Principles of Managing Uncertain Data

Lecture 2: Databases, Queries, Constraints
Table of Contents

1 Database Model
2 Queries
3 Constraints
4 References
Table of Contents

1 Database Model
2 Queries
3 Constraints
4 References
A relation $r$ consists of a heading $(A_1, \ldots, A_m)$, which is a sequence $(A_1, \ldots, A_m)$ of distinct attributes, and a body, which is a finite collection of tuples $t = (a_1, \ldots, a_m)$ of values.

- We assume some infinite domain of values (or constants)
A relation \( r \) consists of a heading \((A_1, \ldots, A_m)\), which is a sequence \((A_1, \ldots, A_m)\) of distinct attributes, and a body, which is a finite collection of tuples \( t = (a_1, \ldots, a_m) \) of values.

- We assume some infinite domain of values (or constants).

- By a convenient abuse of notation, a relation is often identified with its body.
  - (e.g., \( t \in r \) means that \( t \) is a tuple in the body of \( r \)).
A relation $r$ consists of a heading $(A_1, \ldots, A_m)$, which is a sequence $(A_1, \ldots, A_m)$ of distinct attributes, and a body, which is a finite collection of tuples $t = (a_1, \ldots, a_m)$ of values.

- We assume some infinite domain of values (or constants).
- By a convenient abuse of notation, a relation is often identified with its body.
  - (e.g., $t \in r$ means that $t$ is a tuple in the body of $r$)
- We may refer to the $i$th value in a tuple $t \in R$ as $t.A_i$ or $t[i]$.
A relation schema has a relation name (or relation symbol) $R$ and a heading $(A_1, \ldots, A_n)$, which is again a sequence of attributes.
A **relation schema** has a **relation name** (or **relation symbol**) $R$ and a heading $(A_1, \ldots, A_n)$, which is again a sequence of attributes

- Denoted by $R(A_1, \ldots, A_m)$
- Or simply $R$ if the attributes are not important or clear from the context
A relation schema has a relation name (or relation symbol) $R$ and a heading $(A_1, \ldots, A_n)$, which is again a sequence of attributes

- Denoted by $R(A_1, \ldots, A_m)$
- Or simply $R$ if the attributes are not important or clear from the context

The arity of $R(A_1, \ldots, A_m)$ is $m$, and is denoted by $ar(R)$.
A **relation schema** has a **relation name** (or **relation symbol**) $R$ and a heading $(A_1, \ldots, A_n)$, which is again a sequence of attributes
- Denoted by $R(A_1, \ldots, A_m)$
- Or simply $R$ if the attributes are not important or clear from the context

The **arity** of $R(A_1, \ldots, A_m)$ is $m$, and is denoted by $ar(R)$

Sometimes the attributes are not important, and we may use just $R/m$ to specify that $R$ is a relation name of arity $m$
Database Schemas

- A *relation schema* has a *relation name* (or *relation symbol*) $R$ and a heading $(A_1, \ldots, A_n)$, which is again a sequence of attributes
  - Denoted by $R(A_1, \ldots, A_m)$
  - Or simply $R$ if the attributes are not important or clear from the context
- The *arity* of $R(A_1, \ldots, A_m)$ is $m$, and is denoted by $ar(R)$
- Sometimes the attributes are not important, and we may use just $R/m$ to specify that $R$ is a relation name of arity $m$
A relation schema has a relation name (or relation symbol) $R$ and a heading $(A_1, \ldots, A_n)$, which is again a sequence of attributes

- Denoted by $R(A_1, \ldots, A_m)$
- Or simply $R$ if the attributes are not important or clear from the context

The arity of $R(A_1, \ldots, A_m)$ is $m$, and is denoted by $\text{ar}(R)$

Sometimes the attributes are not important, and we may use just $R/m$ to specify that $R$ is a relation name of arity $m$

A schema is a pair $\mathcal{S} = (\mathcal{R}, \Sigma)$, where $\mathcal{R}$ is a set of relation schemas with distinct names, and $\Sigma$ is a set of constraints over $\mathcal{R}$
A relation schema has a relation name (or relation symbol) \( R \) and a heading \((A_1, \ldots, A_n)\), which is again a sequence of attributes
- Denoted by \( R(A_1, \ldots, A_m) \)
- Or simply \( R \) if the attributes are not important or clear from the context

The arity of \( R(A_1, \ldots, A_m) \) is \( m \), and is denoted by \( ar(R) \)

Sometimes the attributes are not important, and we may use just \( R/m \) to specify that \( R \) is a relation name of arity \( m \)

A schema is a pair \( S = (\mathcal{R}, \Sigma) \), where \( \mathcal{R} \) is a set of relation schemas with distinct names, and \( \Sigma \) is a set of constraints over \( \mathcal{R} \)
- \( \mathcal{R} \) is often called a signature
- We later discuss languages of constraints
A relation $r$ is said to be over a relation schema $R$ if $r$ and $R$ have the same heading.
A relation $r$ is said to be over a relation schema $R$ if $r$ and $R$ have the same heading.

A database (or instance) $I$ of a schema $S = (\mathcal{R}, \Sigma)$ associates with every relation name $R$ a relation $R^I$ over $R$, such that all the constraints in $\Sigma$ are satisfied.
A relation $r$ is said to be over a relation schema $R$ if $r$ and $R$ have the same heading.

A database (or instance) $I$ of a schema $S = (\mathcal{R}, \Sigma)$ associates with every relation name $R$ a relation $R^I$ over $R$, such that all the constraints in $\Sigma$ are satisfied.

We denote by $\text{Inst}(S)$ the set of all the instances of $S$. 
Logical Viewpoint

- It is convenient and common to view the database as a *logical structure*
  - vocabulary = signature + built-in predicates (e.g., <, >, =), and interpretation = instance (more formality later)
It is convenient and common to view the database as a *logical structure*

- vocabulary = signature + built-in predicates (e.g., $<$, $>$, $=$), and interpretation = instance (more formality later)

- Database queries are viewed as logical formulas $\varphi(x)$ over the database: $Q(I) = \{a \mid I \models \varphi(a)\}$
Logical Viewpoint

- It is convenient and common to view the database as a *logical structure*
  - vocabulary = signature + built-in predicates (e.g., <, >, =), and interpretation = instance (more formality later)

- Database queries are viewed as logical formulas \( \varphi(x) \) over the database: 
  \[ Q(I) = \{ a \mid I \models \varphi(a) \} \]

- But there are some significant restrictions:
  - We usually have only relation (no function) symbols
  - We usually consider only finite structures (cf. finite-model theory)
  - Queries should be independent of the domain outside the database (cf. relational calculus, discussed later)
Consider the set \( F = \{ \varphi_i \mid i = 1, 2, \ldots \} \) of sentences where \( \varphi_i \) is the sentence “\( R \) has at least \( i \) distinct tuples”

Each \( \varphi_i \) is expressible in first-order logic
On Finite Structures

- Consider the set $F = \{ \varphi_i \mid i = 1, 2, \ldots \}$ of sentences where $\varphi_i$ is the sentence “$R$ has at least $i$ distinct tuples”
- Each $\varphi_i$ is expressible in first-order logic
- Every finite subset of $F$ has a finite model, but there is no finite model for $F$
Consider the set $F = \{ \varphi_i \mid i = 1, 2, \ldots \}$ of sentences where $\varphi_i$ is the sentence “$R$ has at least $i$ distinct tuples”

- Each $\varphi_i$ is expressible in first-order logic
- Every finite subset of $F$ has a finite model, but there is no finite model for $F$
- Hence, the compactness theorem no longer holds in the finite
Table of Contents

1. Database Model
2. Queries
3. Constraints
4. References
The queries we will consider are such that produce a relation from the database.

Formally, a query $Q$ over a schema $S$ is associated with a heading $(A_1, \ldots, A_k)$, and it maps every instance $I \in \text{Inst}(S)$ into a relation with that heading.
Example

Find all pairs \((s, c)\) of students and courses, such that there exists a course number \(x\) where both \(\text{Takes}(s, x)\) and \(\text{Course}(x, c)\) hold.

<table>
<thead>
<tr>
<th>student</th>
<th>cno</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>1</td>
<td>1 AI</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
<td>2 DB</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
<td>3 PL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>student</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>AI</td>
</tr>
<tr>
<td>Alon</td>
<td>AI</td>
</tr>
<tr>
<td>Ahuva</td>
<td>DB</td>
</tr>
</tbody>
</table>
A special case is where $k = 0$, and then the result either contains the empty tuple or is empty; in this case we say that the query is *Boolean*. 

Example: Is it the case that for some $x$, both $\text{Takes}('Ahuva', x)$ and $\text{Course}(x, 'DB')$ hold?
A special case is where \( k = 0 \), and then the result either contains the empty tuple or is empty; in this case we say that the query is **Boolean**.

Example: Is it the case that for some \( x \), both \( \text{Takes('Ahuva', x)} \) and \( \text{Course}(x, 'DB') \) hold?

We often denote

\[
Q(I) = \{(\)} \quad \text{by} \quad Q(I) = \text{true} \quad \text{or} \quad I \models Q
\]

\[
Q(I) = \emptyset \quad \text{by} \quad Q(I) = \text{false} \quad \text{or} \quad I \not\models Q
\]
Boolean Queries

- A special case is where $k = 0$, and then the result either contains the empty tuple or is empty; in this case we say that the query is *Boolean*

- Example: Is it the case that for some $x$, both Takes('Ahuva', $x$) and Course($x$, 'DB') hold?

- We often denote
  - $Q(I) = \{()\}$ by $Q(I) = \text{true}$ or $I \models Q$
  - $Q(I) = \emptyset$ by $Q(I) = \text{false}$ or $I \not\models Q$

- Boolean queries are very important in the analysis query languages (expressiveness, complexity, optimization and equivalence, etc.)
Relational Algebra

- Introduced by Codd [Cod72]
- Used by existing database systems, mainly for internal query-plan optimization
Relational Algebra

- Introduced by Codd [Cod72]
- Used by existing database systems, mainly for internal query-plan optimization
- A collection of operations over relations
  - Unary: \( r \rightarrow t \); binary: \( (r, s) \rightarrow t \)
Relational Algebra

- Introduced by Codd [Cod72]
- Used by existing database systems, mainly for internal query-plan optimization
- A collection of operations over relations
  - Unary: $r \rightarrow t$; binary: $(r, s) \rightarrow t$
- Queries via:
  1. Applying the operators to the database relations
  2. Composition
Algebraic Operators

- Union ($\cup$), difference ($-$)
  - $R \cup S$ and $R - S$ allowed if $R$ and $S$ are *union compatible*, that is, they have the same heading

- Cartesian product ($\times$)
  - $R \times S$ allowed only when $R$ and $S$ have disjoint headings

- Projection ($\pi$)
  - $\pi_{A_1', \ldots, A_k'}(R)$ allowed if $A_1', \ldots, A_k'$ are distinct attributes of $R$

- Selection ($\sigma$)
  - $\sigma_\varphi(R)$ allowed if $\varphi$ is a condition over the attributes of $R$ (e.g., $A_1 = A_2$ or $A_1 \neq A_2$)

- Renaming ($\rho$)
  - $\rho_{A \rightarrow B}(R)$ allowed if $A$ is an attribute of $R$ and $B$ is not an attribute of $R$
Example

\[ T \times \rho_{cno \rightarrow c} C \]

<table>
<thead>
<tr>
<th>student</th>
<th>cno</th>
<th>c</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>1</td>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
<td>2</td>
<td>DB</td>
</tr>
<tr>
<td>Ahuva</td>
<td>1</td>
<td>2</td>
<td>DB</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
<td>3</td>
<td>PL</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
<td>3</td>
<td>PL</td>
</tr>
</tbody>
</table>
### Example

**Takes**

<table>
<thead>
<tr>
<th>student</th>
<th>cno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>1</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>

**Course**

<table>
<thead>
<tr>
<th>cno</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>2</td>
<td>DB</td>
</tr>
<tr>
<td>3</td>
<td>PL</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{cno}=c}(\text{Takes} \times \rho_{\text{cno}\rightarrow c}\text{Course}) \]

<table>
<thead>
<tr>
<th>student</th>
<th>cno</th>
<th>c</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>1</td>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
<td>2</td>
<td>DB</td>
</tr>
</tbody>
</table>
### Example

**Takes**

<table>
<thead>
<tr>
<th>student</th>
<th>cno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>1</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>

**Course**

<table>
<thead>
<tr>
<th>cno</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>2</td>
<td>DB</td>
</tr>
<tr>
<td>3</td>
<td>PL</td>
</tr>
</tbody>
</table>

\[
\pi_{\text{student}, \text{cno}, \text{cname}} \left( \sigma_{\text{cno}=c} \left( \text{Takes} \times \rho_{\text{cno} \rightarrow c} \text{Course} \right) \right)
\]

<table>
<thead>
<tr>
<th>student</th>
<th>cno</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
<td>DB</td>
</tr>
</tbody>
</table>

In short: Takes \( \bowtie \) Course (natural join)
Example

### Takes

<table>
<thead>
<tr>
<th>student</th>
<th>cno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>1</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
</tr>
</tbody>
</table>

### Course

<table>
<thead>
<tr>
<th>cno</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AI</td>
</tr>
<tr>
<td>2</td>
<td>DB</td>
</tr>
<tr>
<td>3</td>
<td>PL</td>
</tr>
</tbody>
</table>

\[
\pi_{\text{student}, \text{cname}}(\sigma_{\text{cno}=c} (\text{Takes} \times \rho_{\text{cno} \rightarrow c} \text{Course}))
\]

<table>
<thead>
<tr>
<th>student</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>AI</td>
</tr>
<tr>
<td>Alon</td>
<td>AI</td>
</tr>
<tr>
<td>Ahuva</td>
<td>DB</td>
</tr>
</tbody>
</table>
Recall: in FO (First-Order Logic), a structure is a triple $(D, \sigma, I)$ where:

- $D$ is a domain of values
- $\sigma$ is a signature (collection of function and relation symbols)
- $I$ is an interpretation (that instantiates $\sigma$ using values from $D$)
Recall: in FO (First-Order Logic), a *structure* is a triple $(D, \sigma, I)$ where:
- $D$ is a *domain* of values
- $\sigma$ is a *signature* (collection of function and relation symbols)
- $I$ is an *interpretation* (that instantiates $\sigma$ using values from $D$)
Recall: in FO (First-Order Logic), a \textit{structure} is a triple \((D, \sigma, I)\) where:

- \(D\) is a \textit{domain} of values
- \(\sigma\) is a \textit{signature} (collection of function and relation symbols)
- \(I\) is an \textit{interpretation} (that instantiates \(\sigma\) using values from \(D\))

In \textit{Relational Calculus} (RC) we view the database schema as the signature, and the database (instance) as the interpretation

- There is no concept of a \textit{domain}, which leads to problems that we later discuss
Example: Division

- Let $R(A, B)$ and $S(B)$ be two relation schemas.
- Recall: $R \div S$ consists of all $A$s of $R$ that occur in $R$ with every $B$ of $S$.
  - Example: $\text{Takes}(A, C') \div \text{Course}(C')$ consists of all the students that take every course.
Example: Division

- Let $R(A, B)$ and $S(B)$ be two relation schemas
- Recall: $R ÷ S$ consists of all $A$s of $R$ that occur in $R$ with every $B$ of $S$
  - Example: Takes$(A, C) ÷$ Course$(C)$ consists of all the students that take every course
- In RA: $\pi_A R \setminus \pi_A ((\pi_A R \times S) \setminus R)$
Example: Division

- Let \( R(A, B) \) and \( S(B) \) be two relation schemas.
- Recall: \( R \div S \) consists of all \( A \)s of \( R \) that occur in \( R \) with every \( B \) of \( S \).
  - Example: \( \text{Takes}(A, C) \div \text{Course}(C) \) consists of all the students that take every course.
- In RA: \( \pi_A R \setminus \pi_A ( (\pi_A R \times S) \setminus R ) \)
- In RC: \( \{(x) \mid \exists z[R(x, z)] \land \forall y[S(y) \rightarrow R(x, y)]\} \)
Formal Definition

- Let $S$ be a schema
Formal Definition

- Let $S$ be a schema
- An atomic formula (over $S$) has one of the following forms:
  - $R(τ_1, \ldots, τ_k)$ where $R$ is a $k$-ary relation symbol and each term $τ_i$ is a variable or a constant (e.g., Takes($x, AI$))
  - $τ_1 = τ_2$ or $τ_1 \neq τ_2$, where each term $τ_i$ is a variable or a constant
Let $S$ be a schema.

An *atomic formula* (over $S$) has one of the following forms:

- $R(\tau_1, \ldots, \tau_k)$ where $R$ is a $k$-ary relation symbol and each term $\tau_i$ is a variable or a constant (e.g., $\text{Takes}(x, \text{AI})$)
- $\tau_1 = \tau_2$ or $\tau_1 \neq \tau_2$, where each term $\tau_i$ is a variable or a constant

An *RC formula* is a formula that is built from atomic formulas using connectives ($\land$, $\lor$, $\neg$, $\rightarrow$) and quantifiers ($\exists$, $\forall$)

- Note: an RC formula may have free (unquantified) variables
  - I omit the recursive definition of free variables
- We denote by $\varphi(x_1, \ldots, x_n)$ a formula that has free variables among $x_1, \ldots, x_n$
Let $\mathcal{S}$ be a schema.

An **atomic formula** (over $\mathcal{S}$) has one of the following forms:

- $R(\tau_1, \ldots, \tau_k)$ where $R$ is a $k$-ary relation symbol and each term $\tau_i$ is a variable or a constant (e.g., $\text{Takes}(x, \text{AI})$)
- $\tau_1 = \tau_2$ or $\tau_1 \neq \tau_2$, where each term $\tau_i$ is a variable or a constant

An **RC formula** is a formula that is built from atomic formulas using connectives ($\land, \lor, \neg, \to$) and quantifiers ($\exists, \forall$)

- Note: an RC formula may have free (unquantified) variables
  - I omit the recursive definition of free variables
- We denote by $\varphi(x_1, \ldots, x_n)$ a formula that has free variables among $x_1, \ldots, x_n$

An **RC query** has the form $\{ (x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n) \}$
**Domain Dependence**

- What is the meaning of:

\[
\{(x) \divides R(x)\}
\]

\[
\{(x) \divides \neg R(x)\}
\]

In FO it makes sense, since we have a domain. In databases we wish to forbid queries that depend on the domain.
What is the meaning of:

\[ \{ (x) \mid R(x) \} \]?
Domain Dependence

- What is the meaning of:
  - \( \{ (x) \mid R(x) \} \)?
  - \( \{ (x) \mid \neg R(x) \} \)?
Domain Dependence

- What is the meaning of:
  - \( \{ (x) \mid R(x) \} \)?
  - \( \{ (x) \mid \neg R(x) \} \)?

- In FO it makes sense, since we have a domain
Domain Dependence

- What is the meaning of:
  - \( \{ x \mid R(x) \} \)?
  - \( \{ x \mid \neg R(x) \} \)?

- In FO it makes sense, since we have a domain

- In databases we wish to *forbid* queries that depend on the domain
What is the meaning of:

- \( \{ (x) \mid R(x) \} \)?
- \( \{ (x) \mid \neg R(x) \} \) ?

In FO it makes sense, since we have a domain.

In databases we wish to *forbid* queries that depend on the domain.

Next, we formalize this idea.
Let $S$ be a schema, $I$ an instance, and
\[ Q = \{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\} \] an RC query
Domain Independence

- Let $S$ be a schema, $I$ an instance, and $Q = \{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}$ an RC query.
- The *active domain* is the set of constants that appear in either $I$ or $Q$. 
Domain Independence

- Let \( S \) be a schema, \( I \) an instance, and

\[
Q = \{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}
\]

an RC query

- The \textit{active domain} is the set of constants that appear in either \( I \) or \( Q \)

- If \( D \) is a domain that contains the active domain, then an \textit{answer} for \( Q \) is an assignment to \( x_1, \ldots, x_n \) that makes \( \varphi(x_1, \ldots, x_n) \) true (in the FO sense) in the interpretation \( I \)
  - Recall, now we have a complete FO signature
Let $S$ be a schema, $I$ an instance, and $Q = \{(x_1, \ldots, x_n) \mid \varphi(x_1, \ldots, x_n)\}$ an RC query.

The **active domain** is the set of constants that appear in either $I$ or $Q$.

If $D$ is a domain that contains the active domain, then an **answer** for $Q$ is an assignment to $x_1, \ldots, x_n$ that makes $\varphi(x_1, \ldots, x_n)$ true (in the FO sense) in the interpretation $I$.

Recall, now we have a complete FO signature.

We say that $Q$ is **domain independent** if for every two domains $D$ and $E$ that contain the active domain we have $Q^D(I) = Q^E(I)$.
Equivalence to RA

- We denote by RA the class of queries that can be phrased in relational algebra, where only equality is allowed in selection predicates.
- Two queries $Q_1$ and $Q_2$ (possibly in different formalisms) over the same schema $S$ are equivalent, denoted $Q_1 \equiv Q_2$, if for all instances $I$ of $S$ we have $Q_1(I) = Q_2(I)$.
Equivalence Between RA and RC

**Theorem [Cod72]**

RA and domain-independent RC have the same expressive power. That is, for all schemas $S$:

- For every RA expression $\alpha$ there is a domain-independent RC query $Q$ such that $\alpha \equiv Q$.
- For every domain-independent RC query $Q$ there is an RA expression $\alpha$ such that $\alpha \equiv Q$. 
Can we test whether a given RC query is domain independent?
Effective Syntax

- *Can we test whether a given RC query is domain independent?*
- Unfortunately, this problem is undecidable [Pao69]
Can we test whether a given RC query is domain independent?

Unfortunately, this problem is undecidable [Pao69].

Nevertheless, there is an effective syntax for domain-independent RC queries; that is, a fragment of safe queries where:

- Every safe query is domain independence
- Safety can be detected in polynomial time
- Every domain-independent RC query is equivalent to some safe query
SQL (Structured Query Language) is natural language to express relational algebra

- SELECT (projection) ... AS (rename)
- FROM (Cartesian product)
- WHERE (selection)
- UNION
- MINUS

And much more, e.g., aggregate operators (e.g., COUNT, SUM), clustering operators (e.g., GROUP BY, HAVING), ranking (e.g., ORDER BY, LIMIT), and more
The Example in SQL

Find all pairs \((s, c)\) of students and courses, such that there exists a course number \(x\) where both \(\text{Takes}(s, x)\) and \(\text{Course}(x, c)\) hold.

```
SELECT S.student, C.cname
FROM Takes T, Course C
WHERE T.cno = C.cno
```
Conjunctive Queries

- Conjunctive Queries (CQs) are SELECT-FROM-WHERE queries (no MINUS, UNION, etc.) such that all the WHERE conditions are equalities among attributes.
Conjunctive Queries

- Conjunctive Queries (CQs) are SELECT-FROM-WHERE queries (no MINUS, UNION, etc.) such that all the WHERE conditions are equalities among attributes.

- CQs are typically represented in the following FOL notation:

\[ Q(x) : \exists y [ \varphi_1(x, y) \land \cdots \land \varphi_k(x, y) ] \]

where:

1. \( x \) and \( y \) are disjoint sequences of variables.
2. Each \( \varphi_i(x, y) \) is an atomic formula of the form \( R(\tau_1, \ldots, \tau_m) \) where \( R \) is an \( m \)-ary relation in the schema and each \( \tau_j \) is either a variable in \( x \), a variable in \( y \), or a constant value (e.g., 7 or 'Ahuva').
3. Every variable in \( x \) occurs at least once on the right hand side.
Conjunctive Queries

- Conjunctive Queries (CQs) are SELECT-FROM-WHERE queries (no MINUS, UNION, etc.) such that all the WHERE conditions are equalities among attributes.

- CQs are typically represented in the following FOL notation:

  \[ Q(x) := \exists y[\varphi_1(x, y) \land \cdots \land \varphi_k(x, y)] \]

  where:
  - \( x \) and \( y \) are disjoint sequences of variables.
Conjunctive Queries

- Conjunctive Queries (CQs) are SELECT-FROM-WHERE queries (no MINUS, UNION, etc.) such that all the WHERE conditions are equalities among attributes.
- CQs are typically represented in the following FOL notation:

\[ Q(x) := \exists y [\varphi_1(x,y) \land \cdots \land \varphi_k(x,y)] \]

where:
- \( x \) and \( y \) are disjoint sequences of variables.
- Each \( \varphi_i(x,y) \) is a an **atomic formula** of the form \( R(\tau_1, \ldots, \tau_m) \) where \( R \) is an \( m \)-ary relation in the schema and each \( \tau_j \) is either a variable in \( x \), a variable in \( y \), or a constant value (e.g., 7 or 'Ahuva').
Conjunctive Queries

- Conjunctive Queries (CQs) are SELECT-FROM-WHERE queries (no MINUS, UNION, etc.) such that all the WHERE conditions are equalities among attributes.
- CQs are typically represented in the following FOL notation:

\[ Q(x) :\exists y[\varphi_1(x, y) \land \cdots \land \varphi_k(x, y)] \]

where:
- \( x \) and \( y \) are disjoint sequences of variables.
- Each \( \varphi_i(x, y) \) is an atomic formula of the form \( R(\tau_1, \ldots, \tau_m) \) where \( R \) is an \( m \)-ary relation in the schema and each \( \tau_j \) is either a variable in \( x \), a variable in \( y \), or a constant value (e.g., 7 or 'Ahuva').
- Every variable in \( x \) occurs at least once on the right hand side.
CQ Terminology and Notation

\[ Q(x) \colon \exists y [\phi_1(x, y) \land \cdots \land \phi_k(x, y)] \]
CQ Terminology and Notation

\[ Q(x) := \exists y[\varphi_1(x, y) \land \cdots \land \varphi_k(x, y)] \]

For simplification, quantification and conjunction are omitted:

\[ Q(x) := \varphi_1(x, y), \cdots, \varphi_k(x, y) \]
CQ Terminology and Notation

\[ Q(x) := \exists y [\varphi_1(x, y) \land \cdots \land \varphi_k(x, y)] \]

For simplification, quantification and conjunction are omitted:

\[ Q(x) := \varphi_1(x, y), \cdots, \varphi_k(x, y) \]

atom

\[ Q(x) \]

head

\[ \varphi_1(x, y), \cdots, \varphi_k(x, y) \]

body
CQ Terminology and Notation

\[ Q(x) := \exists y[\varphi_1(x, y) \land \cdots \land \varphi_k(x, y)] \]

For simplification, quantification and conjunction are omitted:

\[
\begin{array}{c}
\hline
\text{atom} \\
Q(x) : - \varphi_1(x, y), \cdots, \varphi_k(x, y) \\
\hline
\end{array}
\]

A variables in \( x \) is called a free or head variable, and a variable in \( y \) is called an existential variable
The Example as CQ

Find all pairs \((s, c)\) of students and courses, such that there exists a course number \(x\) where both \(\text{Takes}(s, x)\) and \(\text{Course}(x, c)\) hold.
The Example as CQ

Find all pairs \((s, c)\) of students and courses, such that there exists a course number \(x\) where both \(\text{Takes}(s, x)\) and \(\text{Course}(x, c)\) hold

\[
Q(s, c) :\text{\texttt{\texttt{\texttt{\texttt{\texttt{Takes}(s, x) \wedge Course(x, c)}}}}}\]

Why Are CQs Interesting?

- This is the class of all the queries that can be phrased in RA when using only:
  - Selection with equality predicate
  - Projection
  - Join

- For that reason, CQs are often called *SPJ* queries
Why Are CQs Interesting?

- This is the class of all the queries that can be phrased in RA when using only:
  - Selection with equality predicate
  - Projection
  - Join
- For that reason, CQs are often called \textit{SPJ} queries
- CQs are the building block of expressive query languages, such as Datalog
Why Are CQs Interesting?

- This is the class of all the queries that can be phrased in RA when using only:
  - Selection with equality predicate
  - Projection
  - Join
- For that reason, CQs are often called *SPJ* queries
- CQs are the building block of expressive query languages, such as Datalog
- Useful queries that are simple enough to perform deep investigation for various database problems
  - As we shall see, there are significant deep insights and algorithms that apply only to conjunctive queries
Why Do We Care about Constraints?

- Allow to enforce database coherence and avoid bugs
- Allow to formally determine what inconsistency means
  - Very relevant to us
- May have dramatic effect on algorithms and complexity
- By focusing on specific classes of constraint languages, we allow for nontrivial analysis
### Functional Dependencies

#### Lab

<table>
<thead>
<tr>
<th>name</th>
<th>faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIPL</td>
<td>EE</td>
</tr>
<tr>
<td>LCL</td>
<td>CS</td>
</tr>
<tr>
<td>SSDL</td>
<td>CS</td>
</tr>
<tr>
<td>STAT</td>
<td>IE</td>
</tr>
</tbody>
</table>

#### LabRoom

<table>
<thead>
<tr>
<th>lab</th>
<th>building</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIPL</td>
<td>Meyer</td>
<td>100</td>
</tr>
<tr>
<td>SIPL</td>
<td>Meyer</td>
<td>101</td>
</tr>
<tr>
<td>LCL</td>
<td>Taub</td>
<td>100</td>
</tr>
<tr>
<td>SSDL</td>
<td>Taub</td>
<td>200</td>
</tr>
</tbody>
</table>

A lab belongs to just one faculty (i.e., name is a key for Lab).

A specific room in a specific building belongs to only one lab.

A lab may have multiple rooms, but all in the same building.
A lab belongs to just one faculty (i.e., name is a key for Lab)
Functional Dependencies

- A lab belongs to just one faculty (i.e., name is a key for Lab)
- A specific room in a specific building belongs to only one lab

<table>
<thead>
<tr>
<th>Lab</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>faculty</td>
<td></td>
</tr>
<tr>
<td>SIPL</td>
<td>EE</td>
<td></td>
</tr>
<tr>
<td>LCL</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>SSDL</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>STAT</td>
<td>IE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LabRoom</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>lab</td>
<td>building</td>
<td>room</td>
</tr>
<tr>
<td>SIPL</td>
<td>Meyer</td>
<td>100</td>
</tr>
<tr>
<td>SIPL</td>
<td>Meyer</td>
<td>101</td>
</tr>
<tr>
<td>LCL</td>
<td>Taub</td>
<td>100</td>
</tr>
<tr>
<td>SSDL</td>
<td>Taub</td>
<td>200</td>
</tr>
</tbody>
</table>
Functional Dependencies

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>lab</td>
</tr>
<tr>
<td>faculty</td>
<td>building</td>
</tr>
<tr>
<td>SIPL</td>
<td>SIPL</td>
</tr>
<tr>
<td>EE</td>
<td>Meyer</td>
</tr>
<tr>
<td>LCL</td>
<td>SIPL</td>
</tr>
<tr>
<td>CS</td>
<td>Meyer</td>
</tr>
<tr>
<td>SSDL</td>
<td>LCL</td>
</tr>
<tr>
<td>CS</td>
<td>Taub</td>
</tr>
<tr>
<td>STAT</td>
<td>SSDL</td>
</tr>
<tr>
<td>IE</td>
<td>Taub</td>
</tr>
</tbody>
</table>

- A lab belongs to just one faculty (i.e., name is a key for Lab)
- A specific room in a specific building belongs to only one lab
- A lab may have multiple rooms, but all in the same building
Formal Definition

- Let \( S \) be a schema.
- A Functional Dependency (FD) over \( S \) is an expression of the form \( R : U \rightarrow V \), where \( U \) and \( V \) are sets of attributes of \( R \).
- An instance \( I \) over \( S \) satisfies the FD \( R : U \rightarrow V \) if for every two tuples \( t_1 \) and \( t_2 \) of \( R^I \):
  
  \[
  t_1 \text{ and } t_2 \text{ agree on } U \implies t_1 \text{ and } t_2 \text{ agree on } V
  \]

  - By “agree on \( W \)” we mean that \( t_1 \) and \( t_2 \) have the same value in every position that corresponds to an attribute of \( W \).
**Formal Definition**

- Let $S$ be a schema.
- A *Functional Dependency* (FD) over $S$ is an expression of the form $R : U \rightarrow V$, where $U$ and $V$ are sets of attributes of $R$.
- An instance $I$ over $S$ satisfies the FD $R : U \rightarrow V$ if for every two tuples $t_1$ and $t_2$ of $R^I$:
  
  $t_1$ and $t_2$ agree on $U \iff t_1$ and $t_2$ agree on $V$

- By “agree on $W$” we mean that $t_1$ and $t_2$ have the same value in every position that corresponds to an attribute of $W$.

- If $U$ and $V$ cover all the attributes of $R$, then $R : U \rightarrow V$ is a *key constraint* and $U$ is said to be a *key* for $R$.

- As a simplified notation, we write $U$ and $V$ by simply listing their attributes (no set notation).
Example Revisited

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>faculty</td>
</tr>
<tr>
<td>SIPL</td>
<td>EE</td>
</tr>
<tr>
<td>LCL</td>
<td>CS</td>
</tr>
<tr>
<td>SSDL</td>
<td>CS</td>
</tr>
<tr>
<td>STAT</td>
<td>IE</td>
</tr>
</tbody>
</table>
### Example Revisited

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>faculty</td>
</tr>
<tr>
<td>SIPL</td>
<td>EE</td>
</tr>
<tr>
<td>LCL</td>
<td>CS</td>
</tr>
<tr>
<td>SSDL</td>
<td>CS</td>
</tr>
<tr>
<td>STAT</td>
<td>IE</td>
</tr>
</tbody>
</table>

- A lab belongs to just one faculty (i.e., lab is a key for Lab)
### Example Revisited

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>lab</td>
</tr>
<tr>
<td>faculty</td>
<td>building</td>
</tr>
<tr>
<td>SIPL</td>
<td></td>
</tr>
<tr>
<td>EE</td>
<td>SIPL</td>
</tr>
<tr>
<td>LCL</td>
<td>SIPL</td>
</tr>
<tr>
<td>CS</td>
<td>LCL</td>
</tr>
<tr>
<td>SSDL</td>
<td>SSDL</td>
</tr>
<tr>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>STAT</td>
<td></td>
</tr>
<tr>
<td>IE</td>
<td></td>
</tr>
</tbody>
</table>

A lab belongs to just one faculty (i.e., lab is a key for Lab)

Lab : name → faculty
Example Revisited

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>faculty</td>
</tr>
<tr>
<td>SIPL</td>
<td>EE</td>
</tr>
<tr>
<td>LCL</td>
<td>CS</td>
</tr>
<tr>
<td>SSDL</td>
<td>CS</td>
</tr>
<tr>
<td>STAT</td>
<td>IE</td>
</tr>
</tbody>
</table>

- A specific room in a specific building belongs to only one lab
Example Revisited

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>faculty</td>
</tr>
<tr>
<td>SIPL</td>
<td>EE</td>
</tr>
<tr>
<td>LCL</td>
<td>CS</td>
</tr>
<tr>
<td>SSDL</td>
<td>CS</td>
</tr>
<tr>
<td>STAT</td>
<td>IE</td>
</tr>
</tbody>
</table>

- A specific room in a specific building *belongs to only one lab*

LabRoom: building room → lab
### Example Revisited

- **Lab**
  - | name | faculty |
  - | SIPL | EE       |
  - | LCL  | CS       |
  - | SSDL | CS       |
  - | STAT | IE       |

- **LabRoom**
  - | lab  | building | room |
  - | SIPL | Meyer    | 100  |
  - | SIPL | Meyer    | 101  |
  - | LCL  | Taub     | 100  |
  - | SSDL | Taub     | 200  |

- A lab may have multiple rooms, *but all in the same building*
Example Revisited

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>faculty</td>
</tr>
<tr>
<td>SIPL</td>
<td>EE</td>
</tr>
<tr>
<td>LCL</td>
<td>CS</td>
</tr>
<tr>
<td>SSDL</td>
<td>CS</td>
</tr>
<tr>
<td>STAT</td>
<td>IE</td>
</tr>
</tbody>
</table>

- A lab may have multiple rooms, *but all in the same building*

LabRoom : lab → building
There are various formalisms that naturally extend FDs to cross-relation dependencies.

Following are two popular examples:

- **Equality-Generating Dependencies (EGDs)**
- **Denial Constraints (DCs)**
The FD Lab : name → faculty can be phrased in FOL as:

\[ \forall x, y, z [\text{Lab}(x, y) \land \text{Lab}(x, z) \rightarrow y = z] \]

An EGD is an expression of the form:

\[ \forall x [\varphi(x) \rightarrow y_1 = y_2] \]

- \( \varphi(x) \) is a conjunction of atomic formulas
- \( y_1 \) and \( y_2 \) are variables in \( x \)
EGDs

- The FD Lab : name → faculty can be phrased in FOL as

\[ \forall x, y, z \left[ \text{Lab}(x, y) \land \text{Lab}(x, z) \rightarrow y = z \right] \]

- An EGD is an expression of the form

\[ \forall x \left[ \varphi(x) \rightarrow y_1 = y_2 \right] \]

  - \( \varphi(x) \) is a conjunction of atomic formulas
  - \( y_1 \) and \( y_2 \) are variables in \( x \)

- Example:

\[
\text{Lab}(l_1, f_1), \text{Lab}(l_2, f_2), \text{LabRoom}(l_1, b, r_1), \text{LabRoom}(l_2, b, r_2) \\
\rightarrow f_1 = f_2
\]
The FD Lab: name → faculty can be phrased in FOL as

\[ \forall x, y, z \neg (\text{Lab}(x, y) \land \text{Lab}(x, z) \land y \neq z) \]

A DC is an expression of the form

\[ \forall x \neg (\varphi(x) \land \psi(x)) \]

- \(x = (x_1, \ldots, x_n)\) is a sequence of variables
- Each \(\varphi(x)\) is a conjunction of atomic formulas
- Each \(\gamma(x)\) is a conjunction of comparisons between two variables in \(x\) (e.g., \(x_1 \neq x_2, x_1 < x_2, x_1 \geq x_2\), etc.)
### Inclusion Dependencies

**Lab**

<table>
<thead>
<tr>
<th>name</th>
<th>faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIPL</td>
<td>EE</td>
</tr>
<tr>
<td>LCL</td>
<td>CS</td>
</tr>
<tr>
<td>SSDL</td>
<td>CS</td>
</tr>
<tr>
<td>STAT</td>
<td>IE</td>
</tr>
</tbody>
</table>

**LabRoom**

<table>
<thead>
<tr>
<th>lab</th>
<th>building</th>
<th>room</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIPL</td>
<td>Meyer</td>
<td>100</td>
</tr>
<tr>
<td>SIPL</td>
<td>Meyer</td>
<td>101</td>
</tr>
<tr>
<td>LCL</td>
<td>Taub</td>
<td>100</td>
</tr>
<tr>
<td>SSDL</td>
<td>Taub</td>
<td>200</td>
</tr>
</tbody>
</table>
### Inclusion Dependencies

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>lab</td>
</tr>
<tr>
<td>faculty</td>
<td>building</td>
</tr>
<tr>
<td>SIPL</td>
<td>Meyer</td>
</tr>
<tr>
<td>LCL</td>
<td>Taub</td>
</tr>
<tr>
<td>SSDL</td>
<td>100</td>
</tr>
<tr>
<td>STAT</td>
<td>101</td>
</tr>
<tr>
<td>IE</td>
<td>200</td>
</tr>
</tbody>
</table>

Every lab in LabRoom should be listed in the Lab relation (foreign key)
Formal Definition

- Let $S$ be a schema
- An *Inclusion Dependency (IND)* over $S$ is an expression $\delta$ of the form
  \[ R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m] \]
  where:
  - $R$ and $S$ are relation name in $S$
    - $R$ and $S$ may be equal
  - $A_1, \ldots, A_m$ are distinct attributes of $R$
  - $B_1, \ldots, B_m$ are distinct attributes of $S$
Formal Definition

- Let $S$ be a schema
- An *Inclusion Dependency* (IND) over $S$ is an expression $\delta$ of the form
  \[
  R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]
  \]
  where:
  - $R$ and $S$ are relation name in $S$
    - $R$ and $S$ may be equal
  - $A_1, \ldots, A_m$ are distinct attributes of $R$
  - $B_1, \ldots, B_m$ are distinct attributes of $S$
- An instance $I$ over $S$ satisfies $\delta$ if
  \[
  \pi_{A_1, \ldots, A_m}(R^I) \subseteq \pi_{B_1, \ldots, B_m}(S^I)
  \]
Example

Consider the relation $\text{Friend}[\text{person}_1, \text{person}_2]$
Example

Consider the relation $\text{Friend}[\text{person}_1, \text{person}_2]$

$\text{Friend}[\text{person}_1, \text{person}_2] \subseteq \text{Friend}[\text{person}_2, \text{person}_1]$

means that friendship is symmetric
Another Example

<table>
<thead>
<tr>
<th>Lab</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>faculty</td>
</tr>
<tr>
<td>SIPL</td>
<td>EE</td>
</tr>
<tr>
<td>LCL</td>
<td>CS</td>
</tr>
<tr>
<td>SSDL</td>
<td>CS</td>
</tr>
<tr>
<td>STAT</td>
<td>IE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LabRoom</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>lab</td>
<td>building</td>
<td>room</td>
</tr>
<tr>
<td>SIPL</td>
<td>Meyer</td>
<td>100</td>
</tr>
<tr>
<td>SIPL</td>
<td>Meyer</td>
<td>101</td>
</tr>
<tr>
<td>LCL</td>
<td>Taub</td>
<td>100</td>
</tr>
<tr>
<td>SSDL</td>
<td>Taub</td>
<td>200</td>
</tr>
</tbody>
</table>

- Every lab should be listed in the Lab relation (foreign key)
Another Example

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>faculty</td>
</tr>
<tr>
<td>SIPL</td>
<td>EE</td>
</tr>
<tr>
<td>LCL</td>
<td>CS</td>
</tr>
<tr>
<td>SSDL</td>
<td>CS</td>
</tr>
<tr>
<td>STAT</td>
<td>IE</td>
</tr>
</tbody>
</table>

- Every lab should be listed in the Lab relation (foreign key)

LabRoom[lab] ⊆ Lab[name]
Tuple-Generating Dependencies

- The IND $\text{LabRoom}[	ext{lab}] \subseteq \text{Lab}[	ext{name}]$ can be phrased in FOL as

\[
\forall x, y, z \left[ \text{LabRoom}(x, y, z) \rightarrow \exists w \left[ \text{Lab}(x, w) \right] \right]
\]
Tuple-Generating Dependencies

- The IND LabRoom[lab] ⊆ Lab[name] can be phrased in FOL as
  \[ \forall x, y, z [\text{LabRoom}(x, y, z) \rightarrow \exists w [\text{Lab}(x, w)]] \]

- A Tuple-Generating Dependency (TGD) is an expression of the form
  \[ \forall x [\varphi(x) \rightarrow \exists y \psi(x, y)] \]
  where \( \varphi(x) \) and \( \psi(x, y) \) are conjunctions of atomic formulas
Tuple-Generating Dependencies

- The IND $\text{LabRoom}(\text{lab}) \subseteq \text{Lab}(\text{name})$ can be phrased in FOL as
  \[ \forall x, y, z [\text{LabRoom}(x, y, z) \rightarrow \exists w [\text{Lab}(x, w)]] \]

- A Tuple-Generating Dependency (TGD) is an expression of the form
  \[ \forall x [\varphi(x) \rightarrow \exists y \psi(x, y)] \]
where $\varphi(x)$ and $\psi(x, y)$ are conjunctions of atomic formulas

- Example:
  \[ \text{Researcher}(p, l), \text{LabRoom}(l, b, r) \rightarrow \exists r'[\text{PersonRoom}(p, b, r')] \]
Question

Can we express that Friends is transitive with TGDs?

Can we express that Friends has no triangles with TGDs?

End of lecture 2

Databases, Queries, Constraints