QOCO: A Query Oriented Data Cleaning System with Oracles

Moria Bergman [1] Tova Milo [1] Slava Novgorodov [1] Wang-Chiew Tan [2]
[1] *Tel-Aviv University* [2] *University of California, Santa Cruz*
[1] {moriaben, milo, slavanov}@post.tau.ac.il [2] tan@cs.ucsc.edu

ABSTRACT

As key decisions are often made based on information contained in a database, it is important for the database to be as complete and correct as possible. For this reason, many data cleaning tools have been developed to automatically resolve inconsistencies in databases. However, data cleaning tools provide only best-effort results and usually cannot eradicate all errors that may exist in a database. Even more importantly, existing data cleaning tools do not typically address the problem of determining what information is missing from a database.

To tackle these problems, we present QOCO, a novel *query oriented* cleaning system that leverages materialized views that are defined by user queries as a trigger for identifying the remaining incorrect/missing information. Given a user query, QOCO interacts with domain experts (which we model as oracle crowds) to identify potentially wrong or missing answers in the result of the user query, as well as determine and correct the wrong data that is the cause for the error(s). We will demonstrate QOCO over a World Cup Games database, and illustrate the interaction between QOCO and the oracles. Our demo audience will play the role of oracles, and we show how QOCO's underlying operations and optimization mechanisms can effectively prune the search space and minimize the number of questions that need to be posed to accelerate the cleaning process.

1. INTRODUCTION

Databases are accessed for the information they contain and key decisions are often made based on the results that are returned. Regardless of the query interface that is provided for accessing information (e.g., free-text search in an internet bookstore or form-based filters for choosing travel destinations in travel agency website), users naturally expect to obtain correct and complete results to a query that is posed against the database. In practice, however, the expectation of always obtaining correct and complete result is difficult to realize since many of the databases are constructed by (semi-)automatically aggregated data from different sources and are likely to contain some inaccuracies and inconsistencies even if individual sources are free of errors.

Proceedings of the VLDB Endowment, Vol. 8, No. 12

Even though data cleaning is a long standing problem that has attracted significant research efforts (e.g., see [5, 6]) for a number of years, the state-of-the-art data cleaning techniques that have been developed cannot usually eradicate all errors in a database. Take as an example, YAGO [9], a database that was built by automatically extracted data from Wikipedia and other sources. Data cleaning techniques have been applied to YAGO and achieved an accuracy of about 95%, namely leaving 5% still erroneous [9]. Even highly curated databases [3] (i.e., databases that were constructed through extensive human effort of verifying and aggregating existing sources) are unlikely to be completely void of errors. At the same time, the sheer volume of such databases also makes it impossible to manually examine each piece of data for its correctness. More importantly, existing data cleaning tools do not usually address the problem of determining what information is missing from a database.

To complement the efforts and overcome the limitations of existing data cleaning techniques, we propose a novel *query oriented* data cleaning approach with oracle crowds. In our framework, materialized views (i.e., query oriented views which are defined through user queries) are used as a trigger for identifying incorrect or missing information. Our premise is that users' queries (and their corresponding materialized views) provide relevant and focussed perspectives of the underlying database and hence, facilitates the discovery of errors. If an error (i.e., a wrong tuple or missing tuple) in the materialized view is detected, our system will interact minimally with a crowd of oracles by asking only pertinent questions. Given a view, we assume the crowd are relevant domain experts (hence, the name "oracle crowds") who are likely to answer questions posed by our system correctly. Answers to questions will help to identify how to clean the underlying database in order to correct the error in the materialized view. More precisely, answers to a certain question will help to identify the next pertinent questions to ask and ultimately, a sequence of edits is derived and applied to the underlying database. These edits will bring the database closer to the state of the ground truth and, at the same time, correct the error in the materialized view. Note that the goal of QOCO is not to clean the entire database, but rather, clean parts of the database, as needed, to rectify the errors in the view. As we will describe, our algorithms effectively prune the search space and minimize the amount of interaction with the crowd while, at the same time, maximize the potential "cleaning benefit" derived from the oracles' answers.

QOCO can be used to complement existing data cleaning techniques. After the data is cleaned with traditional techniques, QOCO can be activated to monitor the views that are served to users/applications. Whenever an error is reported in a view, QOCO can take over to clean the underlying database by interacting with the crowd.

Demonstration. We will demonstrate QOCO with an example database of World Cup Games. The data that we use for the demonstration was extracted from sport websites (e.g. [12]) and is therefore partially incorrect and incomplete due to errors in the automatic website scraping tools. Although some of the errors in the data could be cleaned with automatic techniques (e.g., by comparing to FIFA official data[1]), we made a deliberate choice to use this dataset. Its true facts about World Cup Games are well-known which makes it easier for us to illustrate our key ideas without the need to delve heavily into the semantics of the data. Moreover, it makes it easy for us to find demo attendees to play the role of the experts and/or users (query requesters).

To illustrate, consider the following simple example. We use D to denote the given dirty database and D_G to denote the correct ground truth database. A small sample of the World Cup Games dataset is depicted in Figure 1, which shows portions of two relations: Games lists the World Cup Games and stores the date, playing teams, stage of the tournament and the final score of each game, and Teams records the name and continent of teams that participated in various World Cup Games. In the figure, the dark gray tuples are the wrong tuples in D (i.e., tuples that do not belong to the ground truth database D_G). The light gray tuples are the tuples that are missing from D (i.e., tuples that are in D_G but do not appear in D). All other tuples are correct (i.e., they belong to both D and D_G).

Games

Date	Winner	Runner-up	Stage	Result
13.07.14	GER	ARG	Final	1:0
11.07.10	ESP	NED	Final	1:0
09.07.06	ITA	FRA	Final	5:3
30.06.02	BRA	GER	Final	2:0
12.07.98	ESP	NED	Final	4:2
17.07.94	ESP	NED	Final	3:1
08.07.90	GER	ARG	Final	1:0
11.07.82	ITA	GER	Final	3:1

Teams

Country	Continent
GER	EU
ESP	EU
ARG	SA
BRA	EU
ITA	EU

Figure 1: World Cup Games database schema

Consider a user query Q_1, defined below, which searches for European teams that won the World Cup at least twice.

(x) :- $\text{Games}(d_1, x, y, Final, u_1), \text{Games}(d_2, x, z, Final, u_2),$
$\text{Teams}(x, EU), d_1 \neq d_2.$

When Q_1 is evaluated against the database D, the query result $Q_1(D)$ (i.e., materialized view of Q_1) consists of two tuples $\{(GER),(ESP)\}$. This output contains wrong answers such as Spain as well as missing ones such as Italy.

Assume that a user reports a wrong or missing answer. Note that in the absence of any knowledge about the ground truth D_G, there are multiple ways to update D so that the wrong answers will no longer be part of the result. For example, to remove (ESP) from $Q_1(D)$, one can remove (ESP, EU) from Teams, or two of the three facts in Games that represent a winning game of Spain. To add the missing tuple (ITA) to the result, one can add the tuple (ITA, EU) to Teams together with a tuple representing a winning game of Italy, e.g., the true fact of 1982 final game or any other true or false tuple representing a real/fake win for Italy.

QOCO could ask the crowd whether Spain is in Europe, or if the tuples representing the World Cup finals in 1994, 1998, or 2010 are correct. Similarly, QOCO could ask the crowd whether there are tuples missing from the query result. We will demonstrate how QOCO selects the most effective questions and efficiently cleans the underlying database

[1]http://www.fifa.com/

Related work. As mentioned above, numerous data cleaning techniques have been proposed in the past (e.g., surveys [5, 6]). Recently, [11] introduced the idea of cleaning only a sample of data to obtain unbiased query results with confidence intervals. QOCO is similar in spirit to [11] in that it uses the crowd to correct query results. However, unlike QOCO , [11] does not propagate the updates back to the underlying database. Another critical difference from [11], as well as from prior data cleaning works, is the support of the *truly open world assumption* through a mechanism that identifies query answers missing from the output, and thus discover and inserts true tuples that are missing from the input database. Also, we propose our query-oriented approach as means to focus resources to the most relevant portions of the underlying data.

Crowdsourcing, or human computation, is a model where humans perform small tasks to help solve challenging problems. Incentives can range from small payments to public recognition and the desire to help scientific progress [8]. It is a powerful tool that has been employed for database cleaning tasks such as entity/conflict resolution [10], duplicate detection [2], and schema matching [13]. These complementary techniques can be used for the initial data cleaning and then refined by our approach. There has also been extensive research in the past on ensuring the quality of answers (whether individual answers or aggregated answers) obtained from the crowd and on methods for evaluating crowd workers' quality [7]. These methods are complementary to our work and can be used here as a preliminary step to identify quality answers or select the experts to which we pose questions.

2. TECHNICAL BACKGROUND

We will briefly present our underlying model. Full details can be found in the full version of the paper [1].

Let D be our underlying relational database and Q be a query defined by a union of conjunctive queries. We adopt the *truly open world assumption* where a fact that is in D can also be true or false, in addition to the well-known *open world assumption* that a fact that is not in D can be true or false. In other words, we assume that a given database can contain mistakes, in addition to being incomplete. The truth of a tuple is given by the ground truth database D_G that contains all true tuples and only them. Hence, a database D is *dirty* w.r.t D_G if $D \neq D_G$. The two databases D and D_G together determine the set of missing/wrong answers w.r.t. a given query.

DEFINITION 2.1. *Types of answers:*

- *(True Answer, True Result) A tuple t is a* true answer *to a query Q and database D if $t \in Q(D)$ and $t \in Q(D_G)$. We call $Q(D_G)$ the* true result *of Q.*
- *(Missing Answer) A tuple t is a* missing answer *to a query Q and database D if $t \in (Q(D_G) - Q(D))$.*
- *(Wrong Answer) A tuple t is a* wrong answer *to a query Q and database D if $t \in (Q(D) - Q(D_G))$.*

A cleaned database D' is achieved through a sequence of updates that are generated from answers to questions posed to the oracle crowd (experts). Each such update is called an *edit*. An *insertion edit* $R(\bar{a})^+$ inserts the tuple $\bar{a}$ into relation R in the database, while a *deletion edit* $R(\bar{a})^-$ removes tuple $\bar{a}$ from R. An update to an existing tuple can be modeled by a deletion followed by an insertion. The result of updating D with an edit e is denoted by $D \oplus e$.

PROBLEM 2.2. (EDIT GENERATION PROBLEM) *Given a database instance D, a ground truth database D_G, and a query Q, interact with the crowd minimally to derive a sequence $e_1, \ldots, e_k$ of edits such that $Q(D') = Q(D_G)$, where $D' = D \oplus e_1 \oplus \ldots \oplus e_k$.*

The set of edits to be performed may be identified by asking the crowd of domain experts questions about the correctness of query answers (or some database tuples related to these answers), as well as asking them to provide missing answers to the query (or missing database tuples related to them).

To delete a wrong answer we seek to remove false tuples from database D as to eliminate all the witnesses for this answer. The complementary case of inserting a missing answer is handled by seeking for a witness for this answer, containing only true tuples. A witness for an answer t is a set of database tuples, $w \subseteq D$, s.t. $Q(w) = t$. A missing answer could also have a witness except that it may also contain tuples that are in $D_G \setminus D$.

To illustrate the main principle of our algorithm, let us assume first for simplicity that there is just one wrong answer in the query result, and then that there is just one missing answer. In what follows, we assume there is a single crowd member who is a *perfect oracle*. A perfect oracle always speaks the truth and knows about D_G. We will later explain how the techniques extend to multiple crowd experts who may provide incorrect answers.

Removing one wrong query answer. The considered problem is to identify a set of corrective updates that need to be performed to remove a wrong tuple from the output. It can be shown to be NP-hard by reducing from the Hitting Set Problem, a well known NP-hard problem (proof given in [1]). Hence, to obtain a practical algorithm, we use a greedy heuristic to determine the next question to ask in order to clean the database. This heuristic is repeatedly applied until we can deduce the exact set of false tuples that should be deleted to remove the wrong answer.

We illustrate this idea next with an example. Full details of the algorithm, as well as experimental study that demonstrates its efficiency, can be found in [1].

EXAMPLE 1. *Consider same query Q_1 which finds European teams that won the World Cup at least twice. Assume that the expert examines the query answers, and finds the answer (ESP) to be wrong. Observe that this answer has three witnesses w_1, w_2, w_3 in D (i.e., $Q_1(w_i)$, $1 \leq i \leq 3$, produces the answer (ESP)).*

	Tuples of the witness
w_1	$t_1 = Game(11.07.10, ESP, NED, final, 1{:}0)$ $t_2 = Game(17.07.94, ESP, NED, final, 3{:}1)$ $t_3 = Team(ESP, EU)$
w_2	$t_2 = Game(17.07.94, ESP, NED, final, 3{:}1)$ $t_4 = Game(12.07.98, ESP, NED, final, 4{:}2)$ $t_3 = Team(ESP, EU)$
w_3	$t_4 = Game(12.07.98, ESP, NED, final, 4{:}2)$ $t_1 = Game(11.07.10, ESP, NED, final, 1{:}0)$ $t_3 = Team(ESP, EU)$

Since (ESP) is a wrong answer, at least one tuple in each witness is wrong. To efficiently prune the search space of tuples to remove, we adopt a greedy approach that asks the expert first about the most frequent tuples (i.e., tuples that appear in the most number of witnesses). Intuitively, if the most frequent tuples are indeed incorrect, they will eliminate all witnesses at once, whereas if found to be correct, they will provide a negative indication about the other tuples in the witnesses. Here tuple t_3 occurs most frequently. The expert will be asked TRUE(t_3)? *and since t_3 is correct ($t_3 \in D_G$), the crowd member will answer YES. The remaining candidate tuples for removal in the witnesses are now, respectively,*

$\{t_1, t_2\}, \{t_2, t_4\}, \{t_4, t_1\}$

As all tuples occur equally often in the witnesses, QOCO *will choose randomly between them. Suppose* QOCO *first poses the question* TRUE(t_1)?. *Since t_1 is correct, the crowd member will answer YES. The remaining candidate tuples for removal in the witnesses are now, respectively,*

$\{t_2\}, \{t_2, t_4\}, \{t_4\}$

At this point the remaining deletions are completely determined: it is clear that both t_2 and t_4 must be deleted in order to eliminate the first and third witness. In doing so, the second witness is eliminated as well. Hence, (ESP) will be removed from the query result. (An analogues procedure is followed if some tuple other than t_1 is first selected.)

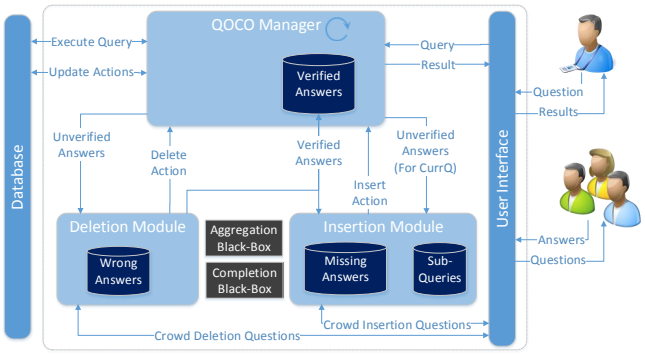

Figure 2: QOCO architecture

Adding one missing answer. Analogously, this considered problem is to identify a set of corrective updates that need to be performed to add a missing answer to the output. Here again, the problem can be shown to be NP-Hard by reducing from the One-3SAT problem (proof given in [1]). We therefore again employ heuristics to effectively identify missing database tuples that, once added, will form together with the already existing tuples a witness to the missing answer. Using provenance information along with ideas from [4] we (recursively) split the query into subqueries whose answers' witnesses can potentially be expanded into a witness to the missing answer. Our heuristic aims to help the crowd members fill in missing data by greedily directing them with facts existing in the underlying database. The details are omitted for space constraints, full details in [1].

3. SYSTEM OVERVIEW

QOCO is implemented with PHP and JavaScript and uses a MySQL database. QOCO's high-level architecture is depicted in Figure 2. The experts, who are answering questions, and the user (requester) who is running the query over the DB, interact with QOCO through the *User Interface*. QOCO has three core modules. *QOCO Manager* is responsible for interacting with the *Database*, and managing the iterations of our iterative algorithm in the general case. It receives the query from the requester and executes it on the *Database*. It performs the needed insert/delete actions identified by the *Deletion module* and *Insertion module* as described in Section 2. All verified correct results are sent to the *Final Results* set, which returns them to the requester. QOCO uses two black-boxes: *Aggregation Black-Box* and *Completion Black-Box*. The *Aggregation Black-Box* aggregates crowd answers to a given question whereas the *Completion Black-Box* determines when the query result is estimated to be complete. The latter notifies QOCO when it estimates that asking the crowd to identify further missing answers is no longer necessary, because the query result is complete with high probability. QOCO exposes an API so that other systems can keep their own UI and invoke QOCO as a service. Any application with an underlying database and queries defined on top of the database can make use of QOCO's services for cleaning data.

Figure 3: QOCO UI: mark incorrect/add missing answers

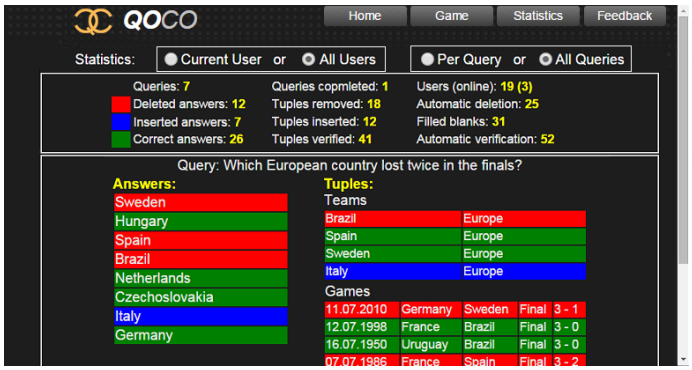

Figure 4: QOCO UI: administrator view

4. DEMONSTRATION SCENARIO

We will demonstrate the functionalities of QOCO on the World Cup Games database. VLDB'15 attendees will be invited to participate in the demonstration by posing queries to the database, checking the answers of queries posed by (other) attendees, identifying (in)correct/missing answers, and assisting in identifying the (in)correct/missing data that is the source for the error.

We use a World Cup Games dataset which we extracted from soccer websites (e.g. [12]). This dataset contains information about World Cup Games since 1930, including dates and scores, names of playing teams, names of players, etc. As mentioned in the Introduction, although some of the errors in the data could be cleaned with automatic techniques, we made a deliberate choice not to fully clean our extracted World Cup data. As previously mentioned, we assume the *truly open world* assumption where errors in data can be either missing true tuples or existing false ones. For example, in the portions of data that we presented in Figure 1, (BRA, EU) is a false tuple while (ITA, EU) is a missing tuple.

To encourage the audience for our demo, we have designed an engaging interactive multiplayer game where players are awarded points for the following: (1) posing queries whose answers highlight incorrect data, (2) answering correctness questions about query answers or database tuples, (3) completing missing information. Players who earn significant points can further explore the World Cup Games data and view interesting World Cup statistics. Since not all VLDB attendees are soccer fans, we have (1) pre-registered with QOCO a set of queries whose answers we know to be incorrect/partial, and (2) prepared a fact-sheet about the World Cup, which we will distribute when needed, to help non-experts successfully play the role of World Cup experts. Those pre-registered queries are trivia-like queries such as "Which European teams lost the World Cup finals at least twice?" or "Which players scored in the 2002 World Cup final?", and they are taken from various soccer trivia websites. The participants will be invited to review the results of these trivia queries or issue new ones.

We begin our demonstration by presenting the game, its goal and rules. We will then allow the audience to play, while we explain the underlying algorithms in QOCO. A player starts the game with answering questions posted by QOCO. The first screen presented to the first player contains a list of answers to a given query executed over our underlying database. The player can mark incorrect answers and/or add new ones (Figure 3). Based on input from players, QOCO decides whether to ask more questions, and which questions to ask. For example, QOCO can decide to verify tuples if a wrong query answer was identified, or add missing data if a missing answer was identified. Players can also pose their own queries using a "Query Builder Tool" page. Once players are done, we offer them to view the Administrator web page (Figure 4), that presents aggregated statistics for all players and all queries together, such as total number of queries processed, total number of edits on the database, the number of missing answers discovered and of wrong answers identified, etc. This page also present interesting information on each query. For example, the missing answers discovered (blue), wrong answers that were deleted (red), and the different database edits determined due to those identified errors.

During the demonstration we will explain the details and nuances of our algorithms, discuss the presented statistics and the decisions taken by the algorithm that yielded these outcomes. We will focus on which questions QOCO posed to the experts (also presented in the Administrator page) and interpret the internal decisions made by QOCO for our audience. In particular, we discuss why certain questions were chosen by the system, explain the effect that experts' answers have on the system's state, and show how the system infers when sufficient information has been collected to determine all the required corrective updates.

Acknowledgements This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071. Tan is partially supported by NSF grant IIS-1450560.

5. REFERENCES

[1] M. Bergman, T. Milo, S. Novgorodov, and W. Tan. Query-oriented data cleaning with oracles. In *ACM SIGMOD*, 2015.

[2] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. *KDD*, pages 39–48, 2003.

[3] P. Buneman, J. Cheney, W. C. Tan, and S. Vansummeren. Curated databases. In *ACM PODS*, pages 1–12, 2008.

[4] A. Chapman and H. V. Jagadish. Why not? In *SIGMOD*, pages 523–534, 2009.

[5] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. 2012.

[6] V. Ganti and A. D. Sarma. *Data Cleaning: A Practical Perspective*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2013.

[7] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. *PVLDB*, 6(2):109–120, 2012.

[8] M. J. Raddick, G. Bracey, P. L. Gay, C. J. Lintott, P. Murray, K. Schawinski, A. S. Szalay, and J. Vandenberg. Galaxy zoo: exploring the motivations of citizen science volunteers. *Astronomy Education Review*, 9(1), 2010.

[9] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge unifying wordnet and wikipedia. In *WWW*, pages 697–706, 2007.

[10] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(10):1483–1494, 2012.

[11] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, pages 469–480, 2014.

[12] World cup history. http://www.worldcup-history.com/.

[13] C. J. Zhang, Z. Zhao, L. Chen, H. V. Jagadish, and C. C. Cao. Crowdmatcher: crowd-assisted schema matching. In *SIGMOD*, pages 721–724, 2014.