Consistent Query Answers in Inconsistent Databases

Jan Chomicki

University at Buffalo
Inconsistent Databases

Database instance $D$:

- a finite first-order structure
- the information about the world

---

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

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

Name → City Salary
**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.

**Example Database**:

<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>
**Inconsistent Databases**

<table>
<thead>
<tr>
<th>Database instance $D$:</th>
<th>Integrity constraints $IC$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• a finite first-order structure</td>
<td>• first-order logic formulas</td>
</tr>
<tr>
<td>• the information about the world</td>
<td>• the properties of the world</td>
</tr>
</tbody>
</table>

**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,...
### Whence Inconsistency?

#### 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**.
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>

Name → City Salary
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 ≤ 25M
Query results not reliable.

```sql
SELECT Name
FROM Employee
WHERE Salary ≤ 25M
```
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>

Horizontal Decomposition

Decomposition into two relations:

- violators
- the rest


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

Name → City Salary

Grove Santa Clara 10M
Name → City Salary

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

Weakening the constraints:

- functional dependencies $\rightarrow$ denial constraints

[Borgida: TODS’85]
Exceptions to Constraints

Weakening the constraints:

- functional dependencies $\rightarrow$ denial constraints

<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

[Borgida: TODS’85]
Exceptions to Constraints

Weakening the constraints:

- functional dependencies \(\rightarrow\) denial constraints

<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>

[Borgida: TODS’85]

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

Name \(\rightarrow\) 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 Repairs

Restoring consistency:

- insertion, deletion, update
- minimal change?
Database Repairs

Restoring consistency:

- insertion, deletion, update
- minimal change?

<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>
Database Repairs

Restoring consistency:

- insertion, deletion, update
- minimal change?

<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>

<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>

<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>Grove</td>
<td>Santa Clara</td>
<td>10M</td>
</tr>
</tbody>
</table>
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 answer:

Query answer obtained in every repair.

[Arenas, Bertossi, Ch.: PODS’99]
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 ≤ 25M

Name → City Salary

Name
Grove
Consistent query answer:
Query answer obtained in every repair.

[Arenas, Bertossi, Ch.: PODS’99]

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

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
Research Goals

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

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

<table>
<thead>
<tr>
<th><strong>Algorithms</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>How to <strong>compute</strong> consistent information.</td>
<td></td>
</tr>
</tbody>
</table>
## Research Goals

<table>
<thead>
<tr>
<th><strong>Formal definition</strong></th>
<th>What constitutes reliable (consistent) information in an inconsistent database.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithms</strong></td>
<td>How to <em>compute</em> consistent information.</td>
</tr>
<tr>
<td><strong>Computational complexity analysis</strong></td>
<td></td>
</tr>
</tbody>
</table>
| • *tractable* vs. intractable classes of queries and integrity constraints  
| • tradeoffs: complexity vs. expressiveness. |
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. \(IC\).
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.
Exponentially many repairs

Example relation $R(A, B)$

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

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 $P_{IC,D} \cup P_Q$. 
### Constraint classes

#### Universal constraints

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

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
\( M[Manager] \subseteq M[Name] \)
**Constraint classes**

<table>
<thead>
<tr>
<th>Universal constraints</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m )</td>
<td>( \forall \neg Par(x) \lor Ma(x) \lor Fa(x) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Denial constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall \neg A_1 \lor \cdots \lor \neg A_n )</td>
</tr>
</tbody>
</table>
### Constraint classes

#### Universal constraints

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

**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

<table>
<thead>
<tr>
<th>Universal constraints</th>
<th>Denial constraints</th>
<th>Functional dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall. \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m$</td>
<td>$\forall. \neg A_1 \lor \cdots \lor \neg A_n$</td>
<td>$X \rightarrow Y$:</td>
</tr>
<tr>
<td>Example</td>
<td>Example</td>
<td>• a key dependency in $F$ if $X$ is a key</td>
</tr>
<tr>
<td>$\forall. \neg Par(x) \lor Ma(x) \lor Fa(x)$</td>
<td>$\forall. \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t$</td>
<td>• a primary-key dependency: only one key exists</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**
\[ \forall. \neg Par(x) \lor Ma(x) \lor Fa(x) \]

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

**Functional dependencies**

**Example primary-key dependency**
Name $\rightarrow$ Address Salary

$X \rightarrow Y$:
- a **key** dependency in $F$ if $X$ is a key
- a **primary-key** dependency: only one key exists

**Example**
Name $\rightarrow$ Address Salary
Constraint classes

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

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

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

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 \( S \)
<table>
<thead>
<tr>
<th>Constraint classes</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal constraints</td>
<td>∀. ( \neg A_1 \lor \cdots \lor \neg A_n \lor B_1 \lor \cdots \lor B_m )</td>
</tr>
<tr>
<td>Denial constraints</td>
<td>∀. ( \neg A_1 \lor \cdots \lor \neg A_n )</td>
</tr>
<tr>
<td>Example</td>
<td>∀. ( \neg Par(x) \lor Ma(x) \lor Fa(x) )</td>
</tr>
<tr>
<td>Example</td>
<td>∀. ( \neg M(n, s, m) \lor \neg M(m, t, w) \lor s \leq t )</td>
</tr>
<tr>
<td>Functional dependencies</td>
<td>X \rightarrow Y:</td>
</tr>
<tr>
<td></td>
<td>• a key dependency in F if X is a key</td>
</tr>
<tr>
<td></td>
<td>• a primary-key dependency: only one key exists</td>
</tr>
<tr>
<td>Example primary-key dependency</td>
<td>Name \rightarrow Address Salary</td>
</tr>
<tr>
<td>Inclusion dependencies</td>
<td>R[X] \subseteq S[Y]:</td>
</tr>
<tr>
<td></td>
<td>• a foreign key constraint if Y is a key of S</td>
</tr>
<tr>
<td>Example foreign key constraint</td>
<td>M[Manager] \subseteq M[Name]</td>
</tr>
</tbody>
</table>
## Query Rewriting

### Building queries that compute CQAs

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

**Rewritten query**

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

Building queries that compute CQAs

- relational calculus (algebra) $\leadsto$ relational calculus (algebra)
- SQL $\leadsto$ SQL
- leads to PTIME data complexity

Query

$Emp(x, y, z)$

Rewritten query

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

## Building queries that compute CQAs

- relational calculus (algebra) \(\approx\) relational calculus (algebra)
- SQL \(\approx\) 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) $\leadsto$ relational calculus (algebra)
- SQL $\leadsto$ 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) $\rightsquigarrow$ relational calculus (algebra)
- SQL $\rightsquigarrow$ 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

[Areas, 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 query

SELECT Name FROM Emp
WHERE Salary ≥ 10K
**SQL query**

SELECT Name FROM Emp
WHERE Salary $\geq$ 10K

**SQL rewritten query**

SELECT e1.Name FROM Emp e1
WHERE e1.Salary $\geq$ 10K AND NOT EXISTS
  (SELECT * FROM EMPLOYEE e2
   WHERE e2.Name = e1.Name AND e2.Salary < 10K)
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 $S \supseteq \{B_1, \ldots, B_m\}$.
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 $A \in \{P_{m+1}(t_{m+1}), \ldots, P_n(t_n)\}$: $A \notin D$ 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
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]
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

\[
\text{emp}(x, y, z) \leftarrow \text{emp}_D(x, y, z), \text{not dubious_emp}(x, y, z).
\]
\[
\text{dubious_emp}(x, y, z) \leftarrow \text{emp}_D(x, y, z), \text{emp}(x, y', z'), y \neq y'.
\]
\[
\text{dubious_emp}(x, y, z) \leftarrow \text{emp}_D(x, y, z), \text{emp}(x, y', z'), z \neq z'.
\]
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

\[ emp(x, y, z) \leftarrow emp_D(x, y, z), \text{not dubious}_\text{emp}(x, y, z). \]
\[ \text{dubious}_\text{emp}(x, y, z) \leftarrow emp_D(x, y, z), emp(x, y', z'), y \neq y'. \]
\[ \text{dubious}_\text{emp}(x, y, z) \leftarrow emp_D(x, y, z), emp(x, y', z'), z \neq z'. \]

Answer sets

• \{emp(Gates, Redmond, 20M), emp(Grove, SantaClara, 10M), \ldots\}
• \{emp(Gates, Redmond, 30M), 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 $\Pi^P_2$-complete
- dlv, smodels, ...
## Logic Programs for computing CQAs

### Logic Programs

- disjunction and classical negation
- checking whether an atom is in all answer sets is $\Pi^P_2$-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 data-complete for co-NP.
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: $V$ is a repair iff $V \models IC$ and $W \not\models IC$ if $W = V \cup M$.

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

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

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: \( V \) is a repair iff \( V \models IC \) and \( W \not\models IC \) if \( W = V \cup M \).

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

1. Positive clauses \( \beta_1 = \phi_1 \land \cdots \land \phi_m \), negative clauses \( \beta_2 = \psi_{m+1} \land \cdots \land \psi_l \).

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></th>
<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>
<td></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, -, \cup )</td>
<td></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>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: binary</td>
</tr>
<tr>
<td>$\sigma, \times, -, \cup$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td>PTIME</td>
<td>co-NPC</td>
<td>co-NPC</td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
</tr>
<tr>
<td>$\sigma, \pi, \times, C_{forest}$</td>
<td>PTIME: $C_{forest}$</td>
<td>co-NPC</td>
<td>co-NPC</td>
</tr>
<tr>
<td>$\sigma, \pi, \times, -, \cup$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</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>$\sigma, \times, -$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME: binary $\Pi_2^p$-complete</td>
</tr>
<tr>
<td>$\sigma, \times, -, \cup$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>$\Pi_2^p$-complete</td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td>PTIME</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>$\Pi_2^p$-complete</td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>$\Pi_2^p$-complete</td>
</tr>
<tr>
<td>$\sigma, \pi, \times, -$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>$\Pi_2^p$-complete</td>
</tr>
</tbody>
</table>

- [Arenas, Bertossi, Ch.: PODS’99]
- [Ch., Marcinkowski: Inf.Comp.’05]
- [Fuxman, Miller: ICDT’05]
- [Staworko, Ph.D.]
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]
### 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 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].

### 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>Name</th>
<th>Dept</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<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 → Dept
Dept → City
The Need for Attribute-based Repairing

Tuple-based repairing leads to information loss.

<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>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:

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

Dept \to 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

Name → Dept
Dept → City

Name → Dept
Dept → City

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

<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>
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</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>Gates</td>
</tr>
<tr>
<td>Gates</td>
</tr>
<tr>
<td>Grove</td>
</tr>
<tr>
<td>Grove</td>
</tr>
</tbody>
</table>

Name $\rightarrow$ Salary
Computing Clean Answers

**SQL query**

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

<table>
<thead>
<tr>
<th>SQL query</th>
<th>SQL rewritten query</th>
</tr>
</thead>
</table>
| SELECT Name  
FROM EmpProb e  
WHERE e.Salary > 15M | SELECT e.Name, SUM(e.Prob)  
FROM EmpProb e  
WHERE e.Salary > 15M  
GROUP BY e.Name |
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>
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*
Computing Clean Answers

**SQL query**

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

**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*

**SQL rewritten query**

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

```
Name | Prob  
-----|-------
Gates| 1.0   
Grove| 0.5   
```

- SELECT e.Name, SUM(e.Prob)
- FROM EmpProb e
- WHERE e.Salary > 15M
- GROUP BY e.Name
Consistent Query Answering: Looking Back

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:
Consistent Query Answering: Looking Back

PODS’99, June 1999

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

Other concurrent events:
Consistent Query Answering: Looking Back

PODS’99, June 1999

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

Other concurrent events:
Consistent Query Answering: Looking Back

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
- **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?
• aggregate constraints
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**
## 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
## 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
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
Inconsistent elephant (by Oscar Reutersvärd)
