Plan of the course

1. Integrity constraints

2. Consistent query answers
Part I

Integrity constraints
Outline of Part I

1. Basic notions

2. Implication of dependencies

3. Axiomatization

4. Applications
   - Database design
   - Data exchange
   - Semantic query optimization

5. Prospects
Integrity constraints (dependencies)

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

**Integrity constraints $\Sigma$:**
- first-order logic *formulas*
- the *properties* of the world
Integrity constraints (dependencies)

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

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

Satisfaction of constraints: $D \models \Sigma$

Formula satisfaction in a first-order structure.
Integrity constraints (dependencies)

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

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

Satisfaction of constraints: $D \models \Sigma$

Formula satisfaction in a first-order structure.

Consistent database: $D \models \Sigma$

<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 $\rightarrow$ City Salary
Integrity constraints (dependencies)

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

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

Satisfaction of constraints: $D \models \Sigma$
Formula satisfaction in a first-order structure.

Consistent database: $D \models \Sigma$

<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 $\rightarrow$ City Salary

Inconsistent database: $D \not\models \Sigma$

<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
The need for integrity constraints

- Examples:
  - Key functional dependency: "every employee has a single address and salary"
  - Denial constraint: "no employee can earn more than her manager"
  - Foreign key constraint: "every manager is an employee"

Roles of integrity constraints:
- Capture the semantics of data: legal values of attributes, object identity, relationships, associations (in value-based data models)
- Reduce data errors, leading to data quality
- Help in database design and query formulation:
  - Usually (usually) no effect on query semantics but...
  - Query evaluation and analysis are affected: indexes, access paths, query containment and equivalence, semantic query optimization (SQO)
- Specify database mappings
The need for integrity constraints

Examples

- **key functional dependency**: “every employee has a single address and salary”
- **denial constraint**: “no employee can earn more than her manager”
- **foreign key constraint**: “every manager is an employee”
The need for integrity constraints

Examples

- **Key functional dependency:** “every employee has a single address and salary”
- **Denial constraint:** “no employee can earn more than her manager”
- **Foreign key constraint:** “every manager is an employee”

Roles of integrity constraints

- Capture the **semantics** of data:
  - Legal values of attributes
  - Object identity
  - Relationships, associations (in value-based data models)
- Reduce data **errors** ⇒ **data quality**
- Help in database **design**
- Help in query **formulation**
- (usually) no effect on query **semantics** but ... query **evaluation** and **analysis** are affected:
  - Indexes, access paths
  - Query containment and equivalence
  - Semantic query optimization (SQO)
- Specify database **mappings**
Constraint enforcement

Enforced by application programs
- Constraint checks inserted into code
- Code duplication and increased application complexity
- Error-prone: different applications can make different assumptions
- Prevent system-level optimizations

Enforced by DBMS
- Constraint checks performed by DBMS ("factored out")
- Violating updates rolled back
- Leads to application simplification and reduces errors
- Enables query optimizations

Not enforced
- Data comes from multiple, independent sources
- Long transactions with inconsistent intermediate states
- Enforcement too expensive
Constraint enforcement

Enforced by application programs

- constraint checks inserted into code
- code duplication and increased application complexity
- error-prone: different applications can make different assumptions
- prevent system-level optimizations

Enforced by DBMS

- constraint checks performed by DBMS (“factored out”)
- violating updates rolled back
- leads to application simplification and reduces errors
- enables query optimizations

But... integrity checks are expensive and inflexible

Not enforced

- data comes from multiple, independent sources
- long transactions with inconsistent intermediate states
- enforcement too expensive
### Constraint enforcement

#### Enforced by application programs
- constraint checks inserted into code
- code duplication and increased application complexity
- error-prone: different applications can make different assumptions
- prevent system-level optimizations

#### Enforced by DBMS
- constraint checks performed by DBMS ("factored out")
- violating updates rolled back
- leads to application simplification and reduces errors
- enables query optimizations
- but ... integrity checks are expensive and inflexible
Constraint enforcement

Enforced by application programs
- constraint checks inserted into code
- code duplication and increased application complexity
- error-prone: different applications can make different assumptions
- prevent system-level optimizations

Enforced by DBMS
- constraint checks performed by DBMS ("factored out")
- violating updates rolled back
- leads to application simplification and reduces errors
- enables query optimizations
- but ... integrity checks are expensive and inflexible

Not enforced
- data comes from multiple, independent sources
- long transactions with inconsistent intermediate states
- enforcement too expensive
Basic issues

Implication
Given a set of ICs $\Sigma$ and an IC $\sigma$, does $D|\Sigma = \Sigma$ imply $D|\sigma = \sigma$ for every database $D$?

Axiomatization
Can the notion of implication be captured by an inference system ("axiomatized")?

Inconsistent databases
1. How to construct a consistent database on the basis of an inconsistent one?
2. How to obtain information unaffected by inconsistency?
Basic issues

Implication

Given a set of ICs $\Sigma$ and an IC $\sigma$, does $D \models \Sigma$ imply $D \models \sigma$ for every database $D$?
Basic issues

Implication
Given a set of ICs $\Sigma$ and an IC $\sigma$, does $D \models \Sigma$ imply $D \models \sigma$ for every database $D$?

Axiomatization
Can the notion of implication be captured by an inference system ("axiomatized")?
Basic issues

**Implication**
Given a set of ICs $\Sigma$ and an IC $\sigma$, does $D \models \Sigma$ imply $D \models \sigma$ for every database $D$?

**Axiomatization**
Can the notion of implication be captured by an inference system ("axiomatized")?

**Inconsistent databases**
1. How to **construct** a consistent database on the basis of an inconsistent one?
2. How to obtain information **unaffected** by inconsistency?
ICs in logical form

Atomic formulas
relational (database) atoms $P(x_1, ..., x_k)$ and equality atoms $x_1 = x_2$ (usually) typed

General form
$\forall x_1, ..., x_k. A_1 \land \cdots \land A_n \Rightarrow \exists y_1, ..., y_l. B_1 \land \cdots \land B_m$.

Subclasses
- full dependencies: no existential variables ($l = 0$)
- tuple-generating dependencies (TGDs): no equality atoms
- equality-generating dependencies (EGDs): $m = 1$, $B_1$ is an equality atom
- functional dependencies (FDs): binary unirelational EGDs
- join dependencies (JDs): TGDs with LHS a multiway join
- denial constraints: $l = 0$, $m = 0$
- inclusion dependencies (INDs): $n = m = 1$, no equality atoms
ICs in logical form

Atomic formulas

- relational (database) atoms $P(x_1, \ldots, x_k)$ and equality atoms $x_1 = x_2$
- (usually) typed
- no constants
ICs in logical form

### Atomic formulas
- relational (database) atoms $P(x_1, \ldots, x_k)$ and equality atoms $x_1 = x_2$
- (usually) typed
- no constants

### General form
\[
\forall x_1, \ldots, x_k. \ A_1 \land \cdots \land A_n \Rightarrow \exists y_1, \ldots, y_l. \ B_1 \land \cdots \land B_m.
\]
ICs in logical form

Atomic formulas
- relational (database) atoms $P(x_1, \ldots, x_k)$ and equality atoms $x_1 = x_2$
- (usually) typed
- no constants

General form
$$\forall x_1, \ldots, x_k. \ A_1 \land \cdots \land A_n \Rightarrow \exists y_1, \ldots, y_l. \ B_1 \land \cdots \land B_m.$$ 

Subclasses
- **full** dependencies: no existential variables ($l = 0$)
- **tuple-generating** dependencies (TGDs): no equality atoms
- **equality-generating** dependencies (EGDs): $m = 1$, $B_1$ is an equality atom
- **functional** dependencies (FDs): binary unirelational EGDs
- **join** dependencies (JDs): TGDs with LHS a multiway join
- **denial** constraints: $l = 0$, $m = 0$
- **inclusion** dependencies (INDs): $n = m = 1$, no equality atoms
Examples

Relations \(NAM(\text{Name}, \text{Address}, \text{Manager}), \ NAS(\text{Name}, \text{Address}, \text{Salary}), \ NM(\text{Name}, \text{Manager}).\)
Examples

Relations $NAM(Name, Address, Manager)$, $NAS(Name, Address, Salary)$, $NM(Name, Manager)$.

**Full TGD**

$$\forall n, a, m, s. \ NAS(n, a, s) \land NM(n, m) \Rightarrow NAM(n, a, m)$$
Examples

Relations $NAM(\text{Name}, \text{Address}, \text{Manager})$, $NAS(\text{Name}, \text{Address}, \text{Salary})$, $NM(\text{Name}, \text{Manager})$.

**Full TGD**

\[ \forall n, a, m, s. \ NAS(n, a, s) \land NM(n, m) \Rightarrow NAM(n, a, m) \]

**Non-full (embedded) TGD**

\[ \forall n, a, m. \ NAM(n, a, m) \Rightarrow \exists s. \ NAS(n, a, s) \]

**Inclusion dependency (IND)**

\[ NAM[\text{Name}, \text{Address}] \subseteq NAS[\text{Name}, \text{Address}] \]
Examples

Relations \(NAM(\text{Name}, \text{Address}, \text{Manager})\), \(NAS(\text{Name}, \text{Address}, \text{Salary})\), \(NM(\text{Name}, \text{Manager})\).

**Full TGD**
\[
\forall n, a, m, s. \ NAS(n, a, s) \land NM(n, m) \Rightarrow NAM(n, a, m)
\]

**Non-full (embedded) TGD**
\[
\forall n, a, m. \ NAM(n, a, m) \Rightarrow \exists s. \ NAS(n, a, s)
\]

**Inclusion dependency (IND)**
\[
NAM[\text{Name}, \text{Address}] \subseteq NAS[\text{Name}, \text{Address}]
\]

**Functional dependency (FD)**
\[
\text{Name} \rightarrow \text{Address}
\]
Implication: from linear-time to undecidable

1. View each attribute as a propositional variable.
2. View each dependency \( A_1 \ldots A_k \rightarrow B \in \Sigma \) as a Horn clause.
3. If \( \sigma = C_1 \ldots C_d \Rightarrow D \), then \( \neg \sigma = C_1 \ldots C_d \land \neg D \) consists of Horn clauses.
4. Thus \( \Sigma \cup \neg \sigma \) is a set of Horn clauses whose (un)satisfiability can be tested in linear time (Dowling, Gallier [DG84]).

Theorem (Chandra, Vardi [CV85])

The implication problem for functional dependencies together with inclusion dependencies is undecidable.
Implication: from linear-time to undecidable

Functional dependencies

1. view each attribute as a propositional variable
2. view each dependency $A_1 \ldots A_k \rightarrow B \in \Sigma$ as a Horn clause $A_1 \land \cdots \land A_k \Rightarrow B$
3. if $\sigma = C_1 \land \cdots \land C_d \Rightarrow D$, then $\neg\sigma = C_1 \land \cdots \land C_d \land \neg D$ consists of Horn clauses
4. thus $\Sigma \cup \neg\sigma$ is a set of Horn clauses whose (un)satisfiability can be tested in linear time (Dowling, Gallier [DG84])
Function dependencies

1. view each attribute as a propositional variable
2. view each dependency $A_1 \ldots A_k \rightarrow B \in \Sigma$ as a Horn clause $A_1 \land \cdots \land A_k \Rightarrow B$
3. if $\sigma = C_1 \land \cdots \land C_d \Rightarrow D$, then $\neg \sigma = C_1 \land \cdots \land C_d \land \neg D$ consists of Horn clauses
4. thus $\Sigma \cup \neg \sigma$ is a set of Horn clauses whose (un)satisfiability can be tested in linear time (Dowling, Gallier [DG84])

Theorem (Chandra, Vardi [CV85])

The implication problem for functional dependencies together with inclusion dependencies is undecidable.
Implication in logic

No restriction to \textit{finite structures}.
Implication in logic

No restriction to finite structures.

Finite and unrestricted implication

- coincide for full dependencies
- if they coincide, then they are decidable
- but not vice versa (FDs and unary INDs)
Implication in logic

No restriction to finite structures.

Finite and unrestricted implication

- coincide for full dependencies
- if they coincide, then they are decidable
- but not vice versa (FDs and unary INDs)

Counterexample

\[ \Sigma = \{ A \rightarrow B, R[A] \subseteq R[B] \} \]
\[ \sigma = R[B] \subseteq R[A] \]
Implication in logic

No restriction to finite structures.

Finite and unrestricted implication

- coincide for full dependencies
- if they coincide, then they are decidable
- but not vice versa (FDs and unary INDs)

Counterexample

\( \Sigma = \{ A \rightarrow B, R[A] \subseteq R[B] \} \)

\( \sigma = R[B] \subseteq R[A] \)

\[
\begin{array}{cc}
A & B \\
1 & 0 \\
2 & 1 \\
3 & 2 \\
4 & 3 \\
\vdots & \\
\end{array}
\]
Implication in logic

No restriction to finite structures.

Finite and unrestricted implication

- coincide for full dependencies
- if they coincide, then they are decidable
- but not vice versa (FDs and unary INDs)

Counterexample

\[ \Sigma = \{ A \rightarrow B, R[A] \subseteq R[B] \} \]
\[ \sigma = R[B] \subseteq R[A] \]

Finite and unrestricted implication do not have to coincide.
Chase

Deciding the implication of full dependencies using chase
apply chase steps using the dependencies in $\Sigma$ nondeterministically, obtaining a sequence of dependencies $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$
stop when no chase steps can be applied to $\tau_n$ (a terminal chase sequence)
if $\tau_n$ is trivial, then $\Sigma$ implies $\sigma$
otherwise, $\Sigma$ does not imply $\sigma$

Trivial dependencies
tgd: LHS contains RHS
egd: RHS $\equiv x = x$

Fundamental properties of the chase
Terminal chase sequence $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$: the LHS of $\tau_n$, viewed as a database $D_n$, satisfies $\Sigma$ if $\tau_n$ is nontrivial, then $D_n$ violates $\sigma$
the order of chase steps does not matter
Deciding the implication of full dependencies using chase

1. **apply** chase steps using the dependencies in \( \Sigma \) nondeterministically, obtaining a sequence of dependencies \( \tau_0 = \sigma, \tau_1, \ldots, \tau_n \)

2. stop when no chase steps can be applied to \( \tau_n \) (a terminal chase sequence)

3. if \( \tau_n \) is trivial, then \( \Sigma \) implies \( \sigma \)

4. otherwise, \( \Sigma \) does not imply \( \sigma \)
Deciding the implication of full dependencies using chase

1. apply chase steps using the dependencies in $\Sigma$ nondeterministically, obtaining a sequence of dependencies $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$
2. stop when no chase steps can be applied to $\tau_n$ (a terminal chase sequence)
3. if $\tau_n$ is trivial, then $\Sigma$ implies $\sigma$
4. otherwise, $\Sigma$ does not imply $\sigma$

Trivial dependencies

- tgd: LHS contains RHS
- egd: RHS $\equiv x = x$
Deciding the implication of full dependencies using chase

1. apply chase steps using the dependencies in $\Sigma$ nondeterministically, obtaining a sequence of dependencies $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$
2. stop when no chase steps can be applied to $\tau_n$ (a terminal chase sequence)
3. if $\tau_n$ is trivial, then $\Sigma$ implies $\sigma$
4. otherwise, $\Sigma$ does not imply $\sigma$

Trivial dependencies

- tgd: LHS contains RHS
- egd: RHS $\equiv x = x$

Fundamental properties of the chase

Terminal chase sequence $\tau_0 = \sigma, \tau_1, \ldots, \tau_n$:

- the LHS of $\tau_n$, viewed as a database $D_n$, satisfies $\Sigma$
- if $\tau_n$ is nontrivial, then $D_n$ violates $\sigma$
- the order of chase steps does not matter
Chase steps

A chase sequence $\tau_0 = \sigma, \tau_1, \ldots$. 
A chase sequence \( \tau_0 = \sigma, \tau_1, \ldots \).

**Applying a chase step using a tgd \( C \)**

1. view the LHS of \( \tau_j \) as a database \( D_j \)
2. find a substitution \( h \) that (1) \( h \) makes the LHS of \( C \) true in \( D_j \), and (2) \( h \) cannot be extended to a substitution that makes the RHS of \( C \) true in that instance
3. apply \( h \) to the RHS of \( C \)
4. add the resulting facts to the LHS of \( \tau_j \), obtaining \( \tau_{j+1} \)
Chase steps

A chase sequence $\tau_0 = \sigma, \tau_1, \ldots$

Applying a chase step using a tgd $C$

1. view the LHS of $\tau_j$ as a database $D_j$
2. find a substitution $h$ that (1) $h$ makes the LHS of $C$ true in $D_j$, and (2) $h$ cannot be extended to a substitution that makes the RHS of $C$ true in that instance
3. apply $h$ to the RHS of $C$
4. add the resulting facts to the LHS of $\tau_j$, obtaining $\tau_{j+1}$

Applying a chase step using an egd $C$

1. view the LHS of $\tau_j$ as a database $D_j$
2. RHS of $C \equiv x_1 = x_2$
3. find a substitution $h$ such that makes the LHS of $C$ true in $D_j$ and $h(x_1) \neq h(x_2)$
4. replace all the occurrences of $h(x_2)$ in $\tau_j$ by $h(x_1)$, obtaining $\tau_{j+1}$
Chase in action

Chase in action
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]

Goal

Show that \{ C_1, C_2 \} implies \ C_3. \]
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]

Goal

Show that \( \{ C_1, C_2 \} \) implies \( C_3 \).

Terminal chase sequence
### Integrity constraints

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]

### Goal

Show that \{ C_1, C_2 \} implies \( C_3 \).

### Terminal chase sequence

\[ \tau_0 = \{ P(x, y) \land P(x, z) \Rightarrow y = z \} \]
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]

Goal

Show that \( \{ C_1, C_2 \} \) implies \( C_3 \).

Terminal chase sequence

\[ \tau_0 = \{ P(x, y) \land P(x, z) \Rightarrow y = z \} \]
\[ \tau_1 = \{ P(x, y) \land P(x, z) \land R(x, y) \Rightarrow y = z \} \]
Chase in action

Integrity constraints

\[ C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y) \]
\[ C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z \]
\[ C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z \]

Goal

Show that \( \{ C_1, C_2 \} \) implies \( C_3 \).

Terminal chase sequence

\[ \tau_0 = \{ P(x, y) \land P(x, z) \Rightarrow y = z \} \]
\[ \tau_1 = \{ P(x, y) \land P(x, z) \land R(x, y) \Rightarrow y = z \} \]
\[ \tau_2 = \{ P(x, y) \land P(x, z) \land R(x, y) \land R(x, z) \Rightarrow y = z \} \]
Chase in action

Integrity constraints

\(C_1 = \forall x, y. \; P(x, y) \Rightarrow R(x, y)\)
\(C_2 = \forall x, y, z. \; R(x, y) \land R(x, z) \Rightarrow y = z\)
\(C_3 = \forall x, y, z. \; P(x, y) \land P(x, z) \Rightarrow y = z\)

Goal

Show that \(\{C_1, C_2\}\) implies \(C_3\).

Terminal chase sequence

\(\tau_0 = \{P(x, y) \land P(x, z) \Rightarrow y = z\}\)
\(\tau_1 = \{P(x, y) \land P(x, z) \land R(x, y) \Rightarrow y = z\}\)
\(\tau_2 = \{P(x, y) \land P(x, z) \land R(x, y) \land R(x, z) \Rightarrow y = z\}\)
\(\tau_3 = \{P(x, y) \land R(x, y) \Rightarrow y = y\}\)
**Chase in action**

### Integrity constraints

\[
C_1 = \forall x, y. \ P(x, y) \Rightarrow R(x, y)
\]
\[
C_2 = \forall x, y, z. \ R(x, y) \land R(x, z) \Rightarrow y = z
\]
\[
C_3 = \forall x, y, z. \ P(x, y) \land P(x, z) \Rightarrow y = z
\]

### Goal

Show that \(\{C_1, C_2\}\) implies \(C_3\).

### Terminal chase sequence

\[
\tau_0 = \{P(x, y) \land P(x, z) \Rightarrow y = z\}
\]
\[
\tau_1 = \{P(x, y) \land P(x, z) \land R(x, y) \Rightarrow y = z\}
\]
\[
\tau_2 = \{P(x, y) \land P(x, z) \land R(x, y) \land R(x, z) \Rightarrow y = z\}
\]
\[
\tau_3 = \{P(x, y) \land R(x, y) \Rightarrow y = y\}: \text{a trivial dependency}
\]
A general perspective

Computational complexity

Testing implication of full dependencies is:

in \text{EXPTIME} (using chase)

\text{EXPTIME-complete} (Chandra et al. \cite{CLM81})

First-order logic

implication of \( \sigma \) by \( \Sigma = \{ \sigma_1, \ldots, \sigma_k \} \)

is equivalent to the unsatisfiability of the formula \( \Phi_{\Sigma, \sigma} \equiv \sigma_1 \land \cdots \land \sigma_k \land \neg \sigma \)

for full dependencies, the formulas \( \Phi_{\Sigma, \sigma} \) are of the form

\[ \exists \ast \forall \ast \phi \]

where \( \phi \) is quantifier-free (Bernays-Schoenfinkel class)

Bernays-Schoenfinkel formulas have the finite-model property and their satisfiability is in \text{NEXPTIME}

Theorem proving

Chase corresponds to a combination of hyperresolution and paramodulation.
Computational complexity

Testing implication of full dependencies is:
- in EXPTIME (using chase)
- EXPTIME-complete (Chandra et al. [CLM81])
A general perspective

Computational complexity

Testing implication of full dependencies is:
- in EXPTIME (using chase)
- EXPTIME-complete (Chandra et al. [CLM81])

First-order logic

- implication of $\sigma$ by $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is equivalent to the unsatisfiability of the formula $\Phi_{\Sigma, \sigma} \equiv \sigma_1 \land \cdots \land \sigma_k \land \neg \sigma$
- for full dependencies, the formulas $\Phi_{\Sigma, \sigma}$ are of the form $\exists^* \forall^* \phi$ where $\phi$ is quantifier-free (Bernays-Schönfinkel class)
- Bernays-Schönfinkel formulas have the finite-model property and their satisfiability is in NEXPTIME
A general perspective

Computational complexity

Testing implication of full dependencies is:
- in EXPTIME (using chase)
- EXPTIME-complete (Chandra et al. [CLM81])

First-order logic

- implication of $\sigma$ by $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is equivalent to the unsatisfiability of the formula $\Phi_{\Sigma,\sigma} \equiv \sigma_1 \land \cdots \land \sigma_k \land \neg \sigma$
- for full dependencies, the formulas $\Phi_{\Sigma,\sigma}$ are of the form $\exists^{*} \forall^{*} \phi$ where $\phi$ is quantifier-free (Bernays-Schönfinkel class)
- Bernays-Schönfinkel formulas have the finite-model property and their satisfiability is in NEXPTIME

Theorem proving

Chase corresponds to a combination of hyperresolution and paramodulation.
Axiomatization

Inference rules specific to classes of dependencies guarantee closure: only dependencies from the same class are derived.

Properties

- Soundness: all the dependencies derived from a given set Σ are implied by Σ.
- Completeness: all the dependencies implied by Σ can be derived from Σ.

Finite set of rules \(\Rightarrow\) implication decidable (but not vice versa).
Axiomatization

Inference rules

- specific to classes of dependencies
- guarantee **closure**: only dependencies from the same class are derived
- **bounded** number of premises
Axiomatization

Inference rules

- specific to classes of dependencies
- guarantee **closure**: only dependencies from the same class are derived
- **bounded** number of premises

Properties

Inference rules capture finite or unrestricted implication:

- **soundness**: all the dependencies derived from a given set $\Sigma$ are implied by $\Sigma$
- **completeness**: all the dependencies implied by $\Sigma$ can be derived from $\Sigma$
- finite set of rules $\Rightarrow$ implication **decidable** (but not vice versa)
Example axiomatization

1. Reflexivity: \( R[X] \subseteq R[X] \)

2. Projection and permutation: If \( R[A_1,\ldots,A_m] \subseteq S[B_1,\ldots,B_m] \), then \( R[A_{i_1},\ldots,A_{i_k}] \subseteq S[B_{i_1},\ldots,B_{i_k}] \) for every sequence \( i_1,\ldots,i_k \) of distinct integers in \( \{1,\ldots,m\} \).

3. Transitivity: If \( R[X] \subseteq S[Y] \) and \( S[Y] \subseteq T[Z] \), then \( R[X] \subseteq T[Z] \).

A derivation

Schemas \( R(ABC) \) and \( S(AB) \):

(1) \( S[AB] \subseteq R[AB] \) (given IND)

(2) \( R[C] \subseteq S[A] \) (given IND)

(3) \( S[A] \subseteq R[A] \) (from (1))

(4) \( R[C] \subseteq R[A] \) (from (2) and (3))
Axiomatizing INDs

- **Reflexivity**: \( R[X] \subseteq R[X] \)
- **Projection and permutation**: If \( R[A_1, \ldots A_m] \subseteq S[B_1, \ldots B_m] \), then \( R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}] \) for every sequence \( i_1, \ldots, i_k \) of distinct integers in \( \{1, \ldots, m\} \).
- **Transitivity**: If \( R[X] \subseteq S[Y] \) and \( S[Y] \subseteq T[Z] \), then \( R[X] \subseteq T[Z] \).
Example axiomatization

Axiomatizing INDs

1. **Reflexivity:** $R[X] \subseteq R[X]

2. **Projection and permutation:** If $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$, then $R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}]$ for every sequence $i_1, \ldots, i_k$ of distinct integers in $\{1, \ldots, m\}$.

3. **Transitivity:** If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.

A derivation

Schemas $R(ABC)$ and $S(AB)$:
Example axiomatization

Axiomatizing INDs

1. **Reflexivity:** $R[X] \subseteq R[X]$

2. **Projection and permutation:** If $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$, then $R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}]$ for every sequence $i_1, \ldots, i_k$ of distinct integers in $\{1, \ldots, m\}$.

3. **Transitivity:** If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.

A derivation

Schemas $R(ABC)$ and $S(AB)$:

(1) $S[AB] \subseteq R[AB]$ (given IND)
Example axiomatization

Axiomatizing INDs

1. **Reflexivity**: $R[X] \subseteq R[X]$

2. **Projection and permutation**: If $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$, then $R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}]$ for every sequence $i_1, \ldots, i_k$ of distinct integers in $\{1, \ldots, m\}$.

3. **Transitivity**: If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.

A derivation

Schemas $R(ABC)$ and $S(AB)$:

(1) $S[AB] \subseteq R[AB]$ (given IND)

(2) $R[C] \subseteq S[A]$ (given IND)
Example axiomatization

**Axiomatizing INDs**

1. **Reflexivity**: $R[X] \subseteq R[X]$
2. **Projection and permutation**: If $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$, then $R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}]$ for every sequence $i_1, \ldots, i_k$ of distinct integers in \{1, \ldots, m\}.
3. **Transitivity**: If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.

**A derivation**

Schemas $R(ABC)$ and $S(AB)$:

1. $S[AB] \subseteq R[AB]$  (given IND)
2. $R[C] \subseteq S[A]$  (given IND)
3. $S[A] \subseteq R[A]$  (from (1))
Axiomatizing INDs

1. **Reflexivity:** \( R[X] \subseteq R[X] \)

2. **Projection and permutation:** If \( R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m] \), then \( R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}] \) for every sequence \( i_1, \ldots, i_k \) of distinct integers in \( \{1, \ldots, m\} \).

3. **Transitivity:** If \( R[X] \subseteq S[Y] \) and \( S[Y] \subseteq T[Z] \), then \( R[X] \subseteq T[Z] \).

A derivation

Schemas \( R(ABC) \) and \( S(AB) \):

1. \( S[AB] \subseteq R[AB] \) (given IND)
2. \( R[C] \subseteq S[A] \) (given IND)
3. \( S[A] \subseteq R[A] \) (from (1))
4. \( R[C] \subseteq R[A] \) (from (2) and (3))
### Review of results

<table>
<thead>
<tr>
<th></th>
<th>Implication</th>
<th>Axiomatization</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDs</td>
<td>PTIME</td>
<td>Yes</td>
</tr>
<tr>
<td>INDs</td>
<td>PSPACE-complete</td>
<td>Yes</td>
</tr>
<tr>
<td>FDs + INDs</td>
<td>Undecidable</td>
<td>No</td>
</tr>
<tr>
<td>Full (typed) dependencies</td>
<td>EXPTIME-complete</td>
<td>Yes</td>
</tr>
<tr>
<td>Join dependencies</td>
<td>NP-complete</td>
<td>No</td>
</tr>
<tr>
<td>First-order logic</td>
<td>Undecidable</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Application: database design

Keys
A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:

$\Sigma$ implies $X \rightarrow U$ for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$.

Decomposition
A decomposition $R = (R_1, \ldots, R_n)$ of a schema $R$ has the lossless join property with respect to a set of FDs $\Sigma$ iff $\Sigma$ implies the join dependency $\Delta \Join [R]$.

Decomposition $(R_1, R_2)$ of $R(ABC)$
Relation schemas:
$R_1(AB)$ with FD $A \rightarrow B$, $R_2(AC)$.

Terminal chase sequence:
$R(x, y, z') \wedge R(x, y', z) \Rightarrow R(x, y, z)$ given JD
$R(x, y, z') \wedge R(x, y, z) \Rightarrow R(x, y, z)$ chase with $A \rightarrow B$.

Jan Chomicki
Database Consistency
June 2008
A set of attributes \( X \subseteq U \) is a key with respect to a set of FDs \( \Sigma \) if:

- \( \Sigma \) implies \( X \rightarrow U \)
- for no proper subset \( Y \) of \( X \), \( \Sigma \) implies \( Y \rightarrow U \)
Keys

A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:

- $\Sigma$ implies $X \rightarrow U$
- for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$

Decomposition

A decomposition $\mathcal{R} = (R_1, \ldots, R_n)$ of a schema $R$ has the lossless join property with respect to a set of FDs $\Sigma$ iff $\Sigma$ implies the join dependency $\bowtie [\mathcal{R}]$. 

($\bowtie$ indicates the lossless join property)
Keys

A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:

- $\Sigma$ implies $X \rightarrow U$
- for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$

Decomposition

A decomposition $R = (R_1, \ldots, R_n)$ of a schema $R$ has the lossless join property with respect to a set of FDs $\Sigma$ iff $\Sigma$ implies the join dependency $\Join [R]$.

Decomposition $(R_1, R_2)$ of $R(ABC)$

Relation schemas: $R_1(AB)$ with FD $A \rightarrow B$, $R_2(AC)$. 
Application: database design

### Keys

A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:

- $\Sigma$ implies $X \rightarrow U$
- for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$

### Decomposition

A decomposition $R = (R_1, \ldots, R_n)$ of a schema $R$ has the **lossless join property** with respect to a set of FDs $\Sigma$ iff $\Sigma$ implies the join dependency $\Join [R]$.

### Decomposition $(R_1, R_2)$ of $R(ABC)$

Relation schemas: $R_1(AB)$ with FD $A \rightarrow B$, $R_2(AC)$.

Terminal chase sequence:
Keys

A set of attributes $X \subseteq U$ is a key with respect to a set of FDs $\Sigma$ if:

- $\Sigma$ implies $X \rightarrow U$
- for no proper subset $Y$ of $X$, $\Sigma$ implies $Y \rightarrow U$

Decomposition

A decomposition $\mathcal{R} = (R_1, \ldots, R_n)$ of a schema $R$ has the lossless join property with respect to a set of FDs $\Sigma$ iff $\Sigma$ implies the join dependency $\Join [\mathcal{R}]$.

Decomposition $(R_1, R_2)$ of $R(ABC)$

Relation schemas: $R_1(AB)$ with FD $A \rightarrow B$, $R_2(AC)$.

Terminal chase sequence:

$R(x, y, z') \land R(x, y', z) \Rightarrow R(x, y, z)$ given JD
### Keys

A set of attributes \( X \subseteq U \) is a **key** with respect to a set of FDs \( \Sigma \) if:

- \( \Sigma \) implies \( X \rightarrow U \)
- for no proper subset \( Y \) of \( X \), \( \Sigma \) implies \( Y \rightarrow U \)

### Decomposition

A decomposition \( \mathcal{R} = (R_1, \ldots, R_n) \) of a schema \( R \) has the **lossless join property** with respect to a set of FDs \( \Sigma \) iff \( \Sigma \) implies the join dependency \( \bowtie [\mathcal{R}] \).

### Decomposition \((R_1, R_2)\) of \(R(ABC)\)

Relation schemas: \( R_1(AB) \) with FD \( A \rightarrow B \), \( R_2(AC) \).

Terminal chase sequence:

\[
R(x, y, z') \land R(x, y', z) \Rightarrow R(x, y, z) \quad \text{given JD}
\]

\[
R(x, y, z') \land R(x, y, z) \Rightarrow R(x, y, z) \quad \text{chase with } A \rightarrow B
\]
Goal

Exchange of data between independent databases with different schemas.
Application: data exchange

Goal

Exchange of data between independent databases with different schemas.

Setting for data exchange

- **source** and **target** schemas
- **source-to-target dependencies**: describe how the data is mapped between source and target
- **target integrity constraints**
Application: data exchange

Goal
Exchange of data between independent databases with different schemas.

Setting for data exchange
- source and target schemas
- source-to-target dependencies: describe how the data is mapped between source and target
- target integrity constraints

Data exchange is a specific scenario for data integration, in which a target instance is constructed.
**Constraints and solutions**

$\phi_S$, $\phi_T$, $\psi_T$ are conjunctions of relational atoms over source and target.

**Source-to-target dependencies $\Sigma_{st}$**

- **tuple-generating dependencies**: $\forall x \ (\phi_S(x) \Rightarrow \exists y \ \psi_T(x, y))$.

**Target integrity constraints $\Sigma_t$**

- **tuple-generating dependencies (tgds)**: $\forall x \ (\phi_T(x) \Rightarrow \exists y \ \psi_T(x, y))$
- **equality-generating dependencies**: $\forall x \ (\phi_T(x) \Rightarrow x_1 = x_2)$. 

**Solution**

Given a source instance $I$, a target instance $J$ is a solution for $I$ if $J$ satisfies $\Sigma_t$ and $(I, J)$ satisfy $\Sigma_{st}$.

A universal solution for $I$ if it is a solution for $I$ and there is a homomorphism from it to any other solution for $I$.

Solutions can contain labelled nulls.

There may be multiple solutions.
Constraints and solutions

$\phi_S$, $\phi_T$, $\psi_T$ are conjunctions of relational atoms over source and target.

**Source-to-target dependencies $\Sigma_{st}$**
- tuple-generating dependencies: $\forall x (\phi_S(x) \Rightarrow \exists y \psi_T(x, y))$.

**Target integrity constraints $\Sigma_t$**
- tuple-generating dependencies (tgds): $\forall x (\phi_T(x) \Rightarrow \exists y \psi_T(x, y))$
- equality-generating dependencies: $\forall x (\phi_T(x) \Rightarrow x_1 = x_2)$.

**Solution**

Given a source instance $I$, a target instance $J$ is
- a solution for $I$ if $J$ satisfies $\Sigma_t$ and ($I$, $J$) satisfy $\Sigma_{st}$
- a universal solution for $I$ if it is a solution for $I$ and there is a homomorphism from it to any other solution for $I$
- solutions can contain labelled nulls
Constraints and solutions

\(\phi_S, \phi_T, \psi_T\) are conjunctions of relational atoms over source and target.

**Source-to-target dependencies \(\Sigma_{st}\)**
- **tuple-generating dependencies:** \(\forall x (\phi_S(x) \Rightarrow \exists y \psi_T(x, y))\).

**Target integrity constraints \(\Sigma_t\)**
- **tuple-generating dependencies (tgds):** \(\forall x (\phi_T(x) \Rightarrow \exists y \psi_T(x, y))\)
- **equality-generating dependencies:** \(\forall x (\phi_T(x) \Rightarrow x_1 = x_2)\).

**Solution**

Given a source instance \(I\), a target instance \(J\) is
- a **solution** for \(I\) if \(J\) satisfies \(\Sigma_t\) and \((I, J)\) satisfy \(\Sigma_{st}\)
- a **universal solution** for \(I\) if it is a solution for \(I\) and there is a homomorphism from it to any other solution for \(I\)
- solutions can contain **labelled nulls**

There may be multiple solutions.
Certain answer

Given a query $Q$ and a source instance $I$, a tuple $t$ is a **certain answer** with respect to $I$ if $t$ is an answer to $Q$ in every solution $J$ for $I$. 
Certain answer

Given a query $Q$ and a source instance $I$, a tuple $t$ is a certain answer with respect to $I$ if $t$ is an answer to $Q$ in every solution $J$ for $I$.

Conjunctive queries

- relational calculus: $\exists, \wedge$
- relational algebra: $\sigma, \pi, \times$
Certain answer

Given a query $Q$ and a source instance $I$, a tuple $t$ is a certain answer with respect to $I$ if $t$ is an answer to $Q$ in every solution $J$ for $I$.

Conjunctive queries

- relational calculus: $\exists, \land$
- relational algebra: $\sigma, \pi, \times$

Query evaluation

1. construct any universal solution $J_0$
2. evaluate the query over $J_0$
3. discard answers with nulls
4. the above returns certain answers for unions of conjunctive queries without inequalities
Applying a variant of the chase [AHV95] to the source instance using target and source-to-target dependencies, obtaining a sequence of instances $I_0 = I, I_1, \ldots, I_n, \ldots$. 
Apply a variant of the chase [AHV95] to the source instance using target and source-to-target dependencies, obtaining a sequence of instances $I_0 = I, I_1, \ldots, I_n, \ldots$.

### Chasing a tgd $C$

1. find a substitution $h$ that (1) $h$ makes the LHS of $C$ true in the constructed instance $I_j$, and (2) $h$ cannot be extended to a substitution that makes the RHS of $C$ true in that instance
2. apply $h$ to the RHS of $C$, mapping the existentially quantified variables to fresh labelled nulls
3. add the resulting facts to $I_j$, obtaining $I_{j+1}$. 

Apply a variant of the chase [AHV95] to the source instance using target and source-to-target dependencies, obtaining a sequence of instances \( l_0, l_1, \ldots, l_n, \ldots \).

**Chasing a tgd \( C \)**

1. find a substitution \( h \) that (1) \( h \) makes the LHS of \( C \) true in the constructed instance \( l_j \), and (2) \( h \) cannot be extended to a substitution that makes the RHS of \( C \) true in that instance
2. apply \( h \) to the RHS of \( C \), mapping the existentially quantified variables to fresh labelled nulls
3. add the resulting facts to \( l_j \), obtaining \( l_{j+1} \).

**Chasing an egd \( C \)**

Find a substitution \( h \) such that makes the LHS of \( C \) true in \( l_j \) and \( h(x_1) \neq h(x_2) \):

- if \( h(x_1) \) and \( h(x_2) \) are constants, then FAILURE
- otherwise, identify \( h(x_1) \) and \( h(x_2) \) in \( l_j \) (preferring constants), obtaining \( l_{j+1} \).
Chase at work

Source and target databases

Source: \( \text{Emp}(N, A), \text{Num}(N, \text{Id}) \)

Target: \( \text{Name}(\text{Id}, N), \text{Addr}(\text{Id}, A) \)

Source-to-target dependencies

\[
\forall n, a. \text{Emp}(n, a) \Rightarrow \exists id. \text{Name}(id, n) \land \text{Addr}(id, a)
\]

\[
\forall n, a, id. \text{Emp}(n, a) \land \text{Num}(n, id) \Rightarrow \text{Name}(id, n)
\]

Target constraints

\( \text{Name}: N \rightarrow \text{Id}, \text{Id} \rightarrow N, \text{Addr}: \text{Id} \rightarrow A \)

Chase sequence

\[
I_0 = \{ \text{Emp}(\text{Li}, \text{LA}), \text{Num}(\text{Li}, 111) \}
\]

\[
I_1 = \{ \text{Emp}(\text{Li}, \text{LA}), \text{Num}(\text{Li}, 111), \text{Name}(\text{id}_1, \text{Li}), \text{Addr}(\text{id}_1, \text{LA}) \}
\]

\[
I_2 = \{ \text{Emp}(\text{Li}, \text{LA}), \text{Num}(\text{Li}, 111), \text{Name}(\text{id}_1, \text{Li}), \text{Addr}(\text{id}_1, \text{LA}), \text{Name}(111, \text{Li}) \}
\]

\[
I_3 = \{ \text{Emp}(\text{Li}, \text{LA}), \text{Num}(\text{Li}, 111), \text{Name}(111, \text{Li}), \text{Addr}(111, \text{LA}) \}
\]
### Source and target databases

**Source:** $Emp(N, A), Num(N, Id)$  
**Target:** $Name(Id, N), Addr(Id, A)$
Chase at work

Source and target databases

Source: $Emp(N, A), Num(N, Id)$ \hspace{1cm} Target: $Name(Id, N), Addr(Id, A)$

Source-to-target dependencies

$\forall n, a. \; Emp(n, a) \Rightarrow \exists id. \; Name(id, n) \land Addr(id, a)$

$\forall n, a, id. \; Emp(n, a) \land Num(n, id) \Rightarrow Name(id, n)$

Target constraints

$Name: N \rightarrow Id, Id \rightarrow N, Addr: Id \rightarrow A.$
Chase at work

Source and target databases

Source: \(\text{Emp}(N, A), \text{Num}(N, Id)\)  
Target: \(\text{Name}(Id, N), \text{Addr}(Id, A)\)

Source-to-target dependencies

\(\forall n, a. \text{Emp}(n, a) \Rightarrow \exists id. \text{Name}(id, n) \land \text{Addr}(id, a)\)

\(\forall n, a, id. \text{Emp}(n, a) \land \text{Num}(n, id) \Rightarrow \text{Name}(id, n)\)

Target constraints

\(\text{Name} : N \rightarrow Id, Id \rightarrow N, \text{Addr} : Id \rightarrow A.\)

Chase sequence
Chase at work

Source and target databases

Source: $Emp(N, A), Num(N, Id)$  
Target: $Name(Id, N), Addr(Id, A)$

Source-to-target dependencies

$\forall n, a. \ Emp(n, a) \Rightarrow \exists id. \ Name(id, n) \land Addr(id, a)$

$\forall n, a, id. \ Emp(n, a) \land Num(n, id) \Rightarrow Name(id, n)$

Target constraints

$Name : N \rightarrow Id, Id \rightarrow N, Addr : Id \rightarrow A.$

Chase sequence

$I_0 = \{Emp(Li, LA), Num(Li, 111)\}$
Chase at work

Source and target databases

Source: $Emp(N, A)$, $Num(N, Id)$  
Target: $Name(Id, N)$, $Addr(Id, A)$

Source-to-target dependencies

$\forall n, a. \ Emp(n, a) \Rightarrow \exists id. \ Name(id, n) \land Addr(id, a)$

$\forall n, a, id. \ Emp(n, a) \land Num(n, id) \Rightarrow Name(id, n)$

Target constraints

$Name : N \rightarrow Id, Id \rightarrow N, Addr : Id \rightarrow A.$

Chase sequence

$I_0 = \{Emp(Li, LA), Num(Li, 111)\}$

$I_1 = \{Emp(Li, LA), Num(Li, 111), Name(id_1, Li), Addr(id_1, LA)\}$
Chase at work

Source and target databases

Source: $Emp(N, A), Num(N, Id)$  
Target: $Name(Id, N), Addr(Id, A)$

Source-to-target dependencies

$\forall n, a. \ Emp(n, a) \Rightarrow \exists id. \ Name(id, n) \land Addr(id, a)$
$\forall n, a, id. \ Emp(n, a) \land Num(n, id) \Rightarrow Name(id, n)$

Target constraints

$Name: N \rightarrow Id, Id \rightarrow N, Addr: Id \rightarrow A.$

Chase sequence

$l_0 = \{Emp(Li, LA), Num(Li, 111)\}$
$l_1 = \{Emp(Li, LA), Num(Li, 111), Name(id_1, Li), Addr(id_1, LA)\}$
$l_2 = \{Emp(Li, LA), Num(Li, 111), Name(id_1, Li), Addr(id_1, LA), Name(111, Li)\}$
Chase at work

Source and target databases

**Source:** \( \text{Emp}(N, A), \text{Num}(N, Id) \)     **Target:** \( \text{Name}(Id, N), \text{Addr}(Id, A) \)

Source-to-target dependencies

\( \forall n, a. \text{Emp}(n, a) \Rightarrow \exists id. \text{Name}(id, n) \land \text{Addr}(id, a) \)

\( \forall n, a, id. \text{Emp}(n, a) \land \text{Num}(n, id) \Rightarrow \text{Name}(id, n) \)

Target constraints

\( \text{Name} : N \rightarrow Id, Id \rightarrow N, \text{Addr} : Id \rightarrow A. \)

Chase sequence

\( I_0 = \{ \text{Emp}(Li, LA), \text{Num}(Li, 111) \} \)

\( I_1 = \{ \text{Emp}(Li, LA), \text{Num}(Li, 111), \text{Name}(id_1, Li), \text{Addr}(id_1, LA) \} \)

\( I_2 = \{ \text{Emp}(Li, LA), \text{Num}(Li, 111), \text{Name}(id_1, Li), \text{Addr}(id_1, LA), \text{Name}(111, Li) \} \)

\( I_3 = \{ \text{Emp}(Li, LA), \text{Num}(Li, 111), \text{Name}(111, Li), \text{Addr}(111, LA) \} \)
Chase termination

For weakly acyclic tgds, each chase sequence is of length polynomial in the size of the input.

Data complexity of computing certain answers in PTIME for unions of conjunctive queries (without inequalities) and constraints that are egds and weakly acyclic tgds co-NP-complete for unions of conjunctive queries (with inequalities) and constraints that are egds and weakly acyclic tgds.
Chase termination

Chase result

- there is a sequence of chase applications that ends in failure: no universal solution
- otherwise: every finite sequence that cannot be extended yields a universal solution
Chase termination

**Chase result**
- there is a sequence of chase applications that ends in failure: no universal solution
- otherwise: every finite sequence that cannot be extended yields a universal solution

**Termination**
For weakly acyclic tgds, each chase sequence is of length polynomial in the size of the input.
Chase termination

**Chase result**
- there is a sequence of chase applications that ends in failure: **no universal solution**
- otherwise: every finite sequence that cannot be extended yields a **universal solution**

**Termination**
For **weakly acyclic tgds**, each chase sequence is of length **polynomial** in the size of the input.

**Data complexity of computing certain answers**
- in **PTIME** for unions of conjunctive queries (without inequalities) and constraints that are egds and weakly acyclic tgds
- **co-NP-complete** for unions of conjunctive queries (with inequalities) and constraints that are egds and weakly acyclic tgds
Application: semantic query optimization

- Rewrite-based cost-based
- Semantic query optimization
  - Rewritings enabled by satisfaction of integrity constraints:
    - join elimination/introduction
    - predicate elimination/introduction
    - eliminating redundancies
Query optimization

- rewrite-based
- cost-based
Application: semantic query optimization

Query optimization
- rewrite-based
- cost-based

Semantic query optimization
Rewritings enabled by satisfaction of integrity constraints:
- join elimination/introduction
- predicate elimination/introduction
- eliminating redundancies
- ...

Jan Chomicki

Database Consistency

June 2008
Preference queries

The winnow operator

\[ \omega \]

(Chomicki [Cho03])

Find the best answers to a query, according to a given preference relation \( \succcurlyeq \).

Relation

Book(Title,Vendor,Price)

Preference: 

\((i_1,v_1,p_1) \succcurlyeq_C (i_2,v_2,p_2) \equiv i_1 = i_2 \land p_1 < p_2\)

Indifference:

\((i_1,v_1,p_1) \sim_C (i_2,v_2,p_2) \equiv i_1 \neq i_2 \lor p_1 = p_2\)
Preference queries

The winnow operator $\omega_C$ (Chomicki [Cho03])

Find the best answers to a query, according to a given preference relation $\succ_C$. 
The winnow operator $\omega_C$ (Chomicki [Cho03])

Find the best answers to a query, according to a given preference relation $\succ_C$.

**Relation** $Book(Title, Vendor, Price)$

**Preference**: $(i_1, v_1, p_1) \succ_C (i_2, v_2, p_2) \equiv i_1 = i_2 \land p_1 < p_2$

**Indifference**: $(i_1, v_1, p_1) \sim_C (i_2, v_2, p_2) \equiv i_1 \neq i_2 \lor p_1 = p_2$
Preference queries

The winnow operator $\omega_C$ (Chomicki [Cho03])

Find the best answers to a query, according to a given preference relation $\succ_C$.

Relation $Book(Title, Vendor, Price)$

Preference: $(i_1, v_1, p_1) \succ_C (i_2, v_2, p_2) \equiv i_1 = i_2 \land p_1 < p_2$

Indifference: $(i_1, v_1, p_1) \sim_C (i_2, v_2, p_2) \equiv i_1 \neq i_2 \lor p_1 = p_2$

<table>
<thead>
<tr>
<th>Book</th>
<th>Title</th>
<th>Vendor</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>The Flanders Panel</td>
<td>amazon.com</td>
<td>$14.75</td>
</tr>
<tr>
<td>$t_2$</td>
<td>The Flanders Panel</td>
<td>fatbrain.com</td>
<td>$13.50</td>
</tr>
<tr>
<td>$t_3$</td>
<td>The Flanders Panel</td>
<td>bn.com</td>
<td>$18.80</td>
</tr>
<tr>
<td>$t_4$</td>
<td>Green Guide: Greece</td>
<td>bn.com</td>
<td>$17.30</td>
</tr>
</tbody>
</table>
Preference queries

The winnow operator $\omega_C$ (Chomicki [Cho03])

Find the best answers to a query, according to a given preference relation $\succ_C$.

Relation $\text{Book}(\text{Title}, \text{Vendor}, \text{Price})$

Preference: $(i_1, v_1, p_1) \succ_C (i_2, v_2, p_2) \equiv i_1 = i_2 \land p_1 < p_2$

Indifference: $(i_1, v_1, p_1) \sim_C (i_2, v_2, p_2) \equiv i_1 \neq i_2 \lor p_1 = p_2$

<table>
<thead>
<tr>
<th>Book</th>
<th>Title</th>
<th>Vendor</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>The Flanders Panel</td>
<td>amazon.com</td>
<td>$14.75</td>
</tr>
<tr>
<td>$t_2$</td>
<td>The Flanders Panel</td>
<td>fatbrain.com</td>
<td>$13.50</td>
</tr>
<tr>
<td>$t_3$</td>
<td>The Flanders Panel</td>
<td>bn.com</td>
<td>$18.80</td>
</tr>
<tr>
<td>$t_4$</td>
<td>Green Guide: Greece</td>
<td>bn.com</td>
<td>$17.30</td>
</tr>
</tbody>
</table>
Given a set of integrity constraints $\Sigma$, $\omega_C(r) = r$ for every relation $r$ satisfying $\Sigma$ iff $\Sigma$ implies the dependency $R(t_1) \land R(t_2) \Rightarrow t_1 \sim_C t_2$.

Example:

$\text{Book}(i_1, v_1, p_1) \land \text{Book}(i_2, v_2, p_2) \Rightarrow i_1 \neq i_2 \lor p_1 = p_2$ is a functional dependency in disguise:

$\text{Book}(i_1, v_1, p_1) \land \text{Book}(i_2, v_2, p_2) \land i_1 = i_2 \Rightarrow p_1 = p_2$.

If this dependency is implied by $\Sigma$, $\omega_C(\text{Book}) = \text{Book}$.

Constraint-generating dependencies (Baudinet et al. [BCW95]) general form:

$\forall t_1, \ldots, t_n. R(t_1) \land \cdots \land R(t_n) \land C(t_1, \ldots, t_n) \Rightarrow C_0(t_1, \ldots, t_n)$

implication of CGDs is decidable for decidable constraint classes implication in PTIME for some classes of CGDs axiomatization not known.
Eliminating redundant occurrences of winnow

Redundant winnow (Chomicki [Cho07b])

Given a set of integrity constraints $\Sigma$, $\omega_C(r) = r$ for every relation $r$ satisfying $\Sigma$ iff $\Sigma$ implies the dependency $R(t_1) \land R(t_2) \Rightarrow t_1 \sim_C t_2$. 
Eliminating redundant occurrences of winnow

**Redundant winnow (Chomicki [Cho07b])**

Given a set of integrity constraints $\Sigma$, $\omega_C(r) = r$ for every relation $r$ satisfying $\Sigma$ iff $\Sigma$ implies the dependency $R(t_1) \land R(t_2) \Rightarrow t_1 \sim_C t_2$.

**Example**

$Book(i_1, v_1, p_1) \land Book(i_2, v_2, p_2) \Rightarrow i_1 \neq i_2 \lor p_1 = p_2$

is a functional dependency in disguise:

$Book(i_1, v_1, p_1) \land Book(i_2, v_2, p_2) \land i_1 = i_2 \Rightarrow p_1 = p_2$.

If this dependency is implied by $\Sigma$, $\omega_C(Book) = Book$. 

constraint-generating dependencies (Baudinet et al. [BCW95])

general form:

$\forall t_1, ..., t_n. R(t_1) \land \ldots \land R(t_n) \land C(t_1, \ldots, t_n) \Rightarrow C_0(t_1, \ldots, t_n)$

implication of CGDs is decidable for decidable constraint classes

implication in PTIME for some classes of CGDs

axiomatization not known
Eliminating redundant occurrences of winnow

Redundant winnow (Chomicki [Cho07b])

Given a set of integrity constraints \( \Sigma \), \( \omega_C(r) = r \) for every relation \( r \) satisfying \( \Sigma \) iff \( \Sigma \) implies the dependency \( R(t_1) \land R(t_2) \Rightarrow t_1 \sim_C t_2 \).

Example

\[
Book(i_1, v_1, p_1) \land Book(i_2, v_2, p_2) \Rightarrow i_1 \neq i_2 \lor p_1 = p_2
\]

is a functional dependency in disguise:

\[
Book(i_1, v_1, p_1) \land Book(i_2, v_2, p_2) \land i_1 = i_2 \Rightarrow p_1 = p_2.
\]

If this dependency is implied by \( \Sigma \), \( \omega_C(Book) = Book \).

Constraint-generating dependencies (Baudinet et al. [BCW95])

- general form: \( \forall t_1, \ldots t_n. R(t_1) \land \cdots \land R(t_n) \land C(t_1, \ldots, t_n) \Rightarrow C_0(t_1, \ldots, t_n) \)
- implication of CGDs is decidable for decidable constraint classes
- implication in PTIME for some classes of CGDs
- axiomatization not known
Prospects for integrity constraints

Schema mapping
second-order dependencies to achieve closure under composition (Fagin et al. \cite{FKPT05})

Data cleaning
conditional functional and inclusion dependencies (Bohannon et al. \cite{BFG07}, Bravo et al. \cite{BFM07})
matching dependencies for object identification (Fan \cite{Fan08})

XML
many different semantics

Semantic Web
knowledge bases and ontologies

Data mining
discovery of FDs and INDs
Prospects for integrity constraints

Schema mapping

- second-order dependencies to achieve closure under composition (Fagin et al. [FKPT05])
Prospects for integrity constraints

Schema mapping
- second-order dependencies to achieve closure under composition (Fagin et al. [FKPT05])

Data cleaning
- conditional functional and inclusion dependencies (Bohannon et al. [BFG+07], Bravo et al. [BFM07])
- matching dependencies for object identification (Fan [Fan08])
### Prospects for integrity constraints

#### Schema mapping
- second-order dependencies to achieve closure under composition (Fagin et al. [FKPT05])

#### Data cleaning
- conditional functional and inclusion dependencies (Bohannon et al. [BFG+07], Bravo et al. [BFM07])
- matching dependencies for object identification (Fan [Fan08])

#### XML
- many different semantics
Prospects for integrity constraints

Schema mapping
- second-order dependencies to achieve closure under composition (Fagin et al. [FKPT05])

Data cleaning
- conditional functional and inclusion dependencies (Bohannon et al. [BFG+07], Bravo et al. [BFM07])
- matching dependencies for object identification (Fan [Fan08])

XML
- many different semantics

Semantic Web
- knowledge bases and ontologies
- extensions of ICs
### Schema mapping
- second-order dependencies to achieve closure under composition (Fagin et al. [FKPT05])

### Data cleaning
- conditional functional and inclusion dependencies (Bohannon et al. [BFG+07], Bravo et al. [BFM07])
- matching dependencies for object identification (Fan [Fan08])

### XML
- many different semantics

### Semantic Web
- knowledge bases and ontologies
- extensions of ICs

### Data mining
- discovery of FDs and INDs
Part II

Consistent query answers
Outline of Part II

6 Motivation

7 Basics

8 Computing CQA
   - Methods
   - Complexity

9 Variants of CQA

10 Conclusions
### 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.
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
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
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
Horizontal Decomposition

Decomposition into two relations:

- violators
- the rest

(De Bra, Paredaens [DBP83])
Horizontal Decomposition

Decomposition into two relations:

- violators
- the rest

(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>

(De Bra, Paredaens [DBP83])
Horizontal Decomposition

Decomposition into two relations:
- violators
- the rest

(De Bra, Paredaens [DBP83])

<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
Exceptions to Constraints

Weakening the constraints:

- functional dependencies $\rightarrow$ denial constraints

(Borgida [Bor85])
Exceptions to Constraints

Weakening the constraints:
- functional dependencies $\rightarrow$ denial constraints

(Borgida [Bor85])

<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>
Exceptions to Constraints

Weakening the constraints:

- functional dependencies → denial constraints

(Borgida [Bor85])

<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'
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)

**Our 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>

Name → City Salary
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>

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

<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>

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

(Arenas, Bertossi, Chomicki [ABC99])
Consistent query answer:

Query answer obtained in every repair.

(Arenas, Bertossi, Chomicki [ABC99])

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

(Arenas, Bertossi, Chomicki [ABC99])

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

SELECT Name
FROM Employee
WHERE Salary ≤ 25M
Consistent Query Answering

Consistent query answer:
Query answer obtained in every repair.

(Arenas, Bertossi, Chomicki [ABC99])

<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
Research Goals

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 compute consistent information.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Computational complexity analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>tractable 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].
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].
A Logical Aside

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.

\[
\begin{array}{|c|c|}
\hline
A & B \\
\hline
a_1 & b_1 \\
a_1 & c_1 \\
a_2 & b_2 \\
a_2 & c_2 \\
\vdots & \\
a_n & b_n \\
a_n & c_n \\
\hline
\end{array}
\]

$A \rightarrow B$

It is impractical to apply the definition of CQA directly.
Query Rewriting

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

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

∀. ¬A₁ ∨ ··· ∨ ¬Aₙ ∨ B₁ ∨ ··· ∨ Bₘ
Constraint classes

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

Example
\[ \forall. \neg \text{Par}(x) \lor \text{Ma}(x) \lor \text{Fa}(x) \]
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
\[ \forall. \neg A_1 \lor \cdots \lor \neg A_n \]
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 \( Y = U \)
- 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 \( Y = U \)

- 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

**Example primary-key dependency**

Name → 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 \( Y = U \)
- 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
\[ M[Manager] \subseteq M[Name] \]
Building queries that compute CQAs

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

Building queries that compute CQAs

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

Query

$Emp(x, y, z)$
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) $\sim$ relational calculus (algebra)
- SQL $\sim$ SQL
- leads to 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) $\sim$ relational calculus (algebra)
- SQL $\sim$ SQL
- leads to PTIME data complexity

**Query**

$Emp(x, y, z)$

**Integrity constraint**

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

**Rewritten query**

$Emp(x, y, z) \land \forall y', z'. \neg Emp(x, y', z') \lor z = z'$
(Arenas, Bertossi, Chomicki [ABC99])

- Integrity constraints: binary universal
- Queries: conjunctions of literals (relational algebra: $\sigma, \times, -$)
<table>
<thead>
<tr>
<th>(Arenas, Bertossi, Chomicki [ABC99])</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integrity constraints:</strong> binary universal</td>
</tr>
<tr>
<td><strong>Queries:</strong> conjunctions of literals (relational algebra: $\sigma, \times, -$)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(Fuxman, Miller [FM07])</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integrity constraints:</strong> primary key functional dependencies</td>
</tr>
<tr>
<td><strong>Queries:</strong> $C_{forest}$</td>
</tr>
<tr>
<td>- a class of conjunctive queries ($\pi, \sigma, \times$)</td>
</tr>
<tr>
<td>- no non-key or non-full joins</td>
</tr>
<tr>
<td>- no repeated relation symbols</td>
</tr>
<tr>
<td>- no built-ins</td>
</tr>
<tr>
<td><strong>Generalization:</strong> conjunctive queries expressed as rooted rules (Wijsen [Wij07])</td>
</tr>
</tbody>
</table>
SQL query

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

```sql
SELECT Name FROM Emp
WHERE Salary \geq 10K
```

SQL rewritten query

```sql
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 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)
```

**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)
Algorithm HProver

INPUT: query $\Phi$ a disjunction of ground literals, 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 $S \supseteq \{B_1, \ldots, B_m\}$.
Computing CQAs Using Conflict Hypergraphs

Algorithm HProver

INPUT: query $\Phi$ a disjunction of ground literals, 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 $S \supseteq \{B_1, \ldots, B_m\}$.

(Chomicki, Marcinkowski, Staworko [CMS04])

- 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, Chomicki [ABC03])
- (Greco, Greco, Zumpano [GGZ03])
- (Calì, Lembo, Rosati [CLR03b])
Logic programs

Specifying repairs as answer sets of logic programs

- (Arenas, Bertossi, Chomicki [ABC03])
- (Greco, Greco, Zumpano [GGZ03])
- (Calì, Lembo, Rosati [CLR03b])

Example

\begin{align*}
\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'.
\end{align*}
Logic programs

Specifying repairs as answer sets of logic programs

- (Arenas, Bertossi, Chomicki [ABC03])
- (Greco, Greco, Zumpano [GGZ03])
- (Calì, Lembo, Rosati [CLR03b])

Example

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

Answer sets

- \{ \text{emp}(\text{Gates, Redmond, 20M}), \text{emp}(\text{Grove, SantaClara, 10M}), \ldots \}\n- \{ \text{emp}(\text{Gates, Redmond, 30M}), \text{emp}(\text{Grove, SantaClara, 10M}), \ldots \}\n
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

- 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 $\Pi^P_2$-complete
- dlv, smodels, ...

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

INFOMIX (Eiter et al. [EFGL03])
- combines CQA with data integration (GAV)
- uses dlv for repair computations
- optimization techniques: localization, factorization
- tested on small-to-medium-size legacy databases
Theorem (Chomicki, Marcinkowski [CM05a])

For primary-key functional dependencies and conjunctive queries, consistent query answering is \textit{data-complete for co-NP}.
Co-NP-completeness of CQA

**Theorem (Chomicki, Marcinkowski [CM05a])**

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 \( W = S \cup M \).

**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 (Chomicki, Marcinkowski [CM05a])

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 $W = S \cup M$.

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 $C_{\text{forest}}$. 
### 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>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></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>

- (Arenas, Bertossi, Chomicki [ABC99])
### 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, Chomicki [ABC99])
- (Chomicki, Marcinkowski [CM05a])
### 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, Chomicki [ABC99])
- (Chomicki, Marcinkowski [CM05a])
- (Fuxman, Miller [FM07])
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 $\Pi^p_2$-complete</td>
</tr>
<tr>
<td>$\sigma, \times, -, \cup$</td>
<td>PTIME</td>
<td>PTIME</td>
<td>PTIME</td>
<td>$\Pi^p_2$-complete</td>
</tr>
<tr>
<td>$\sigma, \pi$</td>
<td>PTIME</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>$\Pi^p_2$-complete</td>
</tr>
<tr>
<td>$\sigma, \pi, \times$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>$\Pi^p_2$-complete</td>
</tr>
<tr>
<td>$\sigma, \pi, \times, -, \cup$</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>co-NPC</td>
<td>$\Pi^p_2$-complete</td>
</tr>
</tbody>
</table>

- (Arenas, Bertossi, Chomicki [ABC99])
- (Chomicki, Marcinkowski [CM05a])
- (Fuxman, Miller [FM07])
- (Staworko, Ph.D., 2007):
  - co-NPC for full TGDs and denial constraints
  - PTIME for acyclic full TGDs and denial constraints
Tuple-based repairs

- asymmetric treatment of insertion and deletion:
  - repairs by minimal deletions only (Chomicki, Marcinkowski [CM05a]): data possibly incorrect but complete
  - repairs by minimal deletions and arbitrary insertions (Calì, Lembo, Rosati [CLR03a]): data possibly incorrect and incomplete

- minimal cardinality changes (Lopatenko, Bertossi [LB07])
Tuple-based repairs

- asymmetric treatment of insertion and deletion:
  - repairs by minimal deletions only (Chomicki, Marcinkowski [CM05a]): data possibly incorrect but complete
  - repairs by minimal deletions and arbitrary insertions (Calì, Lembo, Rosati [CLR03a]): data possibly incorrect and incomplete
- minimal cardinality changes (Lopatenko, Bertossi [LB07])

Attribute-based repairs

- (A) ground and non-ground repairs (Wijsen [Wij05])
- (B) project-join repairs (Wijsen [Wij06])
- (C) repairs minimizing Euclidean distance (Bertossi et al. [BBFL05])
- (D) repairs of minimum cost (Bohannon et al. [BFFR05])
The Explosion of Semantics

**Tuple-based repairs**
- asymmetric treatment of insertion and deletion:
  - repairs by minimal deletions only (Chomicki, Marcinkowski [CM05a]): data possibly incorrect but complete
  - repairs by minimal deletions and arbitrary insertions (Calì, Lembo, Rosati [CLR03a]): data possibly incorrect and incomplete
- minimal cardinality changes (Lopatenko, Bertossi [LB07])

**Attribute-based repairs**
- (A) ground and non-ground repairs (Wijsen [Wij05])
- (B) project-join repairs (Wijsen [Wij06])
- (C) repairs minimizing Euclidean distance (Bertossi et al. [BBFL05])
- (D) repairs of minimum cost (Bohannon et al. [BFFR05])

**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>
```

*EmpDept*

Name \(\rightarrow\) Dept

Dept \(\rightarrow\) City
The Need for Attribute-based Repairing

Tuple-based repairing leads to information loss.

- **EmpDept**
  - **Name** | **Dept** | **Location**
  - John  | Sales | Buffalo
  - Mary  | Sales | Toronto

- **Name** → **Dept**
- **Dept** → **City**
Repair the lossless join decomposition:

\[ \pi_{Name, Dept}(EmpDept) \bowtie \pi_{Dept, Location}(EmpDept) \]
Repair the lossless join decomposition:

\[ \pi_{\text{Name}, \text{Dept}}(\text{EmpDept}) \Join \pi_{\text{Dept}, \text{Location}}(\text{EmpDept}) \]
Attribute-based Repairs through Tuple-based Repairs (Wijsen [Wij06])

Repair the lossless join decomposition:

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

<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>Mary</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>Toronto</td>
</tr>
</tbody>
</table>

Name → Dept
Dept → City

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

(Andritsos, Fuxman, Miller [AFM06])

- 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
Probabilistic framework for “dirty” databases

(Andritsos, Fuxman, Miller [AFM06])

- potential duplicates identified and grouped into clusters
- worlds ★ 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>
SQL query

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

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

<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>
## Technology

- **practical methods** for CQA for a subset of SQL:
  - restricted conjunctive/aggregation queries, primary/foreign-key constraints
  - quantifier-free queries, denial constraints/acyclic TGDs/JDs
  - LP-based approaches for expressive query/constraint languages

- emergence of **generic techniques**

- 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/acyclic TGDs/JDs
  - LP-based approaches for expressive query/constraint languages
- emergence of **generic** techniques
- implemented in **prototype systems**
- tested on **medium-size databases**

The CQA Community

- over 30 active researchers
- over 100 publications (since 1999)
- overview papers [BC03, Ber06, Cho07a, CM05b]
- 2007 SIGMOD Doctoral Dissertation Award (Ariel Fuxman)
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
“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
Taking Stock: Largely Open Issues

Applications

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

CQA in context

- taming the semantic explosion
- 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

CQA in context
- taming the semantic explosion
- CQA and data cleaning
- CQA and schema matching/mapping

Foundations
- repair checking
- 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

CQA in context
- taming the semantic explosion
- CQA and data cleaning
- CQA and schema matching/mapping

Foundations
- repair checking
- defining measures of consistency
- more refined complexity analysis, dynamic aspects
M. Arenas, L. Bertossi, and J. Chomicki.
Consistent Query Answers in Inconsistent Databases.

M. Arenas, L. Bertossi, and J. Chomicki.
Answer Sets for Consistent Query Answering in Inconsistent Databases.

P. Andritsos, A. Fuxman, and R. Miller.
Clean Answers over Dirty Databases.
In IEEE International Conference on Data Engineering (ICDE), 2006.

S. Abiteboul, R. Hull, and V. Vianu.
Foundations of Databases.
Addison-Wesley, 1995.

L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko.
Complexity and Approximation of Fixing Numerical Attributes in Databases Under Integrity Constraints.

L. Bertossi and J. Chomicki.
Query Answering in Inconsistent Databases.
M. Baudinet, J. Chomicki, and P. Wolper.
Constraint-Generating Dependencies.

L. Bertossi.
Consistent Query Answering in Databases.
*SIGMOD Record*, 35(2), June 2006.

A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification.

Conditional Functional Dependencies for Data Cleaning.
In *IEEE International Conference on Data Engineering (ICDE)*, 2007.

Extending Dependencies with Conditions.
A. Borgida.
Language Features for Flexible Handling of Exceptions in Information Systems. 

J. Chomicki.
Preference Formulas in Relational Queries. 

J. Chomicki.
Consistent Query Answering: Five Easy Pieces. 
Keynote talk.

J. Chomicki.
Semantic optimization techniques for preference queries. 

A. Chandra, H.R. Lewis, and J.A. Makowsky.
Embedded Implicational Dependencies and their Inference Problem. 

A. Calì, D. Lembo, and R. Rosati.
On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases.

A. Calì, D. Lembo, and R. Rosati.
Query Rewriting and Answering under Constraints in Data Integration Systems.

J. Chomicki and J. Marcinkowski.
Minimal-Change Integrity Maintenance Using Tuple Deletions.

J. Chomicki and J. Marcinkowski.
On the Computational Complexity of Minimal-Change Integrity Maintenance in Relational Databases.

Computing Consistent Query Answers Using Conflict Hypergraphs.

A. Chandra and M. Vardi.
The Implication Problem for Functional and Inclusion Dependencies is Undecidable. 

P. De Bra and J. Paredaens. 
Conditional Dependencies for Horizontal Decompositions. 
In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 123–141, 1983.

W.F. Dowling and J. H. Gallier. 
Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. 

T. Eiter, M. Fink, G. Greco, and D. Lembo. 
Efficient Evaluation of Logic Programs for Querying Data Integration Systems. 

W. Fan. 
Dependencies Revisited for Improving Data Quality. 
Invited tutorial.

Data Exchange: Semantics and Query Answering. 

A. Fuxman and R. J. Miller.
ConQuer: Efficient Management of Inconsistent Databases.

A. Fuxman and R. J. Miller.
First-Order Query Rewriting for Inconsistent Databases.

G. Greco, S. Greco, and E. Zumpano.
A Logical Framework for Querying and Repairing Inconsistent Databases.

A. Lopatenko and L. Bertossi.
Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics.

J. Wijsen.
Database Repairing Using Updates.
J. Wijsen.

J. Wijsen.
On the Consistent Rewriting of Conjunctive Queries Under Primary Key Constraints.