Consistent Query Answering: Five Easy Pieces[*]

Jan Chomicki

Dept. of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260-2000
chomicki@cse.buffalo.edu

Abstract. Consistent query answering (CQA) is an approach to querying inconsistent databases without repairing them first. This invited talk introduces the basics of CQA, and discusses selected issues in this area. The talk concludes with a summary of other relevant work and an outline of potential future research topics.

1 Introduction

The notion of *inconsistency* has been extensively studied in many contexts. In classical logic, an inconsistent set of formulas implies every formula (*triviality*). In databases, a database instance is inconsistent if it does not satisfy integrity constraints (*constraint violation*). Those two kinds of inconsistency are closely related: an inconsistent database instance, together with the integrity constraints, may be represented as an inconsistent set of formulas. However, triviality is not a problem in the database context because the semantics of query answers does not take integrity constraints into account.

Nowadays more and more database applications have to rely on multiple, often autonomous sources of data. Even if the sources are 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. Similarly, different sensors may register inconsistent readings of the same quantity that need to be resolved.

In order to deal with inconsistency in a flexible manner, database research and practice have developed different approaches that we will illustrate using a simple example.

Example 1. Consider a database schema consisting of two unary relations P_1 and P_2, and the denial integrity constraint $\forall x.\ (\neg P_1(x) \vee \neg P_2(x))$. Assume a database instance consists of the following facts: $\{P_1(a), P_1(b), P_2(a)\}$. Under *prevention* (usual constraint enforcement), such instance could not arise: only one of $P_1(a)$ and $P_2(a)$ could be inserted into the database. Under *ignorance* (constraint non-enforcement), no distinction is made between $P_1(a)$ and $P_1(b)$,

[*] Research supported by NSF grant IIS-0119186.

T. Schwentick and D. Suciu (Eds.): ICDT 2007, LNCS 4353, pp. 1–17, 2007.

despite that the latter, not being involved in a constraint violation, appears to represent more reliable information. Under *isolation* [18], both $P_1(a)$ and $P_2(a)$ are dropped (or ignored in query answering). Under *weakening* [6,49], $P_1(a)$ and $P_2(a)$ are replaced by $P_1(a) \vee P_2(a)$. Allowing *exceptions* [15] means that the constraint is weakened to $\forall x.\ (\neg P_1(x) \vee \neg P_2(x) \vee x = a)$, but query answering is not affected.

A seperate class of responses to inconsistency is based on the notion of *repair*: a consistent instance minimally different from the original one, in this case $\{P_1(a), P_1(b)\}$ or $\{P_2(a), P_1(b)\}$. Such repairs can be *materialized* [30] or *virtual. Virtual repairing*, which is usually called *consistent query answering* [3], does not change the database but rather returns query answers true in all repairs (*consistent* query answers). So the query asking for all such consistent x that $P_1(x)$ is true, returns only $x = b$.

Example 2. Consider the following instance of a relation *Location*, which records the current location of employees. Such a relation may be built from active-badge sensor readings, employee calendars, and other data sources.

Name	*Campus*	*Building*
Mary	North	Bell
Mary	North	Knox
John	South	Hayes

and the functional dependency *Name* → *Campus Building*. Note that for both employees the database contains a single campus, while two different buildings are recorded for Mary, 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 consistent answer to the query Q_1:

```
SELECT * FROM Location
```

is just the tuple (John,South,Hayes), 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:

```
SELECT Name, Campus FROM Location
```

has two consistent answers: (Mary,North) and (John,South), because Q_2 returns those two tuples in both repairs. Using Q_2 the user can extract reliable information about the *campus* location of both employees, despite the fact that the information about the *building* where Mary is located is inconsistent.

Consistent query answering was first proposed by Arenas et al. [3]. That paper was followed by numerous further papers that explored several different dimensions of consistent query answering:

- different notions of repair minimality (leading to different semantics for consistent query answers);
- different classes of queries and integrity constraints;
- different methods of computing consistent query answers.

What follows is a collection of short essays about consistent query answering (CQA). They address the following topics: the basic concepts and results of Arenas et al. [3] (Section 2), computational complexity of CQA (Section 3), referential integrity (Section 4), and improving the informativeness of CQA in the presence of aggregation or probabilistic information (Section 5). The last essay (Section 6) briefly summarizes other research on CQA and outlines some directions for future work. The essay topics were chosen to introduce the central concepts of CQA and illustrate the breadth of the area. By no means is this article a comprehensive survey of CQA research; such surveys are already available elsewhere [8,11,25].

2 The Basics

We use the standard setting of relational databases. We assume the existence of a database schema R, which is a finite set of relation names and the associated arities. Database instances are mappings from the database schema to finite sets of tuples of the appropriate arity. We also assume that relation columns can be labelled by attributes and typed by associating with them one of the two domains: uninterpreted constants or rational numbers.

Given a database schema R and the built-in predicates over the numeric domain $(=, \neq, <, >, \leq, \geq)$, we define the *first-order language* $\mathcal{L}_R$ in the standard way. Database instances can be viewed as first-order structures over $\mathcal{L}_R$. The built-in predicates have infinite, fixed extensions.

2.1 Integrity Constraints

Integrity constraints are closed first-order $\mathcal{L}_R$-formulas. In the sequel we will denote: relation symbols by $P, P_1, \ldots, P_m$, atomic formulas by $A_1, \ldots, A_n$, tuples of variables and constants by $\bar{t}, \bar{x}, \ldots$, and quantifier-free formulas referring to built-in predicates by φ.

In this paper we consider the following basic classes of integrity constraints:

1. *Universal integrity constraints*: $\forall^*.\ A_1 \vee \cdots \vee A_n \vee \neg A_{n+1} \vee \cdots \vee \neg A_m \vee \varphi$.
2. *Denial constraints*: $\forall^*.\ \neg A_1 \vee \cdots \vee \neg A_m \vee \varphi$.
3. *Binary constraints*: universal constraints with at most two occurrences of database literals.
4. *Functional dependencies (FDs)*: $\forall \bar{x}, \bar{y}, \bar{z}, \bar{y}', \bar{z}'.\ (\neg P(\bar{x}, \bar{y}, \bar{z}) \vee \neg P(\bar{x}, \bar{y}', \bar{z}') \vee \bar{y} = \bar{y}')$. A more familiar formulation of the above FD is $X \rightarrow Y$ where X is

the set of attributes of P corresponding to $\bar{x}$ and Y the set of attributes of P corresponding to $\bar{y}$ (and $\bar{y}'$).

5. *Referential integrity constraints*, known as *inclusion dependencies (INDs)*: $\forall\bar{x},\bar{y}.\ \exists\bar{z}.\ (\neg P_2(\bar{x},\bar{y}) \vee P_1(\bar{y},\bar{z}))$. Again, this is often written as $P_2[Y_2] \subseteq P_1[Y_1]$ where Y_1 (resp. Y_2) is the set of attributes of P_1 (resp. P_2) corresponding to $\bar{y}$.

Given a set of FDs and INDs IC and a relation P_1 with attributes U, a *key* of P_1 is a minimal set of attributes X of P_1 such that IC entails the FD $X \rightarrow U$. In that case, we say that each FD $X \rightarrow Y \in IC$ is a *key* dependency and each IND $P_2[Y] \subseteq P_1[X] \in IC$ is a *foreign key constraint*. If, additionally, X is the only key of P_1, then both kinds of dependencies are termed *primary*.

Definition 1. *Given an instance I of a database schema R and a set of integrity constraints IC, we say that I is* consistent *if $I \vDash IC$ in the standard model-theoretic sense;* inconsistent *otherwise.*

Queries are formulas over the same language $\mathcal{L}_R$ as the integrity constraints. *Conjunctive queries* [23,1] are queries of the form $\exists^*.\ A_1 \wedge \cdots \wedge A_n \wedge \varphi$ where φ is a conjunction of built-in atomic formulas.

Definition 2. *A tuple $\bar{t}$ is an* answer *to a query $Q(\bar{x})$ in an instance I iff $I \models Q(\bar{t})$.*

2.2 Repairs and Consistent Query Answers

We introduce now the framework of Arenas et al. [3]. It is based on two fundamental notions: *repair* and *consistent query answer*. The symmetric difference Δ is used to capture the distance between two instances I and I': $\Delta(I,I') = (I - I') \cup (I' - I)$[1].

We assume that we are dealing with *satisfiable* sets of integrity constraints.

Definition 3. *Given a set of integrity constraints IC and database instances I and I', we say that I' is a* repair *of I w.r.t. IC if $I' \vDash IC$ and there is no instance I'' such that $I'' \vDash IC$ and $\Delta(I,I'') \subset \Delta(I,I')$.*

We denote by $Repairs_{IC}(I)$ the set of repairs of I w.r.t. IC. This set is nonempty for satisfiable sets of constraints.

Given a query $Q(\bar{x})$ to a database instance I, we want as *consistent* answers those result tuples that are unaffected by the violations of IC, even when I violates IC.

Definition 4. *A tuple $\bar{t}$ is a* consistent answer *to a query $Q(\bar{x})$ in a database instance I w.r.t. a set of integrity constraints IC iff $\bar{t}$ is an answer to the query $Q(\bar{x})$ in every repair I' of I w.r.t. IC. We can define* true *being a consistent answer to a Boolean query in a similar way.*

[1] The difference is defined component-wise for every relation symbol in the schema.

Note: If the set of integrity constraints IC is clear from the context, we omit it for simplicity.

The notion of consistent query answer corresponds to the notion of *certain* answer, developed in the context of incomplete databases by Lipski and Imieliński [45,42], because repairs can be viewed as *possible worlds.* In some cases, as in Example 1, one can represent the set of repairs w.r.t FDs as a disjunctive database or a table with OR-objects [43]: disjunctive information is used to model resolved conflicts. The correspondence in the other direction breaks down, however, already in very simple cases [5].

2.3 Query Rewriting

Arenas et al. [3] propose a method to compute consistent query answers based on *query rewriting.* Given a set of integrity constraints IC, a query Q is rewritten into a query Q^{IC} such that for every instance I the set of answers to Q^{IC} in I is equal to the set of consistent answers to Q in I. Typically, we expect Q and Q^{IC} to belong to the same class of queries, for example first-order queries or SQL. In such a case, the computation of consistent query answers can be done using the same query engine.

The method proposed in [3] is relatively simple and draws on earlier work in semantic query optimization [22]. The rewriting applies to and produces first-order queries. When a literal in the query can be resolved with an integrity constraint the resolvent forms a *residue.* All such residues are then conjoined with the literal to form its expanded version. If a literal that has been expanded appears in a residue, the residue has to be further expanded until no more changes occur. For denial constraints, however, only a single expansion is necessary.

Example 3. Consider Example 2, the FD $f_1 : \mathit{Name} \rightarrow \mathit{Building}$ and the query $\mathit{Location}(x, y, z)$. The rewritten query Q^{f_1}:

$$Q^{f_1} \equiv \mathit{Location}(x, y, z) \wedge \forall y', z'. \ \big(\mathit{Location}(x, y', z') \Rightarrow z = z'\big).$$

Clearly, the rewritten query can be formulated in SQL.

2.4 Limitations

The notion of repair (Definition 3) has been revisited many times since the publication of [3]. For instance, one can minimize the *cardinality* of the set of changes [50], as opposed to minimizing the *set* of changes under set inclusion, as in Definition 3. Moreover, attribute-based changes were considered in [14,58,59] (this issue is discussed further in Section 6). Surprisingly, the notion of consistent query answer (Definition 4) has been almost universally adopted in the recent literature of the subject (but see Section 5).

The scope of the query rewriting method detailed above (and further developed in [21]) is quite limited. It applies to first-order queries without disjunction or quantification, and binary universal integrity constraints. So, for example, the query Q_2 from Example 2 cannot be handled. A rewriting method that can handle a rather large subset of conjunctive queries at the price of limiting integrity constraints to primary key FDs was recently proposed by Fuxman and Miller [38]. This method, which can handle Q_2 from Example 2, is discussed in more detail in the next section. Adding quantification *and* disjunction has proved to be much harder: the approach of [38] has been extended in that direction by Lembo et al. [47]. The paper [38] was further generalized to include *exclusion dependencies* by Grieco et al. [41].

3 Computational Complexity

We note first that already in the presence of a single *primary key dependency* there are inconsistent relation instances with exponentially many repairs [5]. Thus Definition 4 does not yield a practical method for computing consistent query answers. Below, we show a number of cases in which the latter problem can be solved in polynomial time, and later we characterize the intractable cases in detail. We limit ourselves to *universal* constraints here; inclusion dependencies are discussed in Section 4. We consider two basic decision problems:

- *repair checking*: Is a database instance a repair of another instance w.r.t. the integrity constraints?
- *consistent query answering (CQA)*: Is a tuple a consistent query answer to a query in a database instance w.r.t. the integrity constraints?

The motivation to study repair checking, in addition to CQA, comes from *data cleaning* where a single, consistent database instance needs to be constructed. The repair-checking algorithms can typically be adapted to yield such an instance.

We adopt here the *data complexity* assumption [1,46,57], which measures the complexity of the problem as a function of the number of tuples in a database instance. The query and the integrity constraints are considered fixed.

3.1 Tractable Cases

It is easy to see that for denial constraints repair checking can be done in polynomial time: check whether a potential repair I' is a subset of the database instance I and satisfies the integrity constraints, and whether adding any other tuple from $I - I'$ to I' leads to a constraint violation.

Query rewriting approaches that produce first-order queries provide polynomial-time algorithms for CQA: rewrite the query and evaluate the rewritten query on the original database. Note that the rewriting of the query is done independently of the database instance, and therefore does not affect data complexity.

However, the original query rewriting approach of [3] was applicable only to very restricted classes of queries and constraints (see the previous section).

Recently, that approach was generalized by Fuxman and Miller [38] to allow restricted existential quantification in queries in the context of primary key FDs. The rewriting method of [38] applies to a class of conjunctive queries C_{forest}, defined using the notion of the join graph of a query. The vertices of the join graph are the query literals; an edge runs from a literal A_i to a different literal A_j if there is an existentially-quantified variable which occurs in a nonkey attribute of A_i and any attribute of A_j. The class C_{forest} consists of those conjunctive queries that have a join graph which is a forest, and which have no repeated relation symbols or built-in predicate symbols.

Example 4. Consider the following query Q:

$$Q \equiv \exists x, y, z. P_1(x, y) \wedge P_2(y, z).$$

Assume that the first attributes of both P_1 and P_2 are primary keys. Then Q expresses a *foreign-key*[2] join and belongs to C_{forest}. Then the rewritten query Q' is:

$$Q' \equiv \exists x, y, z. P_1(x, y) \wedge P_2(y, z) \wedge \forall y'. \big(P_1(x, y') \Rightarrow \exists z'. P_2(y', z')\big).$$

Another way to obtain tractability is through the notion of *conflict graph*[5,26]. The vertices of the conflict graph are the tuples in the database; an edge connects two vertices if they violate together an integrity constraint (we assume binary constraints for the moment). A conflict graph is a compact, polynomial-size representation of the set of all repairs of the database: the repairs correspond to maximal independent sets of the graph. The conflict graph can be used to compute consistent answers to queries. Chomicki and Marcinkowski [26] describe a polynomial-time algorithm for CQA that is applicable to quantifier-free queries and denial integrity constraints[3]. The algorithm enumerates repairs, trying to show that a tuple is not a consistent answer. The crucial observation is that only fixed-size fragments of repairs need to be considered.

3.2 Intractable Cases

Computational complexity analysis helps to delineate the boundary between the tractable and the intractable cases. We start the discussion of the relevant complexity results by recalling one of the fundamental results in this area.

Theorem 1. *For conjunctive queries and primary key FDs, CQA is co-NP-complete.*

[2] Note that while the variable y plays here the role of a foreign key, the corresponding *foreign-key constraint* is not taken into account in CQA. We discuss such constraints in Section 4.

[3] To deal with denial constraints, conflict graphs are generalized to *conflict hypergraphs.*

Proof. We describe here the proof of [26] because it is the simplest and was – to our knowledge – chronologically the first. A different proof appears in [19].

The proof is by reduction from MONOTONE 3-SAT. Let $\beta = \phi_1 \wedge \ldots \phi_m \wedge \psi_{m+1} \ldots \wedge \psi_l$ be a conjunction of propositional clauses, such that all occurrences of variables in ϕ_i are positive and all occurrences of variables in ψ_i are negative. To encode such formulas, we use two binary relation schemas P_1 and P_2, each with two attributes of which the first is the key. We build a database instance I such that $I(P_1)$ contains the tuple (i, p) if the variable p occurs in the clause ϕ_i, and $I(P_2)$ contains the tuple (i, p) if the variable p occurs in the clause ψ_i. The query $Q \equiv \exists x, y, z.\ \big(P_1(x, y) \wedge P_2(z, y)\big)$. Now there is an assignment which satisfies β iff there exists a repair of I in which Q is false.

In the above proof, the query Q does not belong to C_{forest} because it contains a join between nonkey attributes, which produces a cycle in the join graph. In the full version of [38], Fuxman and Miller show that several other natural relaxations of the C_{forest} property also lead to co-NP-completeness of CQA. For a class C^* of conjunctive queries, they prove a *dichotomy* result: CQA for each query in C^* is in P iff the join graph of the query does not contain a cycle.

For universal constraints, repair checking is co-NP-complete and CQA Π_2^p-complete in most cases, as recently shown by Staworko et al. [53]. The computational complexity of consistent query answering is summarized in Figure 1. For the purpose of exposition, we refer there to subsets of relational algebra instead of sublanguages of first-order logic. For each result, we cite the primary source, except for those that follow from the definitions or other results.

	Primary keys	*Arbirary keys*	*Denial*	*Universal*
$\sigma, \times, -$	P	P[3]	P	Π_2^p-complete[53] P(binary)[3]
$\sigma, \times, -, \cup$	P	P	P[26]	Π_2^p-complete
σ, π	P[26]	co-NPC[26]	co-NPC	Π_2^p-complete[53]
$\sigma, \pi, \times$	co-NPC[26] P(C_{forest})[38]	co-NPC	co-NPC	Π_2^p-complete
$\sigma, \pi, \times, -, \cup$	co-NPC	co-NPC	co-NPC	Π_2^p-complete

Fig. 1. Complexity of CQA: relational algebra

Proving co-NP-completeness of CQA for a class of first-order queries C is sufficient to show that unless P=NP there is no query rewriting method that (a) returns first-order queries, and (b) is applicable to all queries in C.

4 Referential Integrity Constraints

For denial constraints, integrity violations can only be removed by *deleting* tuples, so all the repairs are subsets of the given database instance. The picture changes when we consider general universal or referential integrity constraints. Violations of such constraints can also be removed by *adding* tuples.

Example 5. Consider a database schema with two relations $P_1(AB)$ and $P_2(C)$, the inclusion dependency $P_1[B] \subseteq P_2[C]$, and the key dependency $A \rightarrow B$. Consider I' such that $I'(P_1) = \{(a,b),(a,c)\}$ and $I'(P_2) = \{b\}$. Then we have the following repairs:

$$I'_1(P_1) = \{(a,b)\}, I'_1(P_2) = \{b\}$$
$$I'_2(P_1) = \{(a,c)\}, I'_2(P_2) = \{b,c\}$$

Allowing repairs constructed using insertions makes sense if the information in the database may be incomplete[4]. The latter is common in data integration applications where the data is pulled from multiple sources, typically without any guarantees on its completeness. On the other hand, if we know that the data in the database is complete but possibly incorrect, as in data warehousing applications, it is natural to consider only repairs constructed using deletions. Current language standards like SQL:1999 [51] allow only deletions in their repertoire of referential integrity actions.

The above considerations have lead to the definition of two new, more restricted classes of repairs:

- *D-repairs*, constructed using a minimal set of deletions [26] (I'_1 in Example 5),
- *I-repairs*, constructed using a minimal set of deletions and some, not necessarily minimal set of insertions [19] (in Example 5, this includes I'_1 and I'_2, as well as any consistent supersets of those).

Each of those classes of repairs leads to a different notion of consistent query answer. We consider first D-repairs. Note the following properties of this class of repairs:

1. Every database instance has a single D-repair w.r.t. any set of INDs, which is obtained by deleting the tuples violating the constraints.
2. Given a set of primary key FDs F and a set of foreign-key constraints IN, every repair of a database instance w.r.t. $F \cup IN$ may be obtained as a repair of the single D-repair of the instance w.r.t. IN (this is because repairing w.r.t. F does not lead to any new violations of IN).

[4] *Incompleteness* here does not mean that the database contains *indefinite information* in the form of nulls or disjunctions [56]. Rather, it means that the *Open World Assumption* is adopted, i.e., the facts missing from the database are not assumed to be false.

The second property implies that one can adapt any polynomial-time method for CQA w.r.t. primary key constraints, for example [38], to compute consistent query answers w.r.t. any set of primary key and foreign-key constraints in polynomial time. However, if one goes beyond this simple setting, the interactions between FDs and INDs get complex, and both repair checking and CQA become quickly intractable [26]. Ultimately, for arbitrary FDs and INDs repair checking is co-NP-complete and CQA Π_2^P-complete. (All of those results hold under the definition of CQA in which D-repairs are substituted for repairs.)

We consider now I-repairs. Calì et al. [19] show that for such repairs in the presence of primary key FDs and arbitrary INDs CQA becomes undecidable. This is shown by a reduction from the implication problem for those constraints which is known to be undecidable [1]. Calì et al. [19] define a class of INDs, called *non-key-conflicting*, for which the interaction between FDs and INDs is limited and consequently CQA is co-NP-complete. Technically, they relate CQA to *conjunctive query containment* under integrity constraints [44]. Calì et al. [19] also analyze repairs (in the sense of Definition 3) in the same setting, obtaining undecidability of CQA in the general case and Π_2^p-completeness of CQA for non-key-conflicting INDs.

5 More Informative Consistent Query Answers

In the previous section we considered varying the notion of repair. Here we keep the original notion of repair but slightly adjust the notion of consistent query answer. The motivation comes from aggregate queries and probabilistic databases.

Example 6. Consider the following instance of the relation $Emp(Name, Salary)$ with the key *Name*:

Name	*Salary*
John	50K
John	60K
Mary	55K

Now the query

```
SELECT MAX(Salary) FROM Emp
```

returns 55K in one repair and 60K in the the other. So there is no consistent answer in the sense of Definition 4.

To provide more informative query answers to aggregation queries, Arenas et al. [4] propose to return the *minimal interval* containing the set of the values of the aggregate function obtained in some repair. In Example 6, the interval [55K,60K] is returned. The paper [4] contains a detailed analysis of the data

	$\lvert F\rvert = 1$	$\lvert F\rvert \geq 2$
MIN(A),MAX(A),SUM(A),AVG(A),COUNT(*)	P	NP-complete
COUNT(A)	NP-complete	NP-complete

Fig. 2. Complexity of scalar aggregation queries

complexity of computing interval answers in the presence of FDs, and showed the influence of the cardinality $|F|$ of the given set of FDs F. The results are summarized in Figure 2.

The tractable cases are typically obtained by query rewriting. The exception is AVG(A), which is computed by an iterative algorithm.

Example 7. The query

```
SELECT MAX(Salary) FROM Emp
```

is rewritten as

```
SELECT SUM(P.MinS), SUM(P.MaxS)
FROM (SELECT MIN(Salary) AS MinS, MAX(Salary) AS MaxS
      FROM Emp GROUP BY Name) P
```

Fuxman and Miller [37] develop a comprehensive framework for rewriting SQL queries with aggregation by combining the methods of [38] and [4]. The framework also allows grouping constructs in queries.

The notion of repair and consistent query answer has been generalized to the context of *probabilistic databases* by Andritsos et al. [2]. In such databases probabilities are associated with individual tuples. Assume the presence of a primary key FD. Then the probabilities of the conflicting tuples sum up to one. A repair also has an associated probability, which is the product of the probabilities of the tuples belonging to the repair. There is a natural way to compute the probability of an answer: sum up the probabilities of the repairs in which the answer appears in the query result. Such answers, with the associated probabilities, are called *clean answers* [2]. Clearly, consistent answers are those clean answers that have probability one. Andritsos et al. [2] present a way to compute clean answers through query rewriting. Their method applies to a class of conjunctive queries closely related to C_{forest} (see Section 3).

Example 8. Figure 3a contains a probabilistic version of the relation *Emp* with the key *Name*. The clean answers to the query

```
SELECT Name FROM Emp WHERE Salary > 52K
```

are shown in Figure 3b. The rewritten query that computes the clean answers is

```
SELECT Name, SUM(Probability) FROM Emp WHERE Salary > 52K
GROUP BY Name
```

Name	*Salary*	Probability
John	50K	0.6
John	60K	0.4
Mary	55K	1

(a)

Name	Probability
John	0.4
Mary	1

(b)

Fig. 3. (a) *Emp* with probabilities; (b) Clean answers

6 Other and Future Work

A major line of work on CQA involves capturing repairs as *answer sets* of logic programs with negation and disjunction [4,39]. Such approaches are quite general, being able to handle arbitrary universal constraints and first-order queries. Determining whether an atom is a member of all answer sets of such a logic program is Π_2^p-complete [28]. Therefore, a direct implementation of CQA using a disjunctive logic programming system like `dlv` [48] or `smodels` [52] is practical only for very small databases. Recently, special optimization techniques in this area have been developed by Eiter et al. [29]. Answer-set-based techniques have been particularly effective in addressing the semantic problems of data integration, where the main issue is how to reconcile repairing the violations of integrity constraints with satisfying the rules describing the mappings between different databases. This issue was addressed in the context of LAV-mappings by Bravo and Bertossi [16,9], and in the context of peer-to-peer mappings by Calvanese et al. [20].

It is natural to consider *preferences* or *priorities* in repairing. For example, if a database violates an FD because of conflicting data coming from different sources such conflicts may be resolved if the sources have different reliability. Similarly, new information may be preferred to old information. In a data cleaning process, preferences are typically encoded using conflict resolution rules. In CQA, more declarative approaches have been pursued. Staworko et al. [55] consider priority relations defined on atoms and discuss various ways in which such relations could be lifted to the level of repairs, yielding *preferred repairs*. Typically, optimization with respect to an additional criterion, represented by the priority, increases the complexity of repair checking and CQA. Flesca et al. [40] define preferred repairs directly, using a numeric utility function.

Repairs in the sense of Definition 3 have been criticised as *too coarse-grained*: deleting a tuple to remove an integrity violation potentially eliminates useful information in that tuple. More fine-grained methods seek to define repairs by minimizing *attribute modifications* [10,14,58]. In particular, Bertossi et al. [10] and Bohannon et al. [14] use various notions of numerical distance between tuples. In both cases the existence of a repair within a given distance of the original database instance turns out to be NP-complete. To achieve tractability, Bertossi et al. [10] propose approximation, and Bohannon et al. [14], heuristics.

Wijsen [59] has recently shown how to combine tuple- and attribute-based repairs in a single framework. To achieve the effect of attribute-based repairing, his approach decomposes an inconsistent relation using a lossless-join decomposition and subsequently joins the obtained projections [5]. PJ-repairs are defined to be the repairs (in the sense of Definition 3) of the resulting relation. Thus, query evaluation methods of Section 3 can be readily applied in that framework.

Another direction is *repairs with nulls.* A repair with nulls can represent a *set* of ground repairs. This is particularly useful when dealing with INDs.

Example 9. Consider a slightly modified database schema from Example 5, consisting now of two relations $P_1(AB)$ and $P_2(CD)$, and an inclusion dependency $P_1[B] \subseteq P_2[C]$. Assume an instance I is as follows: $I(P_1) = \{(a, b)\}$ and $I(P_2) = \emptyset$. This instance has a repair I_1 where $I_1(P_1) = \emptyset, I_1(P_2) = \emptyset$. However, there are also infinitely many repairs I' of the form $I'(P_1) = \{(a, b)\}, I'(P_2) = \{b, \alpha\}$ where α is a constant. All such repairs can be represented as a single repair I_{null} where $I_{null}(P_1) = \{(a, b)\}, I_{null}(P_2) = \{(b, null)\}$

Notice that nulls can also be used to represent a resolved version of an inconsistency associated with an FD, as in Example 2: there would be a single repair consisting of the tuples (Mary,North,null) and (John,South,Hayes). Since the formal semantics of nulls [56] is based on possible worlds that are closely related to repairs, it should be feasible to incorporate repairs with nulls into the CQA framework, using a common semantic basis. This has not been done yet, however. Bravo and Bertossi [17] take a different, more syntactic approach that simulates *SQL nulls* (whose semantic problems are well known [56]) within a logic programming approach to repair specification.

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 [54] and DTDs with functional dependencies in [34]. Staworko et al. [54] propose to base repair minimality on *tree edit distance* [13], while Flesca et al. [34] use an approach more akin to that of [3]. More expressive integrity constraint languages for XML, for example [32], should be considered next.

There are now several prototype CQA systems: CONQUER [37] (based on query rewriting), HIPPO [27] (based on conflict hypergraphs), and INFOMIX [29] (based on answer set programming). Those systems are capable of handling databases with several million tuples.

Considering the number of researchers, projects, and publications involved, consistent query answering seems to be enjoying significant interest as a research topic. Below we identify some of the current and future challenges in this area (in addition to those mentioned earlier):

Coping with semantic heterogeneity. The number of different repair semantics proposed so far, particularly when one considers variations involving nulls and priorities, may overwhelm a potential user. The ways need to be found to provide guidance which semantics are appropriate for specific applications. Also, methods

[5] For a consistent relation such transformation is an identity but for an inconsistent one this is usually not the case.

that unify various approaches within a single framework, as for example [59], should be studied.

Integration with other tools. Ultimately, repairing and CQA should become tools in data integration toolboxes. We have already mentioned incorporating CQA in several data integration frameworks [16,9,20] but still more work is needed in the context of data exchange [31]. Also, integration of CQA with data cleaning seems to be in order. CQA is unnecessarily conservative in the presence of data errors and duplicates. Moreover, integrity violations in an integrated database are often due to structural or semantic discrepancies between the sources, and thus the quality of schema matching/mapping clearly influences the usefulness of CQA [24]. In a broader sense, CQA fits within the framework of *data quality management* [7].

Applications. Very few real-life applications of repairing and consistent query answering have been reported so far. Franconi et al. [36] summarize an application of attribute-based repairing in the area of census data. Flesca et al. [35,33] describe a tool for acquiring and repairing balance-sheet data. That work is notable for its use of *aggregation constraints*. Generally, it seems that most potential applications find repairing more useful than CQA. In many cases, the data in the database is relatively *static*, so it makes sense to invest a considerable effort into its cleaning and repairing. Such data can then be repeatedly used. CQA appears to be more suitable to *dynamic* environments, particularly those that require real-time decisions based on the available data.

Acknowledgments

The collaboration and the interaction with the following people are gratefully acknowledged: Marcelo Arenas, Leopoldo Bertossi, Wenfei Fan, Jerzy Marcinkowski, Sławomir Staworko, and Jef Wijsen.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2. P. Andritsos, A. Fuxman, and R. Miller. Clean Answers over Dirty Databases. In *IEEE International Conference on Data Engineering (ICDE)*, 2006.
3. M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.
4. M. Arenas, L. Bertossi, and J. Chomicki. Answer Sets for Consistent Query Answering in Inconsistent Databases. *Theory and Practice of Logic Programming*, 3(4–5):393–424, 2003.
5. M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 296(3): 405–434, 2003.
6. C. Baral, S. Kraus, J. Minker, and V. S. Subrahmanian. Combining Knowledge Bases Consisting of First-Order Theories. *Computational Intelligence*, 8:45–71, 1992.

7. C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer, 2006.
8. L. Bertossi. Consistent Query Answering in Databases. *SIGMOD Record*, 35(2), June 2006.
9. L. Bertossi and L. Bravo. Consistent Query Answers in Virtual Data Integration Systems. In Bertossi et al. [12], pages 42–83.
10. L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. Complexity and Approximation of Fixing Numerical Attributes in Databases Under Integrity Constraints. In *International Workshop on Database Programming Languages*, pages 262–278. Springer, LNCS 3774, 2005.
11. L. Bertossi and J. Chomicki. Query Answering in Inconsistent Databases. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*, pages 43–83. Springer-Verlag, 2003.
12. L. Bertossi, A. Hunter, and T. Schaub, editors. *Inconsistency Tolerance*. Springer-Verlag, 2004.
13. P. Bille. A Survey on Tree Edit Distance and Related Problems. *Theoretical Computer Science*, 337(1-3):217–239, 2003.
14. P. Bohannon, M. Flaster, W. Fan, and R. Rastogi. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *ACM SIGMOD International Conference on Management of Data*, pages 143–154, 2005.
15. A. Borgida. Language Features for Flexible Handling of Exceptions in Information Systems. *ACM Transactions on Database Systems*, 10(4):565–603, 1985.
16. L. Bravo and L. Bertossi. Logic Programs for Consistently Querying Data Integration Systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 10–15, 2003.
17. L. Bravo and L. Bertossi. Semantically Correct Query Answers in the Presence of Null Values. In *EDBT Workshops (IIDB)*. Springer, 2006.
18. F. Bry. Query Answering in Information Systems with Integrity Constraints. In *IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems*, pages 113–130. Chapman &Hall, 1997.
19. A. Calì, D. Lembo, and R. Rosati. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 260–271, 2003.
20. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Inconsistency Tolerance in P2P Data Integration: An Epistemic Logic Approach. In *International Workshop on Database Programming Languages*, pages 90–105. Springer, LNCS 3774, 2005.
21. A. Celle and L. Bertossi. Querying Inconsistent Databases: Algorithms and Implementation. In *International Conference on Computational Logic*, pages 942–956. Springer-Verlag, LNCS 1861, 2000.
22. U. S. Chakravarthy, J. Grant, and J. Minker. Logic-Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.
23. A. Chandra and P. Merlin. Optimal Implementation of Conjunctive Queries in Relational Databases. In *ACM SIGACT Symposium on the Theory of Computing (STOC)*, pages 77–90, 1977.
24. J. Chomicki. Consistent Query Answering: Opportunities and Limitations. In *DEXA Workshops (LAAIC)*, pages 527–531. IEEE Computer Society Press, 2006.
25. J. Chomicki and J. Marcinkowski. On the Computational Complexity of Minimal-Change Integrity Maintenance in Relational Databases. In Bertossi et al. [12], pages 119–150.

26. J. Chomicki and J. Marcinkowski. Minimal-Change Integrity Maintenance Using Tuple Deletions. *Information and Computation*, 197(1-2):90–121, 2005.
27. J. Chomicki, J. Marcinkowski, and S. Staworko. Computing Consistent Query Answers Using Conflict Hypergraphs. In *International Conference on Information and Knowledge Management (CIKM)*, pages 417–426. ACM Press, 2004.
28. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.
29. T. Eiter, M. Fink, G. Greco, and D. Lembo. Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In *International Conference on Logic Programming (ICLP)*, pages 163–177, 2003.
30. S. M. Embury, S. M. Brandt, J. S. Robinson, I. Sutherland, F. A. Bisby, W. A. Gray, A. C. Jones, and R. J. White. Adapting Integrity Enforcement Techniques for Data Reconciliation. *Information Systems*, 26(8):657–689, 2001.
31. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, 336(1):89–124, 2005.
32. W. Fan and J. Simeon. Integrity Constraints for XML. *Journal of Computer and System Sciences*, 66(1):254–201, 2003.
33. B. Fazzinga, S. Flesca, F. Furfaro, and F. Parisi. DART: a Data Acquisition and Repairing Tool. In *EDBT Workshops (IIDB)*. Springer, 2006.
34. S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Querying and Repairing Inconsistent XML Data. In *Web Information Systems Engineering*, pages 175–188. Springer, LNCS 3806, 2005.
35. S. Flesca, F. Furfaro, and F. Parisi. Consistent Query Answers on Numerical Databases under Aggregate Constraints. In *International Workshop on Database Programming Languages*, pages 279–294. Springer, LNCS 3774, 2005.
36. E. Franconi, A. L. Palma, N. Leone, S. Perri, and F. Scarcello. Census Data Repair: a Challenging Application of Disjunctive Logic Programming. In *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 561–578. Springer-Verlag, LNCS 2250, 2002.
37. A. Fuxman and R. J. Miller. ConQuer: Efficient Management of Inconsistent Databases. In *ACM SIGMOD International Conference on Management of Data*, pages 155–166, 2005.
38. A. Fuxman and R. J. Miller. First-Order Query Rewriting for Inconsistent Databases. In *International Conference on Database Theory (ICDT)*, pages 337–351. Springer, LNCS 3363, 2005. Full version to appear in JCSS.
39. G. Greco, S. Greco, and E. Zumpano. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1389–1408, 2003.
40. S. Greco, C. Sirangelo, I. Trubitsyna, and E. Zumpano. Preferred Repairs for Inconsistent Databases. In *International Conference on Database and Expert Systems Applications (DEXA)*, pages 44–55, 2004.
41. L. Grieco, D. Lembo, R. Rosati, and M. Ruzzi. Consistent Query Answering under Keys and Exclusion Dependencies: Algorithms and Experiments. In *International Conference on Information and Knowledge Management (CIKM)*, pages 792–799. ACM Press, 2005.
42. T. Imieliński and W. Lipski. Incomplete Information in Relational Databases. *Journal of the ACM*, 31(4):761–791, 1984.
43. T. Imieliński, S. Naqvi, and K. Vadaparty. Incomplete Objects - A Data Model for Design and Planning Applications. In *ACM SIGMOD International Conference on Management of Data*, pages 288–297, Denver, Colorado, May 1991.

44. D. S. Johnson and A. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *Journal of Computer and System Sciences*, 28(1):167–189, 1984.
45. W. Lipski Jr. On Semantic Issues Connected with Incomplete Information Databases. *ACM Transactions on Database Systems*, 4(3):262–296, 1979.
46. P. C. Kanellakis. Elements of Relational Database Theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 17, pages 1073–1158. Elsevier/MIT Press, 1990.
47. D. Lembo, R. Rosati, and M. Ruzzi. On the First-Order Reducibility of Unions of Conjunctive Queries over Inconsistent Databases. In *EDBT Workshops (IIDB)*, 2006.
48. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 2006. To appear.
49. J. Lin and A. O. Mendelzon. Merging Databases under Constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1996.
50. A. Lopatenko and L. Bertossi. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. In *International Conference on Database Theory (ICDT)*, 2007. To appear.
51. Jim Melton and Alan R. Simon. *SQL:1999 Understanding Relational Language Components*. Morgan Kaufmann, 2002.
52. P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138(1-2):181–234, June 2002.
53. S. Staworko and J. Chomicki. Consistent Query Answers in the Presence of Universal Constraints. Manuscript, October 2006.
54. S. Staworko and J. Chomicki. Validity-Sensitive Querying of XML Databases. In *EDBT Workshops (dataX)*. Springer, 2006.
55. S. Staworko, J. Chomicki, and J. Marcinkowski. Priority-Based Conflict Resolution in Inconsistent Relational Databases. In *EDBT Workshops (IIDB)*. Springer, 2006.
56. R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 10, pages 307–356. Kluwer Academic Publishers, Boston, 1998.
57. M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.
58. J. Wijsen. Database Repairing Using Updates. *ACM Transactions on Database Systems*, 30(3):722–768, 2005.
59. J. Wijsen. Project-Join Repair: An Approach to Consistent Query Answering Under Functional Dependencies. In *International Conference on Flexible Query Answering Systems (FQAS)*, 2006.