PhD course on

View-based query processing

Data integration – lecture 2

Riccardo Rosati
Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”
{rosati}@dis.uniroma1.it

Corso di Dottorato in Ingegneria Informatica, Università di Roma “La Sapienza”
Course overview

1. Introduction to view-based query processing [Lenzerini]

2. Conjunctive query evaluation [Gottlob]

3. Data exchange [Gottlob]

4. **Data integration** [De Giacomo, Rosati]

5. Data integration through ontologies [De Giacomo]

6. View-based query processing over semistructured data [Calvanese]

7. Reasoning about views [Lenzerini]
Lecture overview

- the role of global integrity constraints
- inclusion dependencies
- query reformulation under inclusion dependencies
  - chase
  - canonical model
  - query rewriting algorithm
- key dependencies
- decidability and separation
Global integrity constraints

- integrity constraints (ICs) posed over the global schema
- specify intensional knowledge about the domain of interest
- add semantics to the information
- but: data in the sources can conflict with global integrity constraints
- the presence of global integrity constraints rises semantic and computational problems
- open research problems
Integrity constraints for relational schemas

Most important ICs for the relational model:

- key dependencies (KDs)
- functional dependencies (FDs)
- inclusion dependencies (IDs)
- foreign keys (FKs)
- exclusion dependencies (EDs)
Inclusion dependencies (IDs)

- an ID states that the presence of a tuple in a relation implies the presence of a tuple in another relation where \( t' \) contains a projection of the values contained in \( t \)
- syntax: \( r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k] \)
- e.g., the ID \( r[1] \subseteq s[2] \)
  corresponds to the FOL sentence

\[
\forall x, y, z . r(x, y, z) \rightarrow \exists x', z' . s(x', x, z')
\]

- IDs are a special form of tuple-generating dependencies
Semantics for GAV systems under integrity constraints

We refer only to databases over a fixed infinite domain $\Gamma$.

Given a source database $C$ for a system $\mathcal{I}$, a global database $\mathcal{B}$ is legal for $(\mathcal{I}, C)$ if:

1. it satisfies the ICs on the global schema

2. it satisfies the mapping, i.e. $\mathcal{B}$ is constituted by a superset of the retrieved global database $\text{ret}(\mathcal{I}, C)$
   - $\text{ret}(\mathcal{I}, C)$ is obtained by evaluating, for each relation in $\mathcal{G}$, the mapping queries over the source database
   - assumption of sound mapping (open-world assumption)
Semantics: Certain Answers

- we are interested in **certain answers**

- a tuple \( t \) is a **certain answer** for a query \( Q \) if \( t \) is in the answer to \( Q \) for **all** (possibly infinite) legal databases for \((\mathcal{I}, \mathcal{C})\)

- the certain answers to \( Q \) are denoted by \( \text{cert}(Q, \mathcal{I}, \mathcal{C}) \)
Example

Global schema: \[ \text{player}(Pname, YOB, Pteam) \]
\[ \text{team}(Tname, Tcity, Tleader) \]

Constraints: \[ \text{team}[Tleader, Tname] \subseteq \text{player}[Pname, Pteam] \]

Mapping:
\[ \text{player} \rightsquigarrow \{ \]
\[ \text{player}(X, Y, Z) \leftarrow s_1(X, Y, Z) \]
\[ \text{player}(X, Y, Z) \leftarrow s_3(X, Y, Z) \]
\[ \text{team} \rightsquigarrow \text{team}(X, Y, Z) \leftarrow s_2(X, Y, Z) \]
Example (cont’d)

Source database $C$

\[ s_1: \boxed{\text{Totti} \ 1971 \ \text{Roma}} \quad s_2: \boxed{\text{Juve} \ \text{Torino} \ \text{Del Piero}} \quad s_3: \boxed{\text{Vieri} \ 1970 \ \text{Inter}} \]

Retrieved global database $ret(\mathcal{I}, C)$

player: \[ \boxed{\text{Totti} \ 1971 \ \text{Roma}} \quad \boxed{\text{Vieri} \ 1970 \ \text{Inter}} \]

team: \[ \boxed{\text{Juve} \ \text{Torino} \ \text{Del Piero}} \]
The ID on the global schema tells us that Del Piero is a player of Juve

All legal global databases for $\mathcal{I}$ have at least the tuples shown above, where $\alpha$ is some value of the domain $\Gamma$. 
The ID on the global schema tells us that Del Piero is a player of Juve.

All legal global databases for $\mathcal{I}$ have at least the tuples shown above, where $\alpha$ is some value of the domain $\Gamma$.

Warning 1 there may be an infinite number of legal databases for $\mathcal{I}$.
Example (cont’d)

<table>
<thead>
<tr>
<th>player</th>
<th>team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Totti</td>
<td>Roma</td>
</tr>
<tr>
<td>Vieri</td>
<td>Inter</td>
</tr>
<tr>
<td>Del Piero</td>
<td>$\alpha$</td>
</tr>
</tbody>
</table>

The ID on the global schema tells us that Del Piero is a player of Juve

All legal global databases for $\mathcal{I}$ have at least the tuples shown above, where $\alpha$ is some value of the domain $\Gamma$.

**Warning 1** there may be an infinite number of legal databases for $\mathcal{I}$

**Warning 2** in case of cyclic IDs, legal databases for $\mathcal{I}$ may be of infinite size
The ID on the global schema tells us that Del Piero is a player of Juve.

All legal global databases for \( \mathcal{I} \) have at least the tuples shown above, where \( \alpha \) is some value of the domain \( \Gamma \).

Consider the query  
\[
q(X, Z) \leftarrow \text{player}(X, Y, Z):
\]
#### Example (cont’d)

<table>
<thead>
<tr>
<th>player</th>
<th>team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Totti</td>
<td>Roma</td>
</tr>
<tr>
<td>Vieri</td>
<td>Inter</td>
</tr>
<tr>
<td>Del Piero</td>
<td>Juve</td>
</tr>
</tbody>
</table>

The ID on the global schema tells us that Del Piero is a player of Juve.

All legal global databases for $I$ have **at least** the tuples shown above, where $\alpha$ is some value of the domain $\Gamma$.

Consider the query $q(X, Z) \leftarrow \text{player}(X, Y, Z)$:

$$\text{cert}(q, I, C) = \{\langle Totti, Roma \rangle, \langle Vieri, Inter \rangle, \langle Del Piero, Juve \rangle\}$$
Query processing under inclusion dependencies

- intuitive strategy: add new facts until IDs are satisfied
- problem: infinite construction in the presence of **cyclic IDs**
- example 1: $r[2] \subseteq r[1]$
  
  suppose $\text{ret}(\mathcal{I}, \mathcal{C}) = \{r(a, b)\}$
  
  1) add $r(b, c_1)$
  2) add $r(c_1, c_2)$
  3) add $r(c_2, c_3)$
  ....
  (infinite construction)
Query processing under inclusion dependencies

- example 2: \( r[1] \subseteq s[1], \ s[2] \subseteq r[1] \)
  suppose \( \text{ret}(I, C) = \{ r(a, b) \} \)

1) add \( s(a, c_1) \)
2) add \( r(c_1, c_2) \)
3) add \( s(c_1, c_3) \)
4) add \( r(c_3, c_4) \)
5) add \( s(c_3, c_5) \)

....

(infinite construction)
The chase

- **chase** of a database: exhaustive application of a set of **rules** that transform the database, in order to make the database consistent with a set of integrity constraints

- the chase for IDs has only one rule, the **ID-chase rule**
• if the schema contains the ID \( r[i_1, \ldots, i_k] \subseteq s[j_1, \ldots, j_k] \)
and there is a fact in \( DB \) of the form \( r(a_1, \ldots, a_n) \)
and there are no facts in \( DB \) of the form \( s(b_1, \ldots, b_m) \)
such that \( a_{i_\ell} = b_{j_\ell} \) for each \( \ell \in \{1, \ldots, k\} \),
then add to \( DB \) the fact \( s(c_1, \ldots, c_m) \),
where for each \( h \) such that \( 1 \leq h \leq m \),
if \( h = j_\ell \) for some \( \ell \) then \( c_h = a_{i_\ell} \)
otherwise \( c_h \) is a new constant symbol
(not occurring already in \( DB \))

• notice: new existential symbols are introduced (skolem terms)
Properties of the chase

- bad news: the chase is in general infinite
- good news: the chase identifies a canonical model
- canonical model = a database that “represents” of all the models of the system
- we can use the chase to prove soundness and completeness of a query processing method
- but: only for positive queries!
why don’t we use a finite number of existential constants in the chase?

example: \( r[1] \subseteq s[1], \ s[2] \subseteq r[1] \)

suppose \( ret(\mathcal{I}, \mathcal{C}) = \{ r(a, b) \} \)

compute \( chase(ret(\mathcal{I}, \mathcal{C})) \) with only one new constant \( c_1 \):

0) \( r(a, b) \);  1) add \( s(a, c_1) \);  2) add \( r(c_1, c_1) \);  3) add \( s(c_1, c_1) \)

this database is not a canonical model for \( (\mathcal{I}, \mathcal{C}) \)

e.g., for the query \( q(X) := r(X, Y), s(Y, Y) : \)

\( a \in q^{chase(ret(\mathcal{I}, \mathcal{C}))} \) while \( a \notin \text{cert}(q, \mathcal{I}, \mathcal{C}) \)

\( \Rightarrow \) unsound method!

(and is unsound for any finite number of new constants)
An algorithm for rewriting CQs under IDs

- basic idea: let’s chase the query, not the data!
- query chase: dual notion of database chase
- IDs are applied from right to left
- advantage: much easier termination conditions! which imply:
  - decidability properties
  - efficiency
Query rewriting under inclusion dependencies

Given a user query $Q$ over $G$

- we look for a rewriting $R$ of $Q$ expressed over $S$
- a rewriting $R$ is perfect if $R^C = cert(Q, \mathcal{I}, C)$ for every source database $C$.

With a perfect rewriting, we can do query answering by rewriting

Note that we avoid the construction of the retrieved global database $ret(\mathcal{I}, C)$
Query rewriting for IDs

**Intuition:** Use the IDs as basic rewriting rules

\[
q(X, Z) \leftarrow \text{player}(X, Y, Z)
\]

\[
\text{team}[T\text{leader}, T\text{name}] \subseteq \text{player}[P\text{name}, P\text{team}]
\]

as a logic rule: \[
\text{player}(W_3, W_4, W_1) \leftarrow \text{team}(W_1, W_2, W_3)
\]
Query rewriting for IDs

**Intuition:** Use the IDs as basic rewriting rules

\[ q(X, Z) \leftarrow \text{player}(X, Y, Z) \]

\[ \text{team}[T\text{leader}, T\text{name}] \subseteq \text{player}[P\text{name}, P\text{team}] \]

as a logic rule:

\[ \text{player}(W_3, W_4, W_1) \leftarrow \text{team}(W_1, W_2, W_3) \]

**Basic rewriting step:**

- **when** the atom unifies with the **head** of the rule
- **substitute** the atom with the **body** of the rule

We add to the rewriting the query

\[ q(X, Z) \leftarrow \text{team}(Z, Y, X) \]
Query Rewriting for IDs: algorithm ID-rewrite

Iterative execution of:

1. **reduction**: atoms that unify with other atoms are eliminated and the unification is applied

2. **basic rewriting step**
The algorithm ID-rewrite

**Input:** relational schema $\Psi$, set of IDs $\Sigma_I$, UCQ $Q$

**Output:** perfect rewriting of $Q$

$Q' := Q$

repeat

$Q_{aux} := Q'$

for each $q \in Q_{aux}$ do

(a) for each $g_1, g_2 \in \text{body}(q)$ do

    if $g_1$ and $g_2$ unify then $Q' := Q' \cup \{\tau(\text{reduce}(q, g_1, g_2))\}$;

(b) for each $g \in \text{body}(q)$ do

    for each $I \in \Sigma_I$ do

        if $I$ is applicable to $g$ then $Q' := Q' \cup \{ q[g/\text{gr}(g, I)] \}$

until $Q_{aux} = Q'$;

return $Q'$
Properties of ID-rewrite

- ID-rewrite terminates
- ID-rewrite produces a perfect rewriting of the input query
  
  more precisely:

  \[ \text{unf}_M(q) = \text{unfolding} \text{ of the query } q \text{ w.r.t. the GAV mapping } M \]

- **Theorem:** \( \text{unf}_M(\text{ID-rewrite}(q)) \) is a perfect rewriting of the query \( q \)

- **Theorem:** query answering in GAV systems under IDs is in PTIME in data complexity (actually in LOGSPACE)
Key dependencies (KDs)

- a KD states that a set of attributes functionally determines all the relation attributes
- syntax: $key(r) = \{i_1, \ldots, i_k\}$
- e.g., the KD $key(r) = \{1\}$ corresponds to the FOL sentence
  \[
  \forall x, y, y', z, z'. r(x, y, z) \land r(x, y', z') \rightarrow y = y' \land z = z'
  \]
- KDs are a special form of equality-generating dependencies
- we assume that only one key is specified on every relation
Query answering under IDs and KDs

- possibility of inconsistencies (recall the sound mapping)
- when \( ret(I, C) \) violates the KDs, no legal database exists and query answering becomes trivial!

**Theorem:** Query answering under IDs and KDs is undecidable.

**Proof:** by reduction from implication of IDs and KDs.
Non-key-conflicting IDs (NKCIDs) are of the form

\[ r_1[A_1] \subseteq r_2[A_2] \]

where \( A_2 \) is not a strict superset of \( \text{key}(r_2) \)

**Theorem (IDs-KDs separation):** Under KDs and NKCIDs:

- if \( \text{ret}(I, C) \) satisfies the KDs
  - then the KDs can be ignored wrt certain answers of a user query \( Q \)
Separation for IDs and KDs

Non-key-conflicting IDs (NKCIDs) are of the form

\[ r_1[A_1] \subseteq r_2[A_2] \]

where \( A_2 \) is not a strict superset of \( key(r_2) \)

**Theorem (IDs-KDs separation):** Under KDs and NKCIDs:
- if \( ret(I, C) \) satisfies the KDs
- then the KDs can be ignored wrt certain answers of a user query \( Q \)

the problem is *undecidable* as soon as we extend the language of the IDs
Separation for IDs and KDs

Non-key-conflicting IDs (NKCIDs) are of the form

\[ r_1[A_1] \subseteq r_2[A_2] \]

where \( A_2 \) is not a strict superset of \( key(r_2) \)

**Theorem (IDs-KDs separation):** Under KDs and NKCIDs:

- if \( ret(I, C) \) satisfies the KDs
- then the KDs can be ignored wrt certain answers of a user query \( Q \)

the problem is **undecidable** as soon as we extend the language of the IDs

foreign keys (FKs) are a special case of NKCIDs
Query processing under separable KIDs and IDs

- global algorithm:
  1. verify consistency of $\text{ret}(\mathcal{I}, \mathcal{C})$ with respect to KIDs
  2. compute ID-rewrite of the input query
  3. unfold the query computed at previous step
  4. evaluate the query over the sources

- the KD consistency check can be done by suitable CQs with inequality

- (exercise: choose a key dependency and write a query that checks consistency with respect to such a key)

- computation of $\text{ret}(\mathcal{I}, \mathcal{C})$ can be avoided (by unfolding the queries for the KD consistency check)
Example: checking KD consistency

relation: player\([Pname, Pteam]\)

key dependency: \(key(player) = \{Pname\}\)

KD (in)consistency query:

\[q() \coloneqq \text{player}(X, Y), \text{player}(X, Z), Y \neq Z\]

\(q\) true iff the instance of \text{player} violates the key dependency
Example: unfolding a KD consistency query

mapping:

\[
\begin{align*}
\text{player}(X, Y) & \leftarrow s_1(X, Y) \\
\text{player}(X, Y) & \leftarrow s_2(X, Y)
\end{align*}
\]

\( q' \) = unfolding of \( q \):

\[
q'(\cdot) = s_1(X, Y), s_1(X, Z), Y \neq Z \lor \\
s_1(X, Y), s_2(X, Z), Y \neq Z \lor \\
s_2(X, Y), s_1(X, Z), Y \neq Z \lor \\
s_2(X, Y), s_2(X, Z), Y \neq Z
\]
Query answering under separable KDs and IDs

Computational characterization:

- **Theorem:** query answering in GAV systems under KDs and NKCIDs is in PTIME in data complexity (actually in LOGSPACE)
Information integration under integrity constraints

- the above algorithms are applicable in information integration systems with GAV mappings and (separable) KDs and IDs
- what happens in the presence of LAV mappings?
- what happens in the presence of other integrity constraints (exclusion dependencies)?
- see next lecture
The inconsistency issue

• ID are “repaired” by the sound semantics
• KD violations are NOT repaired
• need for a more “tolerant” semantics
• see next lecture
More expressive queries

- under KDs and FKS, can we go beyond CQs?
- union of CQs (UCQs): YES
  \[ \text{ID-rewrite}(q_1 \lor \ldots \lor q_n) = \text{ID-rewrite}(q_1) \lor \ldots \lor \text{ID-rewrite}(q_n) \]
- recursive queries: NO
- answering recursive queries under KDs and FKS is undecidable
  [Calvanese & Rosati, 2003]
- (same undecidability result holds in the presence of IDs only)