Database Repairs and Consistent Query Answering: Origins and Further Developments

Leopoldo Bertossi*
leopoldo.bertossi@relational.ai
RelationalAI Inc.
Toronto, Canada

ABSTRACT

In this article we review the main concepts around database repairs and consistent query answering, with emphasis on tracing back the origin, motivation, and early developments. We also describe some research directions that has spun from those main concepts and the original line of research. We emphasize, in particular, fruitful and recent connections between repairs and causality in databases.

CCS CONCEPTS

• **Data management systems → databases**; • **Integrity constraints**;

KEYWORDS

databases, database repairs, consistent query answering, integrity constraints, data integration, answer-set programs, causality

ACM Reference Format:
Leopoldo Bertossi. 2019. Database Repairs and Consistent Query Answering: Origins and Further Developments. In *38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS'19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3294052.3322190

1 INTRODUCTION

The notions of database repair and consistent query answering (CQA) explicitly appeared in a PODS paper in 1999 [3], on databases that may not satisfy (i.e. may be inconsistent with) a given set of integrity constraints (ICs). That paper, co-authored with Marcelo Arenas and Jan Chomicki, has received many citations and has been the basis for a large body of research, until these days, and even beyond the data management community. In particular, the notion of "repair" has become ubiquitous in several areas of computer science. Most of the lines of research on repairs and CQA are well-known in the community of theoretical data management, but possible less known in more applied communities. The same applies, more generally, to some newer "conceptual" applications to other areas of data management and knowledge representation. Without attempting to provide a comprehensive review, the goal of this article is to revisit the early days, and briefly and partially fill the gap in relation to lesser known directions the original ideas have taken. There are several early reviews of the area [12, 18, 49]. The more recent monograph [14] is a rather comprehensive survey of the "core" of this area up to a certain time point. A recent survey of the vast area of CQA for conjunctive queries (CQs) under key constraints can be found in [109].

Since it not possible to introduce, in a such a limited space, formal definitions that can be found in the original papers, the presentation will be based on representative examples.

2 THE BEGINNINGS

Right after finishing my PhD in Mathematics, in mathematical logic, and working at the Catholic University of Chile (PUC), I decided to do research on applications of logic to computer science, more specifically, to knowledge representation and data management. Crucial influences in those days of the late 80s and early 90s were Ray Reiter and Alberto Mendelzon. The latter gave in 1988 an inspiring tutorial in Chile on "Logic in Databases". I was lucky to start collaboration with Ray Reiter in 1990, when I was visiting the University of Toronto. Not long after that, he started doing research on the situation calculus [97], in particular on applications to the specification of database updates and the dynamics of a database [96].[1] So, I joined in, and started working in the area.

Around that time I joined the Department of Computer Science of the School of Engineering of the PUC. Soon I noticed that there was very little research going on there. On the other side, the department was full of students, and most of them of excellent quality. Given this situation, I developed a tree-tier plan: (a) I started teaching an undergraduate course on logic for computer science, which included apart from classical logic, models of computation, elements of complexity, logic programming, and elements of knowledge representation. (This was the only course that did expose students to some computer science.) The course had a sort of a hands-on workshop associated to it, where students learned about automated theorem proving with Otter, Prolog, and at some point also answer-set programming with DLV.[2] (b) I started inviting the best students in that course to do research with me, to motivate them to pursue graduate studies (in the hope they would come back; Chileans do come back). (c) I started inviting researchers from abroad to give tutorials and engage with the local students, to create a research

*Also affiliated to Carleton University, Ottawa, Canada, and member of the "Millenium Institute for Foundational Research on Data (IMFD, Chile)

PODS'19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6227-6/19/06…$15.00
https://doi.org/10.1145/3294052.3322190

[1] It is interesting to observe that mathematical logic has been broadly applied to static aspects of databases, but very little to their dynamic aspects, which involve updates, transactions, dynamic constraints, active rules, hypothetical queries, etc.
[2] Several brilliant students helped me create that applied appendix to the course, most prominently, Cristian Ferretti and Marcelo Arenas.

atmosphere around them. Materializing this plan was not without toil; it was a colonizing and also political undertaking in may ways, but it certainly paid off. A seed of relevant research activity was sown; and several of the now established computer science researchers in Chile were exposed to this experience. I want to believe it did provide some inspiration to them.

In this context, and since I was working on dynamic aspects of databases, I invited Jan Chomicki to visit the PUC. He was a prominent researcher in temporal databases. At the same time, Marcelo Arenas was starting his masters in computer science under my supervision (we had started doing research together before that, while he was still an undergraduate student, as I described in the previous paragraph); and I invited him to join the conversations. In this way, the three of us ended up staring at a blank board, asking ourselves about how to take off. As usual in this kind of meetings, we started wandering in different directions, and at some point we became interested in comparing databases in relation to consistency, asking when can one say a database is more consistent than another database (both for the same schema). As usual again, we did not attack that question, but we became intrigued by the question about what an inconsistent database can tell us. Which led to the more precise question, but still not quite, about what is consistent in an inconsistent database. This question was further refined into "what are the consistent answers to a query posed to an inconsistent database, and how can we obtain them?" Although this question was still unprecise, it was the real start. We did not ask ourselves if this was a fashionable line of research, a well-known open problem, or if someone else was doing something along similar lines. And we started playing with examples as the one that follows.

Example 2.1. Consider the database instance D below, and the inclusion dependency

$$ID\colon\ \forall x\forall y\forall z(Supply(x,y,z)\ \rightarrow\ Articles(z)), \tag{1}$$

requiring that the items shipped according to table *Supply* are all among the official list of items displayed in table *Articles*.

Supply	Company	Receiver	Item
	C_1	R_1	I_1
	C_2	R_2	I_2
	C_2	R_1	I_3

Articles	Item
	I_1
	I_2

This instance is inconsistent with respect to (wrt.) *ID*, i.e. it does not satisfy *ID*, usually denoted with $D \not\models ID$. Clearly there is a problem with the last tuple of table *Supply*, but the information in the other tuples of the database seems to be fine. Now, if we pose the query about the items that are supplied, i.e. the conjunctive query

$$Q(z)\colon\ \exists x\exists y Supply(x,y,z), \tag{2}$$

i.e. a projection on the third attribute, we intuitively expected to obtain as consistent answers only I_1 and I_2, but how? □

Not long before these conversations, I had invited Jack Minker to visit the PUC. He gave a tutorial on semantic query optimization, which is about optimizing query answering by taking advantage of integrity constraints that are satisfied by the database. This can be done via query rewriting, by appending to the query at hand "residues" from the ICs. They impose additional conditions on the query, narrowing down the search space [46]. We though we could try out locally enforcing, possibly unsatisfied, ICs on the query results (rather than on the whole database) through similar rewritings.

Example 2.2. (example 2.1 cont.) *ID* in (1) is logically equivalent to the (implicitly universally quantified) clause

$$\neg Supply(x,y,z) \vee Articles(z), \tag{3}$$

whose first, negative disjunct can be canceled (resolved [86]) with the (positive) query atom $Supply(x,y,z)$ in (2), leaving the residue $Articles(z)$ of *ID*. This residue is appended to the original query, obtaining a rewritten query:

$$Q'(z)\colon\ \exists x\exists y(Supply(x,y,z) \wedge Articles(z)). \tag{4}$$

The new query posed to the original, inconsistent database does return the intuitively expected answers. □

3 FIRST NOTIONS AND DEVELOPMENTS

The next natural research steps were about clarifying two questions: (a) The rewriting mechanism was just that, a computational approach to the original question, but what was the *semantics* of consistent answers to a query posed to an inconsistent database. (b) Assuming that the rewriting-based approach was in agreement with that semantics, what was its the scope of applicability.

3.1 Repairs and consistent query answering

In relation to question (a) above, the intuition that we were developing was that the consistent data inside the inconsistent database are those that are invariant under the process of restoring the consistency of the database. Those consistent data persisted under update actions on the database that made the database consistent. Of course, those update actions have to be restricted somehow. After all, making the database in Example 2.1 empty produces a consistent instance, but this is an unintended kind of repair process. Accordingly, the consistency-restoring updates had to be "minimal" in some way. At the same time we decided to accept only deletions and insertions of tuples as admissible updates (the rewriting approach we were considering worked at the tuple, i.e. atomic, level). This led to the notion of *database repair*.

Example 3.1. (example 2.1 cont.) A repair of the original instance D is any instance D' for the same schema that satisfies *ID*, is obtained from D by inserting or deleting tuples, and minimally differs from D, in the sense that the symmetric difference, $D\Delta D' := (D \setminus D') \cup (D' \setminus D)$, is minimal under set-inclusion. This is a reason why these repairs are also known as *S-repairs*, referring to set-minimality.

For example, $D_1 = \{Supply(C_1,R_1,I_1), Supply(C_2,R_2,I_2), Articles(I_1), Articles(I_2)\}$, obtained deleting $Supply(C_2,R_1,I_3)$ from D is a repair, with $D\Delta D_1 = \{Supply(C_2,R_1,I_3)\}$. Another repair is $D_2 = \{Supply(C_1,R_1,I_1), Supply(C_2,R_2,I_2), Supply(C_2,R_1,I_3), Articles(I_1), Articles(I_2), Articles(I_3)\}$, obtained inserting $Articles(I_3)$ into D, with $D\Delta D_2 = \{Articles(I_3)\}$.

Since $D\Delta D_1$ and $D\Delta D_2$ are incomparable under set-inclusion, both are S-repairs. However, $D_3 = \{Supply(C_1,R_1,I_1), Articles(I_1), Articles(I_2)\}$, obtained deleting $Supply(C_2,R_1,I_3)$ and $Supply(C_2,R_2,I_2)$ from D, is consistent, but not an S-repair: $D\Delta D_1 \subsetneq D\Delta D_3 = \{Supply(C_2,R_1,I_3), Supply(C_2,R_2,I_2)\}$.

Looking at the two obtained repairs, we can see that the data in D that persist across repairs are the first two tuples in table *Supply* and the two tuples in table *Articles*. □

Next, given a database D, a set of ICs Σ, and a query $Q(\bar{x})$, one can define the set of *consistent answers to* Q wrt. D and Σ, denoted $Cons(Q, D, \Sigma)$, as the set of those answers that are *obtained from every repair* when the query is posed to them. So, more formally: $\bar{a} \in Cons(Q, D, \Sigma)$ iff $\bar{a} \in Q(D')$ for every repair D' of D wrt. Σ. (As usual, $Q(D)$ denotes the sets of answer to query Q from D.)

Example 3.2. (example 3.1 cont.) For the query $Q(z)$ in (2), $Q(D_1) = \{I_1, I_2\}$, $Q(D_2) = \{I_1, I_2, I_3\}$. Then, $Cons(Q, D, \{ID\}) = \{I_1, I_2\}$. □

Example 3.3. The following instance D violates the key constraint KC: $Name \rightarrow Salary$ that requires the employee salary to functionally depend upon the employee name, i.e. every employee should have at most one salary.

Employee	*Name*	*Salary*
	page	*5K*
	page	*8K*
	smith	*3K*
	stowe	*7K*

There are two repairs, D_1 and D_2 obtained by tuple deletions:

Employee	*Name*	*Salary*
	page	*5K*
	smith	*3K*
	stowe	*7K*

Employee	*Name*	*Salary*
	page	*8K*
	smith	*3K*
	stowe	*7K*

They are consistent with KC and minimally depart from D. Now, if we want the consistent answers to the query about all tuples in the database, i.e. to $Q_1(x, y)$: $Employee(x, y)$?, we obtain: $Cons(Q_1, D, \{KC\}) = \{\langle smith, 3K\rangle, \langle stowe, 7K\rangle\}$. Now, if the query is only about employee names, i.e. $Q_2(x)$: $\exists y Employee(x, y)$?, we obtain: $Cons(Q_2, D, \{KC\}) = \{\langle smith\rangle, \langle stowe\rangle, \langle page\rangle\}$ since we find *page* as employee in each repair. □

We can see that obtaining consistent answers amounts to posing a query to a class of *possible worlds*, in this case, the class of repairs of the original instance. In this sense, the consistent answers correspond to a form of *certain answers* [74]. This notion, defined in model-theoretic terms (through the class of repairs) captures the initial intuition, but may be impractical for the computation of consistent answers: We may want to avoid computing all repairs, materialize them, and posing the same query to each of them, to finally collect the answers in common. Actually, it is easy to produce examples of databases that have exponentially many repairs in the size of the database. To start, we might try out the rewriting-based approach.

Example 3.4. (example 3.3 cont.) The key constraint KC can be rewritten into a sentence of first-order (FO) predicate logic as

$$\forall x \forall y \forall z (Employee(x, y) \wedge Employee(x, z) \rightarrow y = z),$$

and in clausal form as

$$\neg Employee(x, y) \vee \neg Employee(x, z) \vee y = z. \quad (5)$$

Now, if query Q_1 is resolved with (5), we obtain a residue to be appended to the query, obtaining the rewritten query:

$$Q_1'(x, y): \ Employee(x, y) \wedge \neg \exists z (Employee(x, z) \wedge z \neq y), \quad (6)$$

which asks about employees with their salaries for which there is no other employee with the same name, but a different salary. This is a query written in a FO language, and then easy to express and answer from a database. If it is posed to the original instance, the ordinary, classical answers to this query from D turn out to be the expected consistent answers to the original query, i.e. those in $Cons(Q_1, D, \{KC\})$. Notice that this rewriting amounts to rewriting the original SQL query:

```
SELECT Name, Salary
FROM   Employee;
```

into the new SQL query:

```
SELECT Name, Salary FROM Employee
WHERE NOT EXISTS (  SELECT *  FROM Employee E
    WHERE E.Name = Name AND E.Salary <> Salary);
```

which is posed to- and answered from the original instance as usual. □

3.2 Complexity of CQA and approximation

The correctness and termination of the rewriting approach was investigated in the Pods'99 paper [3], and positive cases were identified. (Termination because there could be interacting ICs which could give rise to a cycle of rewritings.) However, there was no claim of universal applicability of the FO rewriting approach when attempted on FO queries and FO ICs. Actually, no long after the appearance of [3], it became clear that the data complexity of consistent query answering (CQA) was bound to be higher than polynomial time. For example, there are Boolean conjunctive queries (BCQs) and functional dependencies (FDs), for which deciding if the query is consistently true, i.e. true in all S-repairs, is *coNP*-complete (in data complexity) [48]. For some BCQs and combinations of FDs and inclusion dependencies, CQA becomes Π_2^P-complete (again, in data complexity, as all the complexity results mentioned in this article) for repairs allowing only tuple deletions [48], and general S-repairs [25].

These complexity results made it clear that the FO rewriting approach had limitations, but its scope remained open. Actually, not long after the emergence of those first complexity results [48, 64], a thorough investigation of the FO rewriting approach and the data complexity of CQA for CQs under key constraints and FDs was started by Jef Wijsen and collaborators. This research program has provided a clear picture of the syntactic cases where FO rewritings are possible, when queries can be consistently answered in polynomial time (but may not be FO rewritable), and when CQA for them becomes intractable (cf. [77, 78, 109] for details and references).

Algorithmic and complexity research on repairs and CQA took many interesting directions. Among others, the following problems have been investigated: Complexity of CQA for aggregate queries under FDs [5], repair checking [1, 48], counting repairs [90], enumerating repairs [84], computing a particular repair [85], etc. Combined complexity results on repair checking and CQA are reported in [9].

Given the high worst-case complexity of CQA, research has also been conducted on tractable approximations to CQA [65, 69–71]. This is a promising line of research that will lead to interesting applications.

3.3 Answer-set programming approaches

In 1999 I invited Michael Kifer to the PUC in Chile. He had written a paper on a very nice multi-valued FO logic with annotations for truth-values structured as a lattice [75]. Since it could be used to do non-trivial reasoning in the presence of inconsistency, and again with the participation of Marcelo Arenas, we attempted to represent in logical terms both repairs and consistent query answers. This effort gave rise to an elegant representation [4], but it was not exploited in algorithmic terms. Short after that, when Pablo Barcelo, student of Engineering at the PUC then, asked me for a research topic for his masters thesis, I proposed to him to develop logic programs with annotations that could be used to both model repairs and do actual computations.[3] All this was inspired by the just-mentioned annotated logic approach and previous results on the use of logic programming for database repairs [6]. This was a successful undertaking that took the form of *answer-set programs* [33] (or logic programs with stable model semantics [67]) with annotations [10, 11]. The resulting *repair programs* are quite simple and general in terms of the ICs that can be handled. The stable models of a program turn out to be the repairs of the given instance, and CQA becomes certain query answering from the program, for which reasoning can be applied. It is worth mentioning that repairs programs and the problems related to repairs and CQA have the same complexity. That is, repair programs have exactly the required expressive power for the task.

Example 3.5. The following is a *denial constraint*, i.e. it prohibits combinations or joins of database atoms:

$$\kappa\colon\ \neg\exists x\exists y(S(x) \wedge R(x,y) \wedge S(y)).$$

The following database instance D violates κ.

R	A	B
ι_1	a_4	a_3
ι_2	a_2	a_1
ι_3	a_3	a_3

S	A
ι_4	a_4
ι_5	a_2
ι_6	a_3

Here, we are introducing, in an unessential manner, global tuple identifiers (tids), which will be useful later on to refer to individual tuples. Those tids will appear in predicates as well, in the first argument and separated from the rest by a semicolon.

Instance D has three S-repairs: $D_1 = \{R(\iota_1; a_4, a_3), R(\iota_2; a_2, a_1), R(\iota_3; a_3, a_3), S(\iota_4; a_4), S(\iota_5; a_2)\}$, $D_2 = \{R(\iota_2; a_2, a_1), S(\iota_4; a_4), S(\iota_5; a_2), S(\iota_6; a_3)\}$, and $D_3 = \{R(\iota_1; a_4, a_3), R(\iota_2; a_2, a_1), S(\iota_5; a_2), S(\iota_6; a_3)\}$.

The repair program contains the atoms in D (with tids) as facts, and the rules:

$$S'(t_1; x, \mathrm{d}) \vee R'(t_2; x, y, \mathrm{d}) \vee S'(t_3; y, \mathrm{d}) \leftarrow S(t_1; x), R(t_2; x, y), S(t_3; y).$$

$$S'(t; x, \mathrm{s}) \leftarrow S(t; x),\ \mathit{not}\ S'(t; x, \mathrm{d}).$$

$$R'(t; x, y, \mathrm{s}) \leftarrow R(t; x, y),\ \mathit{not}\ R'(t; x, y, \mathrm{d}).$$

Here, $t_1, \ldots, t$ are variables. The annotation d is constant denoting that the tuple is deleted from the original database, and annotation constant s denotes that the tuple stays in the repair. Here, the first rule captures in its body (i.e. antecedent) a violation of κ, and the head (i.e. the consequent) offers all the alternative tuples deletions that can solve that violation. The last two rules basically represent inertia: the repairs keep the original tuples that have not been deleted. Predicates R' and S' are nicknames for R and S, and have an extra argument to accommodate the annotation.

This repair-program has three stable models, with repair D_1 corresponding to the model $M_1 = \{R'(\iota_1; a_4, a_3, \mathrm{s}), R'(\iota_2; a_2, a_1, \mathrm{s}), R'(\iota_3; a_3, a_3, \mathrm{s}), S'(\iota_4; a_4, \mathrm{s}), S'(\iota_5; a_2, \mathrm{s}), S'(\iota_6; a_3, \mathrm{d})\} \cup D$, in the sense that D_1 is read off from M_1 by keeping only the tuples annotated with s. □

This is a simple example, but hopefully conveys the gist. If there are interacting ICs, i.e. that repair actions for one of them may affect satisfaction of another one, it is necessary to use a couple of extra annotations to capture a transition process (cf. the cited papers and [14] for more examples). The important points to keep in mind are: (a) there is a one-to-one correspondence between S-repairs and stable models; (b) the minimality of repairs is captured by the minimality of stable models; and (c) doing CQA becomes reasoning with the repair program. In [43] it is reported on *ConsEx*, a system for CQA based on repair programs that run on top of the answer-set programming system *DLV* [82]. *ConsEx*, for "consistency extractor", uses magic-sets for query optimization.

Repair programs have been used beyond the strict domain of CQA, as we will see in Section 7. Other independent approaches to repairs and CQA using answer-set programs are found in [6, 55, 68]. As a final remark, it is worth mentioning that repair programs as those in Example 3.5 for DCs can be transformed into non-disjunctive, unstratified programs [43]; and repair programs in general can be transformed into theories written in second-order (SO) predicate logic, and in some cases, via elimination of SO quantifiers, into FO theories [13]. This is achieved by taking advantage of a compilation of logic programs with stable model semantics into circumscriptive theories [61].

4 OTHER KINDS OF REPAIRS

As we saw in Example 2.1, inconsistencies wrt. inclusion dependencies can be solved by inserting or deleting tuples. However, we might want to accept just deletions, as done in [48]. Even more, we may not want to restrict ourselves to repairs based on minimal sets of insertions and/or deletions of tuples. In fact, other kinds of repairs have been proposed and investigated. In this direction, different *repair semantics* differ at least in two aspects: (a) the admissible repair actions; and (b) the minimality criterion, i.e. what are the consistent instances that are the "closest" to the original one. Several abstract repair semantics based on optimality criteria have been introduced and investigated in [103], with complexity results reported in [57].

Repairs based on changes of attribute values, a.k.a. "attribute-based repairs", have also been investigated and applied. For this, values for these updates can be taken from the data domain [63, 108]. Special mention deserve attribute-based repairs of databases with numerical values, numerical queries, and subject to numerical constraints [12, 20, 62]. That scenario opens completely new research challenges. In the rest of this section we give some more details on a few specific repairs semantics.

[3] Pablo has different, spicer memories of this first encounter.

4.1 Cardinality-repairs

Cardinality-repairs, a.k.a. *C-repairs*, were first considered in [6]. They are S-repairs, as in Section 3.1, that also minimize the *number* of tuples deleted or inserted, i.e. $|D \Delta D'|$. In Example 3.1, D_1 and D_2 are also C-repairs, because both delete a single tuple.

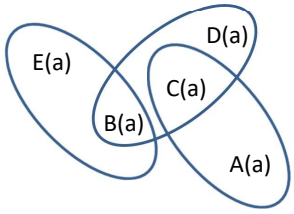


Figure 1: Conflict hyper-graph

Example 4.1. Consider $D = \{A(a), B(a), C(a), D(a), E(a)\}$, and the set of DCs

$$\Sigma = \{\neg\exists x(B(x) \wedge E(x)),\ \neg\exists x(B(x) \wedge C(x) \wedge D(x)),\ \neg\exists x(A(x) \wedge C(x))\}.$$

One can build the so-called *conflict hyper-graph* [5, 48], whose nodes are the database tuples, and hyper-edges connect tuples that simultaneously violate a DC, as shown in Figure 1. Among the S-repairs, which are maximal independent sets in the hyper-graph, we find $D_1 = \{B(a), C(a)\}$, $D_2 = \{C(a), D(a), E(a)\}$, $D_3 = \{A(a), B(a), D(a)\}$, $D_4 = \{E(a), D(a), A(a)\}$. They are all subset-maximal consistent subinstances of D. However, only D_2, D_3 and D_4 are also C-repairs, but not D_1. □

C-repairs and CQA under them were investigated in detail in [87, 88], where complexity results were obtained. This is interesting in its own right, but the complexity bound turned out to be useful to investigate the complexity of causality in databases (cf. Section 7). The complexity of computational problems related to C-repairs tends to be higher than for S-repairs. However, C-repairs seem to behave better than S-repairs under updates of the original database [87]. (By the way, the investigation of repairs and CQA under updates has received little attention; [87] just started to scratch the surface in this direction.) C-repairs were further investigated in [1]. As shown in [6, 15], repair-programs can be produced for the specification of C-repairs and CQA under them. Those programs use *weak program-constraints* [82] to capture the numerical minimization.

Example 4.2. (example 3.5 cont.) The models of the repair program corresponding to C-repairs can be obtained by adding to the program the following weak program constraints (WCs):

$$\Leftarrow R(t, \bar{x}), R'(t, \bar{x}, \mathrm{d}),$$
$$\Leftarrow S(t, \bar{x}), S'(t, \bar{x}, \mathrm{d}).$$

They can be violated by the models (meaning that the rule body becomes true), but the number of violations (value combinations in the rule body) has to be kept to a minimum. The non-minimally violating models are discarded.[4] With these WCs we are minimizing the number of deleted tuples. □

[4] In contrast, a *hard program constraint*, i.e. of the form $\leftarrow Body(\bar{x})$, would eliminate all the models where the body becomes true.

4.2 Null-based repairs: tuple level

Example 4.3. (example 2.1 cont.) Let us modify the table *Articles* in Example 2.1 as follows:

Articles	Item	Cost
	I_1	50
	I_2	30

and the inclusion dependency as follows:

$$ID':\ \forall x \forall y \forall z(Supply(x, y, z) \rightarrow \exists v\ Articles(z, v)), \qquad (7)$$

This is a more general case of inclusion dependency that the one in (1). Both are usually called tuple-generating dependencies (*tgds*). In this case, $D \not\models ID'$ due to the absence of value I_3 in the table *Articles*.

In this case, we may consider two repairs, one just deletes the third tuple from table *Supply*, and another repair inserts the tuple $\langle I_3, \mathsf{NULL}\rangle$ into *Articles*. □

The constant NULL may not have a preassign semantics, and different ones could be considered. However, a natural one is that for the single null in SQL databases. For example, it cannot be used to satisfy joins, to make true comparison atoms, or as values for key attributes, etc. To keep everything in a predicate logic-based setting, the semantics of SQL nulls can be "logically reconstructed", as done in [24] and [25, sec. 4]. (A deeper investigation of the logical foundations of SQL nulls can be found in [71] and references therein.) Other forms of nulls, e.g. multiple, labeled nulls, can be used for the same problem [92].

This repair semantics shown in Example 4.3 was applied in [25] to peer data-exchange systems, where there may be inter-peer mappings of the form (7). Peers exchange data at query-answering time using those mappings, but need to keep their local data consistent. Hence the need for local, peer-level repairs. Those repairs can be specified using repair programs when the local *tgds* are acyclic. (There have been other applications of repairs in peer-data-exchange systems, e.g. [42].)

4.3 Null-based repairs: attribute level

A particularly interesting and natural class of attribute-based repairs relies on changes of attribute values, but only by a null value that behaves as in SQL databases. This kind of repairs is particularly appropriate for DCs.

Example 4.4. (example 3.5 cont.) With the same database D and DC $\kappa:\ \neg\exists x \exists y(S(x) \wedge R(x, y) \wedge S(y))$, repairs can be obtained by "minimally" changing attribute values by a special constant NULL. Since it behaves as in SQL, it cannot be used to satisfy a join. Here we have two attribute-based repairs via NULL replacement; in each of them NULL is preventing the satisfaction of a join in κ:

R	A	B
ι_1	a_4	a_3
ι_2	a_2	a_1
ι_3	a_3	a_3

S	A
ι_4	a_4
ι_5	a_2
ι_6	NULL

R	A	B
ι_1	a_4	NULL
ι_2	a_2	a_1
ι_3	a_3	NULL

S	A
ι_4	a_4
ι_5	a_2
ι_6	a_3

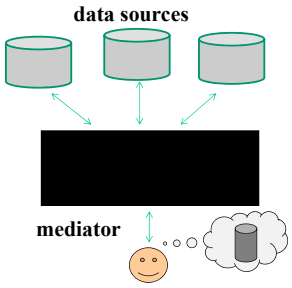
The one on the left-hand side is characterized by the set of changes $\{\iota_6[1]\}$, meaning that the value in the first attribute (or position) of the tuple with tuple ID ι_6 is changed into NULL (the tids use position 0). Similarly, the repair on the right-hand side is characterized by

the set of changes $\{\iota_1[2], \iota_3[2]\}$. Both sets of changes are minimal under set inclusion. □

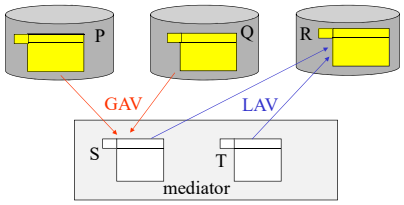
This kind of repairs were used in [24] to hide a sensitive view contents in a database. This is achieved by virtually repairing the database wrt. an IC that specifies that the view has to be empty. Those repairs were also used to define causality for query answers at the attribute level [15] (cf. Section 7). Corresponding repair-programs can be found in [15].

5 VIRTUAL DATA INTEGRATION

In a virtual data integration system, there are independent data sources that are brought together through a *mediator*, which, without materializing data imported from the sources and without being a database, offers a database-like interface, with its own global schema. The mediated schema may be different from those for the sources, which may not even share the same schema. The mediator receives queries from a user and answers them by sending queries back to the sources, collecting the results, integrating them into a query answer.



Crucial components of this architecture are the *mappings* that establish the correspondences between the global schema and the local schemata. This is usually done through view definitions, according to two main paradigms: (a) Global-as-view (GAV), under which predicates at the global level are defined as views over the source relations; and (b) Local-as-view (LAV), under which the local relations (predicates) are defined as views over the global schema. For example, the picture below shows a case of GAV, where the global predicate S is defined as a view over local predicates P and Q. It also shows a case of LAV, where the local predicate R is defined as a view over global predicates T and S. (One usually goes for only one of the two approaches, but we use the picture to illustrate both.)



The semantics of a virtual integration system is given through a collection of admissible global instances [81], i.e. a possible-world semantics. Of course, being a virtual approach, we avoid materializing those global instances.

Example 5.1. The two universities in Ottawa want to virtually integrate their databases, the sources, that are as follows:

Sources: Carleton U.

CUstds	Number	Name
	101	john
	102	mary

SpecCU	Number	Field
	101	alg
	102	ai

Ottawa U.

OUstds	Number	Name
	103	claire
	104	peter

SpecOU	Number	Field
	103	db

We are assuming the tables at the top have the student number as a key, and this constraint is satisfied, but this is not relevant for the moment. The mediator has a single, global relation schema:

$$Stds(Number, Name, Univ, Field).$$

Under GAV, this predicate is defined as a view in terms of source predicates. We use two Datalog rules that define the view:

$$Stds(x, y, \text{'cu'}, z) \leftarrow CUstds(x, y), SpecCU(x, z). \quad (8)$$

$$Stds(x, y, \text{'ou'}, z) \leftarrow OUstds(x, y), SpecOU(x, z). \quad (9)$$

A query posed to the mediator, in a language associated to its global schema, could be about "names of students who study the same field at both universities":

$$Ans(x) \leftarrow Stds(z, x, \text{'cu'}, u), Stds(w, x, \text{'ou'}, u),$$

which can be answered from the sources via unfolding of (8) and (9).

Under LAV, we could define, for example, the first table of Carleton U. as a view over the global $Stds$:

$$CUstds(x, y) \leftarrow Stds(x, y, \text{'cu'}, z). \quad \square$$

Now, what do we do if we want to impose ICs on the global schema? There is no material instance on which to check or enforce those global ICs, and still we would like to see them satisfied, but when and where? The only sensible answer seems to be at the level of global query answering. This is a perfect, if not unavoidable, scenario for CQA.

Example 5.2. (example 5.1 cont.) Let us assume that we have now the following table for Ottawa U.

OUstds	Number	Name
	103	claire
	104	peter
	101	**sue**

It satisfies its local key constraint. However, if we want to impose the global FD: $Stds: Number \rightarrow Name$, there will be problems with the student number 101, that is shared by the two universities, but for different students. The mediator, who may be aware of the global IC, could only check the sources through queries (often of a limited kind), but it cannot update the sources. So, something along the lines of CQA has to be done. For example, the global query about students numbers and names, i.e. $Q(x, y)$: $\exists u \exists z Stds(x, y, u, z)$, could be consistently answered via the rewriting into $Q'(x, y)$: $\exists u \exists z Stds(x, y, u, z) \wedge \neg \exists y' u' z' (Stds(x, y', u', z') \wedge y \neq y' \wedge u \neq u')$, where we are using the fact that the local key constraints are satisfied (and the mediator knows). □

Repairs and CQA under virtual data integration were investigated in several papers [19, 32, 37–39]. The example above is simple and can be handled via a FO rewriting. In other cases, more expressive rewriting languages are needed, such as that of answer-set

programs that specify the virtual repairs of the already virtual global admissible instances [19].

6 DATA CLEANING

The notion of repair naturally appears in data cleaning and data quality assessment. After all, consistency is one of the dimensions of data quality, together with completeness, accuracy, freshness, etc. Instead of ICs, one can use formalized contexts to express data quality concerns; and in this way the context acts as semantic information on the database at hand. If the quality restrictions are not satisfied, then one can, possibly virtually, repair the database. The quality data is, as before, persistent under the repairs, and can be extracted as *quality answers* to queries (the natural generalization of the notion of consistent answer). This is exactly the approach developed in [22, 23].

Conditional functional dependencies (CFDs) were introduced in [58] as extensions of FDs, in order to capture common data quality concerns for which FDs alone are not expressive enough. Consider, for example, a database D containing the following table (taken from [58]):

CC	AC	Phone	Name	Street	City	Zip
44	131	1234567	mike	mayfield	NYC	EH4 8LE
44	131	3456789	rick	crichton	NYC	EH4 8LE
01	908	3456789	joe	mtn ave	NYC	07974

The two following FDs

$$[CC, AC, Phone] \rightarrow [Street, City, Zip]$$
$$[CC, AC] \rightarrow [City]$$

are satisfied by D. They are "global" ICs that may not capture natural data quality requirements, e.g. as related to specific data values. Instead, the CFD

$$[CC = 44, Zip] \rightarrow [Street],$$

expressing a FD of *Street* upon *Zip when* the country code is 44, is not satisfied anymore, and data cleaning may be necessary. CQA and database repairs for CFDs have been investigated in [76, 111]. Cf. [60, 98] for more connections between data quality and database repairs.

The original paper on repairs and CQA was not meant to be a contribution to data cleaning, but rather a scientific undertaking that would lead to model and compute what are the consistent data in an inconsistent database. Nevertheless, repairs still found a way into data cleaning, and have been used in many classical data cleaning problems [28, 31, 111], in particular, in entity resolution (deduplication, record-matching) with entity-linking dependencies [28, 34, 35], and the combination of entity resolution and repairs wrt. classical ICs [59, 66].

In general terms, the notion of repair has been relevant to data cleaning in that it enabled the characterization of what is a good and legal solution to a data cleaning problem, as represented by a new, clean instance or a class thereof. Maybe working with all admissible repairs is too complex, but one can think of choosing one or some of them, and also of weakening the notion of certain answer behind CQA, by considering what is true in most repairs, or introducing probability distributions, etc. Recent application of repairs to data cleaning with probabilistic elements can be found in [52, 98].

7 A CAUSALITY CONNECTION

In data management, we need to understand and compute why certain results are obtained or not, e.g. query answers or violations of semantic conditions. A database system could provide some *explanations*. The notion of *causality* can be used in this direction.

A notion of causality-based explanation for a query result was introduced in [91], as follows. Given a relational instance D and a Boolean conjunctive query (BCQ) Q, a tuple $\tau \in D$ is a *counterfactual cause* for Q if $D \models Q$ and $D \setminus \{\tau\} \not\models Q$. Next, a tuple $\tau \in D$ is an *actual cause* for Q if there is a *contingency set* $\Gamma \subseteq D$, such that τ is a counterfactual cause for Q in $D \setminus \Gamma$. This notion is based on [72]. The strength of an actual cause is commonly captured by the notion of *responsibility*: The responsibility of an actual cause τ for Q is defined by $\rho_D^Q(\tau) := \frac{1}{|\Gamma|+1}$, where $|\Gamma|$ is the smallest size of a contingency set for τ (and 0, otherwise). High responsibility tuples provide more interesting explanations [47].

Example 7.1. (example 3.5 cont.) Consider the same database D. Let us ignore the tids in this example. The query

$$Q: \exists x \exists y (S(x) \wedge R(x, y) \wedge S(y))$$

is satisfied by D: $D \models Q$. Tuple $S(a_3)$ is one of the causes for Q to be true in D, actually a counterfactual cause: If $S(a_3)$ is removed from D, Q is no longer true. Its responsibility is $\frac{1}{1+|\emptyset|} = 1$.

Tuple $R(a_4, a_3)$ is an actual cause for Q with contingency set $\{R(a_3, a_3)\}$: If $R(a_3, a_3)$ is removed from D, Q is still true, but further removing $R(a_4, a_3)$ makes Q false. Its responsibility is $\frac{1}{1+1} = \frac{1}{2}$ (its smallest contingency sets have size 1). Similarly, $R(a_3, a_3)$ and $S(a_4)$ are actual causes, with responsibility $\frac{1}{2}$. □

There are several computational problems related to causality in databases, among them: (a) Computing causes, and deciding if a tuple is a cause; (b) computing responsibilities; (c) computing most responsible actual causes (MRAC); (d) deciding if a tuple has responsibility above a threshold. By now there is a relatively clear picture of the complexity of these problems for CQs and unions thereof (UCQs). Some of these complexity and algorithmic results were obtained in [26] through connections of causality to database repairs and to *consistency-based diagnosis* [95].

Consider a BCQ: $Q: \exists \bar{x}(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$, and assume that Q is true in D. It turns out that the causes for Q to be true in D and their contingency sets can be obtained from database repairs. In fact, the negation of the query, $\neg Q$, is logically equivalent to a DC $\kappa(Q)$: $\neg \exists \bar{x}(P_1(\bar{x}_1) \wedge \cdots \wedge P_m(\bar{x}_m))$; and Q holds in D iff D is inconsistent wrt. $\kappa(Q)$. The S-repairs of D wrt. $\kappa(Q)$ are associated to causes and their minimal contingency sets; and C-repairs are associated to causes, minimum contingency sets, and maximum responsibilities: A database tuple τ is an actual cause with subset-minimal contingency set Γ iff $D \setminus (\Gamma \cup \{\tau\})$ is an S-repair, in which case its responsibility is $\frac{1}{1+|\Gamma|}$. Furthermore, τ is an actual cause with minimum-cardinality contingency set Γ iff $D \setminus (\Gamma \cup \{\tau\})$ is a C-repair, in which case, τ is an MRAC. (Conversely, repairs can be obtained from causes and their contingency sets.) [26].

Computing/deciding actual causes can be done in polynomial time in data for CQs and UCQs [26, 91]. The definition of causality we just gave can be applied to any monotone query, e.g. to Datalog

queries. It turns out that for them, cause computation can be *NP*-complete, a result that can be obtained via a connection between causality and Datalog abduction [27].

Returning to causes for CQs and their responsibilities, we can take advantage of the connection to C-repairs. It turns out that most computational problems related to repairs, in particular, C-repairs, are provably hard (in data complexity) [87], and the bounds can be applied to causality problems. For example, we can obtain that: (a) deciding if a tuple has responsibility above a certain threshold is *NP*-complete for UCQs (responsibility decision problem); (b) computing $\rho_D(\tau)$ is $FP^{NP(log(n))}$-complete for BCQs (this is the *functional*, non-decision, version of (a)); (c) deciding if a tuple is a most responsible cause is $P^{NP(log(n))}$-complete for BCQs [26].

Since ASPs can be used to specify, compute and query S- and C-repairs, another interesting consequence of the causality/repair connection is the possibility of using repair programs as a basis for causality-related specifications and computations [15].

Example 7.2. (examples 3.5 and 7.1 cont.) Consider the same database D and the DC:

$$\kappa(Q): \ \neg\exists x\exists y(S(x) \wedge R(x,y) \wedge S(y)),$$

associated to the BCQ Q for which we are finding causes. The repair program is an in Example 3.5, and its three stable models (or answer sets) are in correspondence with the S-repairs. Now, cause and responsibility computation becomes query answering on extended repair-programs, as we will show next.

First, causes (and tuples in general) will be represented by their tids. So, if we want to compute causes, we add the query rules to the repair program:

$$\begin{aligned} Ans(t) &\leftarrow R'(t,x,y,\mathrm{d}) \\ Ans(t) &\leftarrow S'(t,x,\mathrm{d}) \end{aligned}$$

The causes become those tuples (with tid) ι that make the auxiliary answer predicate *Ans* true wrt. the repair program plus the query rules, i.e. $\Pi \models_{brave} Ans(\iota)$, where $\models_{brave}$ indicates a "brave" consequence of the program, in the sense that $Ans(\iota)$ becomes true in *some* model of Π.

In order to compute responsibilities, we need to specify and compute contingency sets associated to causes. We introduce a new predicate, $CauCon(t,t')$, standing for "t is actual cause, and t' is a member of the former's contingency set", which is specified as follows: For each pair of predicates P_i, P_j in $\kappa(Q)$, add the rule:

$$CauCon(t,t') \leftarrow P_i'(t,\bar{x}_i,\mathrm{d}),\ P_j'(t',\bar{x}_j,\mathrm{d}),\ t \neq t',$$

which is indicating that t' is deleted together with t. More specifically, in this example:

$$\begin{aligned} CauCon(t,t') &\leftarrow S'(t,x,\mathrm{d}), R'(t',u,v,\mathrm{d}), \\ CauCon(t,t') &\leftarrow S'(t,x,\mathrm{d}), S'(t',u,\mathrm{d}), t \neq t', \\ CauCon(t,t') &\leftarrow R'(t,x,y,\mathrm{d}), S'(t',u,\mathrm{d}), \\ CauCon(t,t') &\leftarrow R'(t,x,y,\mathrm{d}), R'(t',u,v,\mathrm{d}), t \neq t'. \end{aligned}$$

For example, from model M_2, corresponding to repair D_2, we obtain: $CauCon(\iota_1,\iota_3)$ and $CauCon(\iota_3,\iota_1)$, corresponding to the difference $D \smallsetminus D_2 = \{R(a_4,a_3), R(a_3,a_3)\}$

Full contingency sets and responsibilities can be specified and computed with extensions of ASP with numerical and set-aggregation, e.g. with DLV-Complex [40, 41, 56], collecting all individual contingency elements for each cause and nothing more:

$$\begin{aligned} preCon(t,\{t'\}) &\leftarrow CauCon(t,t'), \\ preCon(t,\#union(C,\{t''\})) &\leftarrow CauCon(t,t''), preCon(t,C), \\ &\quad\ not\ \#member(t'',C), \\ Con(t,C) &\leftarrow preCon(t,C), not\ aux(t,C), \\ aux(t,C) &\leftarrow CauCon(t,t'), \#member(t',C). \end{aligned}$$

In order to compute a cause's responsibilities, all we need is the rule: $preresp(t,n) \leftarrow \#count\{t' : CauCon(t,t')\} = n$. To obtain the responsibility for a cause ι, we pose the following query to the extended program, say Π^e: $\Pi^e \models_{brave} preresp(\iota,x)$?, and we keep the minimum value m returned for x (we may get different answers from different models), then $\rho_D^Q(\iota) = \frac{1}{1+m}$.

If we want maximum responsibility causes, we can concentrate on C-repairs, which, as we saw in Example 4.2, can be obtained by adding weak programs constraint to the extended program we have so far. □

It is worth mention that the expressive power and data complexity of ASPs with WCs [51, 82] is exactly what is required for maximum-responsibility computation [26].

7.1 Attribute-Level causes

One can naturally define causes at the attribute level, rather than tuple level, by appealing to attribute-based repairs of the kind introduced in Section 4.3. We illustrate the idea with an example (a detailed treatment can be found in [15]).

Example 7.3. (example 4.4 cont.) Consider the same instance with its attribute-based repairs wrt. the DC κ, whose associated conjunctive query is: $Q: \exists x\exists y(S(x) \wedge R(x,y) \wedge S(y))$. It holds $D \models Q$. The sets of changes $\{\iota_6[1]\}$ and $\{\iota_1[2], \iota_3[2]\}$ associated to the two minimal repairs allow us to define $\iota_6[1]$, the first attribute value of the sixth tuple as a counterfactual cause for Q. Similarly, $\iota_1[2]$ becomes an actual cause with contingency set $\Gamma = \{\iota_3[2]\}$, and the other way around. □

Repair programs for attribute-based repairs can be used as a basis to specify and compute attribute-level causes and their associated responsibilities [15].

7.2 Causality under integrity constraints

The problem of defining and investigating causes for database queries under ICs was addressed in [27]. In this scenario it is assumed that a given set Σ of ICs has to be satisfied. So, we start assuming that $D \models \Sigma$. Next, for τ to be actual cause for an answer $\bar{a}$ to a monotone query $Q(\bar{a})$, the associated contingency set Γ has to satisfy: (a) $D \smallsetminus \Gamma \models \Sigma$; (b) $D \smallsetminus \Gamma \models Q(\bar{a})$; (c) $D \smallsetminus (\Gamma \cup \{\tau\}) \models \Sigma$; and (d) $D \smallsetminus (\Gamma \cup \{\tau\}) \not\models Q(\bar{a})$. The responsibility of a cause τ under Σ, denoted $\rho_D^{Q,\Sigma}$, is defined as before.

Example 7.4. Consider the database instance D below

Dep	*DName*	*TStaff*
ι_1	Computing	John
ι_2	Philosophy	Patrick
ι_3	Math	Kevin

Course	*CName*	*TStaff*	*DName*
ι_4	COM08	John	Computing
ι_5	Math01	Kevin	Math
ι_6	HIST02	Patrick	Philosophy
ι_7	Math08	Eli	Math
ι_8	COM01	John	Computing

Let us first consider the query:

(A) $Q(x)$: $\exists y \exists z(Dep(y, x) \wedge Course(z, x, y))$.

John is an answer, denoted $\langle \text{John} \rangle \in Q(D)$; and the causes are: (a) ι_1, counterfactual; (b) ι_4, with single minimal contingency set $\Gamma_1 = \{\iota_8\}$; (c) ι_8, with single minimal contingency set $\Gamma_2 = \{\iota_4\}$. Then, $\rho_D^{Q(\text{John})}(\iota_1) = 1$, and $\rho_D^{Q(\text{John})}(\iota_4) = \rho_D^{Q(\text{John})}(\iota_8) = \frac{1}{2}$.

Now, let us consider an inclusion dependency:[5]

$$\psi: \quad \forall x \forall y \, (Dep(x, y) \rightarrow \exists u \, Course(u, y, x)),$$

which is satisfied by D.

Under ψ, ι_4 and ι_8 are not actual causes anymore, and ι_1 is still a counterfactual cause, i.e. $\rho_D^{Q(\text{John}),\psi}(\iota_1) = 1$, and $\rho_D^{Q(\text{John}),\psi}(\iota_4) = \rho_D^{Q(\text{John}),\psi}(\iota_8) = 0$.

Consider now the query about the first conjunct in (A):

(B) $Q_1(x)$: $\exists y \, Dep(y, x)$.

John is still an answer, i.e. $\langle \text{John} \rangle \in Q_1(D)$. Under ψ, we obtain for Q_1 the same causes with the same minimal contingency sets as for Q, which is logically equivalent to Q_1 under ψ: $Q \equiv_\psi Q_1$.

And now the query about the second conjunct in (A):

(C) $Q_2(x)$: $\exists y \exists z Course(z, x, y)$.

It holds $\langle \text{John} \rangle \in Q_2(D)$. Without considering ψ, ι_4 and ι_8 are the only actual causes, with contingency sets $\Gamma_1 = \{\iota_8\}$ and $\Gamma_2 = \{\iota_4\}$, resp., that is: $\rho_D^{Q(\text{John})}(\iota_1) = 0$, and $\rho_D^{Q(\text{John})}(\iota_4) = \rho_D^{Q(\text{John})}(\iota_8) = \frac{1}{2}$.

Now under ψ, ι_4 and ι_8 are still actual causes, but we lose their previous contingency sets. Now, the smallest contingency sets are: $\Gamma_3 = \{\iota_8, \iota_1\}$ for ι_4; and $\Gamma_4 = \{\iota_4, \iota_1\}$ for ι_8. With them, the responsibilities decrease: $\rho_D^{Q_2(\text{John}),\psi}(\iota_4) = \rho_D^{Q_2(\text{John}),\psi}(\iota_8) = \frac{1}{3}$. We can see that ι_1 is still not an actual cause, but affects the responsibility of the actual causes. □

We can see that causes and responsibilities are affected by ICs. However, both are preserved under logical equivalence of queries under ICs. More interestingly, without ICs, deciding causality for CQs is tractable, but their presence may make the complexity grow: There are a CQ Q and an inclusion dependency ψ, for which deciding causality is *NP*-complete [27]. Finally, ASPs for computation of causes and responsibilities under ICs can be produced [15].

There is still much to investigate around the connection between database repairs and causality in databases. In a different direction, an alternative notion to that of responsibility was introduced and investigated in [102].

8 OTHER DIRECTIONS

Far from pretending to give a full account of the many other directions in which repairs and CQA have been investigated, we mention just a few of them.

Repairs have been defined and investigated for data warehouses and multidimensional databases [8, 21, 44, 45, 110]; also in spatial databases [93, 99], and temporal databases [50, 93]. In data exchange, the data sent to a target instance may collide with the local target constraints, and repairs may be considered [106]. Notions found in data exchange, such as universal solutions, now applied to repairs, can be found in [105], together with a deep investigation of data complexity of CQA, and some undecidability results (for the latter, see also [7]). Repairs of databases using probabilistic elements and repairs of probabilistic databases [104] have been investigated in [2, 69, 83].

An active area of research is that of formal ontologies, and, in particular, ontology-based data access (OBDA) [94]. In OBDA it is not unlikely that the combination of data, rules and constraints produces inconsistencies. In that case, repairs of- and consistent query answering (CQA) on the inconsistent ontology have to be considered. There are different ways of doing this, in terms of the ontological language (e.g. some Description Logic or a Datalog$^{\pm}$ program class), the kind of repairs considered, and the way CQA is done [29, 30, 53, 54, 79, 80, 89, 100, 107].

As a final note, let me say that it has been fascinating to see how the area has evolved since its inception in 1999. We can see that research on repairs and CQA is still going strong after twenty years. As to the problem we first started thinking about in those early days, that of measuring the degree of inconsistency of a database, it turns out that database repairs can be used as a basis for such a task. We have taken the first steps in that direction [16, 17], but that is a different story.

[5] Tuple deletions do not affect the satisfaction of monotone ICs, such as DCs. Causes and inclusion dependencies were considered without a formalization in [101].

ACKNOWLEDGMENTS

My deep gratitude goes to my coauthors of the Pods'99 paper, Marcelo Arenas and Jan Chomicki. Working with them gave me the best moments in my days at the PUC in Chile. We had a good time and stimulating exchanges of ideas. I am grateful to the several coauthors and former students who have worked with me in this area. Special gratitude goes to all those who have done research in the area of repairs and CQA. This is a community effort after all. I appreciate the support from RelationalAI while writing this paper, and the great human and research environment it provides.

REFERENCES

[1] Afrati, F. and Kolaitis, P. Repair Checking in Inconsistent Databases: Algorithms and Complexity. Proc. ICDT 2009.
[2] Andritsos, P., Fuxman, A. and Miller, R.J. Clean Answers over Dirty Databases: A Probabilistic Approach. Proc. ICDE 2006.
[3] Arenas, M., Bertossi, L. and Chomicki, J. Consistent Query Answers in Inconsistent Databases. Proc. PODS, 1999.
[4] Arenas, M., Bertossi, L. and Kifer, M. Applications of Annotated Predicate Calculus to Querying Inconsistent Databases. Proc. Computational Logic 2000, Springer LNCS 1861, pp. 926-941.
[5] Arenas, M., Bertossi, L., Chomicki, J., He, X., Raghavan, V. and Spinrad, J. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 2003, 296(3):405-434.
[6] Arenas, M., Bertossi, L. and Chomicki, J. Answer Sets for Consistent Query Answers. *Theory and Practice of Logic Programming*, 2003, 3(4&5):393-424.
[7] Arenas, M and Bertossi, L. On the Decidability of Consistent Query Answering. Proc. AMW 2010, CEUR-WS, Vol-619.
[8] Ariyan, S. and Bertossi, L. 2013. A multidimensional data model with subcategories for flexibly capturing summarizability. Proc. SSDBM 2013.
[9] Arming, S., Pichler, R. and Sallinger, E. Complexity of Repair Checking and Consistent Query Answering. Proc. ICDT 2016.
[10] Barcelo, P. and Bertossi, L. Logic Programs for Querying Inconsistent Databases. Proc. PADL 2003, Springer LNCS 2562.
[11] Barcelo, P., Bertossi, L. and Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In *Semantics in Databases*, Springer LNCS 2582, 2003, pp. 7-33.
[12] Bertossi, L. Consistent Query Answering in Databases. *ACM Sigmod Record*, 2006, 35(2):68-76.
[13] Bertossi, L. From Database Repair Programs to Consistent Query Answering in Classical Logic. Proc. AMW 2009, CEUR-WS, Vol-450.
[14] Bertossi, L. *Database Repairing and Consistent Query Answering*. Morgan & Claypool, Synthesis Lectures on Data Management, 2011.

[15] Bertossi, L. Characterizing Causes for Query Answers in Databases via Database Repairs and their Computation through Repair Programs. Corr Arxiv Paper cs.DB/1712.01001. Revised and extended version of paper in Proc. FoIKs, 2018, Springer LNCS 10833, pp. 55-76.

[16] Bertossi, L. Measuring and Computing Database Inconsistency via Repairs. Proc. SUM 2018, Springer LNAI 11142, pp. 368-372.

[17] Bertossi, L. Repair-Based Degrees of Database Inconsistency. To appear in Proc. LPNMR 2019. Extended version as Corr Arxiv Paper cs.DB/1809.10286, 2018.

[18] Bertossi, L and Chomicki, J. Query Answering in Inconsistent Databases. In *Logics for Emerging Applications of Databases*, Springer, 2003, pp. 43-83.

[19] Bertossi, L. and Bravo, L. Consistent Query Answers in Virtual Data Integration Systems. In *Inconsistency Tolerance*, Springer LNCS 3300, 2005, pp. 42-83.

[20] Bertossi, L., Bravo, L., Franconi, E. and Lopatenko, A. The Complexity and Approximation of Fixing Numerical Attributes in Databases Under Integrity Constraints. *Information Systems*, 2008, 33(4):407-434.

[21] Bertossi, L., Bravo, L. and Caniupan-Marileo, M. Consistent Query Answering in Data Warehouses. Proc. AMW 2009, CEUR-WS, Vol-450.

[22] Bertossi, L., Rizzolo, F. and Lei, J. Data Quality is Context Dependent. Proc. WS on Enabling Real-Time Business Intelligence (BIRTE 2010). Collocated with VLDB 2010. Springer LNBIP 84, 2011, pp. 52-67.

[23] Bertossi, L. and Rizzolo, F. Contexts and Data Quality Assessment. Corr Arxiv Paper cs.DB/1608.04142. Extended version of [22].

[24] Bertossi, L. and Li, L. Achieving Data Privacy through Secrecy Views and Null-Based Virtual Updates. *IEEE Trans. Knowledge and Data Engineering*, 2013, 25(5):987-1000.

[25] Bertossi, L. and Bravo, L. Consistency and Trust in Peer Data Exchange Systems. *Theory and Practice of Logic Programming*, 2017, 17(2):148-204. Extended version as Corr Arxiv Paper cs.DB/1606.01930.

[26] Bertossi, L. and Salimi, B. From Causes for Database Queries to Repairs and Model-Based Diagnosis and Back. *Theory of Computing Systems*, 2017, 61(1):191-232.

[27] Bertossi, L. and Salimi, B. Causes for Query Answers from Databases: Datalog Abduction, View-Updates, and Integrity Constraints. *International Journal of Approximate Reasoning*, 2017, 90:226-252.

[28] Beskales, G., Soliman, M., Ilyas, I. and Ben-David, S. Modeling and Querying Possible Repairs in Duplicate Detection. Proc. VLDB 2009.

[29] Bienvenu, M. On The Complexity of Consistent Query Answering in the Presence of Simple Ontologies. Proc. AAAI 2012.

[30] Bienvenu, M., Bourgaux, C. and Goasdoue, F. Computing and Explaining Query Answers over Inconsistent DL-Lite Knowledge Bases. *J. Artificial Intelligence Research*, 2019, 64:563-644.

[31] Bohannon, P, Flaster, M., Fan, F. and Rastogi, R. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. Proc. SIGMOD 2005.

[32] Bravo, L. and Bertossi, L. Logic Programs for Consistently Querying Data Integration Systems. Proc. IJCAI 2003.

[33] Brewka, G., Eiter, T. and Truszczynski, M. Answer Set Programming at a Glance. *Comm. of the ACM*, 2011, 54(12):93-103.

[34] Burdick, D., Fagin, R., Kolaitis, P., Popa, L. and Tan, W-C. A Declarative Framework for Linking Entities. *ACM Trans. Database Syst.*, 2016, 41(3):17:1-17:38.

[35] Burdick, D., Fagin, R., Kolaitis, P., Popa, L. and Tan, W-C. Expressive Power of Entity-Linking Frameworks. *J. Comput. Syst. Sci.*, 2019, 100:44-69.

[36] Calautti, M., Libkin, L. and Pieris, A. An Operational Approach to Consistent Query Answering. Proc. PODS 2018.

[37] Calì, A., Lembo, D. and Rosati, R. On The Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. Proc. PODS 2003.

[38] Calì, A., Lembo, D. and Rosati, R. Query rewriting and Answering under Constraints in Data Integration Systems. Proc. IJCAI 2003.

[39] Calì, A., Calvanese, D., De Giacomo, G. and Lenzerini, M. Data Integration under Integrity Constraints. *Inf. Syst.*, 2004, 29(2):147-163.

[40] Calimeri, F. Cozza, S. Ianni, G. and Leone, N. Computable Functions in ASP: Theory and Implementation. Proc. ICLP 2008.

[41] Calimeri, F. Cozza, S. Ianni, G. and Leone, N. An ASP System with Functions, Lists, and Sets. Proc. LPNMR 2009.

[42] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M. and Rosati, R. Inconsistency Tolerance in P2P Data Integration: An Epistemic Logic Approach. *Information Systems*, 2008, 33:360Ü384.

[43] Caniupan-Marileo, M. and Bertossi, L. The Consistency Extractor System: Answer Set Programs for Consistent Query Answering in Databases. *Data & Know. Eng.*, 2010, 69(6):545-572.

[44] Caniupan-Marileo, M., Bravo, L. and Hurtado, C. Repairing inconsistent dimensions in data warehouses. *Data & Know. Eng.*, 2012, 79-80:17-39.

[45] Caniupan-Marileo, M., Vaisman, A. and Arredondo, R. Efficient repair of dimension hierarchies under inconsistent reclassification. *Data & Know. Eng.*, 2015, 95:1-22.

[46] Chakravarthy, U. S., Grant, J. and Minker, J. Logic-Based Approach to Semantic Query Optimization. *ACM Trans. Database Syst.*, 1990, 15(2):162-207.

[47] Chockler, H. and Halpern, J. Y. Responsibility and Blame: A Structural-Model Approach. *J. Artif. Intell. Res.*, 2004, 22:93-115.

[48] Chomicki, J. and Marcinkowski, J. Minimal-Change Integrity Maintenance using Tuple Deletions. *Inf. and Comput.*, 2005, 197(1-2):90-121.

[49] Chomicki, J. Consistent Query Answering: Five Easy Pieces. Proc. ICDT 2007.

[50] Chomicki, J. and Wijsen, J. Consistent Query Answering for Atemporal Constraints over Temporal Databases. Proc. TIME 2016.

[51] Dantsin, E., Eiter, T., Gottlob, G. and Voronkov, A. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 2001, 33(3):374-425.

[52] De Sa, C., Ilyas, I., Kimelfeld, B., Re, C. and Rekatsinas, T. A Formal Framework For Probabilistic Unclean Databases. Proc. ICDT 2019.

[53] Du, J., Qi, G. and Shen, Y.-D. Weight-Based Consistent Query Answering over Inconsistent $\mathcal{SHIQ}$ Knowledge Bases. *KAIS*, 2012 34:335-371.

[54] Eiter, T., Fink, M. and Stepanova, D. Computing Repairs of Inconsistent DL-Programs over EL Ontologies. *J. Artif. Intell. Res.*, 2016, 56:463-515.

[55] Eiter, T., Fink, M., Greco, G. and Lembo, D. Repair Localization for Query Answering from Inconsistent Databases. *ACM Trans. Database Syst.*, 2008, 33(2):10:1-10:51.

[56] Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T. and Ielpa, G. Design and Implementation of Aggregate Functions in the DLV System. *Theory and Practice of Logic Programming*, 2008, 8(5-6):545-580.

[57] Fagin, R., Kimelfeld, B. and Kolaitis, P. Dichotomies in the Complexity of Preferred Repairs. Proc. PODS 2015.

[58] Fan, W., Geerts, F., Jia, X. and Kementsietsidis, A. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.*, 2008, 33(2):6:1-6:48.

[59] Fan, W., Ma, S., Tang, N. and Yu, W. Interaction between Record Matching and Data Repairing. *J. Data and Information Quality*, 2014, 4(4):16:1-16:38.

[60] Fan, W. and Geerts, F. *Foundations of Data Quality Management.* Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2012.

[61] Ferraris, P., Lee, J. and Lifschitz, V. Stable Models and Circumscription. *Artif. Intell.*, 2011, 175(1):236-263.

[62] Flesca, S., Furfaro, F. and Parisi, F. Querying and Repairing Inconsistent Numerical Databases. *ACM Trans. Database Syst.*, 2010, 35(2):14:1-14:50.

[63] Franconi, E., Laureti Palma, A., Leone, N., Perri, S. and Scarcello, F. Census Data Repair: a Challenging Application of Disjunctive Logic Programming. Proc. LPAR 2001.

[64] Fuxman, A. and Miller, R. J. First-Order Query Rewriting for Inconsistent Databases. *J. Comput. Syst. Sci.*, 2007, 73(4):610-635.

[65] Geerts, F., Pijcke, F. and Wijsen, J. First-Order under-Approximations of Consistent Query Answers. *Int. J. Approx. Reasoning*, 2017, 83:337-355.

[66] Geerts, F., Mecca, G., Papotti, P. and Santoro, D. The LLUNATIC Data-Cleaning Framework. *PVLDB*, 2013, 6(9):625-636.

[67] Gelfond, M and Vladimir Lifschitz, M. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Comput.*, 1991, 9(3/4):365-386.

[68] Greco, G., Greco, S. and Zumpano, E. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Trans. Knowl. Data Eng.*, 2003, 15(6):1389-1408.

[69] Greco, S. and Molinaro, C. Approximate Probabilistic Query Answering over Inconsistent Databases. Proc. ER 2008.

[70] Greco, S., Molinaro, C. and Trubitsyna, I. Computing Approximate Query Answers over Inconsistent Knowledge Bases. Proc. IJCAI 2018.

[71] Guagliardo, P. and Libkin, L. Correctness of SQL Queries on Databases with Nulls. *SIGMOD Record*, 2017, 46(3):5-16.

[72] Halpern, J. and Pearl, J. Causes and Explanations: A Structural-Model Approach: Part 1. *British J. Philosophy of Science*, 2005, 56:843-887.

[73] Ilyas, I. and Chu, X. *Trends in Cleaning Relational Data: Consistency and Deduplication.* Foundations and Trends in Databases, 2015, 5(4):281-393.

[74] Imielinski, T. and Lipski Jr., W. Incomplete Information in Relational Databases. *J. ACM*, 1984, 31(4):761-791.

[75] Kifer, M. and Lozinskii, E. A Logic for Reasoning with Inconsistency. *J. Autom. Reasoning*, 1992, 9(2):179-215.

[76] Kolahi, S. and Lakshmanan, L. On Approximating Optimum Repairs for Functional Dependency Violations. Proc. ICDT 2009.

[77] Koutris, P. and Wijsen, J. Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. *ACM Trans. Database Syst.*, 2017, 42(2):9:1-9:45.

[78] Koutris, P. and Wijsen, J. Consistent Query Answering for Primary Keys in Logspace. Proc. ICDT 2019.

[79] Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M. and Savo, D. F. Inconsistency-Tolerant Semantics for Description Logics. Proc. RR 2010.

[80] Lembo, D., Lenzerini, M., Rosati, R., Santarelli, V., Domenico, F. and Thorstensen, V. Mapping Repair in Ontology-based Data Access Evolving Systems. Proc. IJCAI, 2017.

[81] Lenzerini, M. Data Integration: A Theoretical Perspective. Proc. PODS 2002.

[82] Leone, N., Pfeifer, G., Faber,W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. The DLV System for Knowledge Representation and Reasoning. *ACM Trans. Comput. Logic.*, 2006, 7(3):499-562.

[83] Lian, X., Chen, L. and Song, S. Consistent Query Answers in Inconsistent Probabilistic Databases. Proc. SIGMOD 2010.
[84] Livshits, E. and Kimelfeld, B. Counting and Enumerating (Preferred) Database Repairs. Proc. PODS 2017.
[85] Livshits, E., Kimelfeld, B. and Roy, S. Computing Optimal Repairs for Functional Dependencies. Proc. PODS 2018.
[86] Lloyd, J. *Foundations of Logic Programming*, Springer, 1987.
[87] Lopatenko, A. and Bertossi, L. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. Proc. ICDT 2007.
[88] Lopatenko, A. and Bertossi, L. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. Corr Arxiv Paper cs.DB/1605.07159. Extended version of [87].
[89] Lukasiewicz, T., Martinez, M. V., Pieris, A. and Simari, G. I. From Classical to Consistent Query Answering under Existential Rules. Proc. AAAI 2015.
[90] Maslowski, D. and Wijsen, J. A Dichotomy in the Complexity of Counting Database Repairs. *J. Comput. Syst. Sci.*, 2013, 79(6):958-983.
[91] Meliou, A., Gatterbauer, W., Moore, K. F. and Suciu, D. The Complexity of Causality and Responsibility for Query Answers and Non-Answers. Proc. VLDB 2010.
[92] Molinaro, C. and Greco, S. Polynomial Time Queries over Inconsistent Databases with Functional Dependencies and Foreign Keys. *Data Knowl. Eng.*, 2010, 69(7):709-722.
[93] Parisi, F. and Grant, J. Repairs and Consistent Answers for Inconsistent Probabilistic Spatio-Temporal Databases. Proc. SUM 2014.
[94] Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M. and Rosati, R. Linking Data to Ontologies. *J. Data Semantics*, 2008, 10:133-173.
[95] Reiter, R. A Theory of Diagnosis from First Principles. *Artif. Intell.*, 1987, 32(1):57-95.
[96] Reiter, R. On Specifying Database Updates. *J. Log. Program.*, 1995, 25(1):53-91.
[97] Reiter, R. *Knowledge in Action.* The MIT Press, 2001.
[98] Rekatsinas, T., Chu, X., Ilyas, I. and Re, C. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB*, 2017, 10(11):1190-1201.
[99] Rodriguez, M. A., Bertossi, L. and Caniupan-Marileo, M. Consistent Query Answering under Spatial Semantic Constraints. *Inf. Syst.*, 2013, 38(2):244-263.
[100] Rosati, R. On the Complexity of Dealing with Inconsistency in Description Logic Ontologies. Proc. IJCAI 2011.
[101] Roy, S. and Suciu, D. A Formal Approach to Finding Explanations for Database Queries. Proc. SIGMOD 2014.
[102] Salimi, B., Bertossi, L., Suciu, D. and Van den Broeck, G. Quantifying Causal Effects on Query Answering in Databases. Proc. TaPP 2016.
[103] Staworko, S., Chomicki, J. and Marcinkowski, J. Prioritized Repairing and Consistent Query Answering in Relational Databases. *Ann. Math. Artif. Intell.*, 2012, 64(2-3):209-246.
[104] Suciu, D., Olteanu, D., Re, C. and Koch, C. *Probabilistic Databases.* Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2011.
[105] ten Cate, B., Fontaine, G. and Kolaitis, P. On the Data Complexity of Consistent Query Answering. *Theory Comput. Syst.*, 2015, 57(4):843-891.
[106] ten Cate, B., Halpert, R. and Kolaitis, P. Exchange-Repairs - Managing Inconsistency in Data Exchange. *J. Data Semantics*, 2016, 5(2):77-97.
[107] Troquard, N., Confalonieri, R., Galliani, P., Peñaloza, R., Porello, D. and Kutz, O. Repairing Ontologies via Axiom Weakening. Proc. AAAI 2018.
[108] Wijsen, J. Database Repairing using Updates. *ACM Trans. Database Syst.*, 2005, 30(3):722-768.
[109] Wijsen, J. A Survey of the Data Complexity of Consistent Query Answering under Key Constraints. Proc. FoIKS 2014.
[110] Yaghmaie, M., Bertossi, L. and Ariyan, S. Repair-Oriented Relational Schemas for Multidimensional Databases. Proc. EDBT 2012.
[111] Yakout, M., Elmagarmid, A., Neville, J., Ouzzani, M. and Ilyas, I. Guided Data Repair. PVLDB, 2011, 4(5):279-289.