Database Consistency: Logic-Based Approaches

Jan Chomicki
University at Buffalo

June 2008

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

Integrity constraints (dependencies)

Database instance D:

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

Integrity constraints Σ:

- 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 Σ:

- 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 Σ:

- first-order logic formulas
- the properties of the world

Satisfaction of constraints: $D \models \Sigma$

Formula satisfaction in a first-order structure.

Consistent database: $D \models \Sigma$

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

Name $\rightarrow$ City Salary

Integrity constraints (dependencies)

Database instance D:

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

Integrity constraints Σ:

- first-order logic formulas
- the properties of the world

Satisfaction of constraints: $D \models \Sigma$

Formula satisfaction in a first-order structure.

Consistent database: $D \models \Sigma$

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

Name $\rightarrow$ City Salary

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

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

Name $\rightarrow$ City Salary

The need for integrity constraints

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 $\Rightarrow$ 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

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

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 Σ and an IC σ, does $D \models \Sigma$ imply $D \models \sigma$ for every database D?

Implication

Given a set of ICs Σ and an IC σ, 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")?

Implication

Given a set of ICs Σ and an IC σ, 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

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

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

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

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

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

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

Full TGD

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

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

Full TGD

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

Non-full (embedded) TGD

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

Inclusion dependency (IND)

$NAM[Name, Address] \subseteq NAS[Name, Address]$

Examples

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

Full TGD

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

Non-full (embedded) TGD

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

Inclusion dependency (IND)

$NAM[Name, Address] \subseteq NAS[Name, Address]$

EGD

$\forall n, a, m, a', m'.\ NAM(n, a, m) \wedge NAM(n, a', m') \Rightarrow a = a'$

Functional dependency (FD)

$Name \rightarrow Address$

Implication: from linear-time to 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 $A_1 \wedge \cdots \wedge A_k \Rightarrow B$
3. if $\sigma = C_1 \wedge \cdots \wedge C_d \Rightarrow D$, then $\neg\sigma = C_1 \wedge \cdots \wedge C_d \wedge \neg D$ 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])

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 $A_1 \wedge \cdots \wedge A_k \Rightarrow B$
3. if $\sigma = C_1 \wedge \cdots \wedge C_d \Rightarrow D$, then $\neg\sigma = C_1 \wedge \cdots \wedge C_d \wedge \neg D$ 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 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

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

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

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

A	B
1	0
2	1
3	2
4	3

...

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

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

A	B
1	0
2	1
3	2
4	3
...	

Finite and unrestricted implication do not have to coincide.

Chase

Deciding the implication of full dependencies using chase

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

Deciding the implication of full dependencies using chase

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

Trivial dependencies

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

Chase

Deciding the implication of full dependencies using chase

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

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 τ_n, viewed as a database D_n, satisfies Σ
- if τ_n is nontrivial, then D_n violates σ
- the order of chase steps does not matter

Chase steps

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

Chase steps

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

Applying a chase step using a tgd C

1. view the LHS of τ_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 τ_j, obtaining τ_{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 τ_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 τ_j, obtaining τ_{j+1}

Applying a chase step using an egd C

1. view the LHS of τ_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 τ_j by $h(x_1)$, obtaining τ_{j+1}

Chase in action

Chase in action

Integrity constraints

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

Chase in action

Integrity constraints

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

Goal

Show that $\{C_1, C_2\}$ implies C_3.

Chase in action

Integrity constraints

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

Goal

Show that $\{C_1, C_2\}$ implies C_3.

Terminal chase sequence

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

$\tau_0 = \{P(x, y) \land P(x, z) \Rightarrow y = z\}$

Chase in action

Integrity constraints

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

Goal

Show that $\{C_1, C_2\}$ implies C_3.

Terminal chase sequence

$\tau_0 = \{P(x, y) \wedge P(x, z) \Rightarrow y = z\}$
$\tau_1 = \{P(x, y) \wedge P(x, z) \wedge R(x, y) \Rightarrow y = z\}$

Chase in action

Integrity constraints

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

Goal

Show that $\{C_1, C_2\}$ implies C_3.

Terminal chase sequence

$\tau_0 = \{P(x,y) \wedge P(x,z) \Rightarrow y = z\}$
$\tau_1 = \{P(x,y) \wedge P(x,z) \wedge R(x,y) \Rightarrow y = z\}$
$\tau_2 = \{P(x,y) \wedge P(x,z) \wedge R(x,y) \wedge R(x,z) \Rightarrow y = z\}$

Chase in action

Integrity constraints

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

Goal

Show that $\{C_1, C_2\}$ implies C_3.

Terminal chase sequence

$\tau_0 = \{P(x,y) \wedge P(x,z) \Rightarrow y = z\}$
$\tau_1 = \{P(x,y) \wedge P(x,z) \wedge R(x,y) \Rightarrow y = z\}$
$\tau_2 = \{P(x,y) \wedge P(x,z) \wedge R(x,y) \wedge R(x,z) \Rightarrow y = z\}$
$\tau_3 = \{P(x,y) \wedge R(x,y) \Rightarrow y = y\}$

Chase in action

Integrity constraints

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

Goal

Show that $\{C_1, C_2\}$ implies C_3.

Terminal chase sequence

$\tau_0 = \{P(x,y) \wedge P(x,z) \Rightarrow y = z\}$
$\tau_1 = \{P(x,y) \wedge P(x,z) \wedge R(x,y) \Rightarrow y = z\}$
$\tau_2 = \{P(x,y) \wedge P(x,z) \wedge R(x,y) \wedge R(x,z) \Rightarrow y = z\}$
$\tau_3 = \{P(x,y) \wedge R(x,y) \Rightarrow y = y\}$: a trivial dependency

A general perspective

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 σ by $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is equivalent to the unsatisfiability of the formula $\Phi_{\Sigma,\sigma} \equiv \sigma_1 \wedge \cdots \wedge \sigma_k \wedge \neg\sigma$
- for full dependencies, the formulas $\Phi_{\Sigma,\sigma}$ are of the form $\exists^*\forall^*\phi$ where ϕ 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 σ by $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ is equivalent to the unsatisfiability of the formula $\Phi_{\Sigma,\sigma} \equiv \sigma_1 \wedge \cdots \wedge \sigma_k \wedge \neg\sigma$
- for full dependencies, the formulas $\Phi_{\Sigma,\sigma}$ are of the form $\exists^*\forall^*\phi$ where ϕ 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

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 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)

Example axiomatization

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 $R[X] \subseteq T[Z]$.

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 $S(AB)$:

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)

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))

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

Review of results

	Implication	Axiomatization
FDs	PTIME	Yes
INDs	PSPACE-complete	Yes
FDs + INDs	Undecidable	No
Full (typed) dependencies	EXPTIME-complete	Yes
Join dependencies	NP-complete	No
First-order logic	Undecidable	Yes

Application: database design

Keys

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

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

Application: database design

Keys

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

- Σ implies $X \rightarrow U$
- for no proper subset Y of X, Σ 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 Σ iff Σ implies the join dependency $\bowtie [\mathcal{R}]$.

Keys

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

- Σ implies $X \to U$
- for no proper subset Y of X, Σ implies $Y \to U$

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 Σ iff Σ implies the join dependency $\bowtie [\mathcal{R}]$.

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

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

Application: database design

Keys

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

- Σ implies $X \rightarrow U$
- for no proper subset Y of X, Σ 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 Σ iff Σ 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:

Application: database design

Keys

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

- Σ implies $X \rightarrow U$
- for no proper subset Y of X, Σ 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 Σ iff Σ 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:

$$R(x,y,z') \wedge R(x,y',z) \Rightarrow R(x,y,z) \quad \text{given JD}$$

Application: database design

Keys

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

- Σ implies $X \rightarrow U$
- for no proper subset Y of X, Σ 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 Σ iff Σ 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:

$$R(x,y,z') \wedge R(x,y',z) \Rightarrow R(x,y,z) \quad \text{given JD}$$

$$R(x,y,z') \wedge R(x,y,z) \Rightarrow R(x,y,z) \quad \text{chase with } A \rightarrow B$$

Application: data exchange

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

ϕ_S, ϕ_T, ψ_T are conjunctions of relational atoms over source and target.

Source-to-target dependencies Σ_{st}

- tuple-generating dependencies: $\forall \mathbf{x}\ (\phi_{\mathbf{S}}(\mathbf{x}) \Rightarrow \exists \mathbf{y}\ \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$.

Target integrity constraints Σ_t

- tuple-generating dependencies (tgds): $\forall \mathbf{x}\ (\phi_{\mathbf{T}}(\mathbf{x}) \Rightarrow \exists \mathbf{y}\ \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$
- equality-generating dependencies: $\forall \mathbf{x}\ (\phi_{\mathbf{T}}(\mathbf{x}) \Rightarrow \mathbf{x_1} = \mathbf{x_2})$.

Constraints and solutions

ϕ_S, ϕ_T, ψ_T are conjunctions of relational atoms over source and target.

Source-to-target dependencies Σ_{st}

- tuple-generating dependencies: $\forall \mathbf{x}\ (\phi_{\mathbf{S}}(\mathbf{x}) \Rightarrow \exists \mathbf{y}\ \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$.

Target integrity constraints Σ_t

- tuple-generating dependencies (tgds): $\forall \mathbf{x}\ (\phi_{\mathbf{T}}(\mathbf{x}) \Rightarrow \exists \mathbf{y}\ \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$
- equality-generating dependencies: $\forall \mathbf{x}\ (\phi_{\mathbf{T}}(\mathbf{x}) \Rightarrow \mathbf{x_1} = \mathbf{x_2})$.

Solution

Given a source instance I, a target instance J is

- a solution for I if J satisfies Σ_t and (I, J) satisfy Σ_{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

ϕ_S, ϕ_T, ψ_T are conjunctions of relational atoms over source and target.

Source-to-target dependencies Σ_{st}

- tuple-generating dependencies: $\forall \mathbf{x}\ (\phi_\mathbf{S}(\mathbf{x}) \Rightarrow \exists \mathbf{y}\ \psi_\mathbf{T}(\mathbf{x}, \mathbf{y}))$.

Target integrity constraints Σ_t

- tuple-generating dependencies (tgds): $\forall \mathbf{x}\ (\phi_\mathbf{T}(\mathbf{x}) \Rightarrow \exists \mathbf{y}\ \psi_\mathbf{T}(\mathbf{x}, \mathbf{y}))$
- equality-generating dependencies: $\forall \mathbf{x}\ (\phi_\mathbf{T}(\mathbf{x}) \Rightarrow \mathbf{x_1} = \mathbf{x_2})$.

Solution

Given a source instance I, a target instance J is

- a solution for I if J satisfies Σ_t and (I, J) satisfy Σ_{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.

Query evaluation (Fagin et al.[FKMP05])

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.

Query evaluation (Fagin et al.[FKMP05])

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: $\sigma, \pi, \times$

Query evaluation (Fagin et al.[FKMP05])

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: $\sigma, \pi, \times$

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

Building a universal solution [FKMP05]

Apply a variant of the chase [AHV95] to the source instance using target and source-to-target dependencies, obtaining a sequence of instances $I_0 = I, I_1, \ldots, I_n, \ldots$.

Building a universal solution [FKMP05]

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}.

Building a universal solution [FKMP05]

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}.

Chasing an egd C

Find a substitution h such that makes the LHS of C true in I_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 I_j (preferring constants), obtaining I_{j+1}.

Chase at work

Chase at work

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

Source-to-target dependencies

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

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

Target constraints

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

Chase at work

Source and target databases

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

Source-to-target dependencies

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

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

Target constraints

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

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

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

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

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

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

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

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

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

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

Target constraints

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

Chase sequence

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

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

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

Source and target databases

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

Source-to-target dependencies

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

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

Target constraints

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

Chase sequence

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

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

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

$$I_3 = \{Emp(Li, LA), Num(Li, 111), Name(111, Li), Addr(111, LA)\}$$

Chase termination

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 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 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

Application: semantic query optimization

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
- ...

Preference queries

The winnow operator ω_C (Chomicki [Cho03])

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

The winnow operator ω_C (Chomicki [Cho03])

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

Relation *Book(Title,Vendor,Price)*

Preference: $(i_1, v_1, p_1) \succ_{C_1} (i_2, v_2, p_2) \equiv i_1 = i_2 \wedge p_1 < p_2$

Indifference: $(i_1, v_1, p_1) \sim_{C_1} (i_2, v_2, p_2) \equiv i_1 \neq i_2 \vee p_1 = p_2$

The winnow operator ω_C (Chomicki [Cho03])

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

Relation *Book(Title,Vendor,Price)*

Preference: $(i_1, v_1, p_1) \succ_{C_1} (i_2, v_2, p_2) \equiv i_1 = i_2 \wedge p_1 < p_2$

Indifference: $(i_1, v_1, p_1) \sim_{C_1} (i_2, v_2, p_2) \equiv i_1 \neq i_2 \vee p_1 = p_2$

Book	*Title*	*Vendor*	*Price*
t_1	The Flanders Panel	`amazon.com`	$14.75
t_2	The Flanders Panel	`fatbrain.com`	$13.50
t_3	The Flanders Panel	`bn.com`	$18.80
t_4	Green Guide: Greece	`bn.com`	$17.30

The winnow operator ω_C (Chomicki [Cho03])

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

Relation *Book(Title,Vendor,Price)*

Preference: $(i_1, v_1, p_1) \succ_{C_1} (i_2, v_2, p_2) \equiv i_1 = i_2 \wedge p_1 < p_2$

Indifference: $(i_1, v_1, p_1) \sim_{C_1} (i_2, v_2, p_2) \equiv i_1 \neq i_2 \vee p_1 = p_2$

Book	*Title*	*Vendor*	*Price*
t_1	The Flanders Panel	`amazon.com`	\$14.75
t_2	The Flanders Panel	`fatbrain.com`	\$13.50
t_3	The Flanders Panel	`bn.com`	\$18.80
t_4	Green Guide: Greece	`bn.com`	\$17.30

Eliminating redundant occurrences of winnow

Eliminating redundant occurrences of winnow

Redundant winnow (Chomicki [Cho07b])

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

Redundant winnow (Chomicki [Cho07b])

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

Example

$$Book(i_1, v_1, p_1) \wedge Book(i_2, v_2, p_2) \Rightarrow i_1 \neq i_2 \vee p_1 = p_2$$

is a functional dependency in disguise:

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

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

Eliminating redundant occurrences of winnow

Redundant winnow (Chomicki [Cho07b])

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

Example

$$Book(i_1, v_1, p_1) \wedge Book(i_2, v_2, p_2) \Rightarrow i_1 \neq i_2 \vee p_1 = p_2$$

is a functional dependency in disguise:

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

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

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

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

Prospects for integrity constraints

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

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

Data mining

- discovery of FDs and INDs

Part II

Consistent query answers

Outline of Part II

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

Ignoring Inconsistency

Query results not reliable.

Query results not reliable.

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

Name $\longrightarrow$ City Salary

Query results not reliable.

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

Name $\longrightarrow$ City Salary

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

$\longrightarrow$

Query results not reliable.

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

Name $\longrightarrow$ City Salary

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

$\longrightarrow$

Name
Gates
Grove

Horizontal Decomposition

Decomposition into two relations:

- violators
- the rest

(De Bra, Paredaens [DBP83])

Decomposition into two relations:

- violators
- the rest

(De Bra, Paredaens [DBP83])


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

Name $\rightarrow$ City Salary

Horizontal Decomposition

Decomposition into two relations:

- violators
- the rest

(De Bra, Paredaens [DBP83])

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

Name → City Salary

Grove	Santa Clara	10M

Name → City Salary

Gates	Redmond	20M
Gates	Redmond	30M

Name → City Salary

Exceptions to Constraints

Weakening the contraints:

- functional dependencies $\longrightarrow$ denial constraints

(Borgida [Bor85])

Weakening the contraints:

- functional dependencies → denial constraints

(Borgida [Bor85])

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

Name → City Salary

Weakening the contraints:

- functional dependencies $\longrightarrow$ denial constraints

(Borgida [Bor85])

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

Name $\rightarrow$ City Salary

$\longrightarrow$

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

Name $\rightarrow$ City Salary

except Name='Gates'

The Impact of Inconsistency on Queries

Traditional view

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

The Impact of Inconsistency on Queries

Traditional view

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

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?

Restoring consistency:

- insertion, deletion, update
- minimal change?

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

Name $\rightarrow$ City Salary

Database Repairs

Restoring consistency:

- insertion, deletion, update
- minimal change?

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

Name → City Salary

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

Name → City Salary

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

Name → City Salary

Consistent query answer:

Query answer obtained in every repair.

(Arenas, Bertossi, Chomicki [ABC99])

Consistent Query Answering

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

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

Name $\rightarrow$ City Salary

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

$\longrightarrow$

Name
Grove

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

Name → City Salary

```
SELECT Name
FROM Employee
WHERE Salary ≥ 10M
```

→

Name
Gates
Grove

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

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.

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

???

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.

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

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

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

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

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

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

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

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

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

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.

Example relation $R(A, B)$

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

A	B
a_1	b_1
a_1	c_1
a_2	b_2
a_2	c_2
...	
a_n	b_n
a_n	c_n

$A \longrightarrow B$

Example relation $R(A, B)$

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

A	B
a_1	b_1
a_1	c_1
a_2	b_2
a_2	c_2
...	
a_n	b_n
a_n	c_n

$A \longrightarrow B$

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

> *the set of answers to* Q^{IC} *in* D = *the set of consistent answers to* Q *in* D *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

the set of answers to Q^{IC} *in* D = *the set of consistent answers to* Q *in* D *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.

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

> *the set of answers to Q^{IC} in D = the set of consistent answers to Q in D w.r.t. IC.*

Representing all repairs

Given IC and D:

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

Logic programs

Given IC, D and Q:

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

Constraint classes

Universal constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$

Constraint classes

Universal constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$

Example

$\forall.\ \neg Par(x) \vee Ma(x) \vee Fa(x)$

Constraint classes

Universal constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$

Denial constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n$

Example

$\forall.\ \neg Par(x) \vee Ma(x) \vee Fa(x)$

Constraint classes

Universal constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$

Example

$\forall.\ \neg Par(x) \vee Ma(x) \vee Fa(x)$

Denial constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n$

Example

$\forall.\ \neg M(n, s, m) \vee \neg M(m, t, w) \vee s \leq t$

Constraint classes

Universal constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$

Denial constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n$

Functional dependencies

$X \rightarrow Y$:

- a key dependency in F if $Y = U$
- a primary-key dependency: only one key exists

Example

$\forall.\ \neg Par(x) \vee Ma(x) \vee Fa(x)$

Example

$\forall.\ \neg M(n, s, m) \vee \neg M(m, t, w) \vee s \leq t$

Constraint classes

Universal constraints

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

Denial constraints

$\forall.\ \neg A_1 \lor \cdots \lor \neg A_n$

Functional dependencies

$X \rightarrow Y$:

- a key dependency in F if $Y = U$
- a primary-key dependency: only one key exists

Example

$\forall.\ \neg Par(x) \lor Ma(x) \lor Fa(x)$

Example

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

Example primary-key dependency

Name $\rightarrow$ Address Salary

Constraint classes

Universal constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$

Denial constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n$

Functional dependencies

$X \rightarrow Y$:

- a key dependency in F if $Y = U$
- a primary-key dependency: only one key exists

Inclusion dependencies

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

- a foreign key constraint if Y is a key of S

Example

$\forall.\ \neg Par(x) \vee Ma(x) \vee Fa(x)$

Example

$\forall.\ \neg M(n, s, m) \vee \neg M(m, t, w) \vee s \leq t$

Example primary-key dependency

Name $\rightarrow$ Address Salary

Constraint classes

Universal constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$

Example

$\forall.\ \neg Par(x) \vee Ma(x) \vee Fa(x)$

Denial constraints

$\forall.\ \neg A_1 \vee \cdots \vee \neg A_n$

Example

$\forall.\ \neg M(n,s,m) \vee \neg M(m,t,w) \vee s \leq t$

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

$M[Manager] \subseteq M[Name]$

Query Rewriting

Building queries that compute CQAs

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

Building queries that compute CQAs

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

Query

$Emp(x, y, z)$

Query Rewriting

Building queries that compute CQAs

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

Query

$Emp(x, y, z)$

Integrity constraint

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

Building queries that compute CQAs

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

Query

$Emp(x, y, z)$

Integrity constraint

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

Building queries that compute CQAs

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

Query

$Emp(x, y, z)$

Integrity constraint

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

Rewritten query

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

The Scope of Query Rewriting

(Arenas, Bertossi, Chomicki [ABC99])

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

The Scope of Query Rewriting

(Arenas, Bertossi, Chomicki [ABC99])

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

(Fuxman, Miller [FM07])

- Integrity constraints: primary key functional dependencies
- Queries: C_{forest}
  - a class of conjunctive queries ($\pi, \sigma, \times$)
  - no non-key or non-full joins
  - no repeated relation symbols
  - no built-ins
- Generalization: conjunctive queries expressed as rooted rules (Wijsen [Wij07])

SQL Rewriting

SQL query

```
SELECT Name FROM Emp
WHERE 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)
```

SQL Rewriting

SQL query

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

SQL rewritten query

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

(Fuxman, Fazli, Miller [FM05])

- 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)

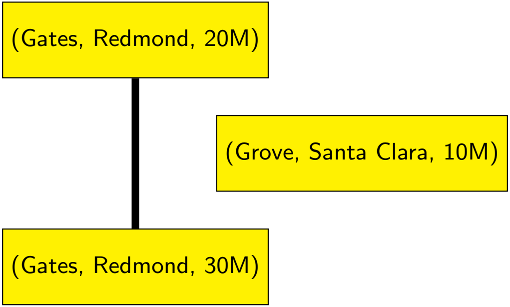
Conflict Hypergraph

Vertices

Tuples in the database.

Edges

Minimal sets of tuples violating a constraint.



Vertices

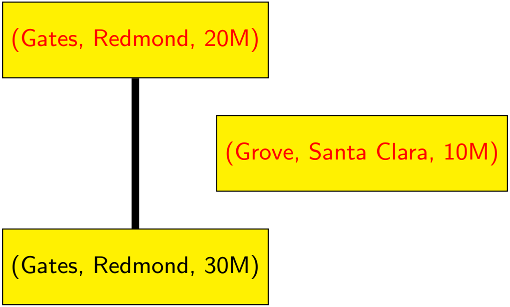
Tuples in the database.

Edges

Minimal sets of tuples violating a constraint.

Repairs

Maximal independent sets in the conflict graph.



Conflict Hypergraph

Vertices

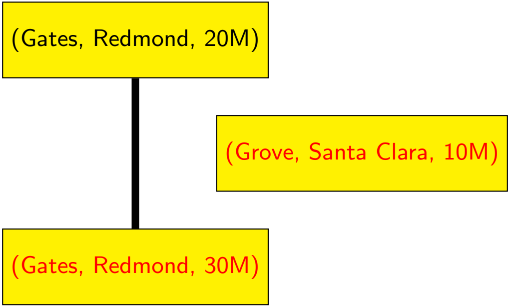
Tuples in the database.

Edges

Minimal sets of tuples violating a constraint.

Repairs

Maximal independent sets in the conflict graph.



Algorithm HProver

INPUT: query Φ a disjunction of ground literals, conflict hypergraph G
OUTPUT: is Φ false in some repair of D w.r.t. IC?
ALGORITHM:

1. $\neg\Phi = P_1(t_1) \wedge \cdots \wedge P_m(t_m) \wedge \neg P_{m+1}(t_{m+1}) \wedge \cdots \wedge \neg P_n(t_n)$
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 $S \supseteq \{B_1, \ldots, B_m\}$.

Algorithm HProver

INPUT: query Φ a disjunction of ground literals, conflict hypergraph G
OUTPUT: is Φ false in some repair of D w.r.t. IC?
ALGORITHM:

1. $\neg\Phi = P_1(t_1) \wedge \cdots \wedge P_m(t_m) \wedge \neg P_{m+1}(t_{m+1}) \wedge \cdots \wedge \neg P_n(t_n)$
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 $S \supseteq \{B_1, \ldots, B_m\}$.

(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])

Specifying repairs as answer sets of logic programs

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

Example

$emp(x, y, z) \leftarrow emp_D(x, y, z), not\ dubious\_emp(x, y, z).$
$dubious\_emp(x, y, z) \leftarrow emp_D(x, y, z), emp(x, y', z'), y \neq y'.$
$dubious\_emp(x, y, z) \leftarrow emp_D(x, y, z), emp(x, y', z'), z \neq z'.$

Logic programs

Specifying repairs as answer sets of logic programs

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

Example

$emp(x,y,z) \leftarrow emp_D(x,y,z), not\ dubious\_emp(x,y,z).$
$dubious\_emp(x,y,z) \leftarrow emp_D(x,y,z), emp(x,y',z'), y \neq y'.$
$dubious\_emp(x,y,z) \leftarrow emp_D(x,y,z), emp(x,y',z'), z \neq z'.$

Answer sets

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

Logic Programs for computing CQAs

Logic Programs

- disjunction and classical negation
- checking whether an atom is in all answer sets is Π_2^p-complete
- `dlv`, `smodels`, ...

Logic Programs for computing CQAs

Logic Programs

- disjunction and classical negation
- checking whether an atom is in all answer sets is Π_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 Π_2^p-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

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.

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 $\beta_1 = \phi_1 \wedge \cdots \wedge \phi_m$, negative clauses $\beta_2 = \psi_{m+1} \wedge \cdots \wedge \psi_l$.
2. Database D contains two binary relations $R(A, B)$ and $S(A, B)$:
   - $R(i, p)$ if variable p occurs in ϕ_i, $i = 1, \ldots, m$.
   - $S(i, p)$ if variable p occurs in ψ_i, $i = m + 1, \ldots, l$.
3. A is the primary key of both R and S.
4. Query $Q \equiv \exists x, y, z. \left(R(x, y) \wedge S(z, y)\right)$.
5. There is an assignment which satisfies $\beta_1 \wedge \beta_2$ 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 $\beta_1 = \phi_1 \wedge \cdots \wedge \phi_m$, negative clauses $\beta_2 = \psi_{m+1} \wedge \cdots \wedge \psi_l$.
2. Database D contains two binary relations $R(A, B)$ and $S(A, B)$:
   - $R(i, p)$ if variable p occurs in ϕ_i, $i = 1, \ldots, m$.
   - $S(i, p)$ if variable p occurs in ψ_i, $i = m + 1, \ldots, l$.
3. A is the primary key of both R and S.
4. Query $Q \equiv \exists x, y, z. \big(R(x, y) \wedge S(z, y)\big)$.
5. There is an assignment which satisfies $\beta_1 \wedge \beta_2$ iff there exists a repair in which Q is false.

□

Q does not belong to C_{forest}.

Data complexity of CQA

	Primary keys	*Arbitrary keys*	*Denial*	*Universal*
$\sigma, \times, -$				
$\sigma, \times, -, \cup$				
σ, π				
$\sigma, \pi, \times$				
$\sigma, \pi, \times, -, \cup$				

Data complexity of CQA

	Primary keys	*Arbitrary keys*	*Denial*	*Universal*
$\sigma, \times, -$	PTIME	PTIME		PTIME: binary
$\sigma, \times, -, \cup$				
σ, π				
$\sigma, \pi, \times$				
$\sigma, \pi, \times, -, \cup$				

- (Arenas, Bertossi, Chomicki [ABC99])

Data complexity of CQA

	Primary keys	*Arbitrary keys*	*Denial*	*Universal*
$\sigma, \times, -$	PTIME	PTIME	PTIME	PTIME: binary
$\sigma, \times, -, \cup$	PTIME	PTIME	PTIME	
σ, π	PTIME	co-NPC	co-NPC	
$\sigma, \pi, \times$	co-NPC	co-NPC	co-NPC	
$\sigma, \pi, \times, -, \cup$	co-NPC	co-NPC	co-NPC	

- (Arenas, Bertossi, Chomicki [ABC99])
- (Chomicki, Marcinkowski [CM05a])

Data complexity of CQA

	Primary keys	*Arbitrary keys*	*Denial*	*Universal*
$\sigma, \times, -$	PTIME	PTIME	PTIME	PTIME: binary
$\sigma, \times, -, \cup$	PTIME	PTIME	PTIME	
σ, π	PTIME	co-NPC	co-NPC	
$\sigma, \pi, \times$	co-NPC PTIME: C_{forest}	co-NPC	co-NPC	
$\sigma, \pi, \times, -, \cup$	co-NPC	co-NPC	co-NPC	

- (Arenas, Bertossi, Chomicki [ABC99])
- (Chomicki, Marcinkowski [CM05a])
- (Fuxman, Miller [FM07])

Data complexity of CQA

	Primary keys	*Arbitrary keys*	*Denial*	*Universal*
$\sigma, \times, -$	PTIME	PTIME	PTIME	PTIME: binary Π_2^p-complete
$\sigma, \times, -, \cup$	PTIME	PTIME	PTIME	Π_2^p-complete
σ, π	PTIME	co-NPC	co-NPC	Π_2^p-complete
$\sigma, \pi, \times$	co-NPC PTIME: C_{forest}	co-NPC	co-NPC	Π_2^p-complete
$\sigma, \pi, \times, -, \cup$	co-NPC	co-NPC	co-NPC	Π_2^p-complete

- (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

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])

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])

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.

EmpDept

Name	*Dept*	*Location*
John	Sales	Buffalo
Mary	Sales	Toronto

Name $\longrightarrow$ Dept

Dept $\longrightarrow$ 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

Name	*Dept*	*Location*
John	Sales	Buffalo

Name → Dept

Dept → City

Name	*Dept*	*Location*
Mary	Sales	Toronto

Name → Dept

Dept → City

Attribute-based Repairs through Tuple-based Repairs (Wijsen [Wij06])

Repair the lossless join decomposition:

$$\pi_{Name,Dept}(EmpDept) \bowtie \pi_{Dept,Location}(EmpDept)$$

Repair the lossless join decomposition:

$$\pi_{Name,Dept}(EmpDept) \bowtie \pi_{Dept,Location}(EmpDept)$$

Name	*Dept*	*Location*
John	Sales	Buffalo
John	Sales	Toronto
Mary	Sales	Buffalo
Mary	Sales	Toronto

Name $\longrightarrow$ Dept

Dept $\longrightarrow$ City

Repair the lossless join decomposition:

$$\pi_{Name,Dept}(EmpDept) \bowtie \pi_{Dept,Location}(EmpDept)$$

Name	*Dept*	*Location*
John	Sales	Buffalo
John	Sales	Toronto
Mary	Sales	Buffalo
Mary	Sales	Toronto

Name → Dept

Dept → City

Name	*Dept*	*Location*
John	Sales	Buffalo
Mary	Sales	Buffalo

Name → Dept

Dept → City

Name	*Dept*	*Location*
John	Sales	Toronto
Mary	Sales	Toronto

Name → Dept

Dept → City

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

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

Salaries with probabilities

EmpProb

Name	*Salary*	Prob
Gates	20M	0.7
Gates	30M	0.3
Grove	10M	0.5
Grove	20M	0.5

SQL query

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

SQL query

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

SQL rewritten query

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

Computing Clean Answers

SQL query

```
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

Name	*Salary*	Prob
Gates	20M	0.7
Gates	30M	0.3
Grove	10M	0.5
Grove	20M	0.5

Name $\rightarrow$ 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
```

EmpProb

Name	*Salary*	Prob
Gates	20M	0.7
Gates	30M	0.3
Grove	10M	0.5
Grove	20M	0.5

Name → Salary

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

→

Computing Clean Answers

SQL query

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

SQL rewritten query

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

EmpProb

Name	*Salary*	Prob
Gates	20M	0.7
Gates	30M	0.3
Grove	10M	0.5
Grove	20M	0.5

Name → Salary

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

→

Name	Prob
Gates	1
Grove	0.5

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

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?

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

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

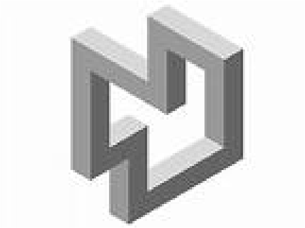
- no deployed applications
- repairing vs. CQA: data and query characteristics
- heuristics for CQA and repairing

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.
In *ACM Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.

M. Arenas, L. Bertossi, and J. Chomicki.
Answer Sets for Consistent Query Answering in Inconsistent Databases.
Theory and Practice of Logic Programming, 3(4–5):393–424, 2003.

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.
In *International Workshop on Database Programming Languages*, pages 262–278. Springer, LNCS 3774, 2005.

L. Bertossi and J. Chomicki.
Query Answering in Inconsistent Databases.

In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*, pages 43–83. Springer-Verlag, 2003.

M. Baudinet, J. Chomicki, and P. Wolper.
Constraint-Generating Dependencies.
In *International Conference on Database Theory (ICDT)*, pages 322–337, Prague, Czech Republic, January 1995. Springer-Verlag, LNCS 893.
Short version in: Proc. 2nd Workshop on Principles and Practice of Constraint Programming, 1994.

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

P. Bohannon, M. Flaster, W. Fan, and R. Rastogi.
A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification.
In *ACM SIGMOD International Conference on Management of Data*, pages 143–154, 2005.

P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis.
Conditional Functional Dependencies for Data Cleaning.
In *IEEE International Conference on Data Engineering (ICDE)*, 2007.

L. Bravo, W. Fan, and S. Ma.
Extending Dependencies with Conditions.

In *International Conference on Very Large Data Bases (VLDB)*, pages 243–254, 2007.

A. Borgida.
Language Features for Flexible Handling of Exceptions in Information Systems.
ACM Transactions on Database Systems, 10(4):565–603, 1985.

J. Chomicki.
Preference Formulas in Relational Queries.
ACM Transactions on Database Systems, 28(4):427–466, December 2003.

J. Chomicki.
Consistent Query Answering: Five Easy Pieces.
In *International Conference on Database Theory (ICDT)*, pages 1–17. Springer, LNCS 4353, 2007.
Keynote talk.

J. Chomicki.
Semantic optimization techniques for preference queries.
Information Systems, 32(5):660–674, 2007.

A. Chandra, H.R. Lewis, and J.A. Makowsky.
Embedded Implicational Dependencies and their Inference Problem.
In *ACM Symposium on Theory of Computing (STOC)*, pages 342–354, 1981.

A. Calì, D. Lembo, and R. Rosati.

On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases.
In *ACM Symposium on Principles of Database Systems (PODS)*, pages 260–271, 2003.

A. Calì, D. Lembo, and R. Rosati.
Query Rewriting and Answering under Constraints in Data Integration Systems.
In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 16–21, 2003.

J. Chomicki and J. Marcinkowski.
Minimal-Change Integrity Maintenance Using Tuple Deletions.
Information and Computation, 197(1-2):90–121, 2005.

J. Chomicki and J. Marcinkowski.
On the Computational Complexity of Minimal-Change Integrity Maintenance in Relational Databases.
In L. Bertossi, A. Hunter, and T. Schaub, editors, *Inconsistency Tolerance*, pages 119–150. Springer-Verlag, 2005.

J. Chomicki, J. Marcinkowski, and S. Staworko.
Computing Consistent Query Answers Using Conflict Hypergraphs.
In *International Conference on Information and Knowledge Management (CIKM)*, pages 417–426. ACM Press, 2004.

A. Chandra and M. Vardi.

The Implication Problem for Functional and Inclusion Dependencies is Undecidable.
SIAM Journal on Computing, 14(3):671–677, 1985.

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.
Journal of Logic Programming, 1(3):267–284, 1984.

T. Eiter, M. Fink, G. Greco, and D. Lembo.
Efficient Evaluation of Logic Programs for Querying Data Integration Systems.
In *International Conference on Logic Programming (ICLP)*, pages 163–177, 2003.

W. Fan.
Dependencies Revisited for Improving Data Quality.
In *ACM Symposium on Principles of Database Systems (PODS)*, pages 159–170, 2008.
Invited tutorial.

R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa.
Data Exchange: Semantics and Query Answering.
Theoretical Computer Science, 336(1):89–124, 2005.

R. Fagin, P. G. Kolaitis, L. Popa, and W-C. Tan.
Composing Schema Mappings: Second-Order Dependencies to the Rescue.
ACM Transactions on Database Systems, 30(4):994–1055, 2005.

A. Fuxman and R. J. Miller.
ConQuer: Efficient Management of Inconsistent Databases.
In *ACM SIGMOD International Conference on Management of Data*, pages 155–166, 2005.

A. Fuxman and R. J. Miller.
First-Order Query Rewriting for Inconsistent Databases.
Journal of Computer and System Sciences, 73(4):610–635, 2007.

G. Greco, S. Greco, and E. Zumpano.
A Logical Framework for Querying and Repairing Inconsistent Databases.
IEEE Transactions on Knowledge and Data Engineering, 15(6):1389–1408, 2003.

A. Lopatenko and L. Bertossi.
Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics.
In *International Conference on Database Theory (ICDT)*, pages 179–193. Springer, LNCS 4353, 2007.

J. Wijsen.
Database Repairing Using Updates.
ACM Transactions on Database Systems, 30(3):722–768, 2005.

J. Wijsen.
Project-Join Repair: An Approach to Consistent Query Answering Under Functional Dependencies.
In *International Conference on Flexible Query Answering Systems (FQAS)*, 2006.

J. Wijsen.
On the Consistent Rewriting of Conjunctive Queries Under Primary Key Constraints.
In *International Workshop on Database Programming Languages*, pages 112–126. Springer, LNCS 4797, 2007.