CONSISTENT QUERY ANSWERING

Recent Developments and Future Directions

Jan Chomicki
Department of Computer Science and Engineering
University at Buffalo, SUNY
Buffalo, NY 12260-2000
USA
chomicki@cse.buffalo.edu

Abstract We summarize here recent research on obtaining consistent information from inconsistent databases. We describe the underlying semantic model and a number of approaches to computing consistent query answers. We conclude by outlining further research directions in this area.

Keywords: Databases, integrity constraints, queries, inconsistency.

Background

We are concerned here with relational databases and knowledgebases. A *database* consists thus of *facts* and *integrity constraints.* A *knowledgebase* can additionally contain *rules* (not necessarily Horn).

A knowledgebase is inconsistent if it implies, in the classical sense, every formula, leading to the trivialization of reasoning. An inconsistent knowledgebase has no model and thus can hardly be viewed as a representation of the real world. However, inconsistency is a common phenomenon in knowledgebases and databases. This apparent paradox can be explained by observing that the stored information is not necessarily *correct* and *complete.*

Inconsistency is often viewed as a *defect,* to be avoided at all costs. Knowledgebases are assumed to be consistent, with their consistency preserved by updates. Moreover, different forms of nonmonotonic reasoning or truth maintenance make sure that inconsistencies do not occur during reasoning.

EXAMPLE 1 *The* Closed World Assumption (CWA) *[49] is a form of nonmonotonic reasoning that infers* $\neg L$ *for every atom L not implied by the knowledgebase. CWA preserves the consistency of the knowledgebase if the latter contains only atomic facts and Horn rules. In the presence of disjunction, however, it is well known that CWA can lead to inconsistencies. Consider the knowledgebase* $p \vee q$. *It does not imply p or q separately, and therefore CWA derives* $\neg p$ *and* $\neg q$. *But* $\{p \vee q, \neg p, \neg q\}$ *is an inconsistent set of formulas. Consequently, weaker versions of CWA were studied [45].*

In relational databases, inconsistencies appear as violations of integrity constraints by the current database instance (a set of facts). This can be viewed as an instance of the more general logical notion of inconsistency, under the Closed World Assumption. Database systems typically *prevent* such violations by cancelling the offending updates to the database. Another possibility is to revise the database by *resolving* inconsistencies.

EXAMPLE 2 *Assume a database contains a fact p and an integrity constraint* $\neg p \vee \neg q$. *Inserting q leads to an integrity violation. A typical DBMS would reject the insertion. Another option is to replace p by* $p \vee q$ *but that requires moving to a richer representational framework in the form of* disjunctive databases [53], *which is beyond the capabilities of real DBMS today.*

However, present-day database applications have to consider a variety of scenarios in which data is not necessarily consistent. Integrity violations may be due to the presence of multiple autonomous data sources. The sources may separately satisfy the constraints, but when they are integrated the constraints may not hold. Moreover, because the sources are autonomous, the violations cannot be simply fixed by removing the data involved in the violations. Integrity constraints may also fail to be enforced for efficiency (e.g., denormalization) or other reasons. Finally, it may often be the case that the consistency of a database is only temporarily violated and further updates or transactions are expected to restore it. In all such cases, traditional approaches to deal with integrity violations fail, and a new approach is required.

Consistent query answers

Bry's approach

Bry [13]was the first to note that the standard notion of query answer needs to be modified in the context of inconsistent databases[1]. He proposed the notion of a *consistent query answer* – a query answer that

is unaffected by integrity violations present in the database. The set of such answers forms a conservative estimate of the reliable information content of a database. Bry's proposal does not require that the database be modified to remove the inconsistencies and thus is suitable to the scenarios, outlined above, in which the removal of inconsistencies is impossible or undesirable.

Bry's definition of consistent query answer is based on provability in minimal logic and expresses the intuition that the part of the database instance involved in an integrity violation should not be involved in the derivation of consistent query answers. This is not quite satisfactory, as one would like to have a semantic, model-theoretic notion of consistent query answer that parallels that of the standard notion of query answer in relational databases. Moreover, the data involved in an integrity violation is not entirely useless and some reliable partial information can be extracted from it.

EXAMPLE 3 *Assume that an instance of the relation* Student *is as follows:*

Name	*Address*
Smith	*Los Angeles*
Smith	*New York*

Assume also that functional dependency $Name \rightarrow Address$ *is given. In Bry's approach, no positive information can be derived from this inconsistent database, although such information is clearly present, for example, we know that there is a student named Smith, and that Smith lives in Los Angeles or New York.*

The crucial observation to address the shortcomings of Bry's approach is as follows: even in the above simple example there is more than one minimal way to restore the consistency of the database. Therefore, *all such ways* should be considered. This brings us to the realm of *belief revision* [29] where minimal change has been extensively studied. Choosing *model-theoretic* revision operators can also yield the desired semantic definition of consistent query answers. These insights form the basis of the approach of Arenas, Bertossi and Chomicki [2], discussed below. The paper [2] provided a framework on which most of the subsequent work in this area is based. We discuss it in detail next.

Repairs and query answers

We assume a fixed relational database schema and a set of integrity constraints IC over this schema. We say that a database instance r is *consistent* if $r \models IC$ in the standard model-theoretic sense (i.e., IC is true in r), and *inconsistent* otherwise.

DEFINITION 4 *Given a database instance* r, *we denote by* $\Sigma(r)$ *the set of formulas* $\{P(\bar{a})|r \models P(\bar{a})\}$, *where the P is a relation name and* $\bar{a}$ *a ground tuple.* $\Sigma(r)$ *is a set of facts corresponding to the instance* r. *The* distance $\Delta(r, r')$ *between instances* r *and* r' *is the symmetric difference:*

$$\Delta(r, r') = (\Sigma(r) - \Sigma(r')) \cup (\Sigma(r') - \Sigma(r)).$$

For the instances r, r', r'', $r' \leq_r r''$ *if* $\Delta(r, r') \subseteq \Delta(r, r'')$, *i.e., if the distance between* r *and* r' *is less than or equal to the distance between* r *and* r''.

DEFINITION 5 *Given database instances* r *and* r', *we say that* r' *is a* repair *of* r *if* r' *is consistent* ($r' \models IC$) *and* r' *is* $\leq_r$-*minimal in the class of consistent database instances.*

EXAMPLE 6 *Suppose the results of an election in which two candidates, Brown and Green are running, are kept in two relations:* BrownVotes *and* GreenVotes.

BrownVotes

County	*Date*	*Tally*
A	*11/07*	*541*
A	*11/11*	*560*
B	*11/07*	*302*

GreenVotes

County	*Date*	*Tally*
A	*11/07*	*653*
A	*11/11*	*780*
B	*11/07*	*101*

Vote tallies in every county should be unique: thus the functional dependency $County \rightarrow Tally$ *should hold in both relations. On the other hand, we may want to keep multiple tallies corresponding to different counts (and recounts). Clearly, both relations will have two repairs, depending on whether the first or the second count for county A is picked. So overall there are 4 repairs of the entire database.*

To define consistent answers to queries defined in some query language, we assume that this language has already a well-defined notion of query answer in a database instance. For example, a k-tuple $(a_1, \ldots, a_k)$ is an answer to a relational calculus query $\phi(x_1, \ldots, x_k)$ in an instance r if $r \models \phi(a_1, \ldots, a_k)$. SQL2 has also a well-defined notion of when a tuple is a query answer. We define by $ans_Q(r)$ the set of all answers to a query Q in an instance r.

DEFINITION 7 [2] *Assume the set of all answers to a query* Q *is a k-ary relation. A k-tuple* $\bar{t}$ *is a* consistent query answer *to* Q *in an instance* r *(in symbols:* $\bar{t} \in cns_Q(r)$*) if for every repair* r' *of* r, $\bar{t} \in ans_Q(r')$.

EXAMPLE 8 *Returning to Example 6, we can see that the only consistent answer to the query:*

```
SELECT *
FROM BrownVotes
```

is the tuple

B	*11/07*	*302*

since the remaining two tuples are in conflict and will not appear in every repair.

Similarly, the only consistent answer to the query

```
SELECT County
FROM BrownVotes
WHERE Tally > 400
```

is A. Note also that the latter answer is not obtained if the conflicting tuples are blocked in the derivation of consistent answers (as in Bry's approach).

From the belief revision point of view, the problem addressed here is that of *revising* the database with the integrity constraints. The definition of repair (Definition 5) follows Winslett's semantics [57]. The notion of consistent query answer (Definition 7) corresponds to the notion of *counterfactual inference* in that semantics. We note, however, that research in belief revision has addressed mostly revising arbitrary prepositional theories with prepositional formulas and focused on the semantics of revised theories. On the other hand, the relational database context requires a first-order approach but assumes a restricted form of the revised theory which is just a set of facts. The latter restriction makes possible an efficient derivation of consistent query answers even for large databases.

Note. A *literal* is of the form $P(x_1, \ldots, x_k)$ (a positive literal) or $\neg P(x_1, \ldots, x_k)$ (a negative literal), where P is a database relation. We will denote by L_i literals and by ϕ a quantifier-free formula containing only built-in predicates. We consider the following classes of integrity constraints:

- *universal* constraints: $\forall\ (L_1 \vee \cdots \vee L_n \vee \phi)$ (*binary* if $n = 2$);
- *denial* constraints: universal constraints with only negative literals;
- *inclusion* dependencies: $\forall\ (L_1 \vee \exists\ L_2)$, where L_1 is a negative and L_2 a positive literal.

Functional dependencies (FDs) are a special case of denial constraints.

EXAMPLE 9 *Consider a relation Student with two attributes Name and Address. The (key) functional dependency* $Name \rightarrow Address$ *can be written as the following denial constraint*;

$$(\forall x)(\forall y)(\forall z)(\neg Student(x, y) \vee \neg Student(x, z) \vee y = z).$$

Computing consistent query answers

We discuss here a number of different mechanisms for computing consistent query answers. Note that Definition 7 suggests that consistent query answers can be computed by evaluating the given query in every repair of the given database. However, this approach is not practical because there may be exponentially many repairs even in very simple cases.

EXAMPLE 10 *Consider the functional dependency* $A \rightarrow B$ *over R and the following family of instances of R, each of which has* $2n$ *tuples (represented as columns):*

A	a_1	a_1	a_2	a_2	$\cdots$	a_n	a_n
B	b_0	b_1	b_0	b_1	$\cdots$	b_0	b_1

Each instance has 2^n *repairs.*

Query rewriting

Query rewriting is based on the following idea: *Given a query Q and a set of integrity constraints, construct a query* Q' *such that for every database instance* r *the set of answers to* Q' *in* r *is equal to the set of consistent answers to Q in* r.

Query rewriting was first proposed in [2] in the context of the domain relational calculus. The approach presented there was based on concepts from semantic query optimization [15], in particular the notion of a *residue.*

Residues are associated with literals of the form $P(\bar{x})$ or $\neg P(\bar{x})$ (where $\bar{x}$ is a vector of different variables of appropriate arity). For each literal $P(\bar{x})$ and each constraint containing $\neg P(\bar{x})$ in its clausal form (possibly after variable renaming), a local residue is obtained by removing $\neg P(\bar{x})$ and the quantifiers for $\bar{x}$ from the (renamed) constraint. For each literal $\neg P(\bar{x})$ and each constraint containing $P(\bar{x})$ in its clausal form (possibly after variable renaming), a local residue is obtained by removing $P(\bar{x})$ and the quantifiers for $\bar{x}$ from the (renamed) constraint. Finally, for

each literal the global residue is computed as the conjunction of all local residues (possibly after normalizing variables).

EXAMPLE 11 *The functional dependency*

$$(\forall x)(\forall y)(\forall z)(\neg Student(x, y) \vee \neg Student(x, z) \vee y = z)$$

produces for $Student(x, y)$ *the following local and global residue*

$$(\forall z)(\neg Student(x, z) \vee y = z)$$

The rewritten query is obtained in several steps. First, for every literal, an expanded version is constructed as the conjunction of this literal and its global residue. Second, the expansion step is iterated by replacing the literals in the residue by their expanded versions, until no changes occur. Finally, the literals in the query are replaced by their final expanded versions.

EXAMPLE 12 *Under the functional dependency*

$$(\forall x)(\forall y)(\forall z)(\neg Student(x, y) \vee \neg Student(x, z) \vee y = z)$$

the query $Student(x, y)$ *is rewritten into*

$$Student(x, y) \wedge (\forall z)(\neg Student(x, z) \vee y = z).$$

In this case, the expansion step is iterated only once.

The above approach is applicable to binary integrity constraints and queries that are conjunctions of literals [2]. For more general queries it is incomplete: The rewritten query does not necessarily produce all consistent answers. For some non-binary constraints, the rewriting may fail to terminate.

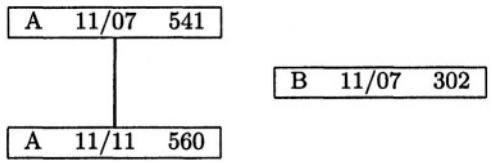

Figure 1. Conflict graph

Clearly, the query-rewriting approach is applicable to SQL queries corresponding to the above class of relational calculus queries.

EXAMPLE 13 *In the database of Example 6, the query*

```
SELECT *
FROM BrownVotes
```

is transformed to:

```
SELECT *
FROM BrownVotes B1
WHERE NOT EXISTS
  SELECT *
  FROM BrownVotes B2
  WHERE B1.County = B2.County
    AND B1.Tally <> B2.Tally.
```

This opens the possibility of using SQL engines for computing consistent query answers. In that way, very large databases can be handled.

Conflict graphs

Although the set of repairs of a database instance may be of exponential size, it can often be compactly represented. For functional dependencies, it is very natural to define the *conflict graph* of the given instance, whose vertices are the tuples in the instance and the edges represent conflicts between tuples. Repairs correspond to maximal independent sets in the conflict graph.

EXAMPLE 14 *The conflict graph of the instance of the relation BrownVotes from Example 6 is represented in Figure 1.*

We show here a nondeterministic algorithm [17, 16] for checking whether *true* is a consistent answer to a ground query Φ in an instance r of a relation P. We assume that the sentence Φ is in CNF, i.e. of the form $\Phi = \Phi_1 \wedge \Phi_2 \wedge \ldots \Phi_l$, where each Φ_i is a disjunction of ground literals. Φ is true in every repair of r if and only if each of the clauses Φ_i is true

in every repair. So it is enough to provide an algorithm that will check if for a given ground clause *true* is a consistent answer.

It is easier to think that we are checking if *true* is **not** a consistent answer. This means that we are checking whether there exists a repair in which $\neg\Phi_i$ is true for some i. But $\neg\Phi_i$ is of the form

$$P(\bar{t}_1) \wedge P(\bar{t}_2) \wedge \ldots \wedge P(\bar{t}_m) \wedge \neg P(\bar{t}_{m+1}) \wedge \ldots \wedge \neg P(\bar{t}_n),$$

where the $\bar{t}_i$'s are tuples of constants. (We assume that all facts in the set $\{P(\bar{t}_1), \ldots, P(\bar{t}_n)\}$ are mutually distinct.)

The nonderministic algorithm selects an edge E_j in the conflict graph for every j, such that $m+1 \leq j \leq n$, $\bar{t}_j \in r$, and $\bar{t}_j \in E_j$, and constructs a set of tuples S, such that

$$S = \{\bar{t}_1, \ldots, \bar{t}_m\} \cup \bigcup_{m+1 \leq j \leq n, \bar{t}_j \in r} (E_j - \{\bar{t}_j\})$$

and *there is no edge* $E \in \mathcal{G}_{F,r}$ *such that* $E \subseteq S$. If the construction of S succeeds, then a repair in which $\neg\Phi_i$ is true can be built by adding to S new tuples from r until the set is maximal independent.

The algorithm is also applicable to denial constraints that generalize functional dependencies. In this case the notion of conflict graph is replaced by that of conflict hypergraph. The algorithm can be extended to deal with nonground queries, too [18]. However, it is still not applicable to queries with quantifiers or general universal constraints.

Typically, the number of conflicts in a database is not large, and thus the conflict graph does not require much space and fits in main memory. In such a case, the above approach is practical even for large databases.

Logic programs

Another approach to computing consistent query answers relies on logical specification of repairs [3, 5]. A logic program (with disjunction and classical negation) Π_1 is constructed on the basis of the given integrity constraints and database instance. Repairs correspond to *answer sets* [30] of this program. A query is also represented using a logic program Π_2 with a distinguished query predicate. The query atoms that belong to every answer set of the program $\Pi_1 \cup \Pi_2$ provide consistent query answers. We demonstrate this construction through an example.

EXAMPLE 15 *Consider Example 3 In this case, the logic program* Π_1 *contains the facts*

$$\mathit{Student}(\text{'Smith'}, \text{'Los Angeles'}).$$
$$\mathit{Student}(\text{'Smith'}, \text{'New York'}).$$

as well as the following rules:

$$\neg Student'(x, y) \vee \neg Student'(x, z) \leftarrow Student(x, y), Student(x, z), y \neq z.$$
$$Student'(x, y) \leftarrow Student(x, y), not\ \neg Student'(x, y).$$
$$\neg Student'(x, y) \leftarrow not\ Student(x, y), not\ Student'(x, y).$$

Here Student' refers to the repaired version of Student. The first rule is responsible for repairing integrity violations. The second and third rules guarantee persistence. The first rule should override the remaining ones. This can be accomplished, for example, using logic programs with exceptions [38]. The first rule will then be a higher-priority exception, and the remaining rules – lower-priority defaults. A logic program with exceptions can be converted into a logic program with disjunction and classical negation [38]. Consider now any relational calculus query Q. Such a query can be converted to a stratified logic program in a standard way.

The approach outlined above is very general as it can handle arbitrary relational calculus queries and binary universal constraints. It has been extended to handle inclusion dependencies under the assumption that null values are allowed in repairs (which slightly changes the semantics of consistent query answers). A similar approach was independently proposed by Greco et al. [34, 33]. That approach can handle arbitrary universal constraints. In [8], another encoding of repairs by logic programs was proposed. That encoding uses additional predicate arguments to represent *annotations,* along the lines of [7]. The resulting program is somewhat simpler because it does not require classical negation (but still uses disjunction). In [54], specification of repairs using ordered logic programs is proposed. In that formulation, disjunctions and multiple versions of the same predicate are not necessary.

Answer sets of logic programs with disjunction and classical negation can be computed using any of the number of systems proposed in the logic programming community: DLV [22] or `smodels` [51]. These implementations can handle only relatively small databases because they work by grounding a logic program and use only main memory.

The paper [23] proposes several optimizations that are applicable to logic programming approaches. One is localization of conflict resolution, another - encoding tuple membership in individual repairs using bitvectors, which makes possible efficient computation of consistent query answers using bitwise operators. However, we have seen in Example 10 even in the presence of one functional dependency there may be *exponentially* many repairs [6]. With only 80 tuples involved in conflicts, the number of repairs may exceed 10^{12}! It is clearly impractical to efficiently manipulate bitvectors of that size.

Aggregation

Aggregation is common in data warehousing applications where inconsistencies are likely to occur. However, in the presence of aggregation operators, the notion of consistent query answer needs to be slightly adjusted.

EXAMPLE 16 *Consider Example 6. The aggregation query*

```
SELECT SUM(Tally)
FROM BrownVotes
```

returns a different answer in every repair. Therefore, it has no consistent answer in the sense of Definition 7. However, it is clear that from the information present in the database we can draw the inference that the returned sum of tallies is not completely unknown but rather falls between 843 *and* 862.

The definition below reflects this intuition.

DEFINITION 17 *[4, 6](a) A consistent answer to an aggregation query Q in a database instance r is the minimal closed interval $I = [a, b]$ such that for every repair r' of r, the scalar value $Q(r')$ of the query Q in r' belongs to I.*
(b) The left and right end-points of the interval I are the greatest lower bound *(glb) and* least upper bound *(lub), respectively, answers to Q in r.*

So in Example 16, the consistent answer is the closed interval [843, 862].

In some cases, consistent answers to aggregation queries can be efficiently computed [4, 6].

EXAMPLE 18 *The consistent answer to the aggregation query*

```
SELECT SUM(Tally)
FROM BrownVotes
```

ic computed by the following query (in the syntax of SQL:1999):

```
WITH Partial(County,MinS,MaxS) AS
   (SELECT County,MIN(Tally),MAX(Tally)
    FROM BrownVotes
    GROUP BY County)

SELECT SUM(MinS) ,SUM(MaxS)
FROM Partial;
```

The computational complexity of consistent answers to first-order and aggregation queries is further discussed below.

Computational complexity

We summarize here the results about the computational complexity of consistent query answers [2, 6, 16, 17]. We adopt the *data complexity* assumption [1, 37, 55] that measures the complexity of the problem as a function of the number of tuples in a given database instance. The given query and integrity constraints are considered fixed.

The query rewriting approach [2] – when it terminates – provides a direct way to establish PTIME-computability of consistent query answers. If the original query is first-order, so is the transformed version. In this way, we obtain a PTIME procedure for computing CQAs: transform the query and evaluate it in the original database. Note that the transformation of the query is done independently of the database instance and therefore, does not affect the data complexity. For example, in Example 10 the query $R(x, y)$ will be transformed (similarly to the query in Example 12) to another first-order query and evaluated in PTIME, despite the presence of an exponential number of repairs. The logic programming approaches described earlier do not have good asymptotic complexity properties because they are all based on Π_2^p-complete classes of logic programs [20].

The paper [16]([17] is an earlier version containing only some of the results) identifies several new tractable classes of queries and constraints. This paper contains the conflict-graph-based algorithm presented earlier, which can easily be shown to work in PTIME. Another tractable class consists of conjunctive queries where all the conjuncts do not share variables and integrity constraints that are functional dependencies, with at most one dependency per relation. The paper [16] also shows that relaxing any of those restrictions leads to co-NP-completeness. Recently, [27] has identified a class of conjunctive queries with relation arity at most 2, for which data complexity of computing consistent query answers is in PTIME, although the answers cannot be computed by query rewriting.

The paper [6] contains a complete classification of tractable and intractable cases of the problem of computing consistent query answers (in the sense of Definition 17) to aggregation queries w.r.t. a set of FDs *F*. Its results can be summarized as follows:

- For all SQL2 aggregate operators except `COUNT(A)`, the problem is in PTIME iff the set of FDs *F* contains at most one nontrivial FD ($|F| \leq 1$).

- For `COUNT(A)`, the problem is NP-complete, even for one nontrivial FD.

- If F is in BCNF and $|F| \leq 2$, then the lub-answer to COUNT (*) queries can be computed in PTIME.

Relaxing any of the above restrictions leads to NP-completeness.

Alternative repair semantics

The definition of repair (Definition 5), while intuitive, is not the only possibly one. It postulates that repairs be obtained by deleting and inserting facts, with deletion and insertion treated symmetrically. If denial constraints are the only constraints present, then only deletions can lead to repairs. However, in the presence of inclusion or tuple-generating dependencies, insertions can also be used to bring about the satisfaction of the constraints.

Several options have been explored in the literature in this context. First, one can ignore insertions and work with deletions only [16]. That is valid if we can assume that the information in the database is not necessarily correct (thus there may be inconsistencies) but it is complete. In fact, referential integrity actions in the SQL:1999 standard are limited to deletions. Second, one can allow deletions only for repairing denial constraints (there is no other way to do it) and use insertions to fix all the other constraints. This is the approach adopted in [14]. It seems suitable to data integration applications where the data is necessarily incomplete. Finally, in some cases it is natural to consider updates of individual attributes, instead of inserting/deleting whole tuples [56].

EXAMPLE 19 *Consider the relation* Emp *with attributes* Name, Salary, *and* Manager, *where* Name *is the primary key. The constraint that* no employee can have a salary greater than that of her manager *is a denial constraint:*

$$\forall n, s, m, s', m'.\ [\neg Emp(n, s, m) \vee \neg Emp(m, s', m') \vee s \leq s'].$$

Consider the following instance of Emp *that violates the constraint:*

Name	Salary	Manager
Jones	*120K*	*Black*
Black	*100K*	*Black*

Under Definition 5, this instance has two repairs: one obtained by deleting the first tuple and the other – by deleting the second tuple. It might be more natural to consider the repairs obtained by adjusting the individual salary values in such a way that the constraint is satisfied.

The first approach leads to finitely many repairs, which guarantees decidability for arbitrary queries and constraints. Some tractable cases

are identified in [16]. In the second approach, there may be infinitely many repairs (in the presence of inclusion dependencies). Some decidable cases and more detailed complexity analysis are presented in [14]. In the third approach, Wijsen proposes to represent all repairs of an instance by a single *trustable tableau.* From this tableau, answers to conjunctive queries can be efficiently obtained. It is not clear, however, what is the computational complexity of constructing the tableau, or even whether the tableau is always of polynomial size.

The minimality criterion constitutes another dimension of the repair definition. The approaches discussed so far minimize the *set* of changes but it is also possible to minimize the *cardinality* of this set. The latter approach has been pursued in the context of belief revision [19]. The paper [5] contains some discussion of how the modified definition of repair can be implemented using a logic-program-based approach.

Future directions

Conflict resolution

The notion of repair can be useful not only in the context of consistent query answering. In some cases, it is necessary to compute a single repair, removing all the integrity violations. For denial constraints, this task can be accomplished in PTIME. However, if inclusion dependencies are added, the task may become co-NP-complete [16].

In general, one would like to use some form of *priority* or *preference* to influence repairing. For example, some piece of information may be more reliable or more up-to-date than another. The issue of computing repairs and consistent query answers in the presence of priorities or preferences is largely open. The work on prioritized logic programming [12, 50] may be relevant in this context.

Data integration and exchange

Assume that we have a collection of (materialized) data sources and a global, virtual database that integrates data from the sources. According to the *local-as-view* approach [42, 46, 52], we can look at the data sources as *views* over the global schema. Now, given a query to the global database, one can generate a *query plan* that extracts the information from the sources [31, 42, 43, 44]. In the global-as-view approach [41], the global database is defined as a view over the data sources.

Sometimes one assumes that certain integrity constraints hold in the global system, and those integrity constraints are used in generating the query plan; actually, there are situations where without integrity con-

straints, no query plan can be generated [21, 32, 35]. The problem is that we can rarely be sure that such global integrity constraints hold. Even in the presence of consistent data sources, a global database that integrates them may become inconsistent. The global integrity constraints are not maintained and could easily be violated. In consequence, data integration is a natural scenario to apply the methodologies presented before. What we have to do is to retrieve consistent information from the global database.

Several new interesting issues appear, among them are: (a) What is a consistent answer in this context? (b) If we are going to base this notion on a notion of repair, what is a repair? Notice that we do not have global *instances* to repair, (c) How can the consistent answers be retrieved from the global systems? What kind of query plans do we need? These and other issues are addressed in [10–11] for the local-as-view approach and in [40] for the global-as-view approach.

Recently, a new kind of scenario for data integration, called *data exchange* [47, 24], has been identified. In this scenario, one considers source and target databases, and the mappings between them are described using source-to-target dependencies that generalize inclusion dependencies. The distinctive feature of this scenario is that an instance of the target database is materialized, using an instance of the source database and the source-to-target dependencies. However, the issue of what to do if, for the given contents of the data sources, there is no target database satisfying the target constraints has not been addressed so far. Such a scenario would clearly require that the construction of the target database be augmented with some form of repairing the inconsistencies.

Data cleaning

The task of data cleaning [36, 48, 28] is to remove errors and inconsistencies in the data submitted to a data warehouse. Clearly, repairing integrity violations is a part of this task. However, there is much more to data cleaning. One has to resolve schematic discrepancies, combine different records describing the same entity, normalize the data etc. Domain knowledge and heuristics play a significant role in data cleaning. Declarative, high-level descriptions of the data cleaning process have been proposed [28]. However, the semantics of the databases resulting from the cleaning process has not been formally characterized so far in the literature.

Other applications

In spatial or spatiotemporal databases inconsistencies arise quite often. For example, there may be inconsistent readings of an object's location or the extent of a forest fire. Spatial or spatiotemporal objects correspond to infinite sets of points. However, those sets are usually finitely representable, e.g., using constraint databases [39]. Fortunately, the notion of repair is applicable not only to finite databases but also to infinite ones. Queries to constraint databases can be rewritten in the same way as those to relational databases, opening the possibility of computing consistent query answers in spatial or spatiotemporal databases.

Relational integrity constraints have been adapted to XML databases [25–26]. However, as XML updates are more complex than relational ones, it is not quite clear how to define repairs and consistent query answers in that context.

Conclusions

We have presented a survey of recent work on consistent query answering. This work has identified relevant expressiveness vs. tractability tradeoffs and proposed a variety of computational approaches. Many semantic and computational issues in this area remain to be explored. New application scenarios promise also to bring a variety of new problems.

A more detailed survey of consistent query answering that contains also an extensive discussion of related work is [9].

Acknowledgments

Thanks go to all those who have collaborated with me in the area of consistent query answering, in particular Marcelo Arenas, Leopoldo Bertossi and Jerzy Marcinkowski. The support of NSF under the grant IIS-0119186 is gratefully acknowledged.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] 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.

[3] M. Arenas, L. Bertossi, and J. Chomicki. Specifying and Querying Database Repairs Using Logic Programs with Exceptions. In *International Conference on Flexible Query Answering Systems (FQAS),* pages 27–41. Springer-Verlag, 2000.

[4] M. Arenas, L. Bertossi, and J. Chomicki. Scalar Aggregation in FD-Inconsistent Databases. In *International Conference on Database Theory (ICDT),* pages 39–53. Springer-Verlag, LNCS 1973, 2001.

[5] 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.

[6] 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.

[7] M. Arenas, L. Bertossi, and M. Kifer. Applications of Annotated Predicate Calculus to Querying Inconsistent Databases. In *International Conference on Computational Logic,* pages 926–941. Springer-Verlag, LNCS 1861, 2000.

[8] P. Barcelo and L. Bertossi. Logic Programs for Querying Inconsistent Databases. In *International Symposium on Practical Aspects of Declarative Languages (PADL),* pages 208–222. Springer-Verlag, LNCS 2562, 2003.

[9] 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.* Springer-Verlag, 2003.

[10] L. Bertossi, J. Chomicki, A. Cortes, and C. Gutierrez. Consistent Answers from Integrated Data Sources. In *International Conference on Flexible Query Answering Systems (FQAS),* Copenhagen, Denmark, October 2002. Springer-Verlag.

[11] L. Bravo and L. Bertossi. Logic Programs for Consistently Querying Data Integration Systems. In *International Joint Conference on Artificial Intelligence (IJCAI),* 2003. To appear.

[12] G. Brewka and T. Eiter. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence,* 109(1-2):297–356, 1999.

[13] F. Bry. Query Answering in Information Systems with Integrity Constraints. In *IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems.* Chapman &Hall, 1997.

[14] A. Cali, 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.

[15] 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.

[16] J. Chomicki and J. Marcinkowski. Minimal-Change Integrity Maintenance Using Tuple Deletions. Technical Report cs.DB/0212004, arXiv.org e-Print archive, December 2002. Under journal submission.

[17] J. Chomicki and J. Marcinkowski. On the Computational Complexity of Consistent Query Answers. Technical Report arXiv:cs.DB/0204010, arXiv.org e-Print archive, April 2002.

[18] J. Chomicki, J. Marcinkowski, and S. Staworko. Computing Consistent Query Answers Using Conflict Hypergraphs. Submitted, September 2003.

[19] M. Dalal. Investigations into a Theory of Knowledge Base Revision. In *National Conference on Artificial Intelligence,* St.Paul, Minnesota, August 1988.

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

[21] O.M. Duschka, M.R. Genesereth, and A.Y. Levy. Recursive Query Plans for Data Integration. *Journal of Logic Programming,* 43(1):49–73, 2000.

[22] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving in DLV. In J. Minker, editor, *Logic-Based Artificial Intelligence,* pages 79–103. Kluwer, 2000.

[23] 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),* 2003.

[24] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *International Conference on Database Theory (ICDT),* pages 207–224. Springer-Verlag, LNCS 2572, 2003.

[25] W. Fan, G. Kuper, and J. Simeon. A Unified Constraint Model for XML. *Computer Networks,* 39(5):489–505, 2002.

[26] W. Fan and J. Simeon. Integrity Constraints for XML. *Journal of Computer and System Sciences,* 66(1):254–201, 2003.

[27] A. Fuxman and R. Miller. Towards Inconsistency Management in Data Integration Systems. In *IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03),* 2003.

[28] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C-A. Saita. Declarative Data Cleaning: Language, Model, and Algorithms. In *International Conference on Very Large Data Bases (VLDB),* pages 371–380, 2001.

[29] P. Gärdenfors and H. Rott. Belief Revision. In D. M. Gabbay, J. Hogger, C, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming,* volume 4, pages 35–132. Oxford University Press, 1995.

[30] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing,* 9(3/4):365–386, 1991.

[31] G. Grahne and A. O. Mendelzon. Tableau Techniques for Querying Information Sources through Global Schemas. In *International Conference on Database Theory (ICDT),* pages 332–347. Springer-Verlag, LNCS 1540, 1999.

[32] J. Grant and J. Minker. A Logic-Based Approach to Data Integration. *Theory and Practice of Logic Programming,* 2(3):323–368, 2002.

[33] G. Greco, S. Greco, and E. Zumpano. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *International Conference on Logic Programming (ICLP),* pages 348–364. Springer-Verlag, LNCS 2237, 2001.

[34] S. Greco and E. Zumpano. Querying Inconsistent Databases. In *International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR),* pages 308–325. Springer-Verlag, LNCS 1955, 2000.

[35] J. Gryz. Query Rewriting using Views in the Presence of Functional and Inclusion Dependencies. *Information Systems,* 24(7):597–612, 1999.

[36] M. Hernandez and S. Stolfo. The Merge/Purge Problem for Large Databases. In *ACM SIGMOD International Conference on Management of Data,* pages 127–138, 1995.

[37] 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.

[38] R. Kowalski and F. Sadri. Logic Programs with Exceptions. *New Generation Computing,* 9(3/4):387–400, 1991.

[39] G. Kuper, L. Libkin, and J. Paredaens, editors. *Constraint Databases.* Springer-Verlag, 2000.

[40] D. Lembo, M. Lenzerini, and R. Rosati. Source Inconsistency and Incompleteness in Data Integration. In *9th International Workshop on Knowledge Representation meets Databases (KRDB'02),* Toulouse, France, 2002.

[41] M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM Symposium on Principles of Database Systems (PODS),* 2002. Invited talk.

[42] A. Y. Levy. Combining Artificial Intelligence and Databases for Data Integration. In *Artificial Intelligence Today,* pages 249–268. Springer-Verlag, LNCS 1600, 1999.

[43] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query-Answering Algorithms for Information Agents. In *National Conference on Artificial Intelligence,* pages 40–47, 1996.

[44] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *International Conference on Very Large Data Bases (VLDB),* pages 251–262, 1996.

[45] J. Minker. On Indefinite Databases and the Closed World Assumption. In *International Conference on Automated Deduction (CADE),* pages 292–308. Springer-Verlag, LNCS 138, 1982.

[46] A. Motro. Multiplex: A Formal Model for Multidatabases and Its Implementation. In *International Workshop on Next Generation Information Technology and Systems (NGITS),* pages 138–158. Springer-Verlag, LNCS 1649, 1999.

[47] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *International Conference on Very Large Data Bases (VLDB),* pages 598–609, 2002.

[48] E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches. *IEEE Data Engineering Bulletin,* 23(4):3–13, 2000.

[49] R. Reiter. On Closed World Databases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases,* pages 55–76. Plenum Press, 1978.

[50] C. Sakama and K. Inoue. Prioritized Logic Programming and its Application to Commonsense Reasoning. *Artificial Intelligence,* 123:185–222, 2000.

[51] P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence,* 138(1-2):181–234, June 2002.

[52] J. D. Ullman. Information Integration Using Logical Views. In *International Conference on Database Theory (ICDT),* pages 19–40. Springer-Verlag, LNCS 1186, 1997.

[53] 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. Kluwer Academic Publishers, Boston, 1998.

[54] D. Van Nieuwenborgh and D. Vermeir. Preferred Answer Sets for Ordered Logic Programs. In *European Conference on Logics for Artificial Intelligence (JELIA),* pages 432–443. Springer-Verlag, LNAI 2424, 2002.

[55] M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing (STOC),* pages 137–146, 1982.

[56] J. Wijsen. Condensed Representation of Database Repairs for Consistent Query Answering. In *International Conference on Database Theory (ICDT),* pages 378–393. Springer-Verlag, LNCS 2572, 2003.

[57] M. Winslett. Reasoning about Action using a Possible Models Approach. In *National Conference on Artificial Intelligence,* 1988.