Consistent Query Answering: Five Easy Pieces

Jan Chomicki

University at Buffalo and Warsaw University

11th International Conference on Database Theory
Barcelona, January 11, 2007
When was Alberto Mendelzon born?

When was Alberto Mendelzon born?

1951 (Renée Miller, SIGMOD Record 2005)
1953 (Leonid Libkin, ICDT 2007)

Inconsistencies cannot both be right; but, imputed to man, they may both be true.

Samuel Johnson
When was Alberto Mendelzon born?

1951 (Renée Miller, SIGMOD Record 2005)
When was Alberto Mendelzon born?

1951 (Renée Miller, SIGMOD Record 2005)

1953 (Leonid Libkin, ICDT 2007)
When was Alberto Mendelzon born?

1951 (Renée Miller, SIGMOD Record 2005)

1953 (Leonid Libkin, ICDT 2007)

Inconsistencies cannot both be right; but, imputed to man, they may both be true.

Samuel Johnson
Inconsistent Databases

**Database instance** $D$:
- A finite first-order structure
- The information about the world
Inconsistent Databases

Database instance $D$:
- a finite first-order structure
- the information about the world

Integrity constraints $IC$:
- first-order logic formulas
- the properties of the world
### Inconsistent Databases

**Database instance $D$:**
- a finite first-order **structure**
- the **information** about the world

**Integrity constraints $IC$:**
- first-order logic **formulas**
- the **properties** of the world

**Satisfaction of constraints:** $D \models IC$

Formula satisfaction in a first-order structure.
Inconsistent Databases

Database instance $D$:  
- a finite first-order structure  
- the information about the world

Integrity constraints $IC$:  
- first-order logic formulas  
- the properties of the world

Satisfaction of constraints: $D \models IC$

Formula satisfaction in a first-order structure.

Inconsistent database: $D \not\models IC$
Inconsistent Databases

**Database instance** $D$:  
- a finite first-order structure  
- the information about the world

**Integrity constraints** $IC$:  
- first-order logic formulas  
- the properties of the world

**Satisfaction of constraints**: $D \models IC$

Formula satisfaction in a first-order structure.

**Inconsistent database**: $D \not\models IC$

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ City Salary
Whence Inconsistency?

Sources of inconsistency:

- integration of independent data sources with overlapping data
- time lag of updates (eventual consistency)
- unenforced integrity constraints
- dataspace systems,...
Sources of inconsistency:
- integration of independent data sources with overlapping data
- time lag of updates (eventual consistency)
- unenforced integrity constraints
- dataspace systems,...

Eliminating inconsistency?
- not enough information, time, or money
- difficult, impossible or undesirable
- unnecessary: queries may be insensitive to inconsistency
Query results *not reliable*. 
Query results **not reliable**.
Query results **not reliable**.

```
SELECT Name
FROM Employee
WHERE Salary \leq 25M
```

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>
Ignoring Inconsistency

Query results **not reliable**.

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

**SELECT Name**  
**FROM Employee**  
**WHERE Salary \( \leq \) 25M**  

Name → City Salary
Horizontal Decomposition

Decomposition into two relations:
- violators
- the rest

Horizontal Decomposition

Decomposition into two relations:
- violators
- the rest


<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary
Horizontal Decomposition

Decomposition into two relations:
- violators
- the rest


<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary

Grove    San Francisco    10M
Name → City Salary

Gates    Redmond    20M
Gates    Redmond    30M
Name → City Salary
Exceptions to Constraints

Weakening the constraints:
- functional dependencies → denial constraints

[Borgida: TODS’85]
Exceptions to Constraints

Weakening the constraints:
- functional dependencies → denial constraints

Name | City     | Salary |
-----|----------|--------|
Gates| Redmond  | 20M    |
Gates| Redmond  | 30M    |
Grove| Santa Clara | 10M   |

Name → City Salary

[Borgida: TODS’85]
Exceptions to Constraints

Weakening the constraints:
- functional dependencies → denial constraints

[Borgida: TODS’85]

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary except Name='Gates'
The Impact of Inconsistency on Queries

Traditional view

- query results defined irrespective of integrity constraints
- query evaluation may be optimized in the presence of integrity constraints (semantic query optimization)
The Impact of Inconsistency on Queries

Traditional view
- query results defined irrespective of integrity constraints
- query evaluation may be optimized in the presence of integrity constraints (semantic query optimization)

“Post-modernist” view
- inconsistency reflects uncertainty
- query results may depend on integrity constraint satisfaction
- inconsistency may be eliminated or tolerated
Database Repairing

Restoring consistency:
- insertion, deletion, update
- minimal change?
- information loss?
Database Repairing

Restoring consistency:
- insertion, deletion, update
- minimal change?
- information loss?

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary
Database Repairing

**Restoring consistency:**
- insertion, deletion, update
- minimal change?
- information loss?

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary

**Name → City Salary**

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

Name → City Salary
Consistent query answer:
Query answer obtained in every repair.

[Arenas, Bertossi, Ch.: PODS'99]
Consistent Query Answering

Consistent query answer:

Query answer obtained in every repair.

[Arenas, Bertossi, Ch.: PODS’99]

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>
Consistent Query Answering

Consistent query answer:
Query answer obtained in every repair.

[Areñas, Bertossi, Ch.: PODS’99]

Name | City     | Salary |
-----|----------|--------|
Gates| Redmond  | 20M    |
Gates| Redmond  | 30M    |
Grove| Santa Clara | 10M   |

SELECT Name FROM Employee WHERE Salary ≤ 25M

Name → City Salary
Consistent Query Answering

**Consistent query answer:**
Query answer obtained in *every repair*.

**[Arenas, Bertossi, Ch.: PODS'99]**

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>20M</td>
</tr>
<tr>
<td>Gates</td>
<td>Redmond</td>
<td>30M</td>
</tr>
<tr>
<td>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>

```
SELECT Name
FROM Employee
WHERE Salary >= 10M
```

Name → City Salary

Name
- Gates
- Grove
1. Motivation

2. Outline

3. Basics

4. Computing CQA
   - Methods
   - Complexity

5. Variants of CQA

6. Conclusions
Formal definition

What constitutes reliable (consistent) information in an inconsistent database.
Research Goals

Formal definition

What constitutes reliable (consistent) information in an inconsistent database.

Algorithms

How to compute consistent information.
### Research Goals

<table>
<thead>
<tr>
<th>Formal definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>What constitutes reliable (consistent) information in an inconsistent database.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to <em>compute</em> consistent information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computational complexity analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>tractable</strong> vs. intractable classes of queries and integrity constraints</td>
</tr>
<tr>
<td>• tradeoffs: complexity vs. expressiveness.</td>
</tr>
</tbody>
</table>
Research Goals

Formal definition
What constitutes reliable (consistent) information in an inconsistent database.

Algorithms
How to compute consistent information.

Computational complexity analysis
- tractable vs. intractable classes of queries and integrity constraints
- tradeoffs: complexity vs. expressiveness.

Implementation
- preferably using DBMS technology.
Research Goals

Formal definition
What constitutes reliable (consistent) information in an inconsistent database.

Algorithms
How to compute consistent information.

Computational complexity analysis
- tractable vs. intractable classes of queries and integrity constraints
- tradeoffs: complexity vs. expressiveness.

Implementation
- preferably using DBMS technology.

Applications
???
Basic Notions

Repair $D'$ of a database $D$ w.r.t. the integrity constraints $IC$:

- $D'$: over the same schema as $D$
- $D' \models IC$
- symmetric difference between $D$ and $D'$ is minimal.
Basic Notions

**Repair** $D'$ of a database $D$ w.r.t. the integrity constraints $IC$:
- $D'$: over the same schema as $D$
- $D' \models IC$
- symmetric difference between $D$ and $D'$ is minimal.

**Consistent query answer** to a query $Q$ in $D$ w.r.t. $IC$:
- an element of the result of $Q$ in every repair of $D$ w.r.t. $IC$. 

Another incarnation of the idea of sure query answers [Lipski: TODS'79].
Basic Notions

Repair $D'$ of a database $D$ w.r.t. the integrity constraints $IC$:

- $D'$: over the same schema as $D$
- $D' \models IC$
- symmetric difference between $D$ and $D'$ is minimal.

Consistent query answer to a query $Q$ in $D$ w.r.t. $IC$:

- an element of the result of $Q$ in every repair of $D$ w.r.t. $IC$.

Another incarnation of the idea of sure query answers [Lipski: TODS’79].
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.
Exponentially many repairs

Example relation $R(A, B)$
- violates the dependency $A \rightarrow B$
- has $2^n$ repairs.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$a_n$</td>
<td>$b_n$</td>
</tr>
<tr>
<td>$a_n$</td>
<td>$c_n$</td>
</tr>
</tbody>
</table>

$A \rightarrow B$
Exponentially many repairs

Example relation $R(A, B)$

- violates the dependency $A \rightarrow B$
- has $2^n$ repairs.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$a_n$</td>
<td>$b_n$</td>
</tr>
<tr>
<td>$a_n$</td>
<td>$c_n$</td>
</tr>
</tbody>
</table>

$A \rightarrow B$

It is impractical to apply the definition of CQA directly.
Computing Consistent Query Answers

Query Rewriting

Given a query $Q$ and a set of integrity constraints $IC$, build a query $Q^{IC}$ such that for every database instance $D$

$$\text{the set of answers to } Q^{IC} \text{ in } D = \text{the set of consistent answers to } Q \text{ in } D \text{ w.r.t. } IC.$$
Computing Consistent Query Answers

**Query Rewriting**

Given a query $Q$ and a set of integrity constraints $IC$, build a query $Q^{IC}$ such that for every database instance $D$

\[
\text{the set of answers to } Q^{IC} \text{ in } D = \text{the set of consistent answers to } Q \text{ in } D \text{ w.r.t. } IC.
\]

**Representing all repairs**

Given $IC$ and $D$:

1. build a space-efficient representation of all repairs of $D$ w.r.t. $IC$
2. use this representation to answer (many) queries.
Computing Consistent Query Answers

**Query Rewriting**

Given a query $Q$ and a set of integrity constraints $IC$, build a query $Q^{IC}$ such that for every database instance $D$

\[
\text{the set of answers to } Q^{IC} \text{ in } D = \text{the set of consistent answers to } Q \text{ in } D \text{ w.r.t. } IC.
\]

**Representing all repairs**

Given $IC$ and $D$:

1. build a space-efficient representation of all repairs of $D$ w.r.t. $IC$
2. use this representation to answer (many) queries.

**Logic programs**

Given $IC$, $D$ and $Q$:

1. build a logic program $P_{IC,D}$ whose models are the repairs of $D$ w.r.t. $IC$
2. build a logic program $P_Q$ expressing $Q$
3. use a logic programming system that computes the query atoms present in all models of $P_{IC,D} \cup P_Q$. 
Universal constraints

∀. ¬A_1 ∨ ⋯ ∨ ¬A_n ∨ B_1 ∨ ⋯ ∨ B_m
### Constraint classes

<table>
<thead>
<tr>
<th>Universal constraints</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀. ¬A_1 ∨ ⋯ ∨ ¬A_n ∨ B_1 ∨ ⋯ ∨ B_m</td>
<td>∀. ¬Par(x) ∨ Ma(x) ∨ Fa(x)</td>
</tr>
</tbody>
</table>
Constraint classes

### Universal constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

### Denial constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \]

### Example
\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]
Constraint classes

**Universal constraints**

\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

**Example**

\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]

**Denial constraints**

\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \]

**Example**

\[ \forall. \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t \]
Constraint classes

Universal constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

Example
\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]

Denial constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \]

Example
\[ \forall. \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t \]

Functional dependencies
\[ X \rightarrow Y: \]
- a key dependency in \( F \) if \( X \) is a key
- a primary-key dependency: only one key exists
Constraint classes

Universal constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

Example
\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]

Denial constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \]

Example
\[ \forall. \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t \]

Functional dependencies
\[ X \rightarrow Y: \]
- a key dependency in \( F \) if \( X \) is a key
- a primary-key dependency: only one key exists

Example primary-key dependency
Name \rightarrow Address Salary
**Constraint classes**

**Universal constraints**

\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

**Example**

\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]

**Denial constraints**

\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \]

**Example**

\[ \forall. \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t \]

**Functional dependencies**

\[ X \rightarrow Y: \]

- a **key** dependency in \( F \) if \( X \) is a key
- a **primary-key** dependency: only one key exists

**Example primary-key dependency**

Name \( \rightarrow \) Address Salary

**Inclusion dependencies**

\[ R[X] \subseteq S[Y]: \]

- a **foreign key** constraint if \( Y \) is a key of \( S \)
### Constraint classes

#### Universal constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m \]

**Example**
\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]

#### Denial constraints
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \]

**Example**
\[ \forall. \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t \]

#### Functional dependencies
\[ X \rightarrow Y: \]
- a **key** dependency in \( F \) if \( X \) is a key
- a **primary-key** dependency: only one key exists

**Example primary-key dependency**
Name \( \rightarrow \) Address Salary

#### Inclusion dependencies
\[ R[X] \subseteq S[Y]: \]
- a **foreign key** constraint if \( Y \) is a key of \( S \)

**Example foreign key constraint**
\( M[Manager] \subseteq M[Name] \)
Query Rewriting

Building queries that compute CQAs

- relational calculus (algebra) $\leadsto$ relational calculus (algebra)
- SQL $\leadsto$ SQL
- leads to \textbf{PTIME} data complexity
Query Rewriting

Building queries that compute CQAs
- relational calculus (algebra) $\mapsto$ relational calculus (algebra)
- SQL $\mapsto$ SQL
- leads to $\text{PTIME}$ data complexity

Query
$\text{Emp}(x, y, z)$
Query Rewriting

Building queries that compute CQAs
- relational calculus (algebra) $\leadsto$ relational calculus (algebra)
- SQL $\leadsto$ SQL
- leads to $\text{PTIME}$ data complexity

Query

$\text{Emp}(x, y, z)$

Integrity constraint

$\forall x, y, z, y', z'. \neg \text{Emp}(x, y, z) \lor \neg \text{Emp}(x, y', z') \lor z = z'$
### Query Rewriting

#### Building queries that compute CQAs
- relational calculus (algebra) $\iff$ relational calculus (algebra)
- SQL $\iff$ SQL
- leads to PTIME data complexity

#### Query

$Emp(x, y, z)$

#### Integrity constraint

$\forall x, y, z, y', z'. \neg Emp(x, y, z) \lor \neg Emp(x, y', z') \lor z = z'$
Query Rewriting

Building queries that compute CQAs
- relational calculus (algebra) $\sim$ relational calculus (algebra)
- SQL $\sim$ SQL
- leads to $\text{PTIME}$ data complexity

Query
\[
\text{Emp}(x, y, z)
\]

Integrity constraint
\[
\forall x, y, z, y', z'. \neg \text{Emp}(x, y, z) \lor \neg \text{Emp}(x, y', z') \lor z = z'
\]

Rewritten query
\[
\text{Emp}(x, y, z) \land \forall y', z'. \neg \text{Emp}(x, y', z') \lor z = z'
\]
The Scope of Query Rewriting

[Arenas, Bertossi, Ch.: PODS’99]

- Queries: conjunctions of literals (relational algebra: $\sigma, \times, -$)
- Integrity constraints: binary universal
The Scope of Query Rewriting

[Arenas, Bertossi, Ch.: PODS’99]
- Queries: conjunctions of literals (relational algebra: $\sigma, \times, -$)
- Integrity constraints: binary universal

[Fuxman, Miller: ICDT’05]
- Queries: $C_{forest}$
  - a class of conjunctive queries ($\pi, \sigma, \times$)
  - no non-key or non-full joins
  - no repeated relation symbols
  - no built-ins
- Integrity constraints: primary key functional dependencies
SQL Rewriting

SQL query

SELECT Name FROM Emp
WHERE Salary ≥ 10K

[SQL rewritten query]

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

[Fuxman, Fazli, Miller: SIGMOD'05]
SQL Rewriting

**SQL query**

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

**SQL rewritten query**

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

[Fuxman, Fazli, Miller: SIGMOD'05] ConQuer: a system for computing CQA conjunctive (C forest) and aggregation SQL queries databases can be annotated with consistency indicators tested on TPC-H queries and medium-size databases
SQL query

SELECT Name FROM Emp
WHERE Salary ≥ 10K

SQL rewritten query

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

[Fuxman, Fazli, Miller: SIGMOD'05]

- **ConQuer**: a system for computing CQAs
- conjunctive \( (C_{forest}) \) and aggregation SQL queries
- databases can be annotated with consistency indicators
- tested on TPC-H queries and medium-size databases
Conflict Hypergraph

**Vertices**

Tuples in the database.

(Gates, Redmond, 20M)

(Grove, Santa Clara, 10M)

(Gates, Redmond, 30M)
Conflict Hypergraph

**Vertices**
- Tuples in the database.

**Edges**
- Minimal sets of tuples violating a constraint.

(Gates, Redmond, 20M)

(Grove, Santa Clara, 10M)

(Gates, Redmond, 30M)
Conflict Hypergraph

- **Vertices**: Tuples in the database.
- **Edges**: Minimal sets of tuples violating a constraint.
- **Repairs**: Maximal independent sets in the conflict graph.

(Gates, Redmond, 20M)

(Grove, Santa Clara, 10M)

(Gates, Redmond, 30M)
Conflict Hypergraph

**Vertices**
Tuples in the database.

**Edges**
Minimal sets of tuples violating a constraint.

**Repairs**
Maximal independent sets in the conflict graph.

(Gates, Redmond, 20M)
(Grove, Santa Clara, 10M)
(Gates, Redmond, 30M)
Computing CQAs Using Conflict Hypergraphs

Algorithm HProver

INPUT: query $\Phi$ a disjunction of ground atoms, conflict hypergraph $G$
OUTPUT: is $\Phi$ false in some repair of $D$ w.r.t. $IC$?

ALGORITHM:

1. $\neg \Phi = P_1(t_1) \land \cdots \land P_m(t_m) \land \neg P_{m+1}(t_{m+1}) \land \cdots \land \neg P_n(t_n)$

2. find a consistent set of facts $S$ such that

   - $S \supseteq \{P_1(t_1), \ldots, P_m(t_m)\}$
   - for every fact $A \in \{P_{m+1}(t_{m+1}), \ldots, P_n(t_n)\}$: $A \not\in D$ or there is an edge $E = \{A, B_1, \ldots, B_m\}$ in $G$ and $S \supseteq \{B_1, \ldots, B_m\}$. 

[Ch., Marcinkowski, Staworko: CIKM’04]

Hippo: a system for computing CQAs in PTIME
quantifier-free queries and denial constraints
only edges of the conflict hypergraph are kept in main memory
optimization can eliminate many (sometimes all) database accesses in
HProver
tested for medium-size synthetic databases
Computing CQAs Using Conflict Hypergraphs

Algorithm HProver

INPUT: query \( \Phi \) a disjunction of ground atoms, conflict hypergraph \( G \)
OUTPUT: is \( \Phi \) false in some repair of \( D \) w.r.t. \( IC \)?
ALGORITHM:

1. \( \neg \Phi = P_1(t_1) \land \cdots \land P_m(t_m) \land \neg P_{m+1}(t_{m+1}) \land \cdots \land \neg P_n(t_n) \)

2. find a consistent set of facts \( S \) such that
   - \( S \supseteq \{P_1(t_1), \ldots, P_m(t_m)\} \)
   - for every fact \( A \in \{P_{m+1}(t_{m+1}), \ldots, P_n(t_n)\} \): \( A \not\in D \) or there is an edge \( E = \{A, B_1, \ldots, B_m\} \) in \( G \) and \( S \supseteq \{B_1, \ldots, B_m\} \).

[Ch., Marcinkowski, Staworko: CIKM’04]

- **Hippo**: a system for computing CQAs in PTIME
- quantifier-free queries and denial constraints
- only edges of the conflict hypergraph are kept in main memory
- optimization can eliminate many (sometimes all) database accesses in HProver
- tested for medium-size synthetic databases
Specifying repairs as answer sets of logic programs

- [Arenas, Bertossi, Ch.: FQAS’00, TPLP’03]
- [Greco, Greco, Zumpano: LPAR’00, TKDE’03]
- [Calì, Lembo, Rosati: IJCAI’03]
Specifying repairs as answer sets of logic programs

- [Arenas, Bertossi, Ch.: FQAS’00, TPLP’03]
- [Greco, Greco, Zumpano: LPAR’00, TKDE’03]
- [Calì, Lembo, Rosati: IJCAI’03]

Example

\begin{align*}
\text{emp}(x, y, z) & \leftarrow \text{emp}_D(x, y, z), \text{not dubious}_\text{emp}(x, y, z). \\
\text{dubious}_\text{emp}(x, y, z) & \leftarrow \text{emp}_D(x, y, z), \text{emp}(x, y', z'), y \neq y'. \\
\text{dubious}_\text{emp}(x, y, z) & \leftarrow \text{emp}_D(x, y, z), \text{emp}(x, y', z'), z \neq z'.
\end{align*}
Logic programs

Specifying repairs as answer sets of logic programs

- [Arenas, Bertossi, Ch.: FQAS’00, TPLP’03]
- [Greco, Greco, Zumpano: LPAR’00, TKDE’03]
- [Calì, Lembo, Rosati: IJCAI’03]

Example

\[
\begin{align*}
\text{emp}(x, y, z) & \leftarrow \text{emp}_D(x, y, z), \text{not dubious}\_\text{emp}(x, y, z). \\
\text{dubious}\_\text{emp}(x, y, z) & \leftarrow \text{emp}_D(x, y, z), \text{emp}(x, y', z'), y \neq y'. \\
\text{dubious}\_\text{emp}(x, y, z) & \leftarrow \text{emp}_D(x, y, z), \text{emp}(x, y', z'), z \neq z'.
\end{align*}
\]

Answer sets

- \{\text{emp}(Gates, Redmond, 20M), \text{emp}(Grove, SantaClara, 10M), \ldots\}\n- \{\text{emp}(Gates, Redmond, 30M), \text{emp}(Grove, SantaClara, 10M), \ldots\}\
Logic Programs for computing CQAs

Logic Programs

- disjunction and classical negation
- checking whether an atom is in all answer sets is $\Pi_2^P$-complete
- dlv, smodels, ...
Logic Programs for computing CQAs

Logic Programs

- disjunction and classical negation
- checking whether an atom is in all answer sets is $\Pi_2^P$-complete
- dlv, smodels, ...

Scope

- arbitrary first-order queries
- universal constraints
- approach unlikely to yield tractable cases
Logic Programs for computing CQAs

**Logic Programs**
- disjunction and classical negation
- checking whether an atom is in all answer sets is \( \Pi_2^P \)-complete
- dlv, smodels, ...

**Scope**
- arbitrary first-order queries
- universal constraints
- approach unlikely to yield tractable cases

**INFOMIX [Eiter et al.: ICLP’03]**
- combines CQA with data integration (GAV)
- uses dlv for repair computations
- optimization techniques: localization, factorization
- tested on small-to-medium-size legacy databases
Co-NP-completeness of CQA

Theorem (Ch., Marcinkowski: Inf. Comp.'05)

For primary-key functional dependencies and conjunctive queries, consistent query answering is \textit{data-complete for co-NP}. 
Theorem (Ch., Marcinkowski: Inf. Comp.’05)

For primary-key functional dependencies and conjunctive queries, consistent query answering is data-complete for co-NP.

Proof.

Membership: $S$ is a repair iff $S \models IC$ and $W \not\models IC$ if $W = S \cup A$.

Co-NP-hardness: reduction from MONOTONE 3-SAT.

1. Positive clauses $\beta_1 = \phi_1 \land \ldots \land \phi_m$, negative clauses $\beta_2 = \psi_{m+1} \land \ldots \land \psi_l$.

2. Database $D$ contains two binary relations $R(A, B)$ and $S(A, B)$:
   - $R(i, p)$ if variable $p$ occurs in $\phi_i$, $i = 1, \ldots, m$.
   - $S(i, p)$ if variable $p$ occurs in $\psi_i$, $i = m + 1, \ldots, l$.

3. $A$ is the primary key of both $R$ and $S$.

4. Query $Q \equiv \exists x, y, z. (R(x, y) \land S(z, y))$.

5. There is an assignment which satisfies $\beta_1 \land \beta_2$ iff there exists a repair in which $Q$ is false.
Co-NP-completeness of CQA

Theorem (Ch., Marcinkowski: Inf. Comp.’05)

For primary-key functional dependencies and conjunctive queries, consistent query answering is data-complete for co-NP.

Proof.

Membership: $S$ is a repair iff $S \models IC$ and $W \not\models IC$ if $W = S \cup A$.

Co-NP-hardness: reduction from MONOTONE 3-SAT.

1. Positive clauses $\beta_1 = \phi_1 \land \ldots \land \phi_m$, negative clauses $\beta_2 = \psi_{m+1} \ldots \land \psi_l$.

2. Database $D$ contains two binary relations $R(A, B)$ and $S(A, B)$:
   - $R(i, p)$ if variable $p$ occurs in $\phi_i$, $i = 1, \ldots, m$.
   - $S(i, p)$ if variable $p$ occurs in $\psi_i$, $i = m + 1, \ldots, l$.

3. $A$ is the primary key of both $R$ and $S$.

4. Query $Q \equiv \exists x, y, z. (R(x, y) \land S(z, y))$.

5. There is an assignment which satisfies $\beta_1 \land \beta_2$ iff there exists a repair in which $Q$ is false.

$Q$ does not belong to $C_{forest}$. 
### Data complexity of CQA

<table>
<thead>
<tr>
<th>Primary keys</th>
<th>Arbitrary keys</th>
<th>Denial</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma, \times, -$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \times, -, \cup$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi, \times, -, \cup$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data complexity of CQA

<table>
<thead>
<tr>
<th></th>
<th>Primary keys</th>
<th>Arbitrary keys</th>
<th>Denial</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma, \times, -$</td>
<td>PTIME</td>
<td>PTIME</td>
<td></td>
<td>PTIME: binary</td>
</tr>
<tr>
<td>$\sigma, \times, -, \cup$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi, \times, -$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[ Arenas, Bertossi, Ch.: PODS’99 ]
### Data complexity of CQA

<table>
<thead>
<tr>
<th></th>
<th>Primary keys</th>
<th>Arbitrary keys</th>
<th>Denial</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma, \times, -$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME: binary</td>
</tr>
<tr>
<td>$\sigma, \times, -, \cup$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td>PTIME</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td></td>
</tr>
<tr>
<td>$\sigma, \pi, \times, -, \cup$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td></td>
</tr>
</tbody>
</table>

- [Arenas, Bertossi, Ch.: PODS’99]
- [Ch., Marcinkowski: Inf.Comp.’05]
## Data complexity of CQA

<table>
<thead>
<tr>
<th></th>
<th>Primary keys</th>
<th>Arbitrary keys</th>
<th>Denial</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sigma, \times, \neg)</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME: binary</td>
</tr>
<tr>
<td>(\sigma, \times, \neg, \cup)</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td></td>
</tr>
<tr>
<td>(\sigma, \pi)</td>
<td>PTIME</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td></td>
</tr>
<tr>
<td>(\sigma, \pi, \times)</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td></td>
</tr>
<tr>
<td>(\sigma, \pi, \times, \neg, \cup)</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td></td>
</tr>
</tbody>
</table>

- [Arenas, Bertossi, Ch.: PODS’99]
- [Ch., Marcinkowski: Inf.Comp.’05]
- [Fuxman, Miller: ICDT’05]
# Data complexity of CQA

<table>
<thead>
<tr>
<th></th>
<th>Primary keys</th>
<th>Arbitrary keys</th>
<th>Denial</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \sigma, \times, \neg )</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>( \text{PTIME: binary} ) ( \Pi^0_2 )-complete</td>
</tr>
<tr>
<td>( \sigma, \times, \neg, \cup )</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>( \Pi^0_2 )-complete</td>
</tr>
<tr>
<td>( \sigma, \pi )</td>
<td>PTIME</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>( \Pi^0_2 )-complete</td>
</tr>
<tr>
<td>( \sigma, \pi, \times )</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>( \Pi^0_2 )-complete</td>
</tr>
<tr>
<td>( \sigma, \pi, \times, \neg, \cup )</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>( \Pi^0_2 )-complete</td>
</tr>
</tbody>
</table>

- [Arenas, Bertossi, Ch.: PODS’99]
- [Ch., Marcinkowski: Inf.Comp.’05]
- [Fuxman, Miller: ICDT’05]
- [Staworko, Ch.: unpublished]
The Semantic Explosion

### Tuple-based repairs

- asymmetric treatment of insertion and deletion:
  - repairs by minimal deletions only [Ch., Marcinkowski: Inf.Comp.’05]: data possibly incorrect but complete
  - repairs by minimal deletions and arbitrary insertions [Calì, Lembo, Rosati: PODS’03]: data possibly incorrect and incomplete
- minimal cardinality changes [Lopatenko, Bertossi: ICDT’07]
The Semantic Explosion

Tuple-based repairs
- asymmetric treatment of insertion and deletion:
  - repairs by minimal deletions only [Ch., Marcinkowski: Inf.Comp.’05]: data possibly incorrect but complete
  - repairs by minimal deletions and arbitrary insertions [Calì, Lembo, Rosati: PODS’03]: data possibly incorrect and incomplete
- minimal cardinality changes [Lopatenko, Bertossi: ICDT’07]

Attribute-based repairs
- (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].
The Semantic Explosion

Tuple-based repairs
- asymmetric treatment of insertion and deletion:
  - repairs by minimal deletions only [Ch., Marcinkowski: Inf. Comp. ’05]: data possibly \textit{incorrect} but \textit{complete}
  - repairs by minimal deletions and arbitrary insertions [Calì, Lembo, Rosati: PODS’03]: data possibly \textit{incorrect} and \textit{incomplete}
- minimal cardinality changes [Lopatenko, Bertossi: ICDT’07]

Attribute-based repairs
- (A) \textit{ground} and \textit{non-ground} repairs [Wijsen: TODS’05]
- (B) \textit{project-join} repairs [Wijsen: FQAS’06]
- (C) repairs minimizing \textit{Euclidean distance} [Bertossi et al.: DBPL’05]
- (D) repairs of minimum \textit{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.
The Need for Attribute-based Repairing

Tuple-based repairing leads to information loss.
The Need for Attribute-based Repairing

Tuple-based repairing leads to **information loss**.

<table>
<thead>
<tr>
<th>EmpDept</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Dept</strong></td>
<td><strong>Location</strong></td>
</tr>
<tr>
<td>John</td>
<td>Sales</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Mary</td>
<td>Sales</td>
<td>Toronto</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ Dept

Dept $\rightarrow$ City
The Need for Attribute-based Repairing

Tuple-based repairing leads to information loss.
Repair a **lossless join decomposition**.

The decomposition:

\[ \pi_{Name, Dept}(EmpDept) \bowtie \pi_{Dept, Location}(EmpDept) \]
Attribute-based Repairs through Tuple-based Repairs

Repair a lossless join decomposition.

The decomposition:

\[ \pi_{Name, Dept}(EmpDept) \times \pi_{Dept, Location}(EmpDept) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Sales</td>
<td>Buffalo</td>
</tr>
<tr>
<td>John</td>
<td>Sales</td>
<td>Toronto</td>
</tr>
<tr>
<td>Mary</td>
<td>Sales</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Mary</td>
<td>Sales</td>
<td>Toronto</td>
</tr>
</tbody>
</table>

Name → Dept
Dept → City
Attribute-based Repairs through Tuple-based Repairs

Repair a lossless join decomposition.

The decomposition:

\[ \pi_{\text{Name}, \text{Dept}}(\text{EmpDept}) \bowtie \pi_{\text{Dept}, \text{Location}}(\text{EmpDept}) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Sales</td>
<td>Buffalo</td>
</tr>
<tr>
<td>John</td>
<td>Sales</td>
<td>Toronto</td>
</tr>
<tr>
<td>Mary</td>
<td>Sales</td>
<td>Buffalo</td>
</tr>
<tr>
<td>Mary</td>
<td>Sales</td>
<td>Toronto</td>
</tr>
</tbody>
</table>

Name → Dept
Dept → City
Probabilistic framework for “dirty” databases

[Andritsos, Fuxman, Miller: ICDE'06]

- 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
  - consistent answer: clean answer with probability 1

---

Salaries with probabilities

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>20M</td>
<td>0.7</td>
</tr>
<tr>
<td>Gates</td>
<td>30M</td>
<td>0.3</td>
</tr>
<tr>
<td>Grove</td>
<td>10M</td>
<td>0.5</td>
</tr>
<tr>
<td>Grove</td>
<td>20M</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Name → Salary
Probabilistic framework for “dirty” databases

[Andritsos, Fuxman, Miller: ICDE'06]

- 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
  - consistent answer: clean answer with probability 1

Salaries with probabilities

<table>
<thead>
<tr>
<th>EmpProb Name</th>
<th>Salary</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>20M</td>
<td>0.7</td>
</tr>
<tr>
<td>Gates</td>
<td>30M</td>
<td>0.3</td>
</tr>
<tr>
<td>Grove</td>
<td>10M</td>
<td>0.5</td>
</tr>
<tr>
<td>Grove</td>
<td>20M</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ Salary
Computing Clean Answers

SQL query

```
SELECT Name
FROM EmpProb e
WHERE e.Salary > 15M
```
Computing Clean Answers

**SQL query**

```
SELECT Name
FROM EmpProb e
WHERE e.Salary > 15M
```

**SQL rewritten query**

```
SELECT e.Name, SUM(e.Prob)
FROM EmpProb e
WHERE e.Salary > 15M
GROUP BY e.Name
```
Computing Clean Answers

**SQL query**

```sql
SELECT Name
FROM EmpProb e
WHERE e.Salary > 15M
```

**SQL rewritten query**

```sql
SELECT e.Name, SUM(e.Prob)
FROM EmpProb e
WHERE e.Salary > 15M
GROUP BY e.Name
```

**EmpProb**

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>20M</td>
<td>0.7</td>
</tr>
<tr>
<td>Gates</td>
<td>30M</td>
<td>0.3</td>
</tr>
<tr>
<td>Grove</td>
<td>10M</td>
<td>0.5</td>
</tr>
<tr>
<td>Grove</td>
<td>20M</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Name → Salary
Computing Clean Answers

SQL query
SELECT Name
FROM EmpProb e
WHERE e.Salary > 15M

SQL rewritten query
SELECT e.Name, SUM(e.Prob)
FROM EmpProb e
WHERE e.Salary > 15M
GROUP BY e.Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>20M</td>
<td>0.7</td>
</tr>
<tr>
<td>Gates</td>
<td>30M</td>
<td>0.3</td>
</tr>
<tr>
<td>Grove</td>
<td>10M</td>
<td>0.5</td>
</tr>
<tr>
<td>Grove</td>
<td>20M</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Name → Salary
Computing Clean Answers

**SQL query**

```
SELECT Name
FROM EmpProb e
WHERE e.Salary > 15M
```

**SQL rewritten query**

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

**EmpProb**

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>20M</td>
<td>0.7</td>
</tr>
<tr>
<td>Gates</td>
<td>30M</td>
<td>0.3</td>
</tr>
<tr>
<td>Grove</td>
<td>10M</td>
<td>0.5</td>
</tr>
<tr>
<td>Grove</td>
<td>20M</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Name → Salary

**Results**

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gates</td>
<td>1</td>
</tr>
<tr>
<td>Grove</td>
<td>0.5</td>
</tr>
</tbody>
</table>
```
PODS’99, June 1999

- Arenas, Bertossi, Ch.: “Consistent Query Answers in Inconsistent Databases.”
PODS’99, June 1999

- Arenas, Bertossi, Ch.: “Consistent Query Answers in Inconsistent Databases.”

Other concurrent events:
PODS’99, June 1999

Arenas, Bertossi, Ch.: “Consistent Query Answers in Inconsistent Databases.”

Other concurrent events:
PODS’99, June 1999

- Arenas, Bertossi, Ch.: “Consistent Query Answers in Inconsistent Databases.”

Other concurrent events:
PODS’99, June 1999

- Arenas, Bertossi, Ch.: “Consistent Query Answers in Inconsistent Databases.”

Other concurrent events:
Taking Stock: Good News

Technology

- **practical methods** for CQA for a subset of SQL:
  - restricted conjunctive/aggregation queries, primary/foreign-key constraints
  - quantifier-free queries/denial constraints
  - LP-based approaches for expressive query/constraint languages
- implemented in **prototype systems**
- tested on **medium-size databases**
Taking Stock: Good News

Technology

- practical methods for CQA for a subset of SQL:
  - restricted conjunctive/aggregation queries, primary/foreign-key constraints
  - quantifier-free queries/denial constraints
  - LP-based approaches for expressive query/constraint languages
- implemented in prototype systems
- tested on medium-size databases

The CQA Community

- over 30 active researchers
- up to 100 publications (since 1999)
- outreach to the AI community (qualified success)
Taking Stock: Initial Progress

Blending in CQA data integration: tension between repairing and satisfying source-to-target dependencies

Extensions:
- nulls: repairs with nulls?
  - clean semantics vs. SQL conformance
- XML notions of integrity constraint and repair
- repair minimality based on tree edit distance?
Taking Stock: Initial Progress

“Blending in” CQA

- **data integration**: tension between repairing and satisfying source-to-target dependencies
- **peer-to-peer**: how to isolate an inconsistent peer?
Taking Stock: Initial Progress

“Blending in” CQA

- **data integration**: tension between repairing and satisfying source-to-target dependencies
- **peer-to-peer**: how to isolate an inconsistent peer?

Extensions

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

- **priorities**:
  - preferred repairs
  - application: conflict resolution

- **XML**
  - notions of integrity constraint and repair
  - repair minimality based on tree edit distance?
Taking Stock: Largely Open Issues
Taking Stock: Largely Open Issues

Applications

- no deployed applications
- repairing vs. CQA: data and query characteristics
- heuristics for CQA and repairing
## Applications
- no deployed applications
- repairing vs. CQA: data and query characteristics
- heuristics for CQA and repairing

## Consolidation
- taming the semantic explosion
- general first-order definability of CQA
- CQA and data cleaning
- CQA and schema matching/mapping
Taking Stock: Largely Open Issues

Applications
- no deployed applications
- repairing vs. CQA: data and query characteristics
- heuristics for CQA and repairing

Consolidation
- taming the semantic explosion
- general first-order definability of CQA
- CQA and data cleaning
- CQA and schema matching/mapping

Foundations
- defining measures of consistency
- more refined complexity analysis
- dynamic aspects
Taking Stock: Largely Open Issues

Applications
- no deployed applications
- repairing vs. CQA: data and query characteristics
- heuristics for CQA and repairing

Consolidation
- taming the semantic explosion
- general first-order definability of CQA
- CQA and data cleaning
- CQA and schema matching/mapping

Foundations
- defining measures of consistency
- more refined complexity analysis
- dynamic aspects
Inconsistent elephant (by Oscar Reutersvärd)
Selected overview papers


L. Bertossi, Consistent Query Answering in Databases. SIGMOD Record, June 2006.
“Five Easy Pieces”

**Bobby:** I’d like a plain omelet. No potatoes, tomatoes instead. A cup of coffee and wheat toast.

**Waitress:** No substitutions.

**Bobby:** What do you mean? You don’t have any tomatoes?

**Waitress:** Only what’s on the menu. You can have a number two - a plain omelet. It comes with cottage, fries, and rolls.

**Bobby:** Yea, I know what it comes with, but it’s not what I want.

**Waitress:** I’ll come back when you make up your mind.

**Bobby:** Wait a minute, I have made up my mind. I’d like a plain omelet, no potatoes on the plate. A cup of coffee and a side order of wheat toast.

**Waitress:** I’m sorry, we don’t have any side orders of toast. I’ll give you a English muffin or a coffee roll.

**Bobby:** What do you mean ”you don’t make side orders of toast”? You make sandwiches, don’t you?

**Waitress:** Would you like to talk to the manager?

**Bobby:** You’ve got bread. And a toaster of some kind?

**Waitress:** I don’t make the rules.

**Bobby:** OK, I’ll make it as easy for you as I can. I’d like an omelet, plain, and a chicken salad sandwich on wheat toast, no mayonnaise, no butter, no lettuce. And a cup of coffee.

**Waitress:** A number two, chicken sal san. Hold the butter, the lettuce, the mayonnaise, and a cup of coffee. Anything else?

**Bobby:** Yeah, now all you have to do is hold the chicken, bring me the toast, give me a check for the chicken salad sandwich, and you haven’t broken any rules.

**Waitress:** You want me to hold the chicken, huh?

**Bobby:** I want you to hold it between your knees.