Consistent Query Answering

Opportunities and Limitations

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

Integrity constraints

Integrity constraints describe valid database instances. Examples:

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

Integrity constraints

Integrity constraints describe valid database instances. Examples:

- functional dependencies (FDs): "*every employee has a single salary.*"
- denial constraints: "*no employee can make more than her manager.*"
- inclusion dependencies (INDs): "*managers have to be employees.*"

The constraints are formulated in first-order logic:

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

Integrity constraints

Integrity constraints describe valid database instances. Examples:

- functional dependencies (FDs): "*every employee has a single salary.*"
- denial constraints: "*no employee can make more than her manager.*"
- inclusion dependencies (INDs): "*managers have to be employees.*"

The constraints are formulated in first-order logic:

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

An inconsistent database violates the constraints.

Traditional view

Integrity constraints are always enforced.

Traditional view

Integrity constraints are always enforced.

Emp

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	20M
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Functional dependency:

$EmpName \rightarrow Address\ Salary$

Traditional view

Integrity constraints are always enforced.

Emp

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	20M
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Functional dependency:

$EmpName \rightarrow Address\ Salary$

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

Ignoring inconsistency

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

$\Rightarrow$

B. Gates	Redmond, WA	20M
A. Grove	Santa Clara, CA	10M

Ignoring inconsistency

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

⇒

B. Gates	Redmond, WA	20M
A. Grove	Santa Clara, CA	10M

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].

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
```

$\Rightarrow$

A. Grove	Santa Clara, CA	10M

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
```

$\Rightarrow$ $\boxed{\text{A. Grove} \quad \text{Santa Clara, CA} \quad \text{10M}}$

But what about

```
SELECT EmpName
FROM Emp
WHERE Salary > 10K
```

$\Rightarrow$ $\boxed{\text{A. Grove}}$

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 > 10K
```

⇒

A. Grove

Partial information cannot be obtained.

Constraints with exceptions

Weaken the constraints [Borgida, TODS'85], without affecting query evaluation.

Constraints with exceptions

Weaken the constraints [Borgida, TODS'85], without affecting query evaluation.

Emp

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	20M
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Constraint with exception:

$$(\forall x, y, z, y', z')\neg[Emp(x, y, z) \land Emp(x, y', z') \land y \neq y' \land x \neq \mathsf{B.Gates}]$$

Data cleaning

Emp

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	20M
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Functional dependency:

$EmpName \rightarrow Address\ Salary$

Data cleaning

Emp

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	20M
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Functional dependency:

$EmpName \rightarrow Address\ Salary$

Repair:

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Data cleaning

Emp

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	20M
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Functional dependency:

$EmpName \rightarrow Address\ Salary$

Repair:

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Some information is lost.

CQA: a query-driven approach

Consider all repairs: possible databases that result from fixing the original database in a minimal way.

Return all the answers that belong to the result of query evaluation in every repair (consistent answers).

Emp

EmpName	*Address*	*Salary*
B. Gates	Redmond, WA	20M
B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

Functional dependency:

$EmpName \rightarrow Address\ Salary$

Repairs:

B. Gates	Redmond, WA	30M
A. Grove	Santa Clara, CA	10M

B. Gates	Redmond, WA	20M
A. Grove	Santa Clara, CA	10M

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

$\Rightarrow$

A. Grove	Santa Clara, CA	10M

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

$\Rightarrow$

A. Grove	Santa Clara, CA	10M

But

```
SELECT EmpName
FROM Emp
WHERE Salary > 10K
```

$\Rightarrow$

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
- not enough information to resolve inconsistencies
- preservation of all data (even erroneous)
- 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

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/SQL queries
3. extensions and new directions:
   - other notions of repair
   - probabilistic databases
4. further active research directions
5. related work
6. lessons of CQA research
7. the CQA community

Constraint classes

Universal: $\forall.\ \neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m.$

Denial: $\forall.\ \neg A_1 \vee \cdots \vee \neg A_n.$

Functional dependencies (FDs): $X \rightarrow Y.$

Inclusion dependencies (INDs): $P[X] \subseteq R[Y].$

Consistent query answers

Arenas, Bertossi, Ch. [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.

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.

A	B
a_1	b_1
a_1	b'_1
a_2	b_2
a_2	b'_2
$\cdots$	
a_n	b_n
a_n	b'_n

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 = the set of consistent answers to Q 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, Ch., PODS'99].

Query rewriting

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

Functional dependencies:

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

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

Query: $Emp(x,y,z)$.

Query rewriting

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

Functional dependencies:

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

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

Query: $Emp(x,y,z)$.

Transformed query:

$$\begin{aligned} Emp(x,y,z) \quad & \wedge \quad (\forall y')(\forall z')(\neg Emp(x,y',z') \vee y = y') \\ & \wedge \quad (\forall y')(\forall z')(\neg Emp(x,y',z') \vee z = z'). \end{aligned}$$

Scope of query rewriting

Query rewriting:

- queries involving conjunctions of literals (*relational algebra*: $\sigma, \times, -$) and binary universal integrity constraints [Arenas, Bertossi, Ch., PODS'99].
- existentially-quantified conjunctions ($\pi, \sigma, \bowtie$) and single-key dependencies [Fuxman, Miller, ICDT'05]:
  - *CTree* queries ($\approx$ no non-key joins)
  - extended to exclusion dependencies [Grieco et al., CIKM'05].

```
SELECT Name
FROM Emp
WHERE Salary > 10K
```

$\longmapsto$

```
SELECT Name
FROM Emp e1
WHERE Salary > 10K
  AND NOT EXISTS
      (SELECT *
       FROM EMPLOYEE e2
       WHERE e2.Name = e1.Name
         AND e2.Salary <= 10K)
```

Experimental results (ConQuer)

[Fuxman, Fazli, Miller, SIGMOD'05].

The system ConQuer:

- back-end: DB2 UDB.
- query rewriting into SQL, producing unnested queries
- queries from the TPC-H workload
- databases can be annotated with consistency indicators
- tested for synthetic databases with 400K–8M tuples, 0–50% conflicts
- relatively little overhead compared to evaluating the original query using the backend

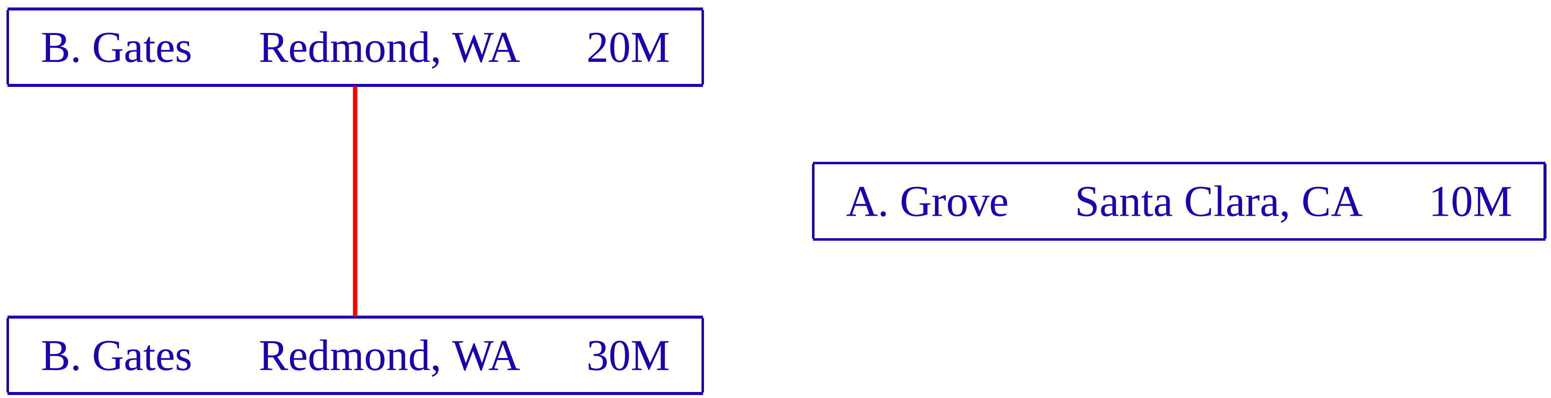
Conflict hypergraph

Denial constraints only.

Vertices: facts in the original instance.

Edges: (minimal) sets of facts that violate some constraint.

Repairs: maximal independent sets.



Computing CQAs using the conflict hypergraph

[Ch., Marcinkowski, I&C, 2005].

Algorithm HProver:

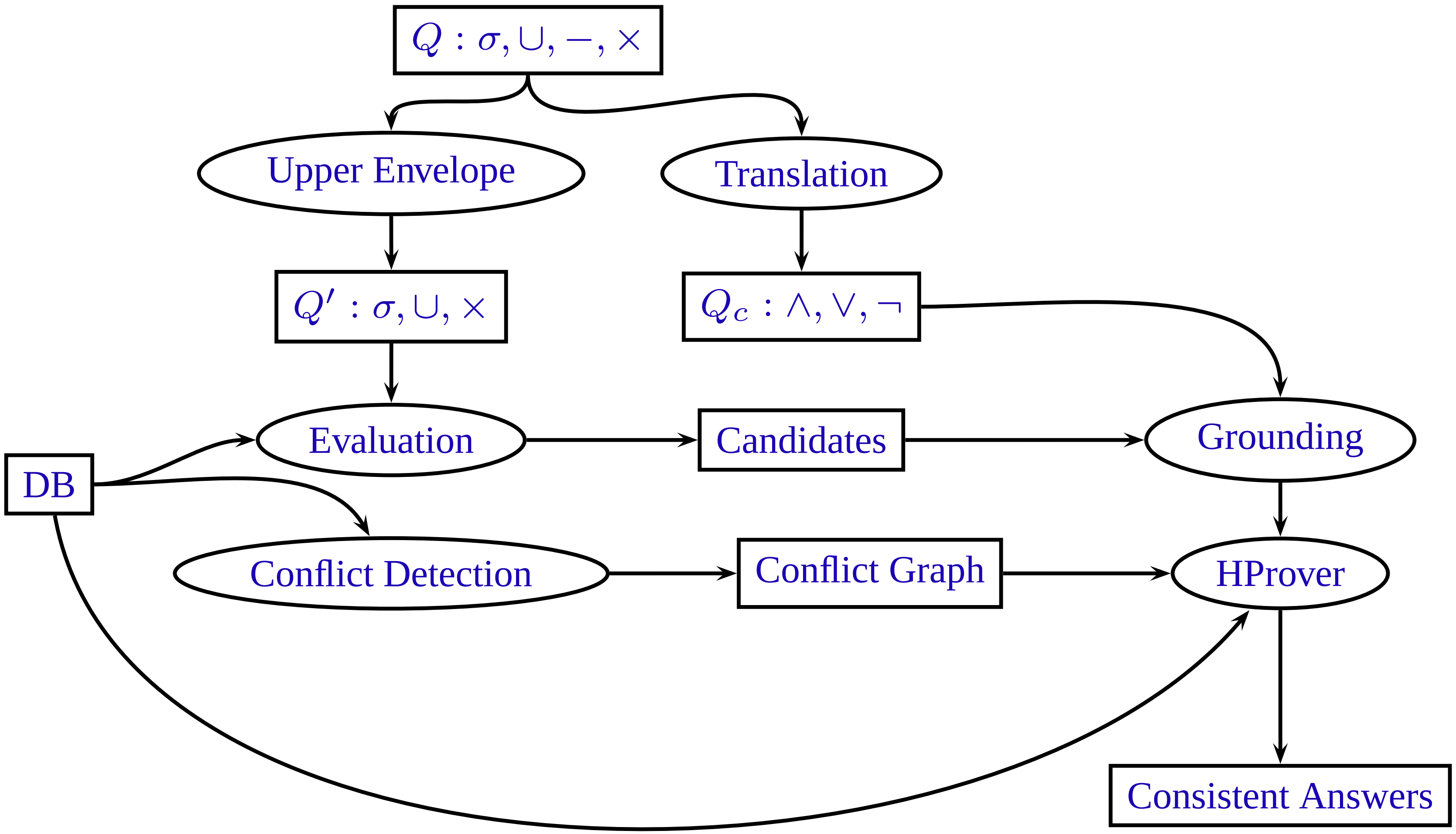
1. input: query Φ a disjunction of ground atoms
2. $\neg\Phi = P_1(t_1) \wedge \cdots \wedge P_m(t_m) \wedge \neg P_{m+1}(t_{m+1}) \wedge \cdots \wedge \neg P_n(t_n)$
3. find a repair including $P_1(t_1), \ldots, P_m(t_m)$ and excluding $P_{m+1}(t_{m+1}), \ldots, P_n(t_n)$ by enumerating the appropriate 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.

P (data complexity) for quantifier-free queries and denial constraints.

Hippo


$Q : \sigma, \cup, -, \times$
Upper Envelope
Translation
$Q' : \sigma, \cup, \times$
$Q_c : \wedge, \vee, \neg$
Evaluation
Candidates
Grounding
DB
Conflict Detection
Conflict Graph
HProver
Consistent Answers

Experimental results (Hippo)

[Ch., 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, Ch., TPLP'03], [Greco, Greco and Zumpano, TKDE'03], [Barcelo, Bertossi, PADL'03], [Eiter et al., ICLP'03]:

- using logic programs with negation and disjunction
- repairs $\equiv$ answer sets
- several different encodings
- implemented using main-memory LP systems (`dlv`, `smodels`)
- Π_2^p-complete problems

Scope:

- arbitrary universal constraints, some inclusion dependencies
- arbitrary first-order queries

The approach of [Arenas, Bertossi, Ch., TPLP'03].

Facts:

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

Rules:

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

Experimental data (INFOMIX)

[Eiter at al., ICLP'03].

The system INFOMIX:

- combines CQA with data integration (GAV)
- relational backend: PostgreSQL
- uses the disjunctive LP system `dlv` for repair computations
- optimization techniques: localization, factorization
- tested on legacy databases of up to 50K tuples.

Data complexity of consistent query answers

	Keys / primary keys)	Denial	Universal
$\sigma, \times, -$	P	P	P(binary)
$\sigma, \times, -, \cup$	P	P	?
σ, π	co-NPC / P	co-NPC	co-NP-hard
$\sigma, \times, \pi$	co-NPC / P(Ctree)	co-NPC	co-NP-hard

Tractable approaches:

- query rewriting
- conflict graphs

Sources: [Ch., Marcinkowski, I&C'05], [Fuxman, Miller, ICDT'05], [Cali, Lembo, Rosati, PODS'03].

Tractable/intractable queries

Tractable (P):

- under any denial constraints:

```
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)
```

Schema:

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

Tractable (P):

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

Schema:

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

Tractable (P):

```
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
```

Alternative frameworks

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

- possibly incorrect but complete: repairs by deletion only [Ch., Marcinkowski, I&C'05]
- possibly incorrect and incomplete: fix FDs by deletion, INDs by insertion [Cali, Lembo, Rosati, PODS'03].

Different notions of minimal repairs:

- minimal cardinality changes [Lopatenko, Bertossi, '06]:
  - tractability of incremental CQA:

    given a consistent database D and a fixed sequence of updates U, what is the complexity of computing CQA over $U(DB)$?

- repairing attribute values.

Attribute-based repairs

Several different approaches:

(A) ground and non-ground repairs [Wijsen, TODS'05]

(B) project-join repairs [Wijsen, FQAS'06]

(C) repairs minimizing Euclidean distance [Bertossi et al., DBPL'05]

(D) repairs of minimum cost [Bohannon et al., SIGMOD'05].

Attribute-based repairs

Several different approaches:

(A) ground and non-ground repairs [Wijsen, TODS'05]

(B) project-join repairs [Wijsen, FQAS'06]

(C) repairs minimizing Euclidean distance [Bertossi et al., DBPL'05]

(D) repairs of minimum cost [Bohannon et al., SIGMOD'05].

Computational complexity:

- (A) and (B): similar to tuple based repairs
- (C) and (D): checking existence of a repair of cost $< K$ NP-complete.

Project-join repairs

PJ-repairs: repairs of a lossless join decomposition.

Project-join repairs

PJ-repairs: repairs of a lossless join decomposition.

EmpDept

Name	*Dept*	*Location*
John	Sales	Buffalo
Mary	Sales	Toronto

Functional dependencies:

$Name \rightarrow Dept$

$Dept \rightarrow Location$

Project-join repairs

PJ-repairs: repairs of a lossless join decomposition.

EmpDept

Name	*Dept*	*Location*
John	Sales	Buffalo
Mary	Sales	Toronto

Functional dependencies:

$Name \rightarrow Dept$

$Dept \rightarrow Location$

Repairs:

John	Sales	Buffalo

Mary	Sales	Toronto

Decomposition

$$\pi_{Name,Dept}(EmpDept) \bowtie \pi_{Dept,Location}(EmpDept)$$

Decomposition

$$\pi_{Name,Dept}(EmpDept) \bowtie \pi_{Dept,Location}(EmpDept)$$

Name	*Dept*	*Location*
John	Sales	Buffalo
John	Sales	Toronto
Mary	Sales	Buffalo
Mary	Sales	Toronto

Decomposition

$$\pi_{Name,Dept}(EmpDept) \bowtie \pi_{Dept,Location}(EmpDept)$$

Name	*Dept*	*Location*
John	Sales	Buffalo
John	Sales	Toronto
Mary	Sales	Buffalo
Mary	Sales	Toronto

PJ-repairs:

John	Sales	Buffalo
Mary	Sales	Buffalo

John	Sales	Toronto
Mary	Sales	Toronto

Probabilistic framework for “dirty” databases

[Andritsos, Fuxman, Miller, ICDE’06].

Framework:

- potential duplicates identified and grouped into clusters
- worlds $\approx$ 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

EmpSal

EmpName	*Salary*	prob
B. Gates	20M	0.7
B. Gates	30M	0.3
A. Grove	10M	0.5
A. Grove	20M	0.5

Functional dependency:

$EmpName \rightarrow Salary$

Query rewriting:

```
SELECT Name
FROM Emp
WHERE Salary > 15M
```

$\longmapsto$

```
SELECT Name, SUM(e.prob)
FROM Emp e
WHERE Salary > 15M
GROUP BY Name
```

Query rewriting:

```
SELECT Name
FROM Emp
WHERE Salary > 15M
```

$\longmapsto$

```
SELECT Name, SUM(e.prob)
FROM Emp e
WHERE Salary > 15M
GROUP BY Name
```

Evaluation of the rewritten query:

B. Gates	20M	0.7
B. Gates	30M	0.3
A. Grove	10M	0.5
A. Grove	20M	0.5

$\Rightarrow$

B. Gates	1
A. Grove	0.5

Further active topics

SQL queries:

- aggregation: glb/lub answers
- grouping

Data integration:

- GAV, LAV, GLAV,...
- tension between repairing and satisfying source-to-target dependencies

P2P:

- how to isolate an inconsistent peer

Nulls:

- repairs with nulls?
- clean semantics vs. SQL conformance

Priorities:

- preferred repairs
- conflict resolution

XML:

- what is an integrity constraint?
- what is a repair?
- minimum edit distance

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
- beyond revision: merging, arbitration

Disjunctive information:

- repair $\equiv$ possible world (sometimes)
- consistent answer $\equiv$ certain answer (sometimes)
- using disjunctions to represent resolved conflicts
- complexity results [Imielinski et al., JCSS'95] do not quite transfer

Lessons of CQA research

Need to tame the semantic explosion:

- different repair semantics overwhelm the potential user

More focus on applications:

- 99% technology, 1% applications
- often only repairing is needed
- heuristics

Integrate with other tools:

- schema matching/mapping
- data cleaning

The CQA community

Some data:

- over 30 active researchers
- 80–100 publications (since 1999)
- papers in major conferences (PODS, SIGMOD, ICDT, ICDE, CIKM, DBPL) and journals (TODS, TKDE, TCS, I&C, AMAI, JAL, TPLP)
- outreach to the AI community (qualified success)
- yearly workshop: IIDB (+LAAIC?)

Selected papers

1. M. Arenas, L. Bertossi, J. Chomicki, "Consistent Query Answers in Inconsistent Databases." PODS'99.
2. L. Bertossi, J. Chomicki, "Query Answering in Inconsistent Databases." In *Logics for Emerging Applications of Databases*, J. Chomicki, R. van der Meyden, G. Saake [eds.], Springer-Verlag, 2003.
3. J. Chomicki and J. Marcinkowski, "On the Computational Complexity of Minimal-Change Integrity Maintenance in Relational Databases." In *Inconsistency Tolerance*, L. Bertossi, A. Hunter, T. Schaub, editors, Springer-Verlag, 2004.
4. L. Bertossi, "Consistent Query Answering in Databases." SIGMOD Record, June 2006.