Scalar Aggregation in FD-Inconsistent Databases

Jan Chomicki
University at Buffalo
http://www.cse.buffalo.edu/~chomicki

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

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:
  - no permission
  - inconsistent information can be useful
  - restoring consistency can be a complex and non deterministic process

Consistent query answers [PODS'99]

How to distinguish between reliable and unreliable information in an inconsistent database?

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.

Scope of this work

Attribute domains: uninterpreted constants, numbers (as in SQL).

Integrity constraints: arbitrary **functional dependencies**.

Aggregation queries:

```
SELECT f(...)
FROM R
```

Standard set of SQL scalar aggregation operators: `MIN`, `MAX`, `COUNT`, `SUM`, `AVG`.

Functional dependency:
$Name\ County \Rightarrow Tally$

Name	*County*	*Date*	*Tally*
Brown	A	11/07	541
Brown	A	11/11	560
Brown	B	11/07	302
Green	A	11/07	653
Green	A	11/11	730
Green	B	11/07	101

Repairs:

Brown	A	11/07	541
Brown	B	11/07	302
Green	A	11/07	653
Green	B	11/07	101

Brown	A	11/11	560
Brown	B	11/07	302
Green	A	11/07	653
Green	B	11/07	101

Brown	A	11/07	541
Brown	B	11/07	302
Green	A	11/11	730
Green	B	11/07	101

Brown	A	11/11	560
Brown	B	11/07	302
Green	A	11/11	730
Green	B	11/07	101

Consistent query answers to aggregation queries

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

returns, in different repairs, two different answers: 843 and 862.

A possible solution:

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

$\Rightarrow \quad [843, 862]$

A consistent query answer for scalar aggregation queries is an **optimal range** of values, represented as a pair

[greatest lower bound, least upper bound]

over all repairs.

Why a range, not a set?

Cons:

- not every element in the range corresponds to the value of the scalar function

Pros:

- no need to introduce complex objects
- always polynomial size

Note: Each representation has to be interpreted as an **indefinite** object anyway.

For the functional dependency $A \rightarrow B$, the instance

A	B
a_1	0
a_1	1
a_2	0
a_2	2
$\cdots$	
a_n	0
a_n	2^{n-1}

leads to **exponentially** many repairs and a **different** value for `SUM(B)` in every repair.

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 = 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 queries.

Specifying repairs as logic programs.

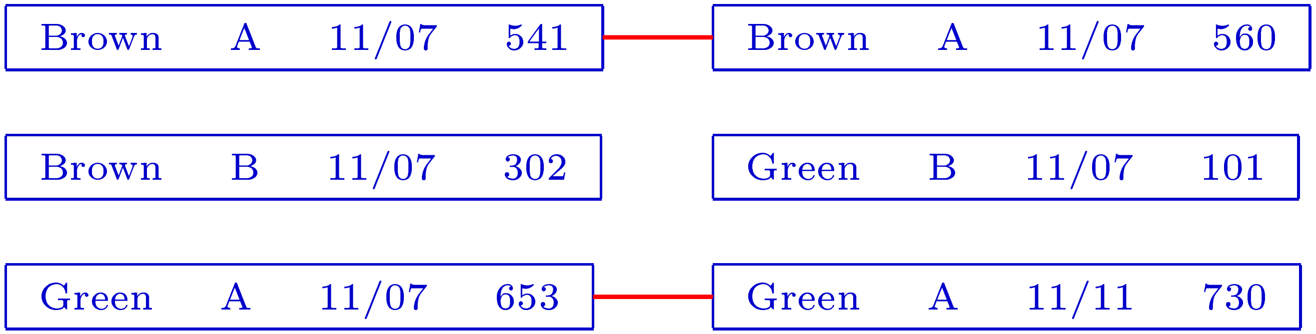
Representing all repairs

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



Another conflict graph

Functional dependencies $A \to B$ and $B \to A$.

Instance:

A	B
a_1	b_1
a_1	b_2
a_2	b_2
a_2	b_1

Conflict graph:

$$\begin{tikzcd}
(a_1, b_1) \arrow[r, dash] \arrow[d, dash] & (a_1, b_2) \arrow[d, dash] \\
(a_2, b_1) \arrow[r, dash] & (a_2, b_2)
\end{tikzcd}$$

Aggregation queries

In simple cases, queries can be transformed to compute optimal result ranges for consistent query answers.

```
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;
```

Data complexity of aggregation queries

	greatest lower bound		least upper bound	
	$\lvert F\rvert = 1$	$\lvert F\rvert \geq 2$	$\lvert F\rvert = 1$	$\lvert F\rvert \geq 2$
`MIN(A)`	PTIME	PTIME	PTIME	NP-complete
`MAX(A)`	PTIME	NP-complete	PTIME	PTIME
`COUNT(*)`	PTIME	NP-complete	PTIME	NP-complete
`COUNT(A)`	NP-complete	NP-complete	NP-complete	NP-complete
`SUM(A)`	PTIME	NP-complete	PTIME	NP-complete
`AVG(A)`	PTIME	NP-complete	PTIME	NP-complete

AVG(A) in PTIME

Contributed by Vijay Raghavan and Jeremy Spinrad, June 2000.

Functional dependency $X \to Y$.

Bins: X-values, colors: Y-values.

Using a **2-OPT** strategy:

Let c be any arbitrary selection of m colors, one from each bin.

`while` $\exists$ a color c'_i in bin i : $AVG(\langle c_1, c_2, ..., c_{i-1}, c'_i, c_{i+1}, ..., c_m \rangle) > AVG(c)$ **`do`**

$\quad c := \langle c_1, c_2, ..., c_{i-1}, c'_i, c_{i+1}, ..., c_m \rangle$

`endwhile`

2-OPT $:= AVG(c)$

To guarantee polynomial number of iterations, the color that gives the **maximum increase** in the value of `AVG` for a given bin needs to be selected.

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 `COUNT(*)` queries

Two functional dependencies.

Indirect approach:

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

Direct approach *[Roger He, December 2000]*:

- bipartite **clique graph**:
  - nodes: cliques in 1-dependency conflict graphs
  - edges: nonempty clique intersections
- finding a maximum matching in the clique graph
- overall complexity $O(\max(e, n\sqrt{n}))$.

Related work

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 aggregation queries

Conjecture: The set of repairs w.r.t. 2 FDs in BCNF can be represented as a finite table with OR-objects.

Future work

Broadening scope

- SQL:
  - more general aggregation queries
  - nested subqueries
  - keys and foreign keys
- integrity: denial constraints, inclusion dependencies
- preferences:
  - source rankings
  - timestamps

New paradigms:

- query reformulation in information integration
- data cleaning
- semistructured data

Algorithms and computational complexity:

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