Consistent Query Answering

Jan Chomicki
Dept. CSE
University at Buffalo
State University of New York
http://www.cse.buffalo.edu/~chomicki
Integrity constraints describe valid database instances. Examples:

- **functional dependencies:** “every employee has a single salary.”
- **denial constraints:** “no employee can make more than her manager.”
- **referential integrity:** “managers have to be employees.”

The constraints are formulated in first-order logic:

\[ \forall n, s, m, s', m'. \neg [\text{Emp}(n, s, m) \land \text{Emp}(m, s', m') \land s > s']. \]

An inconsistent database violates the constraints.
Traditional view

Integrity constraints are always enforced.

<table>
<thead>
<tr>
<th>EmpName</th>
<th>Address</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>B. Gates</td>
<td>Redmond, WA</td>
<td>20M</td>
</tr>
<tr>
<td>B. Gates</td>
<td>Redmond, WA</td>
<td>30M</td>
</tr>
<tr>
<td>A. Grove</td>
<td>Santa Clara, CA</td>
<td>10M</td>
</tr>
</tbody>
</table>

Functional dependency: $EmpName \rightarrow Address \extrm{ Salary}$

This instance cannot arise but ... consider data integration.
Ignoring inconsistency

SELECT *  
FROM Emp  
WHERE Salary < 25M

⇒

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>B. Gates</td>
<td>Redmond, WA</td>
<td>20M</td>
</tr>
<tr>
<td>A. Grove</td>
<td>Santa Clara, CA</td>
<td>10M</td>
</tr>
</tbody>
</table>

The result is not fully reliable.
Quarantining inconsistency

The facts involved in an inconsistency are not used in the derivation of query answers [Bry, IICIS'97].

```
SELECT * FROM Emp
WHERE Salary < 25M
```

⇒

```
A. Grove  Santa Clara, CA  10M
```

But what about

```
SELECT EmpName FROM Emp
WHERE Salary > 1M
```

⇒

```
A. Grove
```

Partial information cannot be obtained.
A middle-ground solution

Consider all repairs: possible databases that result from fixing the original database.

Return all the answers that belong to the result of query evaluation in every repair (consistent answers).
SELECT * FROM Emp WHERE Salary < 25M

⇒

A. Grove Santa Clara, CA 10M

But

SELECT EmpName FROM Emp WHERE Salary > 1M

⇒

B. Gates
A. Grove
Inconsistent databases

There are many situations when users want/need to live with inconsistent databases:

- integration of heterogeneous databases with overlapping information
- the consistency of the database will be restored by executing further transactions
- inconsistency wrt “soft” integrity constraints (those that we hope to see satisfied but do not/cannot check) process
- not enough information to resolve inconsistencies
- preservation of all data (even erroneous).
Research goals

Formal definition of reliable ("consistent") information in an inconsistent database.

Computational mechanisms for obtaining consistent information.

Computational complexity analysis:

- tractable vs. intractable classes of queries and integrity constraints
- trade-off: complexity vs. expressiveness.

Implementation:

- preferably using DBMS technology.
Plan of the talk

1. repairs and consistent query answers
2. computing consistent query answers to relational algebra/calculus queries
3. computational complexity
4. aggregation queries
5. alternative frameworks
6. related recent and current work
7. future directions.
Consistent query answers

Arenas, Bertossi, Chomicki [PODS’99].

Repair:

• a database that satisfies the integrity constraints

• difference from the given database is minimal (the set of inserted/deleted facts is minimal under set inclusion).

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 \(r\).
\[ \text{Emp} \]

<table>
<thead>
<tr>
<th>EmpName</th>
<th>Address</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>B. Gates</td>
<td>Redmond, WA</td>
<td>20M</td>
</tr>
<tr>
<td>B. Gates</td>
<td>Redmond, WA</td>
<td>30M</td>
</tr>
<tr>
<td>A. Grove</td>
<td>Santa Clara, CA</td>
<td>10M</td>
</tr>
</tbody>
</table>

Functional dependency:
\[ \text{EmpName} \rightarrow \text{Address} \quad \text{Salary} \]

Repairs:

<table>
<thead>
<tr>
<th>B. Gates</th>
<th>Redmond, WA</th>
<th>30M</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Grove</td>
<td>Santa Clara, CA</td>
<td>10M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B. Gates</th>
<th>Redmond, WA</th>
<th>20M</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Grove</td>
<td>Santa Clara, CA</td>
<td>10M</td>
</tr>
</tbody>
</table>
A logical aside

Belief revision:

- semantically: repairing $\equiv$ revising the database with integrity constraints
- consistent query answers $\equiv$ counterfactual inference.

Logical inconsistency:

- inconsistent database: database facts together with integrity constraints form an inconsistent set of formulas
- trivialization of reasoning does not occur because constraints are not used in relational query evaluation.
Computational issues

There are too many repairs to evaluate the query in each of them.

\[
\begin{array}{cc}
A & B \\
 a_1 & b_1 \\
a_1 & b'_1 \\
a_2 & b_2 \\
a_2 & b'_2 \\
\vdots \\
a_n & b_n \\
a_n & b'_n \\
\end{array}
\]

Under the functional dependency \( A \rightarrow B \), this instance has \( 2^n \) repairs.
Computing consistent query answers

Query rewriting: 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 (many) queries.

Specifying repairs as logic programs.
Query rewriting

First-order queries transformed using semantic query optimization techniques: [Arenas, Bertossi, Chomicki, PODS’99].

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, obtain a local residue by removing $\neg p(\bar{x})$ and the quantifiers for $\bar{x}$ from the constraint

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

• for each literal, global residue $=$ conjunction of local residues.
Functional dependencies:

\[(\forall x)(\forall y)(\forall z)(\forall y')(\forall z')(\neg Emp(x, y, z) \lor \neg Emp(x, y', z') \lor y = y')\]

\[(\forall x)(\forall y)(\forall z)(\forall y')(\forall z')(\neg Emp(x, y, z) \lor \neg Emp(x, y', z') \lor z = z')\]

Query:

\[Emp(x, y, z).\]

Local residues:

\[(\forall y')(\forall z')(\neg Emp(x, y', z') \lor y = y').\]

\[(\forall y')(\forall z')(\neg Emp(x, y', z') \lor z = z').\]
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.
Functional dependencies:

\[
(\forall x)(\forall y)(\forall z)(\forall y')(\forall z')(\neg Emp(x, y, z) \lor \neg Emp(x, y', z') \lor y = y')
\]

\[
(\forall x)(\forall y)(\forall z)(\forall y')(\forall z')(\neg Emp(x, y, z) \lor \neg Emp(x, y', z') \lor z = z')
\]

Query:

\[Emp(x, y, z).\]

Transformed query:

\[Emp(x, y, z) \land (\forall y')(\forall z')(\neg Emp(x, y', z') \lor y = y')\]

\[\land (\forall y')(\forall z')(\neg Emp(x, y', z') \lor z = z').\]
**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>
Scope of query rewriting

Query rewriting:

- queries involving conjunctions of literals (relational algebra: $\sigma, \bowtie, -$) and binary universal integrity constraints [Arenas, Bertossi, Chomicki, PODS’99].

- existentially-quantified conjunctions ($\pi, \sigma, \bowtie$) and single-key dependencies (under certain syntactic restrictions) [Fuxman, Miller, ICDT’05].
SELECT Name
FROM Emp
WHERE Salary > 1M

SELECT Name
FROM Emp e1
WHERE Salary > 1M
    AND NOT EXISTS
        (SELECT *
            FROM EMPLOYEE e2
            WHERE e2.Name = e1.Name
            AND e2.Salary <= 1M)
Conflict hypergraph

Denial constraints only.

Vertices:

- facts in the original instance.

Edges:

- (minimal) sets of facts that violate some constraint.

Repair: a maximal independent set.
Ground queries

Observations:
- the query is in CNF $\Rightarrow$ each conjunct can be processed separately
- all repairs satisfy $\Phi \iff$ no repair satisfies $\neg \Phi$

Algorithm HProver:
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 repair including $P_1(t_1), \ldots, P_m(t_m)$ and excluding $P_{m+1}(t_{m+1}), \ldots, P_n(t_n)$ by enumerating the appropriate edges.

Excluding a fact $A$:
- $A$ is not in the original instance, or
- $A$ belongs to an edge $\{A, B_1, \ldots, B_k\}$ in the conflict hypergraph and $B_1, \ldots, B_k$ belong to the repair.
Properties of HProver

HProver works in \textbf{PTIME} (data complexity):

- \( n - m \) choices from a set of polynomial size
- if all choices successful, a repair can be completed.

Generalizing to \textbf{open, quantifier-free queries}:

- possible bindings for free variables come from evaluating an upper envelope of the original query
$Q : \sigma, \cup, -, \times$

Upper Envelope

$Q' : \sigma, \cup, \times$

Translation

$Q_c : \land, \lor, \neg$

Evaluation

Candidates

Grounding

Conflict Detection

Conflict Graph

HProver

Consistent Answers

DB

27
Experimental results

[Chomicki, Marcinkowski, Staworko, CIKM’04].

The system Hippo:

- back-end: PostgreSQL
- conflict hypergraph (edges) in main memory
- optimization can eliminate many (sometimes all) database accesses in HProver
- tested for synthetic databases with up to 200K tuples, 2% conflicts
- computing consistent query answers using the conflict hypergraph faster than evaluating transformed queries
- relatively little overhead compared to evaluating the original query using the backend
Specifying repairs as logic programs

[Arenas, Bertossi, Chomicki, FQAS’00, TPLP’03], [Greco, Greco and Zumpano, ICLP’01, TKDE’03], [Barcelo, Bertossi, NMR’02, PADL’03]:

- using logic programs with negation and disjunction
- repairs $\equiv$ answer sets
- implemented using main-memory LP systems (dlv, smodels)
- $\Pi_2^p$-complete problems

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

\[ \text{Emp('B.Gates', 'Redmond WA', 20K)}. \]
\[ \text{Emp('B.Gates', 'Redmond WA', 30K)}. \]
\[ \text{Emp('A.Grove', 'Santa Clara CA', 10K)}. \]

Rules:

\[ \neg \text{Emp'}(x, y, z) \lor \neg \text{Emp'}(x, y', z') \leftarrow \text{Emp}(x, y, z), \text{Emp}(x, y', z'), y \neq y'. \]
\[ \neg \text{Emp'}(x, y, z) \lor \neg \text{Emp'}(x, y', z') \leftarrow \text{Emp}(x, y, z), \text{Emp}(x, y', z'), z \neq z'. \]
\[ \text{Emp'}(x, y, z) \leftarrow \text{Emp}(x, y, z), \text{not} \ \neg \text{Emp'}(x, y, z). \]
\[ \neg \text{Emp'}(x, y, z) \leftarrow \text{not} \ \text{Emp}(x, y, z), \text{not} \ \text{Emp'}(x, y, z). \]
## Summary

<table>
<thead>
<tr>
<th>Integrity constraints</th>
<th>Query rewriting</th>
<th>Conflict hypergraph</th>
<th>Logic programs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Binary universal/single-key FDs</td>
<td>Denial</td>
<td>Universal+INDs</td>
</tr>
<tr>
<td></td>
<td>$\sigma, \times, -/\pi, \sigma, \times$</td>
<td>$\sigma, \times, -, \cup$</td>
<td>$\sigma, \pi, \times, -, \cup$</td>
</tr>
<tr>
<td>Queries</td>
<td>PTIME</td>
<td>PTIME</td>
<td>$\Pi^p_2$</td>
</tr>
<tr>
<td>Data complexity</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data complexity

PTIME
Tractable/intractable queries

Tractable (PTIME):

• under any denial constraints:

```sql
SELECT * FROM P
UNION (SELECT * FROM Q
     EXCEPT SELECT * FROM R)
```
Schema:

CREATE TABLE P(A PRIMARY KEY, B);
CREATE TABLE Q(C PRIMARY KEY, D)

**Tractable (PTIME):**

SELECT Q.D
FROM   P, Q
WHERE P.B = Q.C

**Intractable (co-NP-complete):**

SELECT Q.D
FROM   P, Q
WHERE P.B = Q.D
Aggregation queries

```
SELECT SUM(Salary)  \Rightarrow [30,40]
FROM Emp
```

A consistent answer to an aggregation query is no longer a single value:

- a set of values, or
- a range of values (polynomial size)
SELECT SUM(Salary) FROM Emp

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

But that works only for a single functional dependency and some aggregation operators!
Consistent answers to aggregation queries

[Arenas, Bertossi, Chomicki, ICDT'01, TCS'03]:

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

Restriction to key dependencies improves tractability!
Alternative frameworks

Different assumptions about database completeness and correctness (in the presence of inclusion dependencies):

- possibly incorrect but complete: repairs by deletion only [Chomicki, Marcinkowski, I&C, 2005]
- possibly incorrect and incomplete: fix FDs by deletion, INDs by insertion [Cali, Lembo, Rosati, PODS’03].

Different notions of minimal repairs:

- minimal set of changes vs. minimal cardinality changes
- repairing attribute values [Wijsen, ICDT’03; Bohannon et al., SIGMOD’05].
Related work

Belief revision:
- revising database with integrity constraints
- revised theory changes with each database update
- emphasis on semantics (AGM postulates), not computation
- complexity results [Eiter, Gottlob, AI’92] do not quite transfer

Disjunctive information:
- repair $\equiv$ possible world (sometimes)
- using disjunctions to represent resolved conflicts
- query languages: representation-specific, relational algebra or calculus
- complexity results [Imielinski et al., JCSS’95] do not transfer
Current and future work

Systems:

- **INFOMIX** [Leone et al., SIGMOD 2005 demo]:
  - LP-based (dlv)
  - large databases (large number of repairs?)

- **ConQuer** [Fuxman, Fazli, Miller, SIGMOD’05]:
  - query rewriting
  - most TPC-H benchmark queries
  - large databases

- **Hippo** [Chomicki, Marcinkowski, Staworko, CIKM 2004]:
  - conflict hypergraph
  - no projection
  - large databases
Broadening scope:

- preferences and priorities [Staworko, Chomicki, Marcinkowski, IIDB 2006]:
  - source rankings, timestamps
  - probabilities [Andritsos, Fuxman, Miller, ICDE 2006]

- data integration and exchange

- data cleaning

- XML [Flesca et al, WISE 2005; Staworko, Chomicki, dataX 2006]

- spatial/spatiotemporal databases.
Selected papers:


