Consistent Query Answering

Sławek Staworko

University of Lille
INRIA Mostrare Project

DEIS 2010
November 9, 2010

\(^1\)Some slides are due to [Cho07]
Overview

1. Motivation
2. Basic notions
3. Computing Consistent Query Answers
4. Complexity Results
5. Alternative Semantics
Motivation
# Traditional Databases

**Database instance $D$:**
- a finite first-order **structure**
- represents the information about the world

**Integrity constraints $\Sigma$:**
- first-order logic **formulas**
- express the properties/rules of the world

**Consistent database**
- Formula satisfaction in a first-order structure $D \models \Sigma$
- RDBMS **ensures** consistency
<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermit</td>
<td>Manager</td>
<td>14.03.1965</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>21.06.1976</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>12.04.1946</td>
</tr>
</tbody>
</table>
### Muppet (CBS)

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermit</td>
<td>Manager</td>
<td>14.03.1965</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>21.06.1976</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>12.04.1946</td>
</tr>
</tbody>
</table>

### Muppet (Vanity Fair)

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermit</td>
<td>Manager</td>
<td>14.03.1965</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>01.04.1936</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>18.06.1942</td>
</tr>
</tbody>
</table>

### Muppet (Federated Database)

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermit</td>
<td>Manager</td>
<td>14.03.1965</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>21.06.1976</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>01.04.1950</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>12.04.1946</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>18.06.1942</td>
</tr>
</tbody>
</table>
Inconsistency

Source of Inconsistency

- integration of independent data sources with overlapping data
- time lag of updates (eventual consistency)
- unenforced integrity constraints (denormalized DBs)

Eliminating inconsistency?

- not enough information, time, or money
- difficult, impossible or undesirable
- unnecessary: queries may be insensitive to inconsistency

Living with inconsistency?

- ignoring inconsistency
- modifying the schema
- exceptions to constraints
- redefining query answers
A (young) woman of taste doesn’t look at the price!

### Muppet

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermit</td>
<td>Manager</td>
<td>14.03.1965</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>21.06.1976</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>01.04.1950</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>12.04.1946</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>18.06.1942</td>
</tr>
</tbody>
</table>

Who’s eligible for senior discount?

\[ Q(x) = \exists y, z. Muppet(x, y, z) \land z \leq 9.11.1950 \]

Standard answer semantics is (in)consistency oblivious

\{Miss Piggy, T. Statler\}
Impact of Inconsistency on Queries

Traditional view
- query results defined irrespective of integrity constraints
- integrity constraints may be used to optimize the query

Our view
- inconsistency leads to uncertainty (possible worlds)
- integrity constraints guide the user when formulating her queries
- query results may depend on satisfaction of integrity constraints
- inconsistency may be eliminated (repairing) or tolerated (consistent query answering)
Basic Notions
Restoring Consistency: Two operations

\[ R[A, B] \subseteq P[A, B] \]

\[ r: \begin{array}{c|c} A & B \\ \hline 1 & 2 \end{array} \quad \quad p: \begin{array}{c|c} A & B \\ \hline 1 & 3 \end{array} \]
Restoring Consistency: Two operations

$R[A, B] \subseteq P[A, B]$

Delete a tuple

$r: \begin{array}{c|c}
A & B \\
1 & 2 \\
\end{array}$

$p: \begin{array}{c|c}
A & B \\
1 & 3 \\
\end{array}$

Insert a tuple

$r: \begin{array}{c|c}
A & B \\
1 & 2 \\
\end{array}$

$p: \begin{array}{c|c}
A & B \\
1 & 3 \\
\end{array}$
A consistent instance obtained by performing a *minimal set* of operations.

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermit</td>
<td>Manager</td>
<td>14.03.1965</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>21.06.1976</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>12.04.1946</td>
</tr>
</tbody>
</table>

\[ r_1: \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermit</td>
<td>Manager</td>
<td>14.03.1965</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>21.06.1976</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>18.06.1942</td>
</tr>
</tbody>
</table>

\[ r_2: \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermit</td>
<td>Manager</td>
<td>14.03.1965</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>21.06.1976</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>18.06.1942</td>
</tr>
</tbody>
</table>

\[ r_3: \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermit</td>
<td>Manager</td>
<td>14.03.1965</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>21.06.1976</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>12.04.1946</td>
</tr>
</tbody>
</table>

\[ r_4: \]
Consistent Query Answers

Consistent Query Answer

Query answer present in every repair.

Who’s eligible for senior discount?

$Q(x) = \exists y, z. \ Muppet(x, y, z) \land z \leq 9.11.1950$

Consistent Answers to $Q(x)$

- T. Statler is a consistent answer to $Q(x)$
- Miss Piggy is not a consistent answer to $Q(x)$ because of $r_2$ and $r_3$

CQA scientifically proven to make you feel much younger!
Naïve Data Cleansing

How about removing all conflicting data?

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>DoB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kermit</td>
<td>Manager</td>
<td>14.03.1965</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>21.06.1976</td>
</tr>
<tr>
<td>Miss Piggy</td>
<td>Diva</td>
<td>01.04.1950</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>12.04.1946</td>
</tr>
<tr>
<td>T. Statler</td>
<td>Old Man</td>
<td>18.06.1942</td>
</tr>
</tbody>
</table>

\[ Q(x) = \exists y, z. Muppet(x, y, z) \land z \leq 9.11.1950 \]

The set of answers to \( Q(x) \) in \( r_0 \) is empty

Radical approaches lead to information loss.
Computing Consistent Query Answers
Warning: Exponentially Many Repairs

There are $2^n$ repairs of this instance w.r.t. the FD $A \rightarrow B$.

It is impractical to apply the definition of CQA directly.
Computing Consistent Query Answers

Query Rewriting

Given a query $Q$ and a set of integrity constraints $\Sigma$, build a query $Q^\Sigma$ such that

$$\text{answers to } Q^\Sigma \text{ in } D = \text{consistent answers to } Q \text{ in } D \text{ w.r.t. } \Sigma$$

for every database $D$.

Representing all repairs

Given a database $D$ and a set of integrity constraints $\Sigma$

1. build a compact representation of all repairs of $D \text{ w.r.t. } \Sigma$
2. use it to compute the consistent answers

Logic programs

Given a database $D$, a set of integrity constraints $\Sigma$, and a query $Q$

1. build a logic program $P_{\Sigma,D}$ whose models represent repairs of $D \text{ w.r.t. } \Sigma$
2. build a logic program $P_Q$ expressing $Q$
3. use a LP system (Smodels, dlv) with cautious evaluation semantics to find answers present in all repairs.
**Query Rewriting Example**

**Database Schema**

\[\text{Muppet}(\text{Name}, \text{Role}, \text{DoB}) \text{ with } \text{Muppet} : \text{Name} \rightarrow \text{Role DoB}\]

**Query**

\[\exists y, z. \text{Muppet}(x, y, z) \land z \leq 9.11.1950\]

**Integrity constraint \text{Muppet} : \text{Name} \rightarrow \text{Role DoB}**

\[\forall x, y, z, y', z'. \neg \text{Muppet}(x, y, z) \lor \neg \text{Muppet}(x, y', z') \lor (y = y' \land z = z')\]

**Rewritten query**

\[\exists y, z. \text{Muppet}(x, y, z) \land z \leq 9.11.1950 \land \not \exists x', y'. \text{Muppet}(x, y', z') \land z' > 9.11.1950\]
Milestones in Query Rewriting

- Arenas, Bertrossi, Chomicki [ABC99]
  - binary universal constraints (includes FDs and full INDs)
  - quantifier-free conjunctive queries

- Fuxman, Miler [FM07]
  - primary key dependencies
  - a class of conjunctive queries $C_{forest}$
    - no cycles (join graph is a forest)
    - no non-key or non-full joins
    - no repeated relation symbols
    - no built-ins

- Wijsen [Wij10]
  - primary key dependencies
  - a class of conjunctive queries $C_{rooted}$
    - semantic definition
    - syntactic (effective) characterization that is:
      - based on a notion of an attack graph
      - sound for conjunctive queries without self-join
      - complete for acyclic conjunctive queries without self-join
Rewriting SQL Queries

### SQL query

SELECT Name FROM Muppet
WHERE DoB ≤ '9.11.1950'

### SQL rewritten query

SELECT m1.Name FROM Muppet m1
WHERE m1.DoB ≤ '9.11.1950' AND NOT EXISTS
   (SELECT * FROM Muppet m2
    WHERE m2.Name = m1.Name AND m2.DoB > '9.11.1950')

(Fuxman, Fazli, Miller [FFM05])

- **ConQuer**: a system for computing CQAs
- conjunctive ($C_{forest}$) and aggregation SQL queries
- databases can be annotated with consistency indicators
- tested on TPC-H queries and medium-size databases
Conflict Hypergraph

Conflict Graph (Arenas et al. [ABC+03b])

- **Vertex**: tuple in the database
- **Edge**: two conflicting tuples
- **Repair**: is a maximal independent set

- (Kermit, 14.03.1965) — (Piggy, 21.06.1976) — (Piggy, 01.04.1950)
- (T. Statler, 12.04.1946) — (T. Statler, 18.06.1942)

**Extensions**

- **Conflict Hypergraph** for denial constraints: hyperedges span on sets of tuples (Chomicki, Marcinkowski)[CM05]
- Extended **Conflict Hypergraph** for universal constraints: hyperedges may contain tuples to be added (S., Chomicki, SC10)
Conflict Hypergraph

Conflict Graph (Arenas et al. [ABC+03b])

- **Vertex**: tuple in the database
- **Edge**: two conflicting tuples
- **Repair**: is a maximal independent set

- (Kermit, 14.03.1965)  
- (T. Statler, 12.04.1946)
- (T. Statler, 18.06.1942)
- (Piggy, 21.06.1976)
- (Piggy, 01.04.1950)
- (Piggy, 09.01.1990)

Extentions

- **Conflict Hypergraph** for denial constraints: hyperedges span on sets of tuples (Chomicki, Marcinkowski) [CM05]
- **Extended Conflict Hypergraph** for universal constraints: hyperedges may contain tuples to be added (S., Chomicki [SC10])
Computing CQAs Using Conflict Hypergraphs

Algorithm HProver

**Input:** $\Phi$ a disjunction of ground literals, conflict hypergraph $G$ of $I$ w.r.t. $\Sigma$

**Output:** NO if $\Phi$ is false in some repair of $D$ w.r.t. $\Sigma$?

1. $\neg \Phi = P_1(t_1) \land \cdots \land P_m(t_m) \land \neg P_{m+1}(t_{m+1}) \land \cdots \land \neg P_n(t_n)$
2. find a consistent set of facts $S$ such that
   - $S$ supports all positive facts i.e., $S \supseteq \{P_1(t_1), \ldots, P_m(t_m)\}$
   - $S$ blocks all negative fact i.e., for every $A \in \{P_{m+1}(t_{m+1}), \ldots, P_n(t_n)\} \setminus D$ there is an edge $\{A, B_1, \ldots, B_m\}$ in $G$ such that $S \supseteq \{B_1, \ldots, B_m\}$. 

![Diagram](visual_representation)
Computing CQA using Conflict Hypergraphs (cont.)

Quantifier-free CNF query $\Psi$

1. compute a superset $A$ of consistent answers (with an envelope expression)
2. ground the query with a candidate tuple $t \in A$ and convert to CNF
   \[ \Psi(t) = \Phi_1 \land \ldots \land \Phi_k \]
3. if for some $\Phi_i$ HProver returns NO then discard $t$
4. otherwise, $t$ is a consistent answer to the query

(Chomicki, Marcinkowski, S. [CMS04])

- **Hippo**: a system for computing CQAs in PTIME
- quantifier-free queries and denial constraints
- only edges of the conflict hypergraph hold in memory
- tested for medium-size synthetic databases
Logic Programs for computing CQAs

Logic Programs [ABC03a, GGZ03, CLR03]
- disjunction and classical negation
- checking whether an atom is in all answer sets is $\Pi^p_2$-complete
- dlv, smodels, ...

Scope
- arbitrary first-order queries and universal constraints
- approach unlikely to yield tractable cases

INFOMIX (Eiter et al. [EFGL03, EFGL08])
- combines CQA with data integration (GAV)
- uses dlv for repair computations
- optimization techniques: localization, factorization
- tested on small-to-medium-size legacy databases

Guess what’s in my MIX !
Summary of Complexity Results
What’s so (coNP-)hard about it?

ϕ = (x₁ ∨ ¬x₂ ∨ x₄) ∧ (x₂ ∨ ¬x₄ ∨ x₃) ∧ (¬x₃ ∨ x₄ ∨ ¬x₁)

Reduction

<table>
<thead>
<tr>
<th>R :</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>x₁ = false</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>x₁ = true</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>Falsifying valuations for clauses</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>x₅ = false</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>x₅ = true</td>
</tr>
</tbody>
</table>

A → B

P : | A₁ | B₁ | A₂ | B₂ | A₃ | B₃ |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

reparis correspond to all valuations of variables

we want all valuations to fail to satisfy ϕ i.e. there always should be one clause whose none of literals isn’t satisfied.

Q = ∃x₁, y₁, x₂, y₂, x₃, y₃. P(x₁, y₁, x₂, y₂, y₃) ∧ R(x₁, y₁) ∧ R(x₂, y₂) ∧ R(x₃, y₃)

Claim

True is the consistent answer to Q iff ϕ ∉ 3SAT
### Constraint classes

<table>
<thead>
<tr>
<th>Constraint class</th>
<th>Example</th>
<th>Example full TGD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal constraints</td>
<td>∀. $A_1 \land \cdots \land A_n \Rightarrow B_1 \lor \cdots \lor B_m$</td>
<td>∀. $Par(x, y) \Rightarrow Ma(x, y) \lor Fa(x, y)$</td>
</tr>
<tr>
<td>Tuple-generating dependencies</td>
<td>∀. $A_1 \land \cdots \land A_n \Rightarrow \exists. B$</td>
<td>∀. $Ma(x, y) \land Ma(x, z) \Rightarrow Sib(y, z)$</td>
</tr>
<tr>
<td>Denial constraints</td>
<td>∀. $\neg(A_1 \land \cdots \land A_n)$</td>
<td>∀. $\neg(M(n, s, m) \land M(m, t, w) \land s &gt; t)$</td>
</tr>
<tr>
<td>Functional dependencies</td>
<td>$X \rightarrow Y$</td>
<td>Example primary-key dependency</td>
</tr>
<tr>
<td></td>
<td>• key dependency: $Y = U$</td>
<td>Name $\rightarrow$ Address Salary</td>
</tr>
<tr>
<td>Inclusion dependencies</td>
<td>$R[X] \subseteq S[Y]$</td>
<td>Example foreign key constraint</td>
</tr>
<tr>
<td></td>
<td>• a foreign key constraint: key $Y$</td>
<td>$M[Manager] \subseteq M[Name]$</td>
</tr>
</tbody>
</table>
Data complexity of CQA

- PTIME for \( \{\sigma, \times, \\} \)-queries and binary universal constraints (FD + full IND) [ABC99]
- PTIME for \( \{\sigma, \times, \\}, \cup \} \)-queries and denial constraints [CM05]
- PTIME for \( \{\pi, \sigma \} \)-queries and primary keys [CM05]
- coNP-complete for \( \{\pi, \sigma, \times \} \)-queries and primary keys, and \( \{\pi, \sigma \} \)-queries and FDs [CM05]
- undecidable for arbitrary functional and inclusion dependencies [CLR03]
- \( \Pi^2_p \)-complete for arbitrary sets of functional and inclusion dependencies (repairs obtained by deletions only) [CM05]
- PTIME for \( \{\pi, \sigma, \times \} \)-queries in \( C_{forest} \) and primary keys [FM07]
- PTIME for quantifier-free queries and acyclic full TGDs, join dependencies, and denial constraints [SC10]
- \( \Pi^p_2 \)-complete for universal constraints [SC10]
**Problem statement**

**Fixed:** \( \Sigma \) the set of integrity constraints  
**Input:** Two databases instances \( D \) and \( D' \)  
**Question:** Is \( D' \) a repair of \( D \) w.r.t. \( \Sigma \) ?

**Motivation**

- Close connections with data-cleaning (the model checking problem for repairs)  
- In some cases repair checking is log-space reducible to CQA [CM05].  
- Negative results highlight limitations of integrity enforcement mechanisms.
Data complexity of Repair Checking

- PTIME for denial constraints [CM05]
- PTIME for FDs and acyclic INDs (deletion only) [CM05]
- coNP-complete for arbitrary FDs and INDs (deletion only) [CM05]
- PTIME for denial constrains and full TGDs [SC10]
- PTIME for weekly acyclic LAV dependencies [AK09]
- PTIME for semi-LAV dependencies [GO10]
- coNP for universal constraints [SC10]
Alternative Semantics
Tuple-based repairs

- asymmetric treatment of insertion and deletion:
  - repairs by minimal deletions only (Ch., Marcinkowski [CM05]): data possibly incorrect but complete
  - repairs by minimal deletions and arbitrary insertions (Cali, Lembo, Rosati [CLR03]): data possibly incorrect and incomplete
- minimal cardinality changes (Lopatenko, Bertossi [LB07]), (Afrati, Kolaitis [AK09])
- preferred repairs ([SCM06],[CGZ09], [MAA04], [GSTZ04], [GL04])
- null values (Bravo, Bertossi [BB06])

Attribute-based repairs

- ground and non-ground repairs (Wijsen [Wij05])
- project-join repairs (Wijsen [Wij06])
- repairs minimizing Euclidean distance (Bertossi et al. [BBFL08])
- repairs of minimum cost (Bohannon et al. [BFFR05])
Probabilistic framework for “dirty” databases (Andritsos, Fuxman, Miller [AFM06])

- Potential duplicates identified and grouped into clusters
- Worlds ≈ repairs: one tuple from each cluster
- World probability: product of tuple probabilities
- Clean answers: in the query result in some (supporting) world
- Clean answer probability: sum of the probabilities of supporting worlds
  - Consistent answer: clean answer with probability 1

XML (S., Chomicki, Filiot [SC06, SFC08])

- Tree edit distance for minimality
- Schema: DTD (regular expressions) and tree automata
- XPath queries.

For more, see surveys

- Chomicki, ICDT’07 [Cho07]
- Bertossi, SIGMO Record [Ber06]
M. Arenas, L. Bertossi, and J. Chomicki.  
Consistent query answers in inconsistent databases.  

M. Arenas, L. Bertossi, and J. Chomicki.  
Answer sets for consistent query answering in inconsistent databases.  

M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad.  
Scalar aggregation in inconsistent databases.  

P. Andritsos, A. Fuxman, and R. J. Miller.  
Clean answers over dirty databases: A probabilistic approach.  
In *International Conference on Data Engineering (ICDE)*, page 30, 2006.

F. Afrati and P. Kolaitis.  
Repair checking in inconsistent databases: Algorithms and complexity.  

L. Bravo and L. E. Bertossi.  
Semantically correct query answers in the presence of null values.  

L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko.
The complexity and approximation of fixing numerical attributes in databases under integrity constraints.


**L. Bertossi.**
Consistent query answering in databases.

**P. Bohannon, M. Flaster, W. Fan, and R. Rastogi.**
A cost-based model and effective heuristic for repairing constraints by value modification.

**L. Caroprese, S. Greco, and E. Zumpano.**
Active integrity constraints for database consistency maintenance.
*IEEE Transactions on Knowledge and Data Engineering*, 21(7):1042–1058, 2009.

**J. Chomicki.**
Consistent query answering: Five easy pieces.

**A Cali, D. Lembo, and R. Rosati.**
On the decidability and complexity of query answering over inconsistent and incomplete databases.

J. Chomicki and J. Marcinkowski.
Minimal-change integrity maintenance using tuple deletions.

Computing consistent query answers using conflict hypergraphs.

T. Eiter, M. Fink, G. Greco, and D. Lembo.
Efficient evaluation of logic programs for querying data integration systems.

T. Eiter, M. Fink, G. Greco, and D. Lembo.
Repair localization for query answering from inconsistent databases.

A. Fuxman, E. Fazli, and R. J. Miller.
Conquer: Efficient management of inconsistent databases.

A. Fuxman and R. J. Miller.
First-order query rewriting for inconsistent databases.
G. Greco, S. Greco, and E. Zumpano.
A logical framework for querying and repairing inconsistent databases.

G. Greco and D. Lembo.
Data integration with preferences among sources.

G. Grahne and A. Onet.
Data correspondence, exchange and repair.

S. Greco, C. Sirangelo, I. Trubitsyna, and E. Zumpano.
Feasibility conditions and preference criteria in querying and repairing inconsistent databases.

A. Lopatenko and L. Bertossi.
Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics.
A. Motro, P. Anokhin, and A. C. Acar.  
Utility-based resolution of data inconsistencies.  

S. Staworko and J. Chomicki.  
Validity-sensitive querying of XML databases.  

S. Staworko and J. Chomicki.  
Consistent query answers in the presence of universal constraints.  

S. Staworko, J. Chomicki, and J. Marcinkowski.  
Preference-driven querying of inconsistent relational databases.  

S. Staworko, E. Filiot, and J. Chomicki.  
Querying regular sets of XML documents.  
In *International Workshop on Logic in Databases (LiD)*, 2008.

J. Wijsen.  
Database repairing using updates.  

J. Wijsen.
Project-join-repair: An approach to consistent query answering under functional dependencies.

J. Wijsen.
On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases.