Consistent Query Answers in Inconsistent Databases

Jan Chomicki
Dept. CSE
University at Buffalo
State University of New York
http://www.cse.buffalo.edu/~chomicki

Joint work with Marcelo Arenas, Leo Bertossi, and Jerzy Marcinkowski, with contributions by Roger He, Vijay Raghavan and Jeremy Spinrad.
Integrity constraints

Integrity constraints describe valid database instances.

Examples:

• functional dependencies: “every student has a single address.”
• denial constraints: “no employee can make more than her manager.”
• referential integrity: “students can enroll only in the offered courses.”
• spatial constraints: “every ship has to be in a body of water.”

The constraints are formulated in first-order logic:

$$\forall n, s, m, s', m'. \neg[Emp(n, s, m) \land Emp(m, s', m') \land s > s'].$$
Inconsistent databases

There are situations when we want/need to live with inconsistent data in a database (data that violates given integrity constraints):

- the consistency of the database will be restored by executing further transactions
- integration of heterogeneous databases with duplicate information
- inconsistency wrt “soft” integrity constraints (those that we hope to see satisfied but do not/cannot check)
- denormalized relations in a data warehouse
- legacy data on which we want to impose semantic constraints
- it is impossible/undesirable to repair the database to restore consistency.

How to distinguish between reliable and unreliable information in an inconsistent database?
Plan of the talk

1. repairs and consistent query answers
2. first-order queries
3. aggregation queries
4. spatial constraints
5. computing consistent query answers
6. related and further work
Consistent query answers [PODS’99]

**Repair:**

- a database that satisfies the integrity constraints
- difference from the given database is minimal (the set of inserted/deleted tuples is minimal under set inclusion)

Typically, more than one repair of a given inconsistent database.

A tuple \((a_1, \ldots, a_n)\) is a **consistent query answer** to a query \(Q(x_1, \ldots, x_n)\) in a database \(r\) if it is an element of the result of \(Q\) in **every repair** of \(r\).
Functional dependency:

\textit{Name County} → \textit{Tally}

<table>
<thead>
<tr>
<th>Name</th>
<th>County</th>
<th>Date</th>
<th>Tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown</td>
<td>A</td>
<td>11/07</td>
<td>541</td>
</tr>
<tr>
<td>Brown</td>
<td>A</td>
<td>11/11</td>
<td>560</td>
</tr>
<tr>
<td>Brown</td>
<td>B</td>
<td>11/07</td>
<td>302</td>
</tr>
<tr>
<td>Green</td>
<td>A</td>
<td>11/07</td>
<td>653</td>
</tr>
<tr>
<td>Green</td>
<td>A</td>
<td>11/11</td>
<td>730</td>
</tr>
<tr>
<td>Green</td>
<td>B</td>
<td>11/07</td>
<td>101</td>
</tr>
</tbody>
</table>

Repairs:

<table>
<thead>
<tr>
<th>Name</th>
<th>County</th>
<th>Date</th>
<th>Tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown</td>
<td>A</td>
<td>11/07</td>
<td>541</td>
</tr>
<tr>
<td>Brown</td>
<td>B</td>
<td>11/07</td>
<td>302</td>
</tr>
<tr>
<td>Green</td>
<td>A</td>
<td>11/07</td>
<td>653</td>
</tr>
<tr>
<td>Green</td>
<td>B</td>
<td>11/07</td>
<td>101</td>
</tr>
<tr>
<td>Brown</td>
<td>A</td>
<td>11/11</td>
<td>560</td>
</tr>
<tr>
<td>Brown</td>
<td>B</td>
<td>11/07</td>
<td>302</td>
</tr>
<tr>
<td>Green</td>
<td>A</td>
<td>11/11</td>
<td>730</td>
</tr>
<tr>
<td>Green</td>
<td>B</td>
<td>11/07</td>
<td>101</td>
</tr>
</tbody>
</table>
Query languages

Ultimately: SQL2.

Now:

- **first-order** queries (equivalently: relational algebra)
- **scalar aggregation** queries.

The definition of consistent query answer may have to be generalized.
Consistent query answers

```
SELECT *  ⇒  Brown  B  11/07  302
FROM Election
WHERE Name = 'Brown'
```

```
SELECT County  ⇒  A
FROM Election
WHERE Name = 'Brown'
   AND Tally > 400
```

```
SELECT SUM(Tally)  ⇒  [843,862]
FROM ELECTION
WHERE Name = 'Brown'
```

A consistent answer to an aggregation query is no longer a single value.
Spatial constraints

A and B are fighting a civil war in a country C and making conflicting claims about the territory occupied by each side.

Integrity constraint: “the areas occupied by A and B form a disjoint partition of the territory of C”.

Infinitely many repairs.

Consistent answer to the query about A’s territory: set difference of A’s and B’s claims.
Computing consistent query answers

**Query transformation**: 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 $r = \text{the set of consistent answers to } Q \text{ in } r$.

**Representing all repairs**: given a set of integrity constraints and a database instance $r$:

1. construct a space-efficient representation of all repairs of $r$
2. use this representation to answer queries.

**Specifying repairs as logic programs.**
There are too many repairs to evaluate the query in each of them.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$b'_1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b'_2$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$a_n$</td>
<td>$b_n$</td>
</tr>
<tr>
<td>$a_n$</td>
<td>$b'_n$</td>
</tr>
</tbody>
</table>

Under the functional dependency $A \rightarrow B$, this instance has $2^n$ repairs.
**Query transformation [PODS’99]**

**First-order queries** transformed using semantic query optimization techniques.

**Residues:**

- associated with single literals $p(\bar{x})$ or $\neg p(\bar{x})$ (only one of each for every database relation $p$)

- for each literal $p(\bar{x})$ and each constraint containing $\neg p(\bar{x})$ in its clausal form (possibly after variable renaming), obtain a local residue 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), obtain a local residue by removing $p(\bar{x})$ and the quantifiers for $\bar{x}$ from the (renamed) constraint

- for each literal, compute the global residue as the conjunction of local residues (possibly after normalizing variables)
The functional dependency

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

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

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

The integrity constraints

$$(\forall x)(\neg p(x) \lor r(x)), (\forall x)(\neg q(x) \lor r(x))$$

produce the following global residues

<table>
<thead>
<tr>
<th>Literal</th>
<th>Residue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(x)$</td>
<td>$r(x)$</td>
</tr>
<tr>
<td>$q(x)$</td>
<td>$r(x)$</td>
</tr>
<tr>
<td>$\neg r(X)$</td>
<td>$\neg p(x) \land \neg q(x)$</td>
</tr>
</tbody>
</table>
Constructing the transformed query

Given a first-order query \( Q \).

**Literal expansion**: for every literal, construct an expanded version as the conjunction of this literal and its global residue.

**Iteration**: the expansion step is iterated by replacing the literals in the residue by their expanded versions, until no changes occur.

**Query expansion**: replace the literals in the query by their final expanded versions.
The functional dependency

$$(\forall x)(\forall y)(\forall z)(\neg \text{Student}(x, y) \lor \neg \text{Student}(x, z) \lor y = z)$$

transforms the query $\text{Student}(x, y)$ into

$$\text{Student}(x, y) \land (\forall z)(\neg \text{Student}(x, z) \lor y = z)$$

For the integrity constraints

$$(\forall x)(\neg p(x) \lor r(x)), (\forall x)(\neg r(x) \lor s(x))$$

<table>
<thead>
<tr>
<th>Literal</th>
<th>Residue</th>
<th>First expansion</th>
<th>Second (final) expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r(x)$</td>
<td>$s(x)$</td>
<td>$r(x) \land s(x)$</td>
<td>$r(x) \land s(x)$</td>
</tr>
<tr>
<td>$p(x)$</td>
<td>$r(x)$</td>
<td>$p(x) \land r(x)$</td>
<td>$p(x) \land r(x) \land s(x)$</td>
</tr>
<tr>
<td>$\neg r(x)$</td>
<td>$\neg p(x)$</td>
<td>$\neg r(x) \land \neg p(x)$</td>
<td>$\neg r(x) \land \neg p(x)$</td>
</tr>
<tr>
<td>$\neg s(x)$</td>
<td>$\neg r(x)$</td>
<td>$\neg s(x) \land \neg r(x)$</td>
<td>$\neg s(x) \land \neg r(x) \land \neg p(x)$</td>
</tr>
</tbody>
</table>
SELECT * 
FROM Election 
WHERE Name = 'Brown'

SELECT * 
FROM Election B1 
WHERE B1.Name = 'Brown' 
AND NOT EXISTS 
(SELECT * 
FROM Election B2 
WHERE B1.County = B2.County 
AND B1.Name = B2.Name 

Query transformation possible for queries involving conjunctions of literals
(relational algebra: selection, join and difference) and binary integrity constraints.
## Data complexity of consistent query answers [submitted]

<table>
<thead>
<tr>
<th>Queries</th>
<th>Functional dependencies</th>
<th>Denial constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>F</td>
<td>= 1$</td>
</tr>
<tr>
<td>$\land, \lor, \neg$</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
<tr>
<td>$\exists$</td>
<td>PTIME</td>
<td>co-NP-complete</td>
</tr>
<tr>
<td>$\exists, \land$</td>
<td>co-NP-complete</td>
<td>co-NP-complete</td>
</tr>
<tr>
<td>(2 literals)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Aggregation queries [ICDT’01]

```
SELECT SUM(Tally)
FROM Election
WHERE Name = 'Brown'

WITH Partial(County, MinS, MaxS) AS
    (SELECT County, MIN(Tally), MAX(Tally)
     FROM Election
     WHERE Name = 'Brown'
     GROUP BY County)

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

But that works only for a single functional dependency and some aggregation operators!
Consistent answers to aggregation queries [ICDT’01]

<table>
<thead>
<tr>
<th></th>
<th>greatest lower bound</th>
<th>least upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
<td>F</td>
</tr>
<tr>
<td>MIN(A)</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
<tr>
<td>MAX(A)</td>
<td>PTIME</td>
<td>NP-complete</td>
</tr>
<tr>
<td>COUNT(*)</td>
<td>PTIME</td>
<td>NP-complete</td>
</tr>
<tr>
<td>COUNT(A)</td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>SUM(A)</td>
<td>PTIME</td>
<td>NP-complete</td>
</tr>
<tr>
<td>AVG(A)</td>
<td>PTIME</td>
<td>NP-complete</td>
</tr>
</tbody>
</table>

How to reduce the computational cost?
Representing all repairs

Not as a formula (as in belief revision) but as a graph.

A set of functional dependencies $F$, a database instance $r$.

Conflict graph:

- nodes: tuples in $r$
- edges: there is an edge $(t_1, t_2)$ if there is a functional dependency $A \rightarrow B \in F$ such that $t_1[A] = t_2[A]$ and $t_1[B] \neq t_2[B]$.
- maximal independent sets: repairs
Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF):

Every functional dependency is a key dependency.

BCNF produces restrictions on the conflict graph that improve tractability.

Property: For one dependency in BCNF, the conflict graph is a union of disjoint cliques.
BCNF and \texttt{COUNT}(*) queries

Two functional dependencies.

Indirect approach:

- conflict graph is \textbf{claw-free} and \textbf{perfect}
- finding maximum independent sets in such graphs can be done in PTIME \((O(n^{5.5}))\)

Direct approach:

- bipartite \textbf{clique graph}:
  - nodes: cliques in 1-dependency conflict graphs
  - edges: nonempty clique intersections
- finding a maximum matching in the clique graph
- overall complexity \(O(n^{1.5})\).
Specifying repairs as logic programs [FQAS’00]

Logic programs with:

- negation in the body and the head
- disjunction
- exceptions (can be eliminated)

Scope:

- arbitrary universal constraints, inclusion dependencies
- arbitrary first-order queries
- queries can be “modalized” and nested

A similar approach has been pursued by Greco and Zumpano [LPAR’00, CODAS’01].
Related work

Belief revision:

- revising database with integrity constraints
- revised theory changes with each database update
- emphasis on semantics (AGM postulates), not computation
- inference of ground literals using theorem proving techniques

Disjunctive information:

- repair $\equiv$ possible world
- using disjunctions to represent resolved conflicts
- constructing a single disjunctive instance
- query languages: representation-specific, relational algebra or calculus
- no tractable classes of aggregation queries
Future work

Broadening scope:

- **SQL:**
  - more general aggregation queries
  - nested subqueries
  - keys and foreign keys
- preferences:
  - source rankings
  - timestamps
- conflict resolution
New paradigms:

- query reformulation in information integration
- data cleaning
- XML

Algorithms and computational complexity:

- efficient algorithms for query processing in special cases
- lower bounds
- approximation
Selected papers:


