Carleton
UNIVERSITY

Consistent Query Answering in Databases

Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

bertossi@scs.carleton.ca
www.scs.carleton.ca/~bertossi

Simon Fraser University, March 2005.

The Context

There are situations when we want/need to live with inconsistent information in a database

With information that contradicts given integrity constraints

- The DBMS does not fully support data maintenance or integrity checking/enforcing
- The consistency of the database will be restored by executing further transactions
- Delayed updates of a datawarehouse
- Integration of heterogeneous databases without a central/global maintenance mechanism

- Inconsistency wrt "soft" integrity constraints we hope to see satisfied, but do not prevent transactions from execution
- User constraints than cannot be checked
- Legacy data on which we want to impose semantic constraints

It may be 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

The Problem

Not all data participate in the violation of the ICs

The inconsistent database can still give us "correct" or consistent answers to queries!

We want to:

- Give a precise definition of consistent answer to a query in an inconsistent database
- Find mechanisms for obtaining such consistent information from the inconsistent database
- Study the computational complexity of the problem

Example

A database instance D

$Employee$	$Name$	$Salary$
	$J.Page$	5,000
	$J.Page$	8,000
	$V.Smith$	3,000
	$M.Stowe$	7,000

FD: $Name \rightarrow Salary$

D violates FD, by the tuples with $J.Page$ in $Name$

There are two possible ways to repair the database in a minimal way if only deletions/insertions of whole tuples are allowed

D_1

Employee	*Name*	*Salary*
	J.Page	5,000
	V.Smith	3,000
	M.Stowe	7,000

D_2

Employee	*Name*	*Salary*
	J.Page	8,000
	V.Smith	3,000
	M.Stowe	7,000

$(M.Stowe, 7,000)$ persists in all repairs, and it does not participate in the violation of the FD

$(J.Page, 8,000)$ does not persist in all repairs, and it does participate in the violation of FD

Repairs and Consistent Answers

Fixed: DB schema and (infinite) domain; a set of first order integrity constraints IC

Definition: (Arenas, Bertossi, Chomicki; PODS 99)

A repair of a database instance D is a database instance D'

- over the same schema and domain
- satisfies IC
- differs from D by a minimal set of changes (insertions or deletions of tuples) wrt set inclusion

Given a query $Q(\bar{x})$ to D, we want as answers all and only those tuples obtained from D that are "consistent" wrt IC (even when D globally violates IC)

Definition: (Arenas, Bertossi, Chomicki; PODS 99)

A tuple $\bar{t}$ is a consistent answer to query $Q(\bar{x})$ in D iff $\bar{t}$ is an answer to query $Q(\bar{x})$ in every repair D' of D:

$$D \models KQ[\bar{t}] \quad :\Longleftrightarrow \quad D' \models Q[\bar{t}] \quad \text{for every repair } D' \text{ of } D$$

A model theoretic definition ...

Example

Inconsistent DB instance D wrt FD: $Name \longrightarrow Salary$

Employee	*Name*	*Salary*
	$J.Page$	5,000
	$J.Page$	8,000
	$V.Smith$	3,000
	$M.Stowe$	7,000

Repairs D_1, resp. D_2

Employee	*Name*	*Salary*
	$J.Page$	5,000
	$V.Smith$	3,000
	$M.Stowe$	7,000

Employee	*Name*	*Salary*
	$J.Page$	8,000
	$V.Smith$	3,000
	$M.Stowe$	7,000

$D \models K\ \ Employee(M.Stowe, 7,000)$

$D \models K\ (Employee(J.Page, 5,000) \vee Employee(J.Page, 8,000))$

$D \models K\ \exists X\, Employee(J.Page, X)$

We can see this is not the same as getting rid of the data that participates in the violation of the IC

Some information is preserved ...

Computing Consistent Answers

So far: a semantic notion of consistent answer from an inconsistent database

We want to compute consistent answers

But not by computing all possible repairs and checking answers in common to all of them

Retrieving consistent answers via computation of all database repairs is not possible/sensible/feasible

Example: A database instance that is inconsistent wrt FD: $X \rightarrow Y$

r	X	Y
	1	0
	1	1
	2	0
	2	1
	.	.
	n	0
	n	1

has 2^n possible repairs!

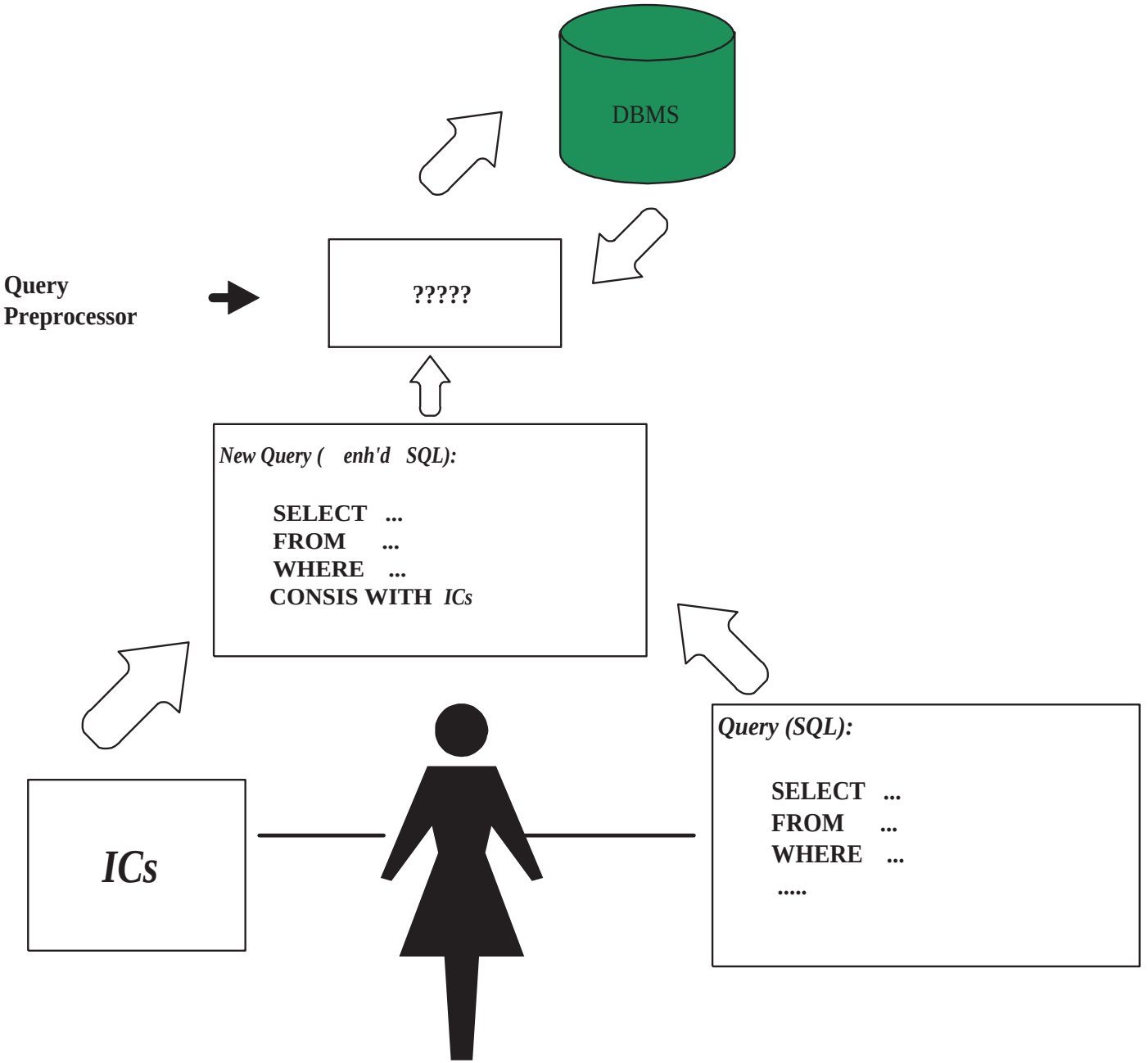
Query Transformation

First-Order queries and constraints

Idea:

- Do not compute the repairs
- Query only the available inconsistent database instance
- Transform the query and pose the new query (as usual)

(Arenas, Bertossi, Chomicki; PODS 99)
(Celle, Bertossi; DOOD 00)


DBMS
Query
Preprocessor
?????
New Query (enh'd SQL):
SELECT ...
FROM ...
WHERE ...
CONSIS WITH ICs
ICs
Query (SQL):
SELECT ...
FROM ...
WHERE ...
.....

Given a query Q to the inconsistent DB D, qualify Q with appropriate information derived from the interaction between Q and the ICs

- To locally satisfy the ICs
- To discriminate between tuples in the answer set
- Inspired by "Semantic Query Optimization" techniques

Consistent answers to $Q(\bar{x})$ in D??

Rewrite query: $Q(\bar{x}) \longmapsto Q'(\bar{x})$

$Q'(\bar{x})$ a new first order query

Retrieve from D the (ordinary) answers to $Q'(\bar{x})$

Example

$IC: \ \forall x(P(x) \rightarrow Q(x))$ $\qquad\qquad$ $D = \{P(a), P(b), Q(b), Q(c)\}$

1. Query to D: $Q(x)$?

If $Q(x)$ holds in D, then $P(x) \rightarrow Q(x)$ holds in D

Elements in Q do not participate in a violation of IC

2. Query: $P(x)$?

If $P(x)$ holds in D, then $Q(x)$ must hold in D in order to satisfy $P(x) \rightarrow Q(x)$

An answer x to "$P(x)$?" is consistent if x is also in table Q

Transform query 2. into: $P(x) \wedge Q(x)$?

Pose this query instead

$Q(x)$ is a residue of $P(x)$ wrt $\forall x(P(x) \rightarrow Q(x))$

Residue can be obtained by resolution between the query literal and the IC

Posing new query to D we get only answer $\{b\}$

For query $Q(x)?$ there is no residue, i.e. every answer to query $Q(x)?$ is also a consistent answer, i.e. we get $\{b, c\}$

3. Query $\neg Q(x)$? (not safe, just for illustration)

Residue wrt $\forall x(P(x) \rightarrow Q(x))$ is $\neg P(x)$

New query: $\neg Q(x) \wedge \neg P(x)$

Answers to this new query (in the active domain): $\emptyset$

No consistent answers ...

Example

$$FD\text{:}\quad \forall XYZ\ (\neg Employee(X,Y)\ \vee\ \neg Employee(X,Z)\ \vee\ Y = Z)$$

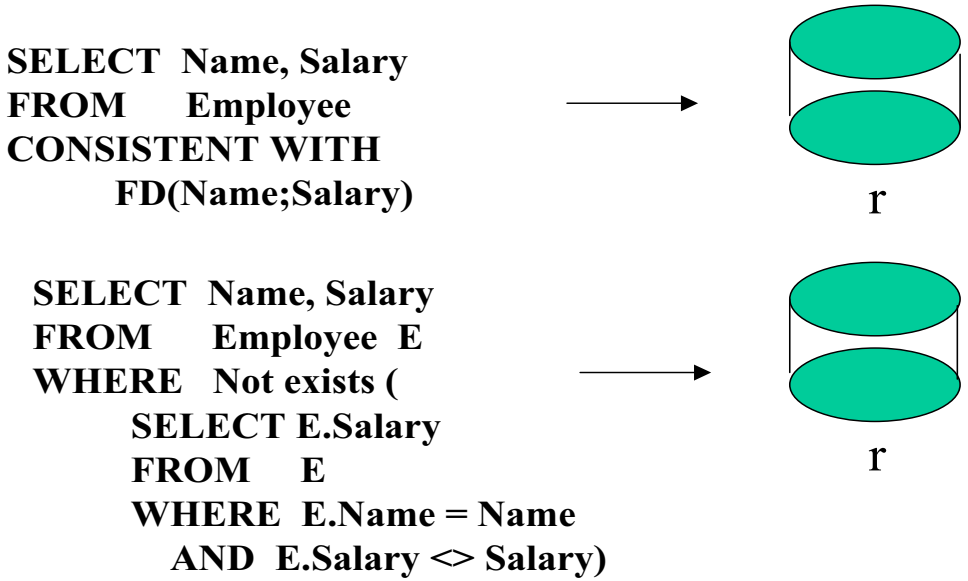
Query: $Employee(X,Y)?$

Consistent answers: $(V.Smith,\ 3{,}000),\ (M.Stowe,\ 7{,}000)$
(but not $(J.Page,\ 5{,}000),\ (J.Page,\ 8{,}000)$)

Can be obtained by means of the transformed query

$$T(Q(X,Y)) := Employee(X,Y)\ \wedge\ \forall Z\ (\neg Employee(X,Z)\ \vee\ Y = Z)$$

... those tuples (X,Y) in the relation for which X does not have and associated Z different from Y ...



Again, the residue $\forall Z\ (\neg Employee(X, Z) \vee Y = Z)$ can be automatically obtained by applying resolution to the query and the FD

In general, T is an iterative operator

Example

IC: $\{R(x) \vee \neg P(x) \vee \neg Q(x),\ P(x) \vee \neg Q(x)\}$

Query: $Q(x)$

$T_1(Q(x)) := Q(x) \wedge (R(x) \vee \neg P(x)) \wedge P(x)$

Apply T again, now to the appended residues

$T_2(Q(x)) := Q(x) \wedge (T(R(x)) \vee T(\neg P(x))) \wedge T(P(x))$

$$T_2(\varphi(x)) = Q(x) \ \wedge \ (R(x) \vee (\neg P(x) \wedge \neg Q(x))) \ \wedge P(x) \wedge (R(x) \vee \neg Q(x))$$

And again:

$$T_3(Q(x)) := Q(x) \ \wedge \ (R(x) \vee (\neg P(x) \wedge T(\neg Q(x)))) \wedge \\ P(x) \wedge (T(R(x)) \vee T(\neg Q(x)))$$

Since $T(\neg Q(x)) = \neg Q(x)$ and $T(R(x)) = R(x)$, we obtain

$$T_3(Q(x)) \ = \ T_2(Q(x))$$

A finite fixed point! Does it always exist?

In general, an infinitary query: $T_\omega(\varphi(x)) \ := \ \bigcup_{n<\omega} \{T_n(\varphi(x))\}$

In the example, $T_\omega(Q(x)) \ = \ \{T_1(Q(x)), T_2(Q(x))\}$

Always finite?

Some Results

There are sufficient conditions on queries and ICs for soundness and completeness of operator T (ABC; PODS 99)

- Soundness: every tuple computed via T is consistent in the semantic sense

$$D \models T_\omega(\varphi)[\bar{t}] \implies D \models K\varphi[\bar{t}]$$

- Completeness: every semantically consistent tuple can be obtained via T

$$D \models K\varphi[\bar{t}] \implies D \models T_\omega(\varphi)[\bar{t}]$$

Natural and useful syntactical classes satisfy the conditions

But incomplete for full FO queries and ICs

There are necessary and sufficient conditions for syntactic termination

- In the iteration process to determine $T_\omega(Q)$ nothing syntactically new is obtained beyond some finite step

There are sufficient conditions for semantic termination

- From some finite step on, only logically equivalent formulas are obtained

In these favorable cases, a FO SQL query can be translated into a new FO SQL query that is posed as usual to the database

Some Limitations

First order query rewriting based approach has limitations (most of them apply to the one based on operator T and to any other; see later ...)

- T is defined and works for some special classes of queries and integrity constraints
- ICs are universal, which excludes referential ICs; and queries are quantifier-free conjunctions of literals
- T does not work for disjunctive or existential queries, e.g. $\exists Y\ Employee(J.Page, Y)$?

FO query reformulation has been slightly extended using other methods

- Hypergraph representation of the DB (the vertices) and its semantic conflicts (the hyperedges)
- Graph based algorithms on original query can be translated into SQL queries (Chomicki, Marcinkowski, Staworko; software demos at EDBT 04)

From the logical point of view:

- We have not logically specified the database repairs
- We have a model-theoretic definition plus an incomplete computational mechanism

- From such a specification $Spec$ we might:
  - Reason from $Spec$
  - Consistently answer queries: $Spec \stackrel{?}{\models} Q(\bar{x})$
  - Derive algorithms for consistent query answering

Consistent query answering is non-monotonic; then a non-monotonic semantics for $Spec$ is expected

Specification in Annotated Logic

We want to specify database repairs, by means of a consistent theory

The database instance D (seen as a set of ground atomic formulas) and the set of integrity constraints IC are mutually inconsistent

Use a different logic, that allows generating a consistent theory!

Use annotated predicate calculus (APC)
(Kifer, Lozinskii; J. Aut. Reas. 92)

Inconsistent classical theories can be translated into consistent annotated theories

Usual annotations: true ($\mathbf{t}$), false ($\mathbf{f}$), contradictory ($\top$), unknown ($\bot$)

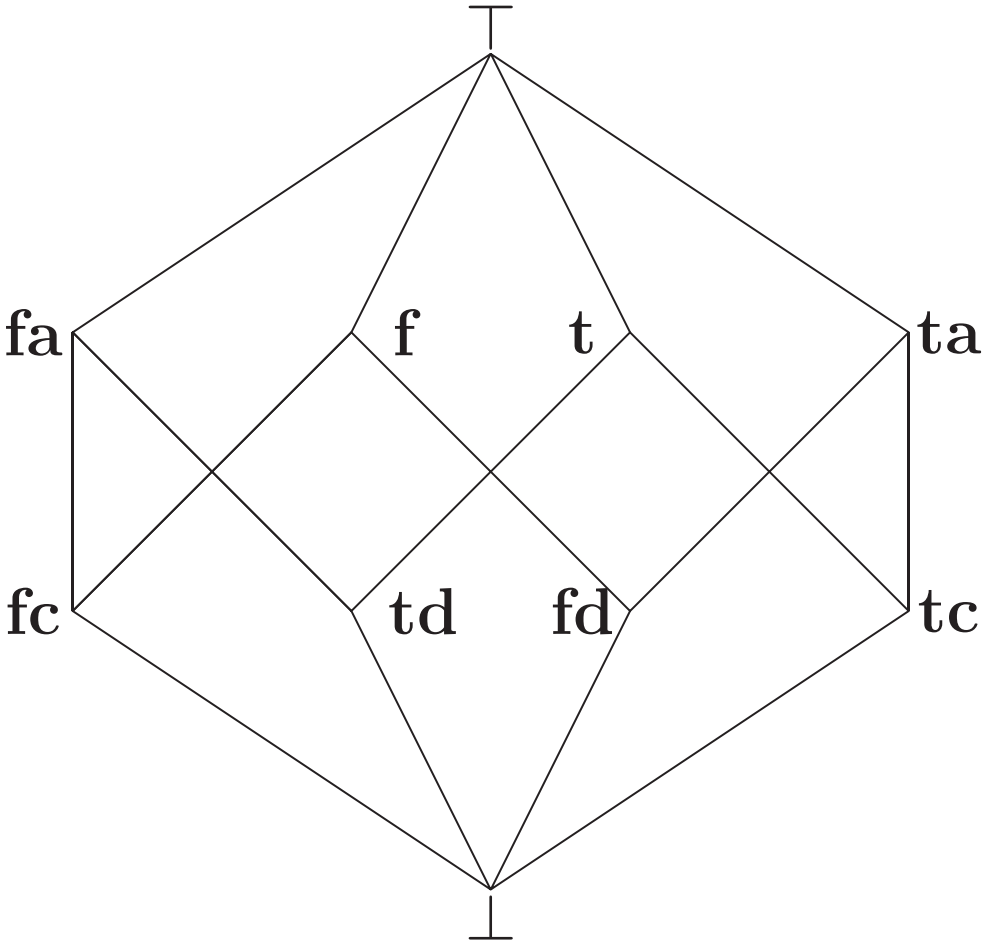
Atoms in an APC theory are annotated with truth values, at the object level, e.g.
$Employee(V.Smith, 3000)$:$\mathbf{t}$, $Employee(V.Smith, X)$:$\mathbf{f}$

Embed both D and IC into a single consistent APC theory (Arenas, Bertossi, Kifer; DOOD 00)

- ICs are hard, not to be given up
- Data is flexible, subject to repairs

Choose an appropriate truth values lattice $\mathcal{L}at$:

- Database values: $\mathbf{t_d}$, $\mathbf{f_d}$
- Constraint values: $\mathbf{t_c}$, $\mathbf{f_c}$
- Advisory values: $\mathbf{t_a}$, $\mathbf{f_a}$ They advise to solve conflicts between $\mathbf{d}$-values and $\mathbf{c}$-values in favor of $\mathbf{c}$-values



Intuitively, ground atoms A for which $A{:}\mathbf{t_a}$ or $A{:}\mathbf{f_a}$ become true are to be inserted into, resp. deleted from D

Generate an APC theory $Spec$ embedding D and IC into APC:

- Translate the constraint:
  $\neg Employee(X,Y) \ \vee \ \neg Employee(X,Z) \ \vee \ Y = Z$
  into
  $Employee(X,Y){:}\mathbf{f_c} \ \vee \ Employee(X,Z){:}\mathbf{f_c} \ \vee \ Y = Z{:}\mathbf{t}$

- Translate database facts, e.g. $Employee(J.Page, 5,000)$ into $Employee(J.Page, 5,000){:}\mathbf{t_d}$

- Plus axioms for unique names assumption, closed world assumption, ...

Navigation in the lattice plus an adequate definition of APC formula satisfaction help solve the conflicts between database facts and constraint facts

- For every $\mathbf{s} \in \mathcal{L}at$, $\bot \leq \mathbf{s} \leq \top$
- $lub(\mathbf{t}, \mathbf{f}) = \top$, $lub(\mathbf{t_c}, \mathbf{f_d}) = \mathbf{t_a}$, etc.
- Use Herbrand structures, i.e sets of ground annotated atoms
- Formula satisfaction: I a structure, $\mathbf{s} \in \mathcal{L}at$, A a classical atomic formula

$I \models A{:}\mathbf{s}$ iff there exists $s' \in \mathcal{L}at$ such that $A{:}\mathbf{s}' \in I$ and $\mathbf{s} \leq \mathbf{s}'$

It can be proved that the database repairs correspond to the models of $Spec$ that make true a minimal set of atoms annotated with $\mathbf{t_a}$, $\mathbf{f_a}$

Change a minimal set of database atoms!!!

From the specification $Spec$ algorithmic and complexity results for consistent query answering can be obtained

Most importantly, this approach motivated a more general and practical approach to specification of database repairs based on logic programs

Specifying Repairs with Logic Programs

The collection of all database repairs can be represented in a compact form

Use disjunctive logic programs with stable model semantics (Barcelo, Bertossi; PADL 03)

Repairs correspond to distinguished models of the program, namely to its stable models

Example: Full inclusion dependency $IC\colon\ \forall\bar{x}(P(\bar{x}) \rightarrow Q(\bar{x}))$

Inconsistent instance $D = \{P(\bar{c}), P(\bar{d}), Q(\bar{d}), Q(\bar{e})\}$

The programs use annotation constants in an extra attribute in the database relations

Annotation	Atom	The tuple $P(\bar{a})$ is ...
$\mathbf{t_d}$	$P(\bar{a}, \mathbf{t_d})$	a fact of the database
$\mathbf{f_d}$	$P(\bar{a}, \mathbf{f_d})$	a fact not in the database
$\mathbf{t_a}$	$P(\bar{a}, \mathbf{t_a})$	advised to be made true
$\mathbf{f_a}$	$P(\bar{a}, \mathbf{f_a})$	advised to be made false
$\mathbf{t}^\star$	$P(\bar{a}, \mathbf{t}^\star)$	true or becomes true
$\mathbf{f}^\star$	$P(\bar{a}, \mathbf{f}^\star)$	false or becomes false
$\mathbf{t}^{\star\star}$	$P(\bar{a}, \mathbf{t}^{\star\star})$	true in the repair
$\mathbf{f}^{\star\star}$	$P(\bar{a}, \mathbf{f}^{\star\star})$	false in the repair

Repair program $\Pi(D, IC)$:

1. The original data: $P(\bar{c}, \mathbf{t_d}) \leftarrow$
   $P(\bar{d}, \mathbf{t_d}) \leftarrow$
   $Q(\bar{d}, \mathbf{t_d}) \leftarrow$
   $Q(\bar{e}, \mathbf{t_d}) \leftarrow$

2. Whatever was true (false) or becomes true (false), gets annotated with $\mathbf{t}^\star$ $(\mathbf{f}^\star)$:

$$P(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x}, \mathbf{t_d})$$
$$P(\bar{x}, \mathbf{t}^\star) \leftarrow P(\bar{x}, \mathbf{t_a})$$
$$P(\bar{x}, \mathbf{f}^\star) \leftarrow \ not\ P(\bar{x}, \mathbf{t_d})$$
$$P(\bar{x}, \mathbf{f}^\star) \leftarrow P(\bar{x}, \mathbf{f_a})$$

... the same for Q ...

3. There may be interacting ICs (not here), and the repair process may take several steps, changes could trigger other changes

   We need annotation constants for the local changes $(\mathbf{t_a}, \mathbf{f_a})$, but also annotations $(\mathbf{t}^\star, \mathbf{f}^\star)$ to provide feedback to the rules that produce local repair steps

   $$P(\bar{x}, \mathbf{f_a}) \ \vee \ Q(\bar{x}, \mathbf{t_a}) \ \leftarrow \ P(\bar{x}, \mathbf{t}^\star), Q(\bar{x}, \mathbf{f}^\star)$$

   One rule per IC; that says how to repair the IC in case of a violation

   Passing to annotations $\mathbf{t}^\star$ and $\mathbf{f}^\star$ allows to keep repairing the DB wrt to all the ICs until the process stabilizes

4. Repairs must be coherent: use denial constraints at the program level to prune undesirable models

$$\leftarrow P(\bar{x}, \mathbf{t_a}), P(\bar{x}, \mathbf{f_a})$$
$$\leftarrow Q(\bar{x}, \mathbf{t_a}), Q(\bar{x}, \mathbf{f_a})$$

5. Annotations constants $\mathbf{t}^{\star\star}$ and $\mathbf{f}^{\star\star}$ are used to read off the literals that are inside (outside) a repair

$$P(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{x}, \mathbf{t_a})$$
$$P(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow P(\bar{x}, \mathbf{t_d}),\ \mathit{not}\ P(\bar{x}, \mathbf{f_a})$$
$$P(\bar{x}, \mathbf{f}^{\star\star}) \leftarrow P(\bar{x}, \mathbf{f_a})$$
$$P(\bar{x}, \mathbf{f}^{\star\star}) \leftarrow \ \mathit{not}\ P(\bar{x}, \mathbf{t_d}),\ \mathit{not}\ P(\bar{x}, \mathbf{t_a}).$$ ... etc.

The program has two stable models (and two repairs):

$$\{P(\bar{c}, \mathbf{t_d}), ..., P(\bar{c}, \mathbf{t}^\star), Q(\bar{c}, \mathbf{f}^\star), Q(\bar{c}, \mathbf{t_a}), P(\bar{c}, \mathbf{t}^{\star\star}), Q(\bar{c}, \mathbf{t}^\star),$$
$$P(d, \mathbf{t}^{\star\star}), Q(d, \mathbf{t}^{\star\star}), \ldots, Q(\bar{c}, \mathbf{t}^{\star\star}), ...\} \equiv$$
$$\{P(\bar{c}), Q(\bar{c}), P(\bar{d}), Q(\bar{d}), Q(\bar{e})\}$$

... insert $Q(\bar{c})$!!

$$\{P(\bar{c}, \mathbf{t_d}), ..., P(\bar{c}, \mathbf{t}^\star), P(\bar{c}, \mathbf{f}^\star), Q(\bar{c}, \mathbf{f}^\star), P(\bar{c}, \mathbf{f}^{\star\star}), Q(\bar{c}, \mathbf{f}^{\star\star}),$$
$$P(d, \mathbf{t}^{\star\star}), Q(d, \mathbf{t}^{\star\star}), \ldots, P(\bar{c}, \mathbf{f_a}), ...\} \equiv$$
$$\{P(\bar{d}), Q(\bar{d}), Q(\bar{e})\}$$

... delete $P(\bar{c})$!!

To obtain consistent answers to a FO SQL query:

1. Transform or provide the query as a logic program (this is standard methodology)

2. Run the query program together with the specification program

   ... under the skeptical or cautious stable model semantics that sanctions as true of a program what is true of all its stable models

Example: (continued)

Consistent answers to query $P(\bar{x}) \wedge \neg Q(\bar{x})$?

Run repair program $\Pi(D, IC)$ together with query program

$$Ans(\bar{x}) \leftarrow P(\bar{x}, \mathbf{t}^{\star\star}), Q(\bar{x}, \mathbf{f}^{\star\star})$$

The two previous stable models become extended with ground Ans atoms

None of them in the intersection of the two models

In consequence, under the skeptical SMS, $Ans = \emptyset$, i.e. no consistent answers, as expected ...

Remarks:

- This is general methodology that works for general FO queries, universal ICs and referential ICs (without cycles)

  One to one correspondence between repairs and stable models of the program

- Existential ICs, like referential ICs, can be handled, with different repair policies, e.g. introduction of null values, cascaded deletions, ...
  (Barcelo,Bertossi,Bravo; LNCS 2582)
  (Bravo,Bertossi; CASCON 04)

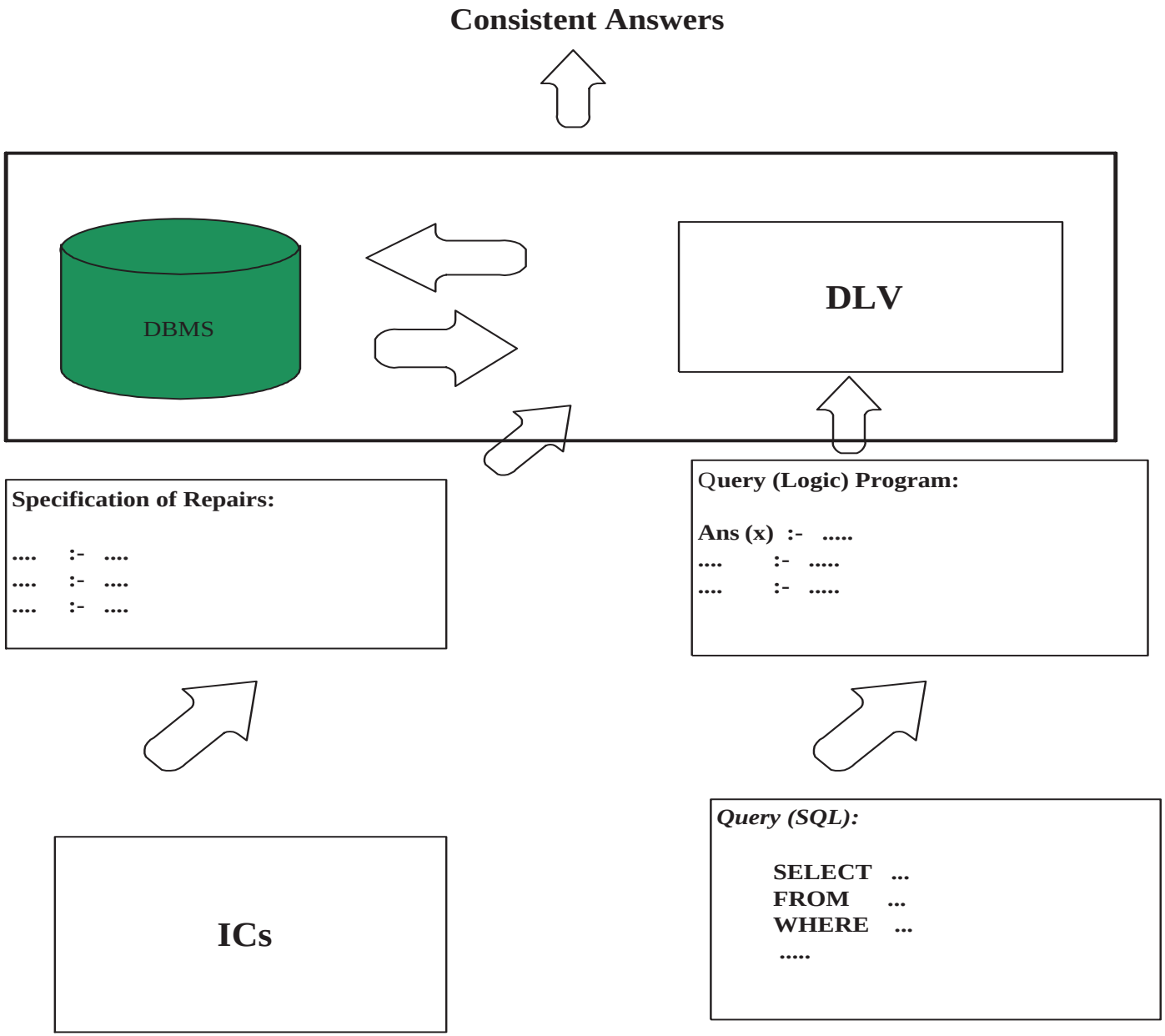
- The same repair program can be used for all the queries, the same applies to the computed stable models

  The query at hand adds a final layer on top (obtaining a split program)

- The program can be optimized in several ways; e.g. avoiding materialization of CWA (Barcelo,Bertossi,Bravo; LNCS 2582), and annotations of DB facts

- We have successfully experimented with the DLV system for computing the stable models semantics
(N. Leone et al.; ACM Transactions on Comp. Logic)

- Related methodologies:
(Arenas, Bertossi, Chomicki; TPLP 03)
(Greco, Greco, Zumpano; IEEE TKDE 03)


Consistent Answers
DBMS
DLV
Specification of Repairs:
.... :- ....
.... :- ....
.... :- ....
Query (Logic) Program:
Ans (x) :- .....
.... :- .....
.... :- .....
ICs
Query (SQL):
SELECT ...
FROM ...
WHERE ...
.....

Aggregation Queries

We have presented first order queries only

What about aggregation queries?

- They are natural and usual in DBs, and part of SQL
- They are crucial in scenarios where inconsistencies are likely to occur, e.g. data integration, in particular, dataware-housing

We will see that aggregation is challenging for consistent answers

A restricted scenario:

- Functional dependencies
- Standard set of SQL-2 scalar aggregation operators: `MIN`, `MAX`, `COUNT(*)`, `COUNT(A)`, `SUM`, and `AVG`
- Atomic queries applying just one of these operators

Redefining Consistent Anwers

Example: A database instance and the FD: $Name \rightarrow Amount$

$Salary$	$Name$	$Amount$
	$V.Smith$	5000
	$V.Smith$	8000
	$P.Jones$	3000
	$M.Stone$	7000

The repairs:

$Salary$	$Name$	$Amount$
	$V.Smith$	5000
	$P.Jones$	3000
	$M.Stone$	7000

$Salary$	$Name$	$Amount$
	$V.Smith$	8000
	$P.Jones$	3000
	$M.Stone$	7000

Query: `MIN(Amount)`?

We should get 3000 as a consistent answer: `MIN(Amount)` returns 3000 in every repair

Query: `MAX(Amount)`?

The maximum, 8000, comes from a tuple that participates in the violation of FD

`MAX(Amount)` returns a different value in each repair: 7000 or 8000

There is no consistent answer as previously defined

Modify the definition of consistent answer:

Definition: A consistent answer to an aggregate query Q in the database instance D is a numerical interval that contains all the answers to Q obtained from the repairs of D

An optimal consistent answer to .... is the smallest interval $[a, b]$ such that ...

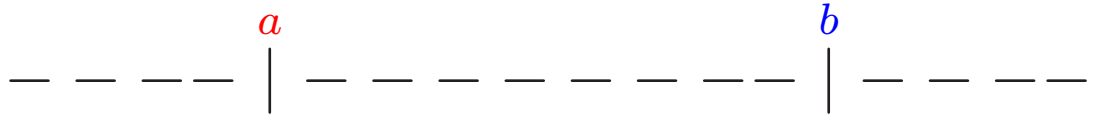
In the example:

- $[6000, 9000]$ is a consistent answer to the query `MAX(Amount)`
- $[7000, 8000]$ is an optimal consistent answer to `MAX(Amount)`

(Arenas, Bertossi, Chomicki; ICDT 01)

Problems: Find and determine

- Algorithms for computing the optimal bounds:



  - a: the max-min answer; and
  - b: the min-max answer

  By querying D only!

- Computational complexities

We need the right tools to attack these problems ...

Graph Representation of Repairs

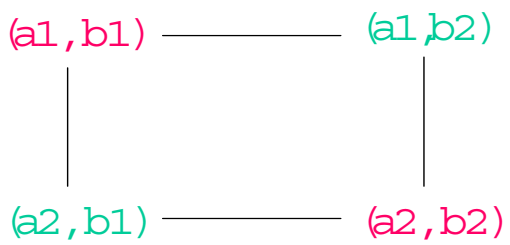
For both purposes it was crucial to appeal to a graph representation of repairs

Given a set of FDs FD and an instance D, the conflict graph $CG_{FD}(D)$ is an undirected graph:

- Vertices are the tuples $\bar{t}$ in D
- Edges are of the form $(\bar{t}_1, \bar{t}_2)$ for which there is a dependency in FD that is simultaneously violated by $\bar{t}_1, \bar{t}_2$

Example: Schema $R(A, B)$ FDs: $A \rightarrow B$ and $B \rightarrow A$

Instance $D = \{(a_1, b_1), (a_1, b_2), (a_2, b_2), (a_2, b_1)\}$



Each repair of D corresponds to a maximal independent set in $CG_{FD}(D)$

... or to a maximal clique in the complement graph

Some Complexity Results

- `MAX(A)` can be different in every repair

  Maximum of the `MAX(A)`'s is `MAX(A)` in D

  Then computing the min max-answer to `MAX(A)` from D is direct — — — — — — — — $\overset{b}{|}$ — —

- Computing directly from D the minimum of the `MAX(A)`'s, i.e. the maximal min-answer to `MAX(A)`, is not that immediate — — $\overset{a}{|}$ — — — — — —

  But still, computing the maximal min-answer to `MAX(A)` for one FD F is in PTIME (in data complexity)

- For more than one FD, the problem of deciding whether the maximal min-answer to `MAX(A)` $\leq k$ is NP-complete

  (reduction from SAT)

- Even for one FD, the problem of deciding if the maximal min-answer to `COUNT(A)` $\leq k$ is NP-complete

  (reduction from HITTING SET)

In general:

	maximal min-answer		minimal max-answer	
	$\|FD\| = 1$	$\|FD\| \geq 2$	$\|FD\| = 1$	$\|FD\| \geq 2$
`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

(Arenas,Bertossi,Chomicki,He,Raghavan,Spinrad; Th. Comp. Sci. 03)

We have identified normalization conditions, e.g. BCNF, (and other conditions) on the DB under which more efficient algorithms can be designed

However, improvements are not impressive

CQA for aggregate queries is an intrinsically complex problem

It seems necessary to approximate optimal consistent answers to aggregate queries, but "maximal independent set" seems to have bad approximation properties ...

Complexity of CQA

When the first order query rewriting approach works (correct and terminating), consistent answers to FO queries can be obtained in PTIME (data complexity)

Graph theoretic techniques for CQA for aggregate queries were extended (hypergraphs now) to:

- Extend the PTIME computation to other classes of FO queries, e.g. with very restricted forms of projection (existential quantifiers), but denial constraints
- Study the complexity of CQA for FO queries for wider classes of integrity constraints, e.g. including referential ICs (but only deletions for repair)

(Chomicki, Marcinkowski; Inf. Comp., to appear)

Some Complexity Results

In those cases where CQA can be done in PTIME, the problem of repair checking can be solved in PTIME

Repair checking is also PTIME for arbitrary FDs and acyclic inclusion dependencies (deletions only)

However: (deletions only)

- For arbitrary FDs and inclusion dependencies, repair checking becomes coNP-complete
- For arbitrary FDs and inclusion dependencies, CQA, i.e. deciding if a tuple is CA, becomes Π_2^P-complete
- (Query answering from disjunctive logic programs under skeptical stable models semantics is also Π_2^P-complete!!)

More complexity theoretic results:
(Cali, Lembo, Rosati; PODS 03)

Among others:

- For arbitrary FDs and inclusion dependencies (in particular, referential ICs), CQA becomes undecidable

Issues?

- Inclusion dependencies repaired through insertions
- Cycles in the set of inclusion dependencies
- Infinite underlying domain that can be used for insertions

Remarks:

- Complexity of query evaluation from disjunctive logic programs (DLPs) coincides with the complexity of CQA
- From this point of view the problem of CQA is not being overkilled by the use of the DLP approach
- However, it is known that for wide classes of queries and ICs, CQA has a lower complexity, e.g. in P time (Chomicki, Marcinkowski; Inf. Comp., to appear) (Fuxman, Miller; ICDT 05)
- It becomes relevant to identify classes of ICs and queries for which the DLP can be automatically "simplified" into, e.g. a FO query

  Or can be evaluated according to the Well-Founded Semantics ...

Discussion

The area of CQA in databases is an active area of research now

Many advances have been achieved since 1999

Many open problems are still open or are subject of ongoing research

- Several implementation issues, in particular in the case of most common SQL queries and constraints

  Specially for ICs that are not maintained by commercial DBMSs

- Research on many issues related to the evaluation of logic programs for consistent query answering (CQA) in the context of databases

- Existing implementations of stable models semantics are based on grounding the rules

  In database applications, this may lead to huge ground programs

- Implementations are geared to computing (some) stable model(s) and answering ground queries

  For database applications, posing and answering open queries is more natural

- Computing all the the stable models completely is undesirable

  Better try generation of "partial" repairs, relative to the ICs that are "relevant" to the query at hand

- Query evaluation based on skeptical stable model semantics should be guided by the query and its relevant information in the database
(Eiter, Fink, G.Greco, Lembo; ICLP 03)

- Magic sets (or similar query-directed methodologies) for evaluating logic programs for CQA
(S.Greco et. al)

- Optimization of the access to the DB, to the relevant portions of it

- Efficient integration of relational databases and answer set programming environments

Application Scenario: Virtual Data Integration

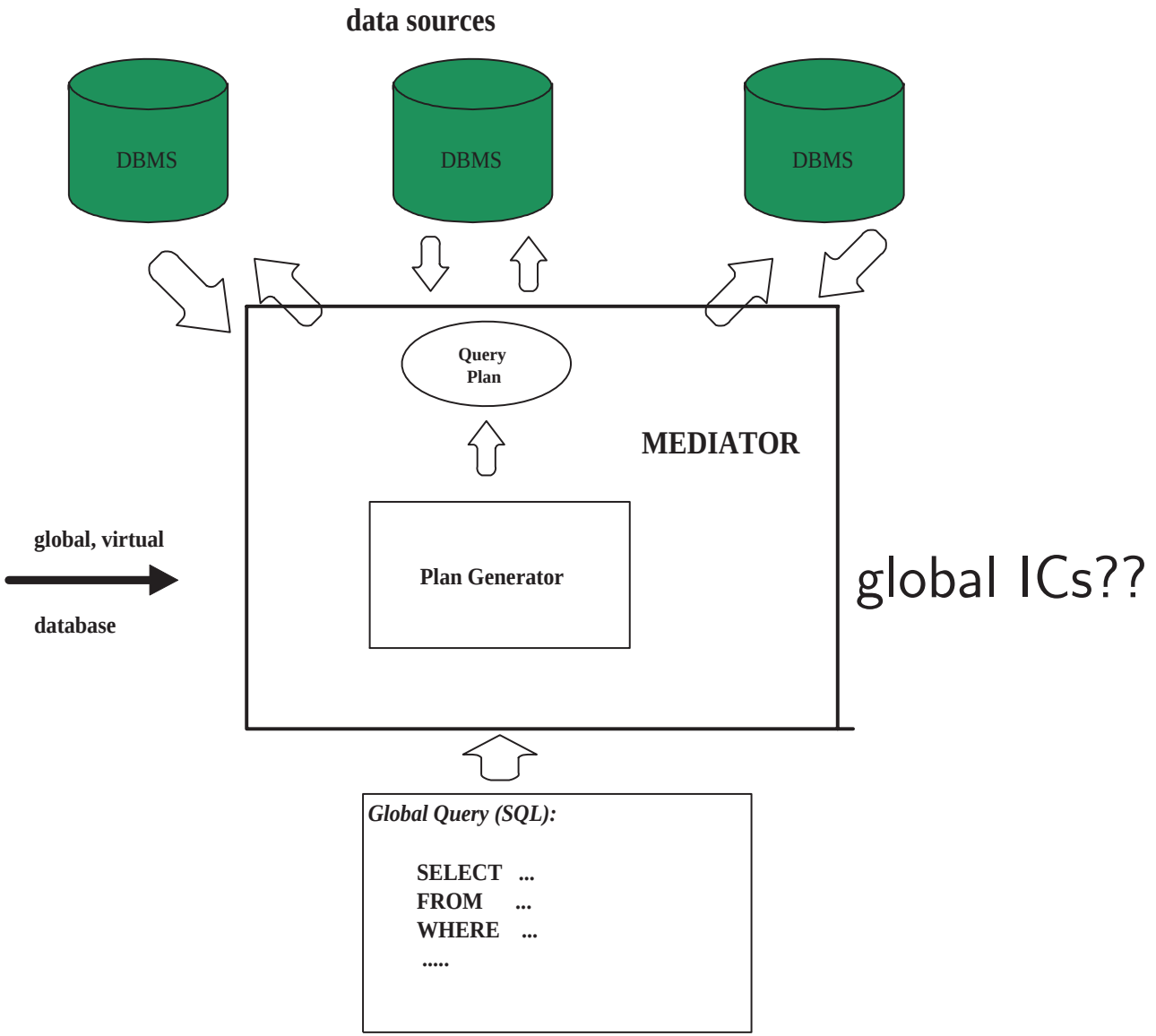
Consider a mediator-based virtual data integration system (VDIS) $\mathcal{G}$, integrating a collection of material data sources $S_1, \ldots, S_n$

Each data source has a local schema and is assumed to be consistent wrt local ICs

System $\mathcal{G}$ offers a database-like interaction schema, but data remains at the sources

Queries can be posed to $\mathcal{G}$: Given a (global) query Q to $\mathcal{G}$, a "query plan" is generated that extracts and combines information from the sources

Usually one assumes that certain ICs hold at the global level, and they are used in the generation of the query plan


data sources
DBMS
DBMS
DBMS
Query Plan
MEDIATOR
Plan Generator
global, virtual
database
Global Query (SQL):
SELECT ...
FROM ...
WHERE ...
.....

global ICs??

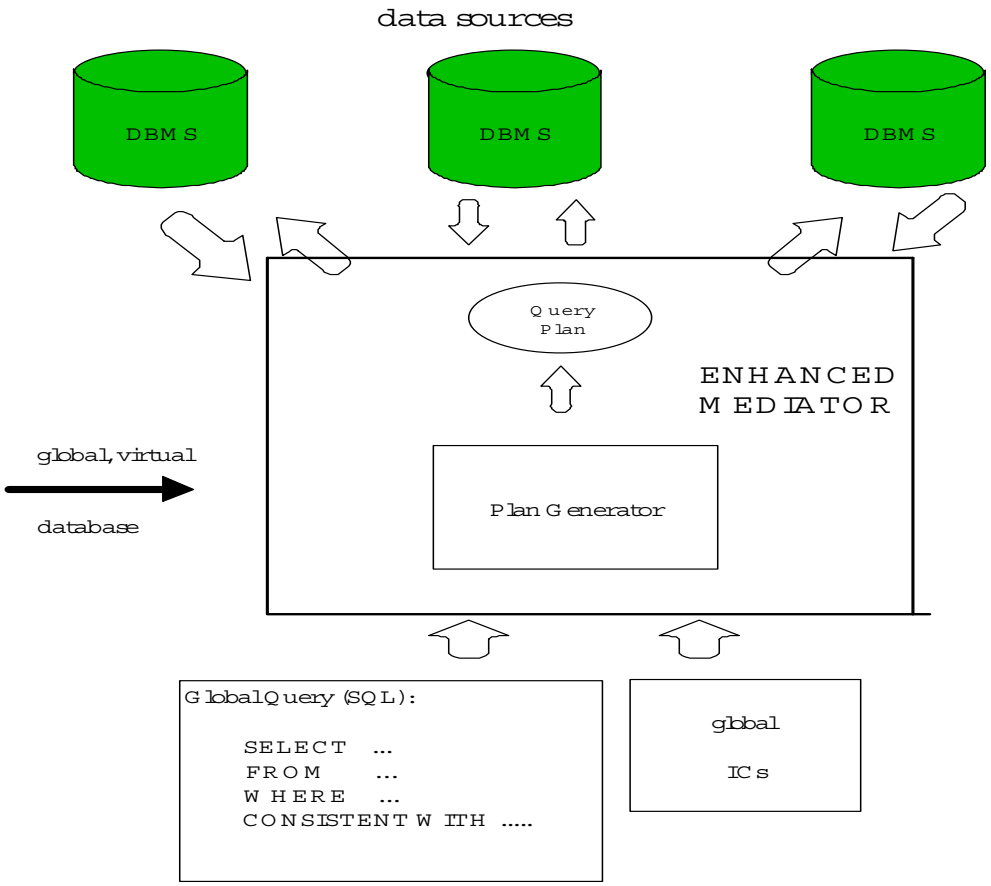
BUT, how can we be sure that such ICs hold?

They are not maintained at the global level

A natural scenario for applying CQA: retrieve only information from the global database that is consistent with IC

New issues appear:

- What is a repair of the global, virtual database?
- What is a CQA?
- How to retrieve consistent information from the global, virtual DB $\mathcal{G}$? At query time ...


data sources
DBMS
DBMS
DBMS
Query Plan
ENHANCED MEDIATOR
Plan Generator
global, virtual database
GlobalQuery (SQL):
SELECT ...
FROM ...
WHERE ...
CONSISTENT WITH .....
global ICs

Work in this direction:

- (Bravo, Bertossi; IJCAI 03)
  (Bravo, Bertossi; J. Appl. Logic, to appear)
- Extension to open, closed and clopen sources
  (Bertossi, Bravo; LNCS 3300)
- Consistency handling, repairs and different semantics for CQA under GAV
  (Lembo, Lenzerini, Rosati; KRDB 02)
  (Cali, Lembo,Rosati; IJCAI 03)
- There are clear connections between query answering in VDISs and query answering in peer-to-peer data exchange systems

Peers exchange data at query answering time according to certain data exchange constraints or data exchange mappings

No central data repository; no centralized management; data resides at peers' sites ...

Relevant literature:

- (Halevy, Ives, Suciu, Tatarinov; ICDE 03)
- (Bertossi, Bravo; P2P&DB 04)
- (Calvanese, De Giacomo, Lenzerini, Rosati; PODS 04)
- (Franconi, Kuper, Lopatenko, Zaihrayeu; P2P&DB 04)
- (Fuxman, Kolaitis, Miller, Tan; PODS 05)