Data Integration: Query Evaluation

Jan Chomicki

University at Buffalo and Warsaw University

March 15, 2007
Data exchange

$\phi_S$, $\phi_T$, and $\psi_T$ are conjunctions of atomic formulas.

**Target integrity constraints $\Sigma_t$**

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

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

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

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...

Jan Chomicki (UB/UW)  Data Integration: Query Evaluation  March 15, 2007  2 / 10
$\phi_S$, $\phi_T$, and $\psi_T$ are conjunctions of atomic formulas.

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

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

**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
Data exchange

$\phi_S$, $\phi_T$, and $\psi_T$ are conjunctions of atomic formulas.

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

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

**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: $\exists, \land$
- 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
Building a universal solution [FKMP05]

Apply exhaustively a variant of the chase [AHV95] to the source instance using target and source-to-target dependencies.

Chasing a tgd

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

Chasing an egd

Find a substitution $h$ such that makes the LHS true 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 $h(x_2)$ (preferring constants).
Apply exhaustively a variant of the chase [AHV95] to the source instance using target and source-to-target dependencies.

### Chasing a tgd

1. find a substitution $h$ that (1) $h$ makes the LHS true in the constructed instance, and (2) $h$ cannot be extended to a substitution that makes the RHS true in that instance
2. apply $h$ to the RHS, mapping the existentially quantified variables to fresh labelled nulls
3. add the resulting facts to the instance.
Apply exhaustively a variant of the chase [AHV95] to the source instance using target and source-to-target dependencies.

**Chasing a tgd**

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

**Chasing an egd**

Find a substitution $h$ such that makes the LHS true 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 $h(x_2)$ (preferring constants).
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 solution
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 solution

Weakly acyclic tgds

- prevent the recurrent generation of labelled nulls
- program dependency graph (PDG) of tgds:
  - nodes: attributes
  - edges: value propagation from LHS to RHS
  - special edges: for existential variables
- weakly acyclic tgd: no cycle in the PDG contains a special edge
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 solution**

Weakly acyclic tgds

- prevent the recurrent generation of labelled nulls
- program dependency graph (PDG) of tgds:
  - nodes: attributes
  - edges: value propagation from LHS to RHS
  - special edges: for existential variables
- weakly acyclic tgd: no cycle in the PDG contains a special edge

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 tgd
- co-NP-complete for unions of conjunctive queries (with inequalities) and constraints that are egds and weakly acyclic tgd
- already co-NP-hard for conjunctive queries and LAV settings (with no target constraints) [AD98]
Local-as-view (LAV)

Setting

- **Source-to-target dependencies:**

  \[ \forall t. R(t) \Rightarrow \phi_T(t) \]

- no target constraints (but FDs can be added)
- queries: sets of Datalog rules (no inequalities).
Local-as-view (LAV)

Setting

- **Source-to-target dependencies:**
  \[
  \forall t. R(t) \Rightarrow \phi_T(t)
  \]
- no target constraints (but FDs can be added)
- queries: sets of Datalog rules (no inequalities).

Query rewriting

- the rewriting produces a set of nonrecursive Datalog rules with function symbols:
  - EDB predicates: source relations
  - IDB predicates: target relations
- function symbols can be eliminated.
Inverse rules for every source-to-target dependency:

\[ \forall x_1, \ldots, x_m. (A \Rightarrow \exists y_1, \ldots, y_k. B_1 \land \cdots \land B_n) \]

produce \( n \) inverse rules \( B'_1 \):

\[ -A, \ldots, -B'_n \]

\( B'_i \) is like \( B_i \), except that each of \( y_1, \ldots, y_k \) is replaced by the (Skolem) term \( f(x_1, \ldots, x_m) \) where \( f \) is a different, unique function symbol.

all the occurrences of the same variable are replaced by the same term

Query evaluation through rewriting:

1. the query rule and the inverse rules are evaluated bottom-up
2. the evaluation terminates
3. only the substitutions that do not contain Skolem terms are returned to the user

Theorem:

Given a source instance \( I \), query evaluation returns the certain answers w.r.t. \( I \).
Query evaluation in LAV

Inverse rules

- for every source-to-target dependency:

\[ \forall x_1, \ldots, x_m.(A \Rightarrow \exists y_1, \ldots y_k. B_1 \land \cdots \land B_n) \]

produce \( n \) inverse rules \( B'_1 : \neg A, \ldots, B'_n : \neg A \)

- \( B'_i \) is like \( B_i \), except that each of \( y_1, \ldots y_k \) is replaced by the (Skolem) term \( f(x_1, \ldots, x_m) \) where \( f \) is a different, unique function symbol.

- all the occurrences of the same variable are replaced by the same term
Inverse rules

- for every source-to-target dependency:

\[ \forall x_1, \ldots, x_m. (A \Rightarrow \exists y_1, \ldots y_k. B_1 \land \cdots \land B_n) \]

produce \( n \) inverse rules \( B'_1 : -A, \ldots, B'_n : -A \)

- \( B'_i \) is like \( B_i \), except that each of \( y_1, \ldots y_k \) is replaced by the (Skolem) term \( f(x_1, \ldots, x_m) \) where \( f \) is a different, unique function symbol.

- all the occurrences of the same variable are replaced by the same term

Query evaluation through rewriting

- the query rule and the inverse rules are evaluated bottom-up
- the evaluation terminates
- only the substitutions that do not contain Skolem terms are returned to the user
Query evaluation in LAV

Inverse rules

- for every source-to-target dependency:

\[ \forall x_1, \ldots, x_m. (A \Rightarrow \exists y_1, \ldots, y_k. B_1 \land \cdots \land B_n) \]

produce \( n \) inverse rules \( B'_1 : \neg A, \ldots, B'_n : \neg A \)

- \( B'_i \) is like \( B_i \), except that each of \( y_1, \ldots, y_k \) is replaced by the (Skolem) term \( f(x_1, \ldots, x_m) \) where \( f \) is a different, unique function symbol.

- all the occurrences of the same variable are replaced by the same term

Query evaluation through rewriting

- the query rule and the inverse rules are evaluated bottom-up
- the evaluation terminates
- only the substitutions that do not contain Skolem terms are returned to the user

Theorem

Given a source instance \( I \), query evaluation returns the certain answers w.r.t. \( I \).
Global-as-view (GAV)

**Setting**

- **Source-to-target dependencies:**
  \[ \forall t. \phi_S(t) \Rightarrow R(t). \]
- no target constraints
- queries: unions of conjunctive queries (defined using Datalog)
Global-as-view (GAV)

Setting

- Source-to-target dependencies:
  \[ \forall t. \phi_S(t) \implies R(t). \]
- no target constraints
- queries: unions of conjunctive queries (defined using Datalog)

Query evaluation by unfolding

1. replace each atom in the query that unifies with the head of a rule with the body of the rule (to which the mgu has been applied)
2. stop when only EDB goals are left
3. take the union \( Q_u \) of all obtained queries
4. the evaluation of \( Q_u \) returns the certain answers
O. Abiteboul and O. Duschka.
Complexity of Answering Queries Using Materialized Views.

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

Data Exchange: Semantics and Query Answering.