

# Semantically Correct Answers from Inconsistent Databases

Leopoldo Bertossi

School of Computer Science

Carleton University

[bertossi@scs.carleton.ca](mailto:bertossi@scs.carleton.ca)

[www.scs.carleton.ca/~bertossi](http://www.scs.carleton.ca/~bertossi)

---

# The Context

We need to live with databases that are inconsistent

With information that contradicts given integrity constraints

There are many reasons, among them

- Inconsistency wrt integrity constraints that current commercial DBMS cannot check or maintain
- User constraints that cannot be checked  
A user wants or needs to impose his/her view of the world (semantics) on data that is out of his/her control

- Legacy data on which we want to impose (new) semantic constraints
- Integration of independent data sources

Each data source may be consistent and have an IC checking mechanism

But the integrated (possibly virtual, mediated) global system not ...

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 process

---

# The Problem

The inconsistent database can still give us “correct” answers to certain queries!

Not all data participates in the violation of the ICs

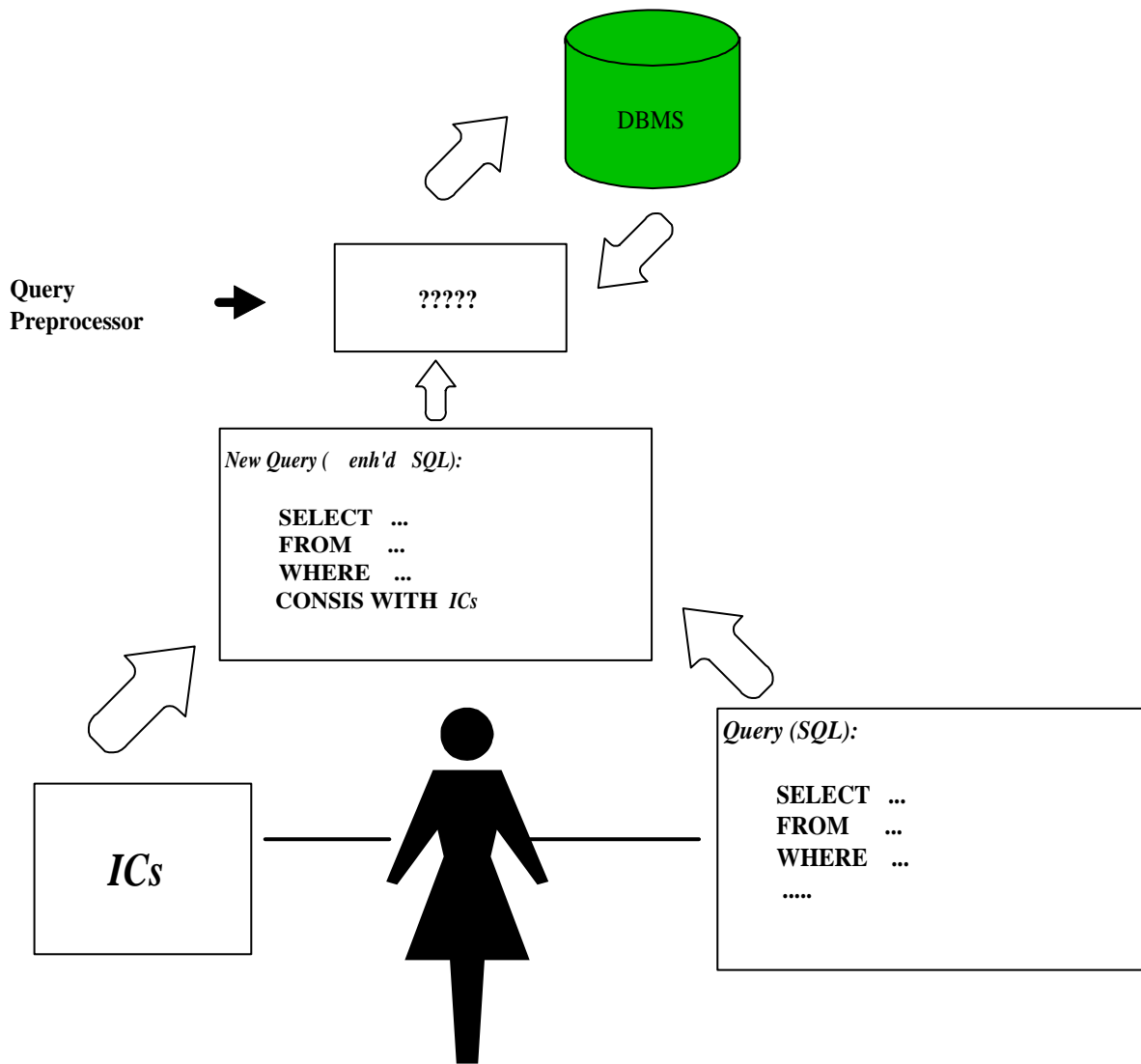
What is “correct” (“consistent”) information in an inconsistent database?

In particular, when we query the DB: what are the “correct answers”?

The research problem requires

- A precise characterization of consistent answers to a query in an inconsistent database
- Mechanisms for retrieving such consistent information from the the database

Without changing the database ...



---

## Consistent Answers

Given a database instance  $r$ , a query  $Q$ , and a set of ICs  $IC$

Tuple  $\bar{t}$  is a **consistent answer** to query  $Q$  in  $r$  wrt  $IC$  whenever  $\bar{t}$  is an answer to  $Q$  in every *repair* of  $r$

Where: a **repair** of a database instance  $r$  is a database instance  $r'$

- over the same schema and domain
- satisfies  $IC$
- differs from  $r$  by a minimal set of changes (insertions/deletions of whole tuples)

Intuitively, consistent answers are invariant under minimal ways of restoring consistency

We use repairs as an auxiliary concept, but we are not interested in repairs per se

We want to **compute** consistent answers, ideally without computing all repairs, but by querying the original instance  $r$

[Arenas, Bertossi, Chomicki. ACM PODS'99]

**Example:**  $r$  inconsistent wrt  $Name \rightarrow Salary$

<i>Employee</i>	<i>Name</i>	<i>Salary</i>
	<i>J.Page</i>	5,000
	<i>J.Page</i>	8,000
	<i>V.Smith</i>	3,000
	<i>M.Stowe</i>	7,000

<i>Repair<sub>1</sub></i>	<i>Name</i>	<i>Salary</i>
	<i>J.Page</i>	5,000
	<i>V.Smith</i>	3,000
	<i>M.Stowe</i>	7,000

<i>Repair<sub>2</sub></i>	<i>Name</i>	<i>Salary</i>
	<i>J.Page</i>	8,000
	<i>V.Smith</i>	3,000
	<i>M.Stowe</i>	7,000

In  $r$  it is consistently true that

- $Employee(M.Stowe, 7,000)$
- $Employee(J.Page, 5,000) \vee Employee(J.Page, 8,000)$
- $\exists X Employee(J.Page, X)$

---

# Addressing the Problem I

A computational mechanism to compute consistent answers

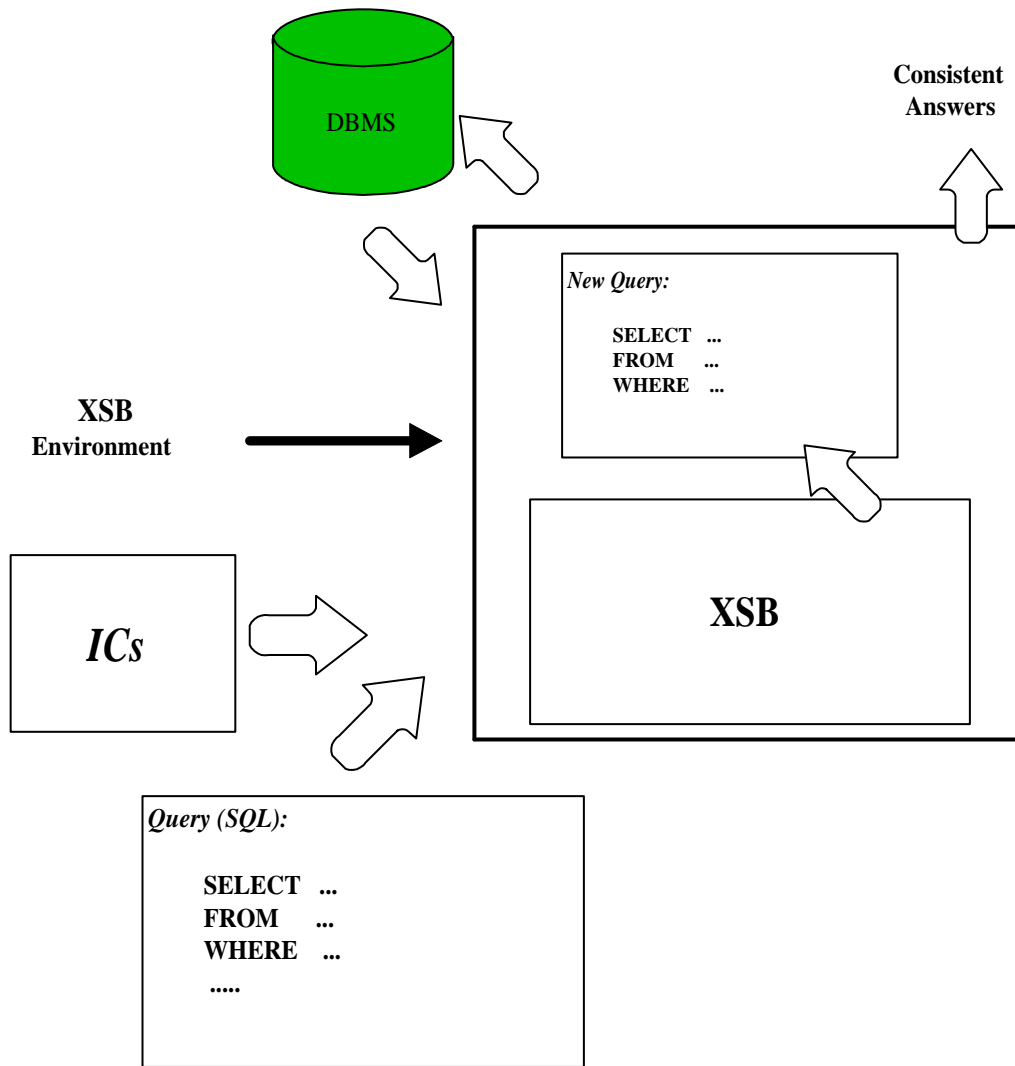
Does not produce the repairs

It queries the only explicitly available database instance

Query is **transformed** and posed as new query

Implementation on top of XSB, a deductive database system, connected to DB2 via ODBC

Input is an SQL query, the algorithm (implemented in XSB) produces a new SQL query that is posed to the DB2 DB



**Example:** The FD:  $Name \rightarrow Salary$  can be written

$$\forall XYZ (\neg Employee(X, Y) \vee \neg Employee(X, Z) \vee Y = Z)$$

Query:  $Employee(X, Y)$ ?

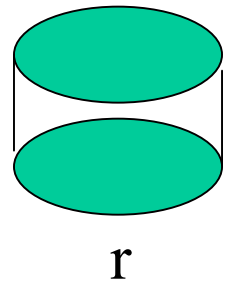
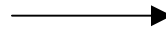
Consistent answers:  $(V.Smith, 3,000)$ ,  $(M.Stowe, 7,000)$   
(but not  $(J.Page, 5,000)$ ,  $(J.Page, 8,000)$ )

Can be obtained by means of the transformed query

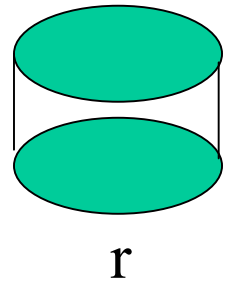
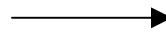
$$T(Employee(X, Y)) := Employee(X, Y) \wedge \\ \forall Z (\neg Employee(X, Z) \vee Y = Z)$$

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

**SELECT Name, Salary  
FROM Employee  
CONSISTENT WITH  
FD(Name;Salary)**



**SELECT Name, Salary  
FROM Employee E  
WHERE Not exists (  
SELECT E.Salary  
FROM E  
WHERE E.Name = Name  
AND E.Salary <> Salary)**



Ordinary answers to new query are the consistent answers to the original query

[Arenas, Bertossi, Chomicki. PODS'99]

[Celle, Bertossi. DOOD'2000]

Methodology based on query transformation restricted to:

- Certain SQL queries, essentially conjunctions of DB tables
- Certain ICs, essentially universal ICs

This covers most ICs found in DB praxis, except for referential ICs

- More expressive queries? Referential ICs?

---

## Addressing the Problem II

Represent in a compact form the collection of all database repairs

Use disjunctive logic (answer set) programs

Repairs correspond to certain distinguished models of the program

To obtain consistent answers to a FO SQL query:

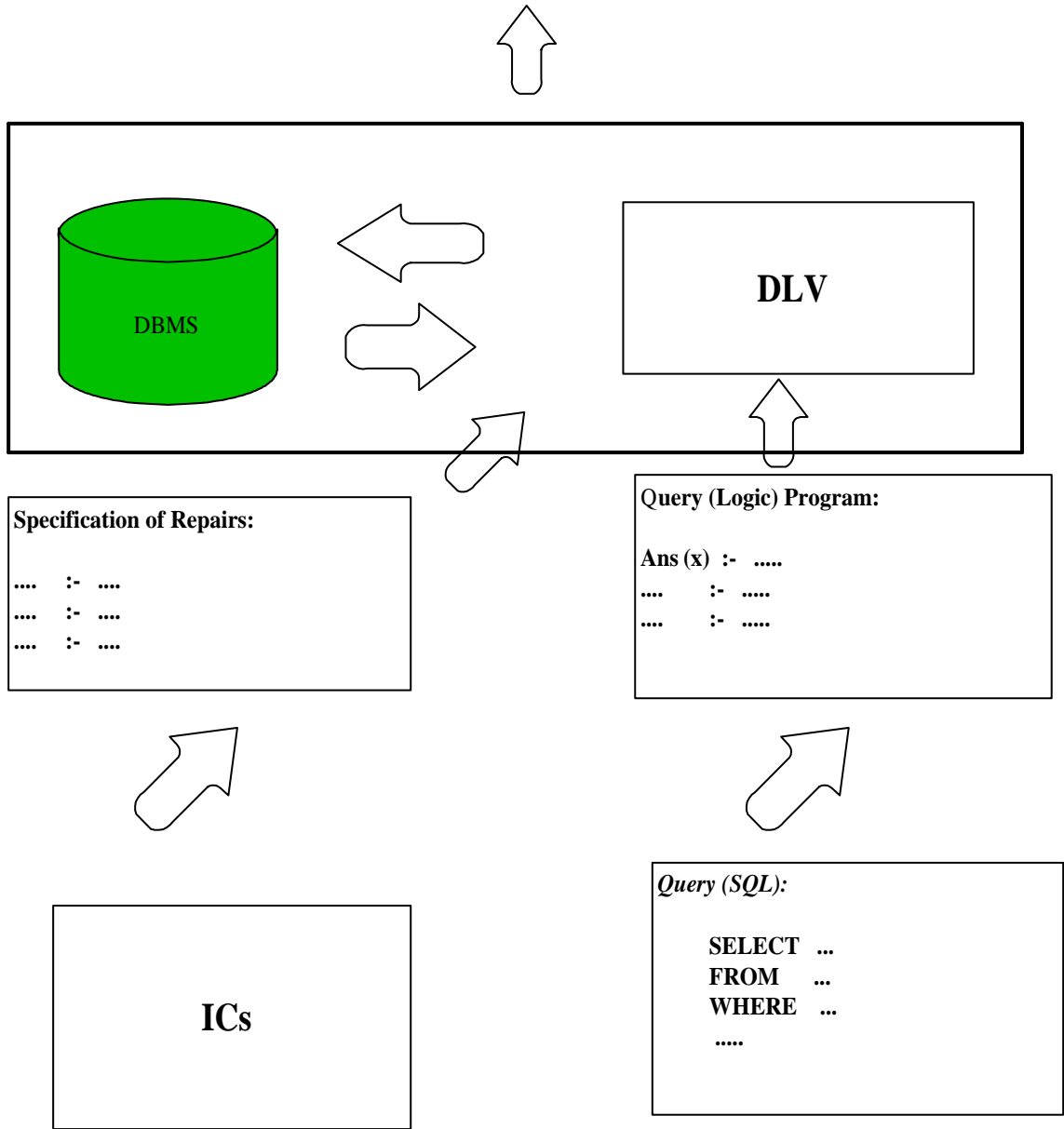
- Transform (internally) the query into a logic program (standard)
- Run that program together with the program that specifies the repairs

Can be implemented on top of DLV, a logic programming system with essentially a stable models semantics that computes the desired models

[Arenas, Bertossi, Chomicki. To appear in TPLP]

[Barcelo, Bertossi. NMR'02, PADL'03]

# Consistent Answers



**Example:** Full inclusion dependency

$$IC: \forall \bar{x} (P(\bar{x}) \rightarrow Q(\bar{x}))$$

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

The programs use annotation constants

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
$\mathbf{t}^*$	$P(\bar{a}, \mathbf{t}^*)$	true or becomes true
$\mathbf{f}^*$	$P(\bar{a}, \mathbf{f}^*)$	false or becomes false
$\mathbf{t}^{**}$	$P(\bar{a}, \mathbf{t}^{**})$	it is true in the repair
$\mathbf{f}^{**}$	$P(\bar{a}, \mathbf{f}^{**})$	it is false in the repair

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

1.  $P(\bar{c}, \mathbf{t}_d) \leftarrow$   
 $P(\bar{d}, \mathbf{t}_d) \leftarrow$   
 $Q(\bar{d}, \mathbf{t}_d) \leftarrow$   
 $Q(\bar{e}, \mathbf{t}_d) \leftarrow$

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

2.  $P(\bar{x}, \mathbf{t}^*) \leftarrow P(\bar{x}, \mathbf{t}_d)$   
 $P(\bar{x}, \mathbf{t}^*) \leftarrow P(\bar{x}, \mathbf{t}_a)$   
 $P(\bar{x}, \mathbf{f}^*) \leftarrow \text{not } P(\bar{x}, \mathbf{t}_d)$   
 $P(\bar{x}, \mathbf{f}^*) \leftarrow P(\bar{x}, \mathbf{f}_a)$

... the same for  $Q$  ...

$$3. \quad P(\bar{x}, \mathbf{f}_a) \vee Q(\bar{x}, \mathbf{t}_a) \leftarrow P(\bar{x}, \mathbf{t}^*), Q(\bar{x}, \mathbf{f}^*)$$

One rule per IC; that says how to repair the IC

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

Repairs must be *coherent*: use denial constraints at the program level, to prune some models

$$4. \quad \leftarrow P(\bar{x}, \mathbf{t}_a), P(\bar{x}, \mathbf{f}_a) \\ \leftarrow Q(\bar{x}, \mathbf{t}_a), Q(\bar{x}, \mathbf{f}_a)$$

Finally, annotations constants  $\mathbf{t}^{**}$  and  $\mathbf{f}^{**}$  are used to read off the literals that are inside (outside) a repair

5.  $P(\bar{x}, \mathbf{t}^{**}) \leftarrow P(\bar{x}, \mathbf{t}_a)$   
 $P(\bar{x}, \mathbf{t}^{**}) \leftarrow P(\bar{x}, \mathbf{t}_d), \text{ not } P(\bar{x}, \mathbf{f}_a)$   
 $P(\bar{x}, \mathbf{f}^{**}) \leftarrow P(\bar{x}, \mathbf{f}_a)$   
 $P(\bar{x}, \mathbf{f}^{**}) \leftarrow \text{not } P(\bar{x}, \mathbf{t}_d), \text{ not } P(\bar{x}, \mathbf{t}_a). \dots \text{ etc.}$

Used to interpret the models as database repairs

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

$$\{P(\bar{c}, \mathbf{t}_d), \dots, P(\bar{c}, \mathbf{t}^*), Q(\bar{c}, \mathbf{f}^*), Q(\bar{c}, \mathbf{t}_a), P(\bar{c}, \mathbf{t}^{**}), Q(\bar{c}, \mathbf{t}^*), Q(\bar{c}, \mathbf{t}^{**}), \dots\} \equiv \{P(\bar{c}), Q(\bar{c}), P(\bar{d}), Q(\bar{d}), Q(\bar{e})\}$$

$$\{P(\bar{c}, \mathbf{t}_d), \dots, P(\bar{c}, \mathbf{t}^*), P(\bar{c}, \mathbf{f}^*), Q(\bar{c}, \mathbf{f}^*), P(\bar{c}, \mathbf{f}^{**}), Q(\bar{c}, \mathbf{f}^{**}), P(\bar{c}, \mathbf{f}_a), \dots\} \equiv \{P(\bar{d}), Q(\bar{d}), Q(\bar{e})\}$$

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

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

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

$$Answer = \emptyset$$

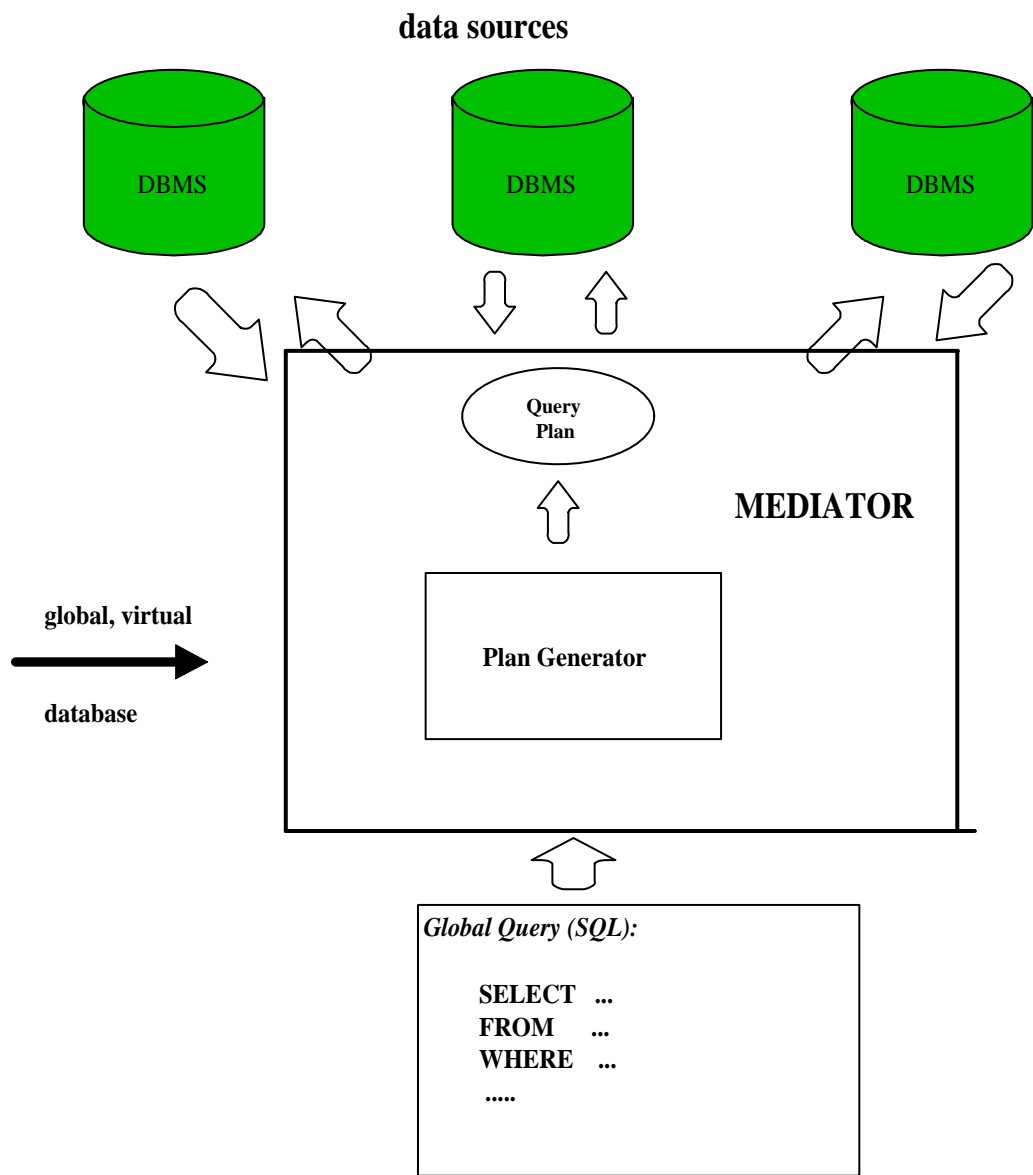
---

# Data Integration

Given a collection of (materialized) data sources  $S_1, \dots, S_n$ , and a global, virtual database  $\mathcal{G}$ , that integrates the data sources

Given a (global) query  $Q$  to  $\mathcal{G}$ , one can generate a *query plan* that extracts and combines the information from the sources

One approach to virtual data integration is the *local as view approach*: data sources are specified as views over the global schema



**Example of the LAV approach:** Sources:

$S_1 : V_1(\textit{title}, \textit{year}, \textit{director}) \leftarrow$   
 $\textit{Movie}(\textit{title}, \textit{year}, \textit{director}, \textit{genre}) \ \&$   
 $\textit{American}(\textit{director}) \ \& \ \textit{year} \geq 1960 \ \&$   
 $\textit{genre} = \textit{Comedy}.$

$S_2 : V_2(\textit{title}, \textit{review}) \leftarrow$   
 $\textit{Movie}(\textit{title}, \textit{year}, \textit{director}, \textit{genre}) \ \&$   
 $\textit{year} \geq 1990 \ \& \ \textit{Review}(\textit{title}, \textit{review}).$

Then, global schema  $G$ :  $\textit{Movie}, \textit{Review}, \textit{American}$

Query to  $G$ : Comedies w/reviews since 1950?

$q(\textit{title}, \textit{review}) \leftarrow$   
 $\textit{Movie}(\textit{title}, \textit{year}, \textit{director}, \textit{director}, \textit{Comedy})$   
 $\& \textit{year} \geq 1950 \& \textit{Review}(\textit{title}, \textit{review})$

Information is in the sources, now, views ...

A query plan:

$q'(\textit{title}, \textit{review}) \leftarrow$   
 $V_1(\textit{title}, \textit{year}, \textit{director}) \& V_2(\textit{title}, \textit{review})$

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

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

They are not maintained at the global level

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

New issues appear:

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

At query time ...

[Bertossi, Chomicki, Cortes, Gutierrez. FQAS'02]:

- (Partial) solution under the LAV approach
- Characterization of (global) consistent query answers
- Given a (conjunctive) SQL query, the query is transformed into a new SQL query
- The new query is transformed into a query written as a logic program
- The logic program query is used as an input for our extension of the *inverse rules algorithm*
- A *query plan* is produced to compute the final (consistent) answers

---

## Ongoing and Future Work

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

Specially those 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

- Optimization of the logic programs for CQA
- Magic sets (or similar query-directed methodologies) for evaluation of logic programs for CQA
- Generation of “partial” repairs, relative to the ICs that are “relevant” to the query at hand
- Efficient integration of databases (DB2) and logic programs (XSB, DLV)
- Implementation under the local-as-view approach of CQS for more expressive queries and ICs