Inconsistency-tolerance in data integration

Riccardo Rosati

Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma
Italy

Problem studied

- how to deal with constraint violations (conflicts) in
  - databases
  - data integration
  - data exchange
  - Semantic Web (ontology-based data access)
  - peer data management
  - ...

Very simple example

HomePage		
name	**URL**	**country**
Georg Gottlob	`benner.dbai.tuwien.ac.at/staff/gottlob`	Austria
Georg Gottlob	`web.comlab.ox.ac.uk/oucl/people/georg.gottlob.html`	UK
Leonid Libkin	`www.cs.toronto.edu/~libkin`	Canada
Leonid Libkin	`www.lfcs.inf.ed.ac.uk/people/profiles/Leonid_Libkin.html`	UK

BelongsTo	
country	**continent**
Austria	EU
UK	EU
Canada	NA

query q: “professors teaching in Europe”

```
  SELECT HomePage.Name
  FROM HomePage, BelongsTo
  WHERE
    HomePage.country = BelongsTo.country
    AND BelongsTo.continent = “EU”
```

Example (cont.)

- “every professor has at most one home page”
  (key constraint on relation HomePage)
- instance violates this key
- we want to evaluate query q...
- ... and still obtain the answer “Georg Gottlob”
  (because both home pages are hosted by European universities)
- ... while we don’t want to get the answer “Leonid Libkin” anymore

How to deal with conflicts?

Traditional off-line solution: **material repair**

- Solution 1: clean the data (before querying)
    - not always possible or convenient

On-line solutions: **virtual repair**

- Solution 2: during query answering, use procedures/trust policies/preferences to resolve the conflicts
    - not always possible
    - e.g., not enough knowledge on data provenance

How to deal with conflicts?

What can be done when all else fails?

- Solution 3: ask the user
- Solution 4: don’t care about conflicts (standard query evaluation)
  - too brave
  - in our example, we also obtain “Leonard Libkin”
- Solution 5: discard all conflicting data (tuples)
  - too cautious
  - in our example, we obtain no answers!
- Solution 6: use **consistent query answering** techniques:
  - obtain meaningful answers from conflicting databases,,,
  - ...through a more “intelligent” (virtual) repair of data (declarative semantics)

Repairs and consistent answers

semantics of consistent query answering (CQA):

- **repair** = database that satisfies the constraints and is at a “minimal distance” from the real database
    - measure: number/sets of tuple insertions and/or deletions
    - (different actual semantics)

- **consistent answer** to q = answer to q in **all** repairs of the database

Example (consistent answers)

repair 1:

name	URL	country
Gottlob	`benner.dbai.tuwien..`	Austria
Libkin	`www.cs.toronto..`	Canada

answer to q:
{Gottlob}

repair 2:

name	URL	country
Gottlob	`benner.dbai.tuwien..`	Austria
Libkin	`www.lfcs.inf.ed.ac..`	UK

answer to q:
{Gottlob,Libkin}

repair 3:

name	URL	country
Gottlob	`web.comlab.ox.uk..`	UK
Libkin	`www.cs.toronto..`	Canada

answer to q:
{Gottlob}

repair 4:

name	URL	country
Gottlob	`web.comlab.ox.uk..`	UK
Libkin	`www.lfcs.inf.ed.ac..`	UK

answer to q:
{Gottlob,Libkin}

Example (consistent answers)

- “Georg Gottlob” is a consistent answer
- “Leonard Libkin” is not a consistent answer

Constraint violations, CWA, and OWA

CWA: data in the DB cannot be neither added nor deleted

what if we move from CWA to OWA?

- very important: many formalizations (data integration, data exchange, ontologies) based on OWA

OWA is able to handle only some kinds of violations:

- positive example: violation of a foreign key constraint
  - can be repaired by **adding** tuples (allowed by OWA)
  - violation interpreted as **incompleteness** of data
- negative example: violation of a key constraint
  - can be repaired only by **deleting** tuples (not allowed by OWA)
  - violation interpreted as **inconsistency** of data

Complexity of consistent query answering

Complexity of CQA depends on:

- the constraint language
- the query language
- (the semantics)

Problem with CQA:

- the number of repairs is in general exponential in the number of conflicting tuples
- computing consistent answers of conjunctive queries is **coNP-hard** (data complexity) for many combinations of queries/constraints
  - e.g., primary key constraints + conjunctive queries

Tractable CQA

how to deal with coNP-hardness?

identify “easy” cases

examples:

- if conflicting data are (very) few...
  - (e.g., when previous data cleaning solves almost all conflicts)

  ... then CQA is tractable
- if (conflicting) data satisfy some locality property (so that repairs can be efficiently factorized)...

  ... then CQA is tractable (Eiter, Fink, Greco, Lembo)
- if the structure of the query (w.r.t. the constraints) allows to look at a “small” number of conflicts (independent of the size of the DB)...

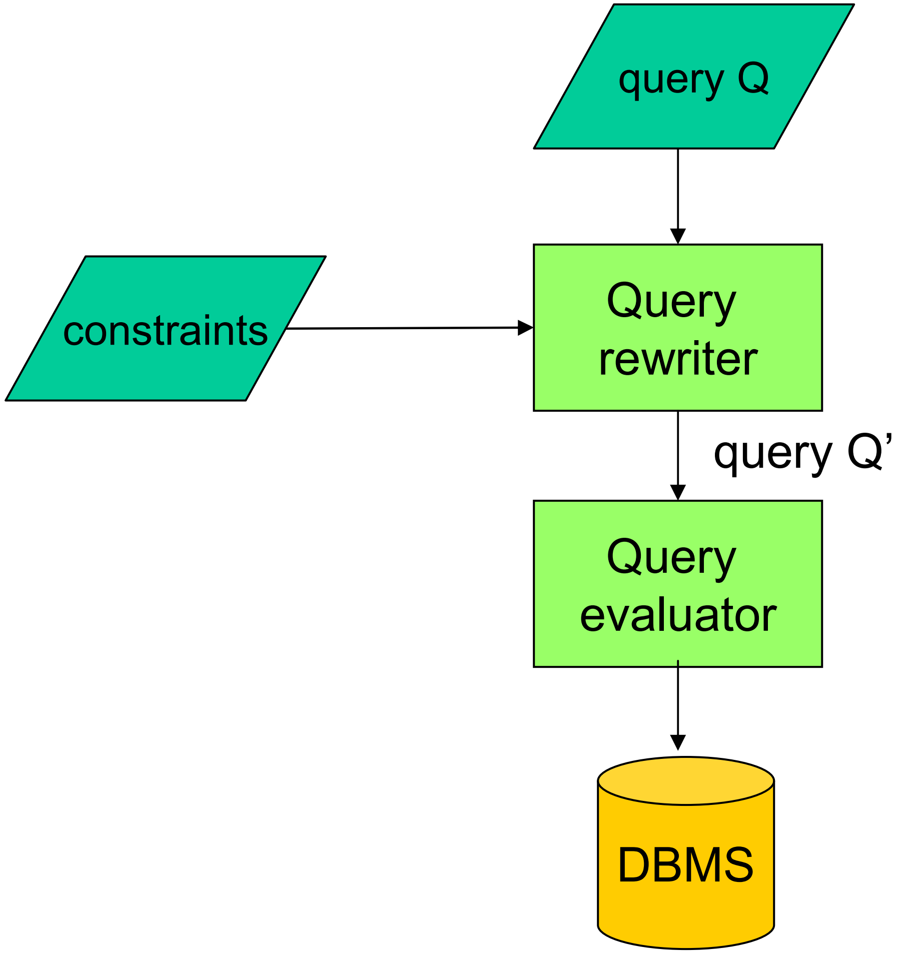
  ... then CQA is tractable

Techniques for CQA

- techniques based on query rewriting:
  1. given query q and constraints C, generate a query q_c
  2. evaluate q_c over the inconsistent DB

- techniques directly accessing the data (based on the constraints)

CQA via query rewriting



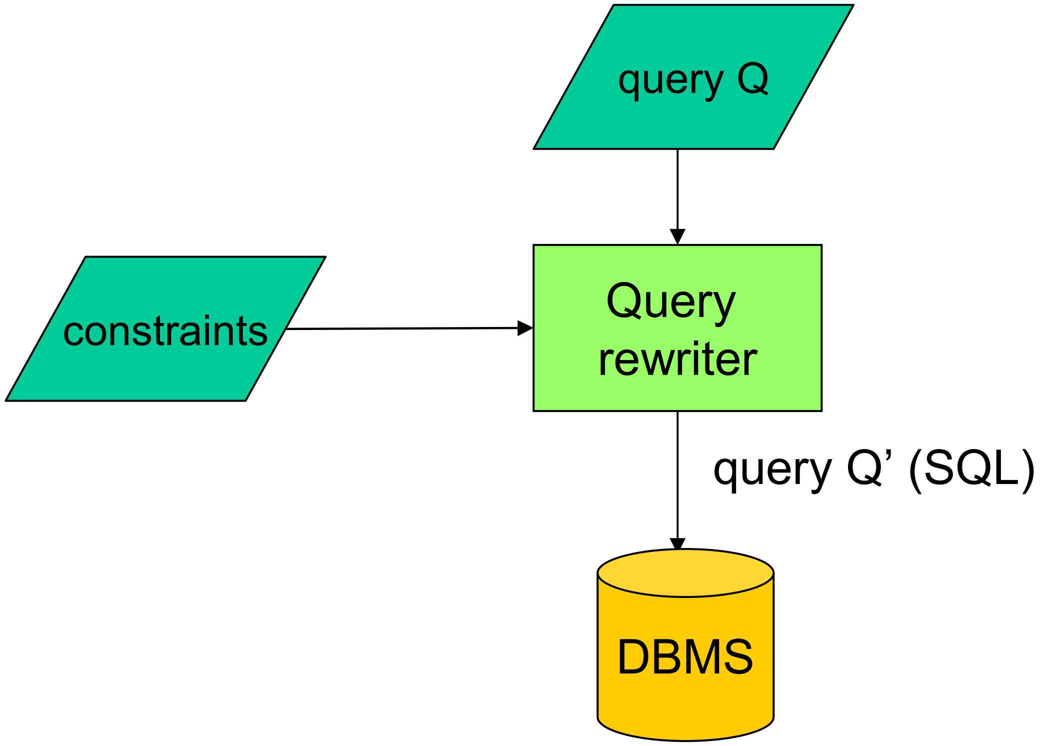
CQA via query rewriting

- techniques based on query rewriting need a coNP-hard query language
- usually, nonmonotonic extensions of datalog
- able to deal with very expressive queries and constraints
  - datalog queries
  - arbitrary “universal” constraints (e.g., EGDs, denials)
  - unable to deal with general “referential” constraints (e.g., foreign keys, TGDs)
- not efficient (in general)
- hard to implement through relational DB technology

CQA via query rewriting

- are there (interesting) combinations of queries and constraints for which CQA can be rewritten in SQL?
- yes!
  - **CQs with acyclic join graphs** + **key constraints** (Fuxman, Miller)
  - extensions to other constraints
    - functional dependencies (Wijsen)
    - disjointness constraints (Lembo, Rosati, Ruzzi)
  - extension to probabilistic databases (Andritsos, Fuxman, Miller)

CQA via SQL query rewriting



CQA in data integration and exchange

- GAV data integration
  - CQs + keys, foreign keys, disjointnesses:
    nonmonotonic datalog rewriting (Calì, Lembo, Rosati)
- LAV data integration
  - (Bertossi, Bravo)
- peer-to-peer data integration:
  nonmonotonic datalog rewriting techniques
  - (Bertossi, Bravo)
  - (Calvanese, De Giacomo, Lenzerini, Lembo, Rosati)
- ontology-based data integration
  - consistent instance checking for DL-Lite (Lembo, Ruzzi)

Systems

- CONQUER (Fuxman, Fazli, Miller)
  - based on SQL rewriting
  - restricted queries + constraints
  - very efficient
- HIPPO (Chomicki, Marcinkowski, Staworko)
  - based on compact representations of repairs (conflict hypergraphs)
  - expressive queries + constraints
- INFOMIX (Leone et al.)
  - based on nonmonotonic datalog rewriting
  - expressive queries + constraints + GAV mappings
  - good experimental results

Open research issues

- semantics:
  - for complex classes of constraints (e.g., keys and foreign keys), no well-established notion of repair (different semantics proposed)
  - same for more complex systems (e.g., LAV/GLAV data integration)
- complexity
  - identification of other (more expressive) tractable combinations of queries and constraints
- algorithms

Questions

- from the application/industrial side, is there a real interest for the development of technologies for inconsistency-tolerance in data integration and data exchange?
    - e.g., are there real applications where “traditional” data cleaning is not sufficient?
- what are the forms of inconsistency-tolerance that are more interesting for current data integration and data exchange applications? e.g.:
    - which classes of queries and constraints?
    - which semantics?
- how far is research from the development of effective methods and techniques for inconsistency-tolerance in data integration?

ANSWERS?

Example (CQA through SQL rewriting)

```
query q: “professors teaching in Europe”
  SELECT HomePage.Name
  FROM HomePage, BelongsTo
  WHERE HomePage.country = BelongsTo.country
  AND BelongsTo.continent = “EU”
```

```
rewritten query:
    SELECT HomePage.Name
    FROM HomePage H1, BelongsTo B1
    WHERE H1.country = B1.country
    AND B1.continent = “EU”
    AND NOT EXISTS
       (SELECT * FROM HomePage H2, BelongsTo B2
        WHERE H2.country = B2.country
        AND B2.continent <> “EU”
        AND B2.name = B1.name)
```