CMPS 277 – Principles of Database Systems

https://courses.soe.ucsc.edu/courses/cmps277/Fall11/01

Lecture #16
Equality-Generating Dependencies

Definition: An equality-generating dependency (egd) is a formula of relational calculus of the form:

\[ \forall x_1,\ldots,x_n(\varphi(x_1,\ldots,x_n) \rightarrow x_i = x_j), \]

where \( \varphi(x_1,\ldots,x_n) \) is a conjunction of atomic formulas (i.e., \( \varphi \) is a conjunctive query)

Examples:

- \[ \forall x_1,x_2,x_3(R(x_1,x_2) \land P(x_2,x_3) \land T(x_2) \rightarrow x_2 = x_3) \]
  - This is an egd, but not a FD.

- \[ \forall x_1,x_2,x_3(R(x_1,x_2) \land R(x_1,x_3) \rightarrow x_2 = x_3) \]
  - This is both an egd and a FD, namely \( A_1 \rightarrow A_2 \).
Tuple-Generating Dependencies

Definition: A tuple-generating dependency (tgd) is a formula of relational calculus of the form:

$$\forall x_1,\ldots,x_n (\varphi(x_1,\ldots,x_n) \rightarrow \exists y_1,\ldots,y_m \psi(x'_1,\ldots,x'_k, y_1,\ldots,y_m)),$$

where

- $\varphi(x_1,\ldots,x_n)$ and $\psi(x'_1,\ldots,x'_k, y_1,\ldots,y_m)$ are conjunctions of atomic formulas
- The variables $x'_1,\ldots,x'_k$ are among the variables $x_1,\ldots,x_n$.

Note: In effect, a tuple-generating dependency asserts that one conjunctive query (namely, the one defined by $\varphi(x_1,\ldots,x_n)$) is contained in another conjunctive query (namely, the one defined by $\exists y_1,\ldots,y_m \psi(x'_1,\ldots,x'_k, y_1,\ldots,y_m)$).
Tuple-Generating Dependencies

Examples:

- Every inclusion dependency is a tuple-generating dependency.

- \( \forall x,y,z \ (E(x,y) \land E(y,z) \rightarrow E(x,z)) \)
  - This is a tgd, but not an ID. It asserts that \( E \) is transitive.

- \( \forall x,y (E(x,y) \rightarrow \exists z (F(x,z) \land F(z,y))) \)
  - This says that for every edge in \( E \), there is a path of length 2 in \( F \).

- \( \forall x,y,z \ (P(x,y,z) \rightarrow R(x,y) \land T(y,z)) \)
  - This says that \( P \) is decomposed to \( R \) and \( T \).
Relational Calculus in Databases

Note:

- Relational calculus has been used in databases in two different ways:
  - As a database query language
  - As a specification language for expressing integrity constraints.

- In what follows, we will see that relational calculus is also used to formalize critical data interoperability tasks, such as
  - Data integration and
  - Data exchange
Data Integration

Query heterogeneous data in different sources via a virtual global schema
Data Exchange

Transform data structured under a *source* schema into data structured under a different *target* schema.
Schema Mappings

- Schema mappings:
  High-level, declarative assertions that specify the relationship between two database schemas.

- Schema mappings constitute the essential building blocks in formalizing and studying data interoperability tasks, including data integration and data exchange.

- Schema mappings help with the development of tools:
  - Are easier to generate and manage (semi)-automatically;
  - Can be compiled into SQL/XSLT scripts automatically.
Schema Mappings

- Schema Mapping $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$
  - Source schema $\mathbf{S}$, Target schema $\mathbf{T}$
  - A set $\Sigma$ of high-level, declarative assertions (constraints) that specify the relationship between $\mathbf{S}$-instances and $\mathbf{T}$-instances.

- $\text{Inst}(\mathbf{M}) = \{ (I, J) : I \text{ is an } \mathbf{S}\text{-instance, } J \text{ is a } \mathbf{T}\text{-instance, and } (I, J) \models \Sigma \}$.
Schema Mappings & Data Exchange

- **Schema Mapping** $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$
  - **Source** schema $\mathbf{S}$, **Target** schema $\mathbf{T}$
  - A set $\Sigma$ of high-level, declarative assertions (constraints) that specify the relationship between $\mathbf{S}$-instances and $\mathbf{T}$-instances.

- **Data Exchange** via the schema mapping $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$
  Transform a given **source** instance $\mathbf{I}$ to a **target** instance $\mathbf{J}$, so that $(\mathbf{I}, \mathbf{J})$ satisfy the specifications $\Sigma$ of $\mathbf{M}$. 
Solutions in Schema Mappings

**Definition:** Schema Mapping \( M = (S, T, \Sigma) \)

If \( I \) is a source instance, then a solution for \( I \) is a target instance \( J \) such that \( (I, J) \) satisfy \( \Sigma \).

**Fact:** In general, for a given source instance \( I \),

- No solution for \( I \) may exist (the constraints *overspecify*)
  or
- Multiple solutions for \( I \) may exist; in fact, *infinitely* many solutions for \( I \) may exist (the constraints *underspecify*).
Schema Mappings: Basic Problems

**Definition**: Schema Mapping \( M = (S, T, \Sigma) \)

- The **existence-of-solutions problem** \( \text{Sol}(M) \): (decision problem)
  Given a source instance \( I \), is there a solution \( J \) for \( I \)?

- The **data exchange problem associated with** \( M \): (function problem)
  Given a source instance \( I \), construct a solution \( J \) for \( I \), provided a solution exists.
 Ideally, schema mappings should be
- expressive enough to specify data interoperability tasks;
- simple enough to be efficiently manipulated by tools.

**Question**: How are schema mappings specified?

**Answer**: Use a high-level, declarative language. In particular, it is natural to try to use relational calculus (first-order logic) as a specification language for schema mappings.

**Fact**: There is a fixed relational calculus sentence specifying a schema mapping $M^*$ such that $\text{Sol}(M^*)$ is undecidable.

Hence, we need to restrict ourselves to well-behaved fragments of relational calculus.
Let us consider some simple tasks that a schema mapping specification language should support:

- **Copy (Nicknaming):**
  - Copy each source table to a target table and rename it.

- **Projection:**
  - Form a target table by projecting on one or more columns of a source table.

- **Decomposition:**
  - Decompose a source table into two or more target tables.

- **Column Augmentation:**
  - Form a target table by adding one or more columns to a source table.

- **Join:**
  - Form a target table by joining two or more source tables.

- **Combinations of the above** (e.g., “join + column augmentation”)
Schema Mapping Specification Languages

- **Copy (Nicknaming):**
  \[ \forall x_1, \ldots, x_n (P(x_1, \ldots, x_n) \rightarrow R(x_1, \ldots, x_n)) \]

- **Projection:**
  \[ \forall x, y, z (P(x, y, z) \rightarrow R(x, y)) \]

- **Decomposition:**
  \[ \forall x, y, z (P(x, y, z) \rightarrow R(x, y) \land T(y, z)) \]

- **Column Augmentation:**
  \[ \forall x, y (P(x, y) \rightarrow \exists z R(x, y, z)) \]

- **Join:**
  \[ \forall x, y, z (E(x, z) \land F(z, y) \rightarrow R(x, y, z)) \]

- **Combinations of the above** (e.g., “join + column augmentation”)
  \[ \forall x, y, z (E(x, z) \land F(z, y) \rightarrow \exists w T(x, y, z, w)) \]
Question: What do all these tasks (copy, projection, decomposition, column augmentation, join) have in common?

Answer:
- They can be specified using tuple-generating dependencies (tgds).
- In fact, they can be specified using a special class of tuple-generating dependencies known as source-to-target tuple generating dependencies (s-t tgds).
The relationship between source and target is given by formulas of relational calculus, called

**Source-to-Target Tuple Generating Dependencies** (s-t tgds)

\[ \forall \mathbf{x} \ (\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \ \psi(\mathbf{x}, \mathbf{y})) , \text{ where} \]

- \( \varphi(\mathbf{x}) \) is a conjunction of atoms over the source;
- \( \psi(\mathbf{x}, \mathbf{y}) \) is a conjunction of atoms over the target;
- \( \mathbf{x} \) and \( \mathbf{y} \) are tuples of variables.

**Example:**

\((\text{Student}(s) \land \text{Enrolls}(s,c)) \rightarrow \exists t \ \exists g \ (\text{Teaches}(t,c) \land \text{Grade}(s,c,g))\)

(here, we have dropped the universal quantifiers in front of s-t tgds)
Schema Mapping Specification Language

- s-t tgds assert that: some **conjunctive** query over the source is **contained** in some other **conjunctive** query over the target.

\[(\text{Student (s) } \land \text{Enrolls(s,c)) } \rightarrow \exists t \exists g (\text{Teaches(t,c) } \land \text{Grade(s,c,g))}\]

- s-t tgds generalize the main specifications used in data integration:
  - They generalize LAV (**local-as-view**) specifications:
    \[P(x) \rightarrow \exists y \psi(x, y),\] where \(P\) is a **source** schema.
  - **Note:** Copy, projection, and decomposition are LAV s-t tgds.
  - They generalize GAV (**global-as-view**) specifications:
    \[\varphi(x) \rightarrow R(x),\] where \(R\) is a **target** relation
    (they are equivalent to full tgds: \(\varphi(x) \rightarrow \psi(x)\), where \(\varphi(x)\) and \(\psi(x)\) are conjunctions of atoms).
  - **Note:** Copy, projection, and join are GAV s-t tgds.
In addition to source-to-target dependencies, we also consider target dependencies, since, after all, the target schema may have its own integrity constraints:

- **Target Tgds**: $\varphi_T(x) \rightarrow \exists y \psi_T(x, y)$

  Dept (did, dname, mgr_id, mgr_name) $\rightarrow$ Mgr (mgr_id, did)
  
  (a target inclusion dependency constraint)

- **Target Equality Generating Dependencies (egds)**: $\varphi_T(x) \rightarrow (x_1 = x_2)$

  (Mgr (e, d_1) $\land$ Mgr (e, d_2)) $\rightarrow$ (d_1 = d_2)

  (a target key constraint)
Schema Mapping $\mathbf{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where

- $\Sigma_{st}$ is a set of source-to-target tgds
- $\Sigma_t$ is a set of target tgds and target egds
Underspecification in Data Exchange

- **Fact:** Given a source instance, multiple solutions may exist.

- **Example:**
  Source relation $E(A,B)$, target relation $H(A,B)$
  \[
  \Sigma: \quad E(x,y) \rightarrow \exists z \ (H(x,z) \land H(z,y))
  \]
  Source instance $I = \{E(a,b)\}$
  Solutions: **Infinitely** many solutions exist
  - $J_1 = \{H(a,b), H(b,b)\}$
  - $J_2 = \{H(a,a), H(a,b)\}$
  - $J_3 = \{H(a,X), H(X,b)\}$
  - $J_4 = \{H(a,X), H(X,b), H(a,Y), H(Y,b)\}$
  - $J_5 = \{H(a,X), H(X,b), H(Y,Y)\}$

  constants:
  \[a, b, \ldots\]

  variables (labelled nulls):
  \[X, Y, \ldots\]
Main issues in data exchange

For a given source instance, there may be multiple target instances satisfying the specifications of the schema mapping. Thus,

- When more than one solution exist, which solutions are “better” than others?

- How do we compute a “best” solution?

- In other words, what is the “right” semantics of data exchange?
Definition (FKMP 2003): A solution is universal if it has homomorphisms to all other solutions (thus, it is a “most general” solution).

- **Constants**: entries in source instances
- **Variables** (labeled nulls): other entries in target instances
- **Homomorphism** $h: J_1 \rightarrow J_2$ between target instances:
  - $h(c) = c$, for constant $c$
  - If $P(a_1,\ldots,a_m)$ is in $J_1$, then $P(h(a_1),\ldots,h(a_m))$ is in $J_2$.

**Claim**: Universal solutions are the preferred solutions in data exchange.
Universal Solutions in Data Exchange

\[ \Sigma \]

Schema \( S \)  
Schema \( T \)

\[ I \rightarrow J \rightarrow J_1, J_2, J_3 \]

Universal Solution

Homomorphisms

\[ h_1, h_2, h_3 \]

Solutions
Example - continued

Source relation $S(A,B)$, target relation $T(A,B)$

$\Sigma : \quad E(x,y) \rightarrow \exists z \ (H(x,z) \land H(z,y))$

Source instance $I = \{E(a,b)\}$

Solutions: Infinitely many solutions exist

- $J_1 = \{H(a,b), H(b,b)\}$ is not universal
- $J_2 = \{H(a,a), H(a,b)\}$ is not universal
- $J_3 = \{H(a,X), H(X,b)\}$ is universal
- $J_4 = \{H(a,X), H(X,b), H(a,Y), H(Y,b)\}$ is universal
- $J_5 = \{H(a,X), H(X,b), H(Y,Y)\}$ is not universal
Universal solutions are analogous to most general unifiers in logic programming.

Uniqueness up to homomorphic equivalence:
If J and J’ are universal for I, then they are homomorphically equivalent.

Representation of the entire space of solutions:
Assume that J is universal for I, and J’ is universal for I’. Then the following are equivalent:
1. I and I’ have the same space of solutions.
2. J and J’ are homomorphically equivalent.
The Existence-of-Solutions Problem

**Question:** What can we say about the existence-of-solutions problem $\text{Sol}(M)$ for a fixed schema mapping $M = (S, T, \Sigma_{st}, \Sigma_t)$ specified by s-t tgds and target tgds and egds?

**Answer:** Depending on the target constraints in $\Sigma_t$:
- $\text{Sol}(M)$ can be trivial (solutions always exist).
  ...
- $\text{Sol}(M)$ can be in PTIME.
  ...
- $\text{Sol}(M)$ can be undecidable.
Proposition: If $M = (S, T, \Sigma_{st}, \Sigma_t)$ is a schema mapping such that $\Sigma_t$ is a set of full (GAV) target tgds, then:

- Solutions always exist; hence, $\text{Sol}(M)$ is trivial.

- There is a Datalog program $\pi$ over the target $T$ that can be used to compute universal solutions as follows:
  
  1. Compute a universal solution $J^*$ for $I$ w.r.t. the schema mapping $M^* = (S, T, \Sigma_{st})$ using the naïve chase algorithm.
  2. Run the Datalog program $\pi$ on $J^*$ to obtain a universal solution $J$ for $I$ w.r.t. $M$.

  Note: The Datalog program $\pi$ is initialized by setting all of its predicates to the values of the relations in $J^*$.

- Consequently, universal solutions can be computed in polynomial time.
The Naïve Chase Algorithm

**Naïve Chase Algorithm** for $M^* = (S, T, \Sigma_{st})$: given a source instance $I$, build a target instance $J^*$ that satisfies each s-t tgd in $\Sigma_{st}$

- by introducing new facts in $J^*$ as dictated by the RHS of the s-t tgd and
- by introducing new values (variables) in $J^*$ each time existential quantifiers need witnesses.

**Example:** $M = (S, T, \Sigma_{st})$ (here $\Sigma_t = \emptyset$)

$$\Sigma_{st} : E(x,y) \rightarrow \exists z(F(x,z) \land F(z,y))$$

The naïve chase returns a relation $F^*$ obtained from $E$ by adding a new node between every edge of $E$.

- If $E = \{(1,2)\}$, then $F^* = \{(1,N),(N,2)\}$ Universal solution for $E$
- If $E = \{(1,2),(2,3),(1,4)\}$, then $F^* = \{(1,M),(M,2),(2,N),(N,3),(1,U),(U,4)\}$
  Universal solution for $E$
The Naïve Chase Algorithm

**Example:** Collapsing paths of length 2 to edges

\[ M = (S, T, \Sigma_{st}) \quad (\text{here } \Sigma_t = \emptyset) \]

\[ \Sigma_{st}: \quad E(x,z) \land E(z,y) \rightarrow F(x,y) \quad \text{ (GAV mapping)} \]

- \[ E = \{ (1,3), (2,4), (3,4) \} \]
  \[ F^* = \{ F(1,4) \} \quad \text{Universal Solution for E} \]

- \[ E = \{ (1,3), (2,4), (3,4), (4,3) \} \]
  \[ F^* = \{ (1,4), (2,3), (3,3), (4,4) \} \quad \text{Universal solution for E} \]
Algorithmic Problems in Data Exchange

\[ M = (S, T, \Sigma_{st}, \Sigma_t) \] is a schema mapping such that \( \Sigma_t \) is a set of full (GAV) target tgds:
- Universal solutions can be computed in polynomial time using Naïve chase for \( \Sigma_{st} \) + Datalog program extracted from \( \Sigma_t \)

**Example:** \[ M = (S, T, \Sigma_{st}, \Sigma_t) \]

\[ \Sigma_{st}: E(x,y) \rightarrow \exists z(F(x,z) \land F(z,y)) \]
\[ \Sigma_t: F(u,w) \land F(w,v) \rightarrow F(u,v) \]

1. The naïve chase returns a relation \( F^* \) obtained from \( E \) by adding a new node between every edge of \( E \).
2. The Datalog program \( \pi \) computes the **transitive closure** of \( F^* \).
Algorithmic Problems in Data Exchang

**Proposition:** If $M = (S, T, \Sigma_{str}, \Sigma_{t})$ is a schema mapping such that $\Sigma_{t}$ is a set of **full target tgds** and **target egds**, then:

- Solutions need not always exist.
- The existence-of-solutions problem $\text{Sol}(M)$ is in PTIME, and may be PTIME-complete.

**Proof:** Reduction from **Path Systems**.
Recall the Datalog program:
$$T(x) : \text{-} A(x)$$
$$T(x) : \text{-} R(x,y,z), T(y), T(z).$$

**Fact:** The following problem is P-complete:
Given sets $A$ and $B$, and a ternary relation $R$, is $B \cap T \neq \emptyset$, where $T$ is the semantics of the above Datalog program.
Algorithmic Problems in Data Exchange

Reducing **Path Systems** to the Existence-of-Solutions Problem \( \text{Sol}(M) \)

- \( \Sigma_{st}: \) 
  - \( A(x) \rightarrow A'(x) \)
  - \( R(x,y,z) \rightarrow R'(x,y,z) \)
  - \( B(x) \rightarrow B'(x) \)
  - \( V(x) \rightarrow V'(x) \)

- \( \Sigma_{t}: \) 
  - \( A'(x) \rightarrow T(x) \)
  - \( T(y) \land T(z) \land R'(x,y,z) \rightarrow T(x) \)
  - \( T(x) \land B'(x) \land V'(u) \rightarrow W(u) \)
  - \( W(u) \land W(v) \rightarrow u = v \)

**Fact:** \( B \cap T \neq \emptyset \) if and only if the instance I has no solution, where I consists of A, B, R, and V = \{0,1\}. 
Algorithmic Problems in Data Exchange

**Question:**

What about arbitrary target tgds and egds?
Undecidability in Data Exchange

**Theorem** (K ..., Panttaja, Tan - 2006):

There is a schema mapping \( M = (S, T, \Sigma_{st}^*, \Sigma_t^*) \) such that:

- \( \Sigma_{st}^* \) consists of a single source-to-target tgd;
- \( \Sigma_t^* \) consists of one egd, one full target tgd, and one (non-full) target tgd;
- The existence-of-solutions problem \( \text{Sol}(M) \) is undecidable.

**Hint of Proof:**

Reduction from the

**Embedding Problem for Finite Semigroups:**

Given a finite partial semigroup, can it be embedded to a finite semigroup?
Reducing the **Embedding Problem for Semigroups** to **Sol(M)**

- $\Sigma_{st}$:  \( R(x,y,z) \rightarrow R'(x,y,z) \)

- $\Sigma_t$:
  - R’ is a partial function:
    \( R'(x,y,z) \land R'(x,y,w) \rightarrow z = w \)
  - R’ is associative
    \( R'(x,y,u) \land R'(y,z,v) \land R'(u,z,w) \rightarrow R'(x,u,w) \)
  - R’ is a total function
    \[
    R'(x,y,z) \land R'(x',y',z') \rightarrow \exists w_1 \ldots \exists w_9 \\
    (R'(x,x',w_1) \land R'(x,y',w_2) \land R'(x,z',w_3)) \\
    (R'(y,x',w_4) \land R'(y,y',w_5) \land R'(x,z',w_6)) \\
    (R'(z,x',w_7) \land R'(z,y',w_8) \land R'(z,z',w_9))
    \]