gap to bridge to help SQL users to get confident about their data and queries, and to tackle real life problems. Such users have often just started to master its basic functionalities, and have been trained with well-defined questions on small datasets. But whenever they are confronted to fuzzy questions on new databases, they can get lost. This situation is common, for examples students confronted to their first internship experiences, or for freshly trained employees in companies that are implementing new data analysis processes to make use of the data they collect daily. Helping these new users to smoothly dive into data exploration is therefore an important challenge that would be beneficial in many domains.

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357847

In this paper, we present ExplIQuE, our exploration interface of SQL databases with query extension. The purpose of this interface is to help users in refining an initial SQL query over a database, through an iterative process, by suggesting a set of possible extensions of this query, that consist in possible additional selection conditions for its `where` clause. These extensions give new queries, that can then be explored and extended, until the desired result is reached. Assistance is provided to understand and select the desired extensions, with metrics and visual representations. These extensions have several objectives:

- Help the user understand her database by paying attention to some interesting attributes they might have overlooked, and observing how the query result is divided and distributed, using 2D visualizations.
- Unblock the user when she does not know where to go or where to start, by providing suggestions even to a very general query.
- Provide a semantic help where most SQL editors only provide syntactic help that is not helpful to understand the content of the database itself.

ExplIQuE is an interface aimed to connect to any database, and to provide, in addition to classic querying options in SQL, extensions to queries, with interactive and intuitive visual support. we offer the audience the possibility to try the various functionalities of ExplIQuE by trying to express an imprecise question over a scientific database on bacterial colonies, through an iterative process.

2 SYSTEM OVERVIEW

2.1 Query extensions

The purpose of extensions is to give suggestions to the user, so that she has directions to refine her initial query Q, by adding additional selection predicates in the `Where` clause of the query. More specifically, we propose, given a query Q, to compute a k extensions set, such that the extensions in this set do not overlap in terms of results, but also cover all the initial tuples returned by Q. Our solution, to compute such extensions, presently being patented[1], is based on two machine learning algorithms:

Clustering In order to identify interesting zone of refinement in Q's answer set (denoted by $ans(Q, d)$), we propose to group similar tuples together using a clustering algorithm. $ans(Q, d)$ is thus divided into k clusters, and each cluster becomes a possible refinement zone for the user. The intuition between this process is that users are formulating queries in order to reach a specific set of tuples, answering a question, and that the results to be found are not random, but likely to contain tuples that fit together and have similar characteristics. We therefore chose to base the identification of refinement zone of the k-means algorithm with a Euclidean distance (see [2]).

Decision tree The second part of our extension process consists in finding a query to describe each refinement zone identified by clustering. To do so, an additional column `label` is added to the dataset from $ans(Q, d)$, and filled with the number of the cluster each tuple has been assigned to. A binary decision tree is then built to discriminate between the different clusters. For each leaf of the tree, the decision path from the root of the tree to this leaf is computed, and the conjunction of decision clause gives a new selection predicate. In order to get as many extensions as clusters, we offer two possibilities. We either limit the depth of the tree to only obtain k leaves, and therefore each leaf gives exactly one extension. Otherwise, we do not limit the depth of the tree, but all the leaves corresponding to the same cluster are joined by a disjunction, forming together a unique extension.

These algorithms require some parameters. Predefined ones are proposed to the user, who can then personalize them using the ExplIQuE's interface. For the clustering, several values of k are used (by default for $k = 2$ to $k = 10$), each giving an extension set of different size. These sets are presented to the user, ranked according to well-known clustering score of the silhouette coefficient [5]. Inside a single extension set, extensions are ranked according to the size of their result, by showing the most general extension first (with the most tuples), and the most specific one last.

The algorithm to compute a k extension set given a fixed value of k and a tree with k-leaves is given in algorithm 1. It should be noted that the data is preprocessed and normalized for clustering, and that the projection of the query is removed, to allow other attributes, that the user might have overlooked, to be included in the extensions.

procedure Extension (Q, d, k);
Input : A query Q over R,
d a database over R,
k the number of extensions
Output : S_c a set of k extensions of Q
if $Q = \pi_X(Q')$; `// remove the projection`
then
| Q = Q'
end
wd = $ans(Q, d)$; `// wd: working data`
lwd = $kmeans$(wd, k) ; `// lwd: labelled wd`
tree = $DecisionTree$(lwd, k)
conjunctions = $getRules$(tree)
$S_c = \{\}$
foreach *c in conjunctions* **do**
| $S_c = S_c \cup \sigma_c(Q)$
end
return S_c;

Algorithm 1: Query extension procedure

2.2 Data visualization

To understand the extensions, and assist the user in choosing or refining one, ExplIQuE offers several hints. First, for each extension, two scores are displayed, in addition to the ranking among extension sets with the silhouette coefficient.

- The *narrowing ratio*, which is the percentage of tuples removed from the initial query when adding the extension.
- The result set size of the extension, when added to the initial query.

[1] Patent number FR1757682 from the 14/08/2017 in France

In addition, two visualizations are available to assist the user in choosing an interesting extension. The first is a scatterplot of the results of the query being extended, where the tuples are grouped by extension. The data is projected on two dimensions using principal component analysis (PCA), and each extension is presented using a different color. Moreover, the visualization is interactive, as the user can see the extension corresponding to the datapoints by moving the mouse over the scatterplot. The purpose of this visualization is to show in one glance to the user the size and dispersion of an extension, as well as how separated each extension is with respect to the others. Such a visualization is presented on figure 2.

In the specific case of *image databases*, where tuples are associated with images, another visualization is possible. The images associated to the results of an extension can be displayed as a mosaic in ExplIQuE. This is useful for the user to easily identify the diversity of data that an extension represents: she might see visually that the extension contains homogeneous images, or on the opposite easily identify outliers. In this demonstration, we propose to use such a database to demonstrate this additional functionality.

2.3 Implementation

ExplIQuE's is implemented as a web interface. The backend relies on Flask[2] framework, and is therefore implemented using Python 3. The clustering and decision trees algorithms are the ones from the *scikit-learn library* [4].

Data is stored in a relational database that can be either MySQL or Oracle. Optional external files can be stored outside the database in a dedicated folder. The user has access to a web interface that is implemented using the React[3] javascript library. A first page allows to connect to the desired database, before accessing to the second page that allows to extend queries over it. A snapshot of this page is presented on figure 2, showing the main functionalities. On the left panel, the user can query the database using SQL, and when needed, ask to extend the current query. If necessary, the schema can be displayed using the link on the top left corner. On the right panel, the extensions are displayed. For a given value of k, a global visualization is displayed, and the corresponding extension is highlighted when the mouse is over a cluster, to link the visualization to the SQL extension it represents. For each extension, additional information such as the reduction ratio is given to the user when she clicks on it. Finally, the extension can be added to the current query in just one click, it can be updated in the query writing zone, and the process can be repeated iteratively until convergence.

As this is an online system, the extensions have to be computed in a reasonable amount of time. This can be challenging on large instances: as a result, we use random sampling, that allows to reach a trade-off between the computing time and the quality of the results. The experimentations on this specific problem are out of the scope of this demonstration paper.

2.4 Experimentation results

A preliminary version of ExplIQuE (see [1]) was experimented with a group of 70 computer science students, who had just started their lessons on SQL[4]. They were divided in two groups, one with access to ExplIQuE (group EXT), the other with a similar interface but with only classic SQL querying possibilities (group NoEXT). They were asked to answer a series of ten questions over a database designed for the test:

- The first three questions were precise and easy to translate into SQL. They were used to assess the participant's level in SQL, and to ensure that the performances of both groups were balanced.
- The other were imprecise questions that had been designed to be deliberately *fuzzy* with a more exploratory purpose. As such, they where not easily translatable to SQL, and required some fumbling around and playing with SQL to answer them using only classic SQL tools.

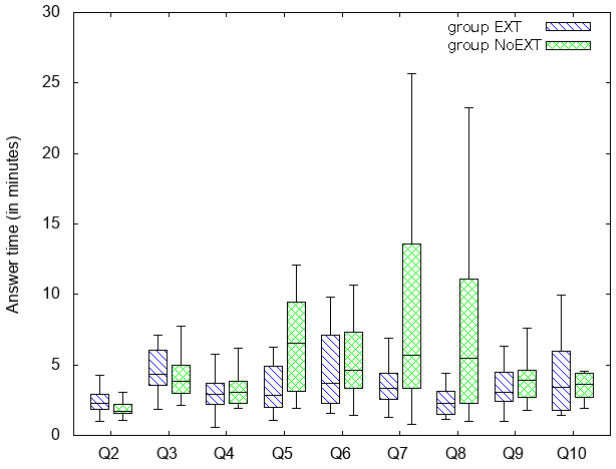

Figure 1: Boxplot of answering time for students with (EXT) and without extensions (NoEXT)

The answering time of each participant to each question was monitored, in order to compare the performances of both groups. The results are presented on figure 1, that shows the boxplot of answering time per question for each group, only when student correctly answered the question. On the first three questions, the results are very similar for both groups, which is what was expected: this questions were easy, and did not require the use of extensions. However, on the other ones, impressive differences can be observed, as students who had access to extensions performed much faster (the difference on question 10 is due to the fact that very few students from group NoEXT had the time to answer it correctly). This experiment therefore showed how extensions can help users in better understanding their data, and to write their SQL queries faster. This is what we want to show in this demonstration.

3 DEMONSTRATION SCENARIO

The audience will have the opportunity to use ExplIQuE's web interface, over a scientific database, in order to answer a list of imprecise questions on the database. The interface is presented on figure 2, showing its main features.

[2] http://flask.pocoo.org/
[3] https://reactjs.org/
[4] more details at https://marielgy.github.io/sql_experimentation/

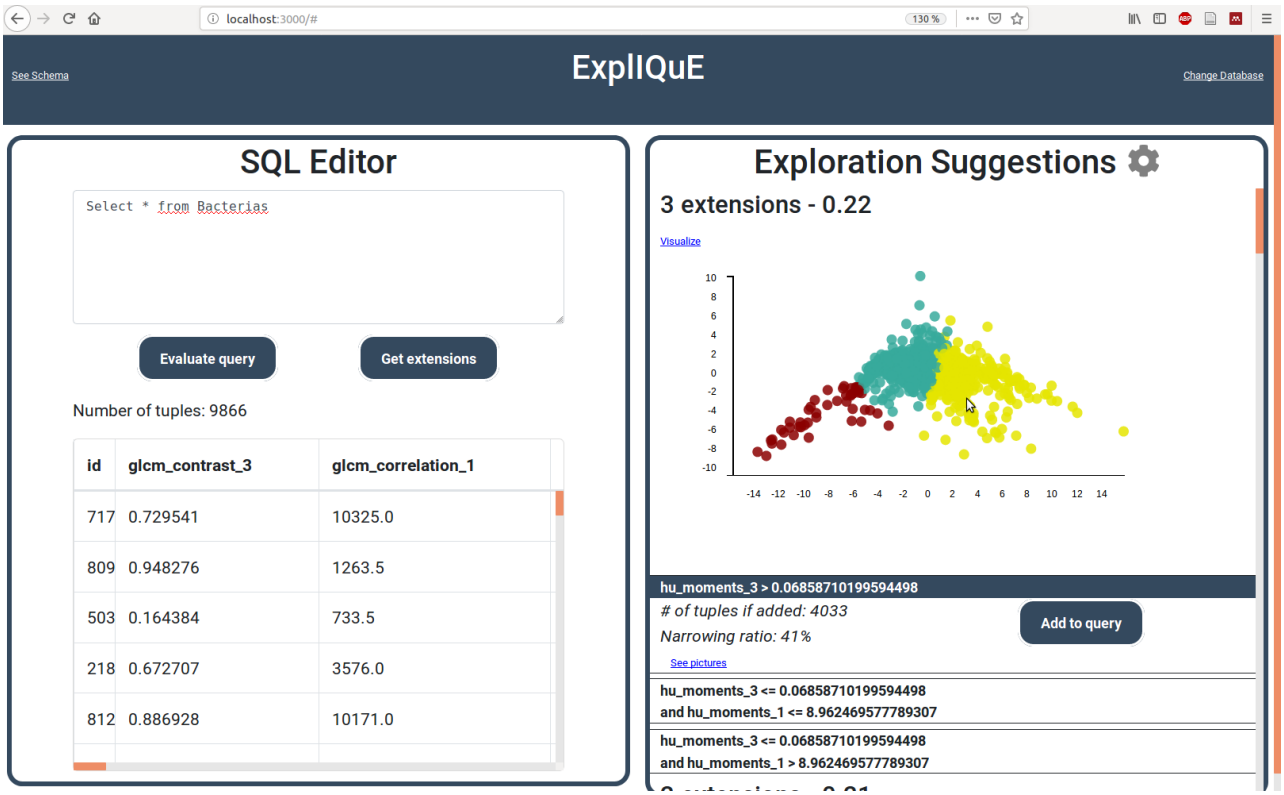

Figure 2: Web interface for ExplIQuE

3.1 Running database

The database used for the demonstration comes from a study where colonies of different bacteria grow on solid plates[5]. It contains 10000 tuples over 29 columns, that correspond to measures on the development of the bacteria. It is both simple (only one table) and difficult, as the content of its columns is not easy to under at first glance, with some confusing column names. It describes the shape, texture, and color of the colonies. Scientists use it to detect colonies with specific characteristics, for example bacteria belonging to the same species. Each tuple is also associated to an image representing the bacteria. We chose this dataset for the audience to be in the conditions where ExplIQuE can be useful, as they are not likely to have prior knowledge on this subject. Moreover, the 29 columns, with names that are not so expressive except for domain experts, will make the extensions useful to understand what they contain and the type of data they represent.

3.2 Audience interaction

The audience will have the opportunity to experience the various features of ExplIQuE. First, as with any DBMS, the audience will be able to browse the database's schema, to understand its structure and content. Moreover, ExplIQuE allows user to evaluate queries like any other DBMS's interface. The audience will then be allowed to play with the dataset, by querying it in a traditional setting. Second, the audience will be asked to answer questions, like the following one: *What are the bacteria that have a very similar texture, and all have a circular shape ?* Using our query extensions, the answer to this question can be found in three iterations, and in less than five minutes. Users will be invited to start with a very general query, and to use the suggestions of the extensions to reach the desired result set. They will also be invited to play with the several visualizations to understand and select the most useful extensions. Finally, we also propose to play with several parameters available to tune the extensions. They will therefore have the possibility to change the number of clusters to produce, and to change how the decision tree is computed.

[5]The authors would like to thank Christopher Pease from Darlington EURL for releasing the dataset

REFERENCES

[1] Marie Le Guilly, Ihab Ilyas, Jean-Marc Petit, and Vasile-Marian Scuturici. 2018. Partitioning queries for data exploration using query extensions. In *BDA 2018 34ème conférence sur la Gestion de Données. Principes, Technologies et Applications.*

[2] S. Lloyd. 2006. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theor.* 28, 2 (Sept. 2006), 129–137. https://doi.org/10.1109/TIT.1982.1056489

[3] Chaitanya Mishra and Nick Koudas. 2009. Interactive query refinement. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology.* ACM, 862–873.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[5] Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.