Scalable Data Exchange with Functional Dependencies

Bruno Marnette$^{1,2}$
Giansalvatore Mecca$^3$
Paolo Papotti$^4$

1: Oxford University Computing Laboratory – Oxford, UK
2: INRIA Saclay, Webdam – Orsay, France
3: Universita della Basilicata – Potenza, Italy
4: Universita Roma Tre – Roma, Italy
Data Exchange

Automatic and scalable exchange of data under constraints

(rule-based)
(very large databases)
(integration, translation)
(constrained schemas)
Standard framework

Fagin, Kolaitis, Miller, Popa, ICDT 2003

Constrained schemas

- **EGDs:** $A(x,y) A(x,z) \rightarrow y = z$ (keys, functional dependencies)
- **TGDs:** $A(x,y) \rightarrow \exists z, B(y,z)$ (foreign keys, inclusion dep.)

Source-to-target dependencies

- **TGDs:** $A(x,y) B(y) \rightarrow \exists z, C(x,z) D(z)$

Databases with labelled nulls: #1,#2,...
Example

Contact

name, phone

<table>
<thead>
<tr>
<th>Alice</th>
<th>01543</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>06513</td>
</tr>
<tr>
<td>Cedric</td>
<td>07945</td>
</tr>
</tbody>
</table>

Address

name, city

<table>
<thead>
<tr>
<th>Alice</th>
<th>Paris</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>Rome</td>
</tr>
<tr>
<td>Joe</td>
<td>London</td>
</tr>
</tbody>
</table>

Person

name, phone, city

\[ C(x,y) \rightarrow \exists z, P(x,y,z) \]
\[ A(x,z) \rightarrow \exists y, P(x,y,z) \]
\[ P(x,y,z), P(x,y',z') \rightarrow y = y', z = z' \]
Example

Contact name, phone

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>01543</td>
</tr>
<tr>
<td>Bob</td>
<td>06513</td>
</tr>
<tr>
<td>Cedric</td>
<td>07945</td>
</tr>
</tbody>
</table>

Address name, city

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Paris</td>
</tr>
<tr>
<td>Bob</td>
<td>Rome</td>
</tr>
<tr>
<td>Joe</td>
<td>London</td>
</tr>
</tbody>
</table>

Person name, phone, city

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>01543</td>
<td>Paris</td>
</tr>
<tr>
<td>Bob</td>
<td>06513</td>
<td>Rome</td>
</tr>
<tr>
<td>Cedric</td>
<td>07945</td>
<td>#1</td>
</tr>
<tr>
<td>Joe</td>
<td>#2</td>
<td>London</td>
</tr>
</tbody>
</table>

C(x,y) → ∃z, P(x,y,z)
A(x,z) → ∃y, P(x,y,z)
P(x,y,z), P(x,y',z') → y = y', z = z'

Desired instance:
Core semantics \[FKP'05\]

**Solution**: satisfies all the dependencies

**Universality**: sound, contained in every solution (~hom)

**Core**: minimal size, minimal redundancy

**Properties**: existence, uniqueness

<table>
<thead>
<tr>
<th>Universal Solution</th>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice #0</td>
<td>Alice #0</td>
</tr>
<tr>
<td>Alice #1</td>
<td>Cedric Denis</td>
</tr>
<tr>
<td>Cedric #2</td>
<td>Cedric Denis</td>
</tr>
<tr>
<td>Denis Elise</td>
<td>Denis Elise</td>
</tr>
<tr>
<td>Cedric #3</td>
<td>#3 Elise</td>
</tr>
<tr>
<td>#3 Elise</td>
<td>#4 Elise</td>
</tr>
</tbody>
</table>

OPTIMISATION

(-60% size)
State of the art

Computing universal solutions:

- A rich literature, not discussed here (see previous talk)
- Several algorithms, based on chase procedures
  - Efficient systems, such as Clio [PVMHF'02]

Computing core solutions:

- Two kinds of algorithms:
  - Recursive optimisation of a universal solution
  - Direct computation with SQL of a core solution
Core-computation algorithms

- Generality
  - Termination
    - Marnette 2009
  - Acyclicity
    - Gottlob, Nash 2006
    - Pichler, Savenkov 2008
    - Fagin, Kolaitis, Popa 2005
  - EGDs only
  - No target dep.

- Efficiency
  - Recursive Algorithms (Polynomial)
    - ten Cate, Chiticariu, Kolaitis, Tan 2009
  - First-Order (SQL)
    - Mecca, Papotti, Raunich 2009

Marnette, Mecca, Papotti - Scalable Data Exchange With Functional Dependencies
Our previous contributions

Marnette, PODS 2009

- Target dependencies (FDs, IDs, Keys.. )
- Assumption: the *oblivious chase* terminates
- Polynomial core-computation (data-complexity)

**Pros:** a general polynomial-time setting

**Cons:** not scalable (possibly hours, for <10K tuples)
Our previous contributions

Mecca, Papotti, Raunich, SIGMOD 2009

\[ +Spicy \]

\{  
\text{Input: scenarios without target dependencies}  
\text{Output: SQL script to compute core solutions}  
\}

- Non-trivial existence! (also shown in [tCCKT'09])
- Efficient rewriting, SQL scripts of good quality

Pros: scalability (few seconds, for millions of tuples)
Cons: no target dependencies
Today's Contribution

Generality

Termination

Marnette, Mecca, Papotti - Scalable Data Exchange With Functional Dependencies

Acyclicity

Efficiency

EGDs only

Recursive Algorithms
(Polynomial)

Core-computation,
Target Dependencies,
In the Scope of SQL

No target dep.

First-Order
(SQL)

Marnette
2009

Gottlob, Nash
2006

Pichler, Savenkov
2008

Fagin, Kolaitis, Popa
2005

ten Cate, Chiticariu, Kolaitis, Tan
2009

Mecca, Papotti, Raunich
2009

Gottlob, Nash
2006

Pichler, Savenkov
2008

Fagin, Kolaitis, Popa
2005

Marnette
2009
Outline of the talk

1) Theoretical limits and practical approach

2) System +Spicy and experimental results

3) Some intuition on the new algorithms
Theoretical limits and practical approach
Compilation: Given scenario $M$, compute an SQL-script $P_M$

- Set of rules of the form $Body \rightarrow Head$
- $Body$: first order query, over the source schema
- $Head$: atoms with skolem functions, over the target schema

Execution and properties:

- Given $I$, executing $P_M$ produces a target instance $P_M(I)$
- Correctness: For every consistent source instance $I$, $P_M(I)$ is a universal solution for $I$ and $M$
- Optimality: In addition, $P_M(I)$ is always a core
Example

\[
\begin{align*}
&\text{Contact} \quad \text{Address} \\
&\text{name, phone} \quad \text{name, city} \\
&\to \\
&\text{Person} \\
&\text{name*, phone, city}
\end{align*}
\]

Script \( P_M \)

\[
\begin{align*}
&\text{Contact}(x,y) \land \text{Address}(x,z) \rightarrow \text{Person}(x,y,z) \\
&\text{Contact}(x,y) \land \neg \text{Address}(x,z) \rightarrow \text{Person}(x,y,F(x)) \\
&\text{Address}(x,z) \land \neg \text{Contact}(x,y) \rightarrow \text{Person}(x,G(x),z)
\end{align*}
\]
Example

Script $P_M$

- $\text{Contact}(x,y) \land \text{Address}(x,z) \rightarrow \text{Person}(x,y,z)$
- $\text{Contact}(x,y) \land \neg \text{Address}(x,z) \rightarrow \text{Person}(x,y,F(x))$
- $\text{Address}(x,z) \land \neg \text{Contact}(x,y) \rightarrow \text{Person}(x,G(x),z)$

$I = \begin{align*}
\text{Alice} & : 01543 & \text{Alice} & : \text{Paris} \\
\text{Bob} & : 06513 & \text{Bob} & : \text{Rome} \\
\text{Cedric} & : 07945 & \text{Joe} & : \text{London}
\end{align*}$

$P_M(I) = \begin{align*}
\text{Alice} & : 01543 & \text{Paris} \\
\text{Bob} & : 06513 & \text{Rome} \\
\text{Cedric} & : 07945 & F(\text{Cedric}) \\
\text{Joe} & : G(\text{Joe}) & \text{London}
\end{align*}$
Theory and practice

**Theorem:** There are scenarios with only one functional dependency for which no correct SQL-script exists.

- s-t TGD: \( \text{Friend}(x,y) \rightarrow \exists g, \ \text{Group}(x,g) \land \text{Group}(y,g) \)
- target FD: \( \text{Group}(x,g1) \land \text{Group}(x,g2) \rightarrow g1=g2 \)
- Need to compute the connected components of Friend

No need to give up:

- Claim: many real-life scenarios remain in the scope of SQL
- Supported by our first experiments (literature, benchmark)
- We have algorithms to recognize the bad cases
Our approach

What we tried first

- Looked for a simple *syntactic* restrictions, ensuring the existence of a correct SQL-script
  - One possibility: *variables affected by at most one FD*
  - Nice on paper, but too restrictive in practice

A more general approach

- Given $M$, compute a “good” SQL-script $P_M$
- Check that this script $P_M$ is correct
- Rewrite $P_M$ into an optimal script, which compute cores
System +Spicy and experimental results
The new version of +Spicy

Previous version, SIGMOD 2009
- Given a scenario without target dependencies, always generates a correct and optimal SQL-script

Now with target dependencies
- Does the same thing for many scenarios with target dependencies (while rejecting the problematic ones)
- Good support for Keys and Functional Dependencies
- Work in progress: Foreign Keys (~Clio [PVMHF'02])
Experimental Results

Choice of scenarios

- From the literature on data exchange and core-computation
- Generated with the STBenchmark [Alexe et al, VLDB'08]

Main results

- Most scenarios in the scope of SQL, despite target dependencies
- Quality of solutions: size reduction 10%-60%
- Script generation: 2s-4m for 22-93 tgds and 25-100 egds
- Script execution: 5s-100s for DBs with 100K-1M tuples
  Drastically faster than recursive algo (>1hour for 10K tuples)
More details in the paper
Some intuition on the new algorithms
Main Algorithms

(1) Overlap Computation
(2) Refined Skolemisation
(3) Test of Correctness
(4) Optimal Rewriting

Input Scenario $M$ (set of dependencies)

$P_M$  

SQL-script

$P'_M$

SQL-script

$P'_M$ is a correct and optimal SQL-script for $M$
(1) Overlap Computation

- Overlap: when two atoms in the head of two rules “unify”
- Combine rules to create new rules, with negation in the body
- Key idea: avoid introducing nulls at the affected positions

\[
\begin{align*}
\text{Contact}(x,y) \rightarrow & \ \exists z, \ \text{Person}(x,y,z) \\
\text{Address}(x,z) \rightarrow & \ \exists y, \ \text{Person}(x,y,z) \\
\text{Person}(x,y,z), \text{Person}(x,y',z') \rightarrow & \ y=y', \ z=z'
\end{align*}
\]

\[
\begin{align*}
\text{Contact}(x,y) \land \text{Address}(x,z) \rightarrow & \ \text{Person}(x,y,z) \\
\text{Contact}(x,y) \land \neg \exists z \ \text{Address}(x,z) \rightarrow & \ \exists z, \ \text{Person}(x,y,z) \\
\text{Address}(x,z) \land \neg \exists y \ \text{Contact}(x,y) \rightarrow & \ \exists y, \ \text{Person}(x,y,z)
\end{align*}
\]
(2) Refined Skolemisation

- Skolemisation: replace variables by uninterpreted functions
- Naïve skolemisation: function of all the universal variables
- The target FDs allow to find better candidates (less variables)
- Our algorithm looks both at the target FDs and the source FDs

\[ A(x, y, z) \rightarrow \exists e, B(x, e) \land C(y, e) \quad \text{(with three FDs)} \]

\[ A(x, y, z) \rightarrow B(x, F(y)) \land C(y, F(y)) \quad (x \rightarrow y \rightarrow F(y)) \]
(3) Test of Correctness

By construction of $P_M$: for all $I$, the instance $P_M(I)$ is *sound*

It remains to check that: for all $I$, the instance $P_M(I)$ is a *solution*

The problem would be undecidable if $P_M$ was too complicated...

**Theorem:** The following problem is decidable in our setting: given $M$ and the corresponding $P_M$, is $P_M$ correct?

- Key argument: *small model property*
- Exponential algorithm (not too bad in practice)
(4) Optimal Rewriting

**Theorem:** Given a correct SQL-script $P_M$ for a scenario $M$, we can always compute an optimal SQL-script

- A non-trivial generalisation of [MPR'09], [tCCKT'09]
  - Strongly based on a *bounded block-width* property
  - A null can not appear in more than $k$ atom
- We support scenarios with target FDs and unbounded blocks
- Technical proof (sketched in the paper)
Example (unbounded blocks)

Scenario $M$

Mother($name$, $name$) → $\exists z$, Parent($x$, $z$) ∧ Child($z$, $y$)
Parent($x$, $y_1$) ∧ Parent($x$, $y_2$) → $y_1 = y_2$

Script $P_M$

Mother($x$, $y$) → Parent($x$, $F(x)$), Child($F(x)$, $y$)

$I = \begin{bmatrix} Alice & Bob \\ Sophie & Cedric \\ Sophie & Denis \end{bmatrix}

P_M(I) = \begin{bmatrix} Alice & #1 \\ #1 & Bob \\ Sophie & #2 \\ #2 & Cedric \\ #2 & Denis \end{bmatrix}$

Where $#1 = F(Alice)$ and $#2 = F(Sophie)$
Conclusion

Experimental evidence

- Real-life scenarios with FDs can be managed with SQL
- Computing cores is very feasible (scales well)

Technical results

- Algorithms of general interest (overlaps, skolemisation)
- The property of correctness is decidable (in our setting)
- Correct SQL-scripts can always be optimised (cores)

Future work

- Other target dependencies (Inclusion Dependences)
- Alternative languages (fixed-points, stratified Datalog)
Experimental evidence

- Real-life scenarios with FDs can be managed with SQL
- Computing cores is very feasible (scales well)

Technical results

- Algorithms of general interest (overlaps, skolemisation)
- The property of correctness is decidable (in our setting)
- Correct SQL-scripts can always be optimised (cores)

Future work

- Other target dependencies (Inclusion Dependences)
- Alternative languages (fixed-points, stratified Datalog)