Consistent Query Answering: Opportunities and Limitations *

Jan Chomicki
Dept. Computer Science and Engineering
University at Buffalo, SUNY
Buffalo, NY 14260-2000, USA
chomicki@buffalo.edu

Abstract

This paper briefly reviews the recent literature on consistent query answering, an approach to handle database inconsistency in a systematic and logical manner based on the notion of repair. It discusses some computational and semantic limitations of consistent query answering, and summarizes selected research directions in this area.

1. Introduction

Nowadays more and more database applications have to rely on multiple, often autonomous sources of data. While the sources may be separately consistent, inconsistency may arise when they are integrated together. For example, different data sources may record different salaries or addresses of the same employee. At the same time, the application may require that the integrated, global database contain a single, correct salary or address.

In consistent query answering, inconsistency is viewed as a logical phenomenon. A database is inconsistent if it violates integrity constraints. Since it is assumed that the real world is consistent, an inconsistent database does not correspond to any state of the real world, and thus is not a source of reliable information. It needs to be repaired before it can be queried. However, there may be many different ways of repairing a database, even if we limit ourselves to the minimal ones. So it is natural to consider the information present in every repaired database. This leads to the notion of consistent query answer (CQA): an element of query result in every repaired database. Consistent query answers provide a conservative “lower bound” on the information contained in the database.

Example 1.1 Consider the following relation Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Brown</td>
<td>Amherst</td>
<td>100K</td>
</tr>
<tr>
<td>John Brown</td>
<td>Amherst</td>
<td>80K</td>
</tr>
<tr>
<td>Bob Green</td>
<td>Clarence</td>
<td>80K</td>
</tr>
</tbody>
</table>

and the functional dependency Name → Address Salary. Note that for both employees the database contains a single, correct address, while two different salaries are recorded for John Brown, violating the functional dependency.

There are two (minimal) repairs: one is obtained by removing the first tuple, the other by removing the second tuple. (Removing more tuples violates repair minimality.) The query $Q_1$

\[
\text{SELECT} * \text{ FROM Employee}
\]

has one consistent answer, (Bob Green,Clarence,80K), because neither of the first two tuples appears in the result of the query in both repairs. On the other hand, the query $Q_2$

\[
\text{SELECT Name, Address FROM Employee WHERE Salary > 70K}
\]

has two consistent answers, (John Brown,Amherst) and (Bob Green,Clarence), because $Q_2$ returns those two tuples in both repairs. Using $Q_2$ the user extracts correct address information for John Brown, despite the fact that the information about Brown’s salary is inconsistent.

The approach outlined and illustrated above was first proposed in [1]. That paper was followed by numerous further papers that explored several different dimensions of consistent query answering:

- different notions of repair minimality;
- different classes of queries and integrity constraints;
- different methods of computing consistent query answers.

In this paper, we first define the basic notions and summarize the main approaches to consistent query answering.
We show when it is practical to compute CQAs and when this task runs into inherent computational obstacles. Subsequently, we examine the assumptions on which the CQA framework is based and its semantical adequacy. We determine in what circumstances consistent query answering is applicable and where it needs to be supplemented by other techniques. We conclude by outlining selected current research directions in the area.

2. Basic notions

We are working in the context of the relational model of data, assuming the standard notions of relation, tuple, attribute, key, foreign key, functional dependency (FD), and inclusion dependency (IND). In addition, we also consider universal integrity constraints of the form $\forall \bar{x}. \varphi(\bar{x})$, where $\varphi$ is quantifier-free, and more restricted denial constraints, in which $\varphi$ is a disjunction of negative literals. We assume that we are dealing with satisfiable sets of constraints. We do not consider nulls.

A database is consistent if it satisfies the given integrity constraints; inconsistent, otherwise. We consider the common query languages for the relational model: relational algebra, relational calculus, and SQL. Each of them has a well-defined notion of the set of query answers $qa_Q(r)$ where $Q$ is the query and $r$ is a database.

A repair $r'$ of a database $r$ is a database over the same schema, which is consistent and minimally different from $r$. We denote the set of repairs of $r$ by $Rep(r)$. Several notions of repair minimality have been proposed in the literature:

- set minimality of the symmetric difference $\Delta(r, r')$ [1] (the most commonly used);
- set minimality of the asymmetric difference $r - r'$, with the assumption that $r' \subseteq r$ [18] or without it [15];
- several different notions of minimality defined attribute-wise [7, 11, 35].

Each of them may lead to a different set of repairs $Rep(r)$. However, the set of consistent query answers $cqa_Q^c(r)$ is always defined as the intersection of the query answers in individual repairs:

$$cqa_Q^c(r) = \bigcap_{r' \in Rep(r)} qa_Q(r').$$

3. Computing CQAs

Retrieving CQAs via the computation of all repairs is not feasible. Even for FDs, the number of repairs may be too large.
Logic programs. Repairs can be specified using logic programs with disjunction and classical negation [2, 4, 26] and correspond to answer sets [25] of such programs. Then CQAs are obtained by skeptical reasoning (computing facts true in every answer set) which is usually available as a primitive in contemporary logic programming systems like dlvl [29]. This is a very general approach that can handle arbitrary first-order queries and universal integrity constraints. The price to be paid is the computational complexity, as skeptical reasoning is \( \Pi^p_2 \)-data-complete. Thus only very small databases can be directly handled by this approach. However, various optimization techniques have been developed in [20], which have the potential to make this approach practical.

Example 3.3 For the Example 1.1, one of the rules obtained using the approach of [2] would be of the form

\[
\neg \text{Emp}'(n, a, s) \lor \neg \text{Emp}'(n, a', s') \\
\leftarrow \text{Emp}(n, a, s), \text{Emp}(n, a', s'), s \neq s'.
\]

Its reading is as follows: If the functional dependency \( \text{Name} \rightarrow \text{Salary} \) is violated (right-hand side), then one of the violating tuples has to be dropped (left-hand side).

4. Computational complexity

We assume here the notion of data complexity [34], i.e., the complexity defined in terms of the number of tuples in the database. [3, 18] show that computing CQAs for conjunctive queries in the presence of FDs is \( \Pi^p_2 \)-complete. [18] shows that adding inclusion dependencies (under repair minimality defined in terms of asymmetric difference) makes the problem \( \Pi^p_2 \)-complete. [15] show that under different repair minimality notions the complexity of this problem can range from \( \Pi_2 \)-complete to undecidable. [35] demonstrates that the complexity of CQA under a notion of attribute-wise repair minimality closely tracks the complexity of the same problem under set-based repair minimality.

5. Semantical issues

We explore here the etiology of inconsistency. There may be many reasons for integrity violations to occur. We examine such reasons in order to delineate the scope of applicability of the CQA framework. We claim that applying it to raw data leads often to undesirable results and loss of information. We conclude that for CQA to return meaningful and reliable information, the data needs to be appropriately prepared by making sure the schema is semantically adequate, detecting and eliminating duplicates, and applying some data cleaning techniques.

We use Example 1.1 and its extensions to illustrate the points made. We will primarily consider violations of the key dependency \( \text{Name} \rightarrow \text{Address Salary} \). Our starting assumption is that the data from multiple sources have been integrated into a single database, over which the integrity constraints are defined.

Semantic inadequacy. The most basic form of inconsistency at this level is due to a semantic inadequacy of the schema. The integrity constraints may fail to be satisfied in the real world. For example, an employee may have more than one address or salary. The proper response in such a case is to modify the schema either by relaxing the violated constraints or by horizontally decomposing the relation into separate parts that satisfy different integrity constraints. For example, the \( \text{Employee} \) relation could be decomposed into \( \text{Employee}_1 \), in which \( \text{Name} \) is still a key, and \( \text{Employee}_2 \), in which this is no longer the case [30]. Or, the functional dependency could be replaced by a weaker form that accommodates exceptions [12]. A fundamental assumption underlying CQA is that the integrity constraints are correct, while the data may be incorrect. Thus, CQA is overly cautious in the case of semantic inadequacy and tries to repair possibly correct information present in the database, which leads to information loss.

Another instance of the semantic inadequacy is when the specified key is not sufficient for distinguishing objects in the real world. For instance, if the first two tuples in the \( \text{Employee} \) relation correspond to different employees named “John Brown,” it is not surprising that the non-key attributes in those tuples conflict! The solution is to come up with a right key. Trying to repair the relation loses information.

Schema misalignment. A more subtle schema-level problem occurs if the database combines data whose semantics is not fully aligned. For example, one source may store the employee’s work address, while another, her home address. So in the integrated database the employee would appear as having two different addresses and violating the functional dependency. The proper response here is to revise the integrated schema so it contains two different address attributes. Again, the CQA approach tries to repair correct information, leading not only to information loss but also to the confusion between semantically different data items.

Object misclassification. The information about an object may be inserted into a wrong relation. For example, in the case of the relations \( \text{Employee}_1 \) and \( \text{Employee}_2 \) discussed above, suppose that the information about an employee with more than one address is wrongly inserted into \( \text{Employee}_1 \), raising an integrity violation. The appropriate response is to transfer this information to \( \text{Employee}_2 \), not to try to repair \( \text{Employee}_1 \). Again, CQA leads to a loss of information.

Data value obsolescence. If data values corresponding to different time instants are simultaneously present in the database, they may conflict. For example, having both old
and new values of the salary of an employee may result in a violation of the functional dependency. The proper response is to clean the data using meta-data, for example in the form of timestamps. However, if such meta-data is not available and there is no way to tell which data is old and which is new, the cautious approach of CQA is suitable. Another approach is to incorporate priority information into CQA [32].

Data value imprecision. There may be multiple readings of a sensor that need to be reconciled to produce a single value, e.g., for the temperature in a room. Again, data cleaning is in order here because one may need to remove outliers, account for different reading granularities etc. However, in some cases it is not possible or desirable to completely clean the data online (imagine several observers gathering information about the size of a crowd or several witnesses reporting on an accident), and in those cases CQA can provide a conservative lower bound on the information in the database.

Hidden duplicates. The same value can often be represented in multiple ways, for example “East Amherst” and “E. Amherst.” So if there are multiple tuples that only differ in the representation of some attribute values, they are likely to be duplicates. The normalization step in data cleaning recognizes such duplicates. CQA treats hidden duplicates as different values which lead to an integrity violation. This prevents the retrieval of the affected attribute values.

Data errors. Erroneous data can often be caught using CHECK constraints, e.g., Salary > 20K. But there are also more subtle errors that creep into data, for example through misspellings, omissions, or transpositions. Detecting such problems is difficult in general. CQA treats correct and (undetected) erroneous values in the same fashion, thus some repairs may contain errors. However, if an erroneous value conflicts with another value, the error will not be propagated to the CQAs.

Update anomalies. Relations are often denormalized for efficiency purposes, which may lead to the violations of non-key functional dependencies which are not maintained for efficiency reasons. For example, consider the Employee relation with two extra attributes Dept and Location, together with a functional dependency Dept → Location. It may happen that different Employee tuples contain different values for the location of a department. However, as long as a query asks only about the department names and not about their locations, all the names are returned as CQAs (provided the tuples containing them are not involved in other conflicts).

Example 5.1 Suppose the Employee relation has the following attributes: Name, Address, Salary, Dept, and Location. It contains two tuples $t_1$ and $t_2$ such that $t_1[Name] \neq t_2[Name]$, $t_1[Dept] = t_2[Dept] = Sales$, $t_1[Location] = New York$, and $t_2[Location] = Chicago$. Then the query

$$\text{SELECT Dept FROM Employee}$$

returns Sales as a consistent answer. On the other hand, the query

$$\text{SELECT Dept, Location FROM Employee}$$

has no consistent answer.

One can define disjunctive CQAs in terms of OR-objects [27]. In the above example, the query

$$\text{SELECT Dept, Location FROM Employee}$$

could, instead of returning no CQAs, return the tuple (Sales, OR(New York, Chicago)) as a disjunctive CQA. It would be natural for disjunctive LP systems like dlv to support the computation of disjunctive query answers.

6. Selected current research

Data integration. In this paper we have assumed that the data in the database has already been integrated at the instance level. Recent research in data integration studies different kinds of mappings between local sources and the global database [28], and investigates how their semantics interacts with that of repairs [6, 13, 16].

Aggregate constraints. [22] studies CQA for integrity constraints that may contain linear arithmetic expressions involving aggregate functions.

Null values. SQL nulls lack formal semantics, while adequate formal approaches to incomplete information lead to intractability [33]. Nulls are useful in repairs under inclusion dependencies, where a repair with nulls can stand for infinitely many repairs without nulls. [14] contains a proposal how to extend the CQA framework to handle nulls.

XML. For the CQA framework to be applicable to XML databases, the basic notions of repair and consistent query answer need to be redefined. This is done for DTDs only in [31] and DTDs with functional dependencies in [21]. [31] proposes to base repair minimality on tree edit distance [10], while [21] uses an approach more akin to that of [1].

References

References


