Uncertainty in Databases

Lecture 2: Essential Database Foundations
Table of Contents

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

1. Database Model
2. Queries
3. Constraints
4. Complexity
5. 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; it is 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; it is 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; it is 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 $S = (R, \Sigma)$, where $R$ is a set of relation schemas with distinct names, and $\Sigma$ is a set of constraints over $R$. 
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; it is 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 database instance (or just instance) $I$ over 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 $Inst(S)$ the set of all the instances over $S$. 
It is convenient and common to view the database as a logical system (first-order of higher-order logic)

- vocabulary = schema + built-in predicates (e.g., \(<\), \(\geq\), \(=\)), and
- structure = instance
It is convenient and common to view the database as a logical system (first-order of higher-order logic)

- vocabulary = schema + built-in predicates (e.g., <, >, =), and structure = instance

- 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 system (first-order of higher-order logic)
  - vocabulary = schema + built-in predicates (e.g., <, >, =), and structure = instance
- Database queries are viewed as logical formulas $\varphi(x)$ over the database: $Q(I) = \{a | I \models \varphi(a)\}$
- But there are some significant restrictions of the logic:
  - 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)
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.
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
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 $I \in Inst(S)$ into a relation with that heading.
### Example

<table>
<thead>
<tr>
<th>Takes</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>student</td>
<td>cno</td>
</tr>
<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>

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>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 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 $I \not\models Q$
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

- \( Q(I) = \{()\} \) 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, the 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 ($\sigma$)
  - $\sigma_{\phi}(R)$ allowed if $\phi$ 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

<table>
<thead>
<tr>
<th>Takes</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>student</td>
<td>cno</td>
</tr>
<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>

\[
\text{Takes} \times \rho_{cno \rightarrow c} \text{Course}
\]

<table>
<thead>
<tr>
<th>student</th>
<th>cno</th>
<th>c</th>
<th>c</th>
<th>corno</th>
<th>corno</th>
<th>corno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Ahuva</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Alon</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Ahuva</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Example

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

<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>

\[ \begin{array}{ll}
\text{Takes} & \text{Course} \\
\hline
\text{student} & \text{cno} & \text{cno} & \text{cname} \\
\hline
\text{Ahuva} & 1 & 1 & \text{AI} \\
\text{Alon} & 1 & 2 & \text{DB} \\
\text{Ahuva} & 2 & 3 & \text{PL} \\
\end{array} \]
### 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>

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

<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**

### Database Model
- Queries
- Constraints
- Complexity
- References

### Relational Algebra
- SQL
- Conjunctive Queries

#### Example

<table>
<thead>
<tr>
<th>Takes</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>student</td>
<td>cno</td>
</tr>
<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>

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

<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>
SQL (Structured Query Language) is a 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 Takes\((s, x)\) and Course\((x, c)\) hold.

\[
\begin{align*}
\text{Takes} & \\
\text{student} & \text{cno} \\
\hline
\text{Ahuva} & 1 \\
\text{Alon} & 1 \\
\text{Ahuva} & 2 \\
\end{align*}
\begin{align*}
\text{Course} & \\
\text{cno} & \text{cname} \\
\hline
1 & \text{AI} \\
2 & \text{DB} \\
3 & \text{PL} \\
\end{align*}
\]

\[
\text{SELECT S.student, C.cname} \\
\text{FROM Takes T, Course C} \\
\text{WHERE T.cno = C.cno}
\]
Conjunctive Queries

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

\[
Q(x) : \neg \exists y \left( \phi_1(x, y) \land \ldots \land \phi_k(x, y) \right)
\]

where:

- \( x \) and \( y \) are disjoint sequences of variables
- Each \( \phi_i(x, y) \) 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
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:
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:

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

where:

- \textbf{x} and \textbf{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 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:
  \[ Q(x) :\exists y \left[ \varphi_1(x, y) \land \cdots \land \varphi_k(x, y) \right] \]

  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').
  - Every variable in \( x \) occurs at least once on the right hand side.
CQ Terminology and Notation

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

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

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

A variables in \( x \) is called a free or head variable, and a variable in \( y \) is called an existential variable.
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:

\[
Q(x) \leftarrow \left\{\begin{array}{c}
\text{atom} \\
\varphi_1(x, y), \cdots, \varphi_k(x, y)
\end{array}\right\}
\]

A variables in \( x \) is called a \textit{free or head} variable, and a variable in \( y \) is called an \textit{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

<table>
<thead>
<tr>
<th>Takes</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>student</td>
<td>cno</td>
</tr>
<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>

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

\[ Q(s, c) :\neg \text{Takes}(s, x) \land \text{Course}(x, c) \]
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 *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
### 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)
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
### Functional Dependencies

<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., 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
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 \( R^I \):

\[ t_1 \text{ and } t_2 \text{ agree on } U \Rightarrow t_1 \text{ and } t_2 \text{ agree on } V \]

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 $U \Rightarrow t_1$ 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><strong>name</strong></td>
<td><strong>faculty</strong></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>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)

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

Database Model
Queries
Constraints
Complexity
References

Functional Dependencies
Generalizing FDs
Inclusion Dependencies
Tuple-Generating 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 specific room in a specific building belongs to only one lab

LabRoom: building room → 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>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 may have multiple rooms, *but all in the same building*
A lab may have multiple rooms, *but all in the same building*

\[ \text{LabRoom} : \text{lab} \rightarrow \text{building} \]
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 \(\text{Lab} : \text{name} \rightarrow \text{faculty}\) 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 \(x\)
EGDs

- The FD \( \text{Lab}: \text{name} \rightarrow \text{faculty} \) 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 \( x \)

- 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>

Every lab in LabRoom should be listed in the Lab relation (foreign key).
### 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>

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

$$\pi_{A_1,\ldots,A_m}(R^I) \subseteq \pi_{B_1,\ldots,B_m}(S^I)$$
Consider the relation \( \text{Friend}[\text{person}_1, \text{person}_2] \)
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>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)
Another Example

<table>
<thead>
<tr>
<th>Lab</th>
<th>LabRoom</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>lab</td>
</tr>
<tr>
<td>SIPL</td>
<td>SIPL</td>
</tr>
<tr>
<td>LCL</td>
<td>SIPL</td>
</tr>
<tr>
<td>SSDL</td>
<td>LCL</td>
</tr>
<tr>
<td>STAT</td>
<td>SSDL</td>
</tr>
</tbody>
</table>

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

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

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

- 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?*
We consider computational problems that involve one or more of the following components:

- Schema $S$
- A set $\Sigma$ of constraints
- A query $Q$
- A database instance $I$

Common complexity measures designed to distinguish one input from another (e.g., instances are far bigger than schemas/queries)

Combined complexity: everything is given as input

Data complexity: $I$ is given as input, everything else is fixed

Formally, we consider infinitely many computational problems $P_{S,\Sigma,Q}$, one per combination of $S$, $\Sigma$, and $Q$. 


Types of Database Complexity

- We consider computational problems that involve one or more of the following components:
  - Schema $S$
  - A set $\Sigma$ of constraints
  - A query $Q$
  - A database instance $I$

- Common complexity measures designed to distinguish one input from another (e.g., instances are far bigger than schemas/queries)
Types of Database Complexity

- We consider computational problems that involve one or more of the following components:
  - Schema $S$
  - A set $\Sigma$ of constraints
  - A query $Q$
  - A database instance $I$

- Common complexity measures designed to distinguish one input from another (e.g., instances are far bigger than schemas/queries)

- **Combined complexity**: everything is given as input

- **Data complexity**: $I$ is given as input, everything else is fixed
  - Formally, we consider infinitely many computational problems $P_{S,\Sigma,Q}$, one per combination of $S$, $\Sigma$ and $Q$
Example: Complexity of CQ Answering

**Problem Def. (Boolean CQ Evaluation)**

Given a schema $S$, a Boolean CQ $Q$ over $S$ and an instance $I$ over $S$, determine whether $Q(I) = \text{true}$. 

We will show that this problem is NP-complete under combined complexity, by reduction from the Clique problem.
Example: Complexity of CQ Answering

**Problem Def. (Boolean CQ Evaluation)**

Given a schema \( S \), a Boolean CQ \( Q \) over \( S \) and an instance \( I \) over \( S \), determine whether \( Q(I) = \text{true} \).

We will show that this problem is NP-complete under combined complexity, by reduction from the Clique problem.
Example: Complexity of CQ Answering

**Problem Def. (Boolean CQ Evaluation)**

Given a schema $S$, a Boolean CQ $Q$ over $S$ and an instance $I$ over $S$, determine whether $Q(I) = \text{true}$.

We will show that this problem is NP-complete under combined complexity, by reduction from the Clique problem.

**Problem Def. (Clique)**

Given a graph $G = (V, E)$ and a number $k$, determine whether $G$ contains a clique of size $k$, that is, a subset $U$ of $V$ such that $|U| = k$ and every two nodes in $U$ are neighbours.
Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, and $k$, construct:

- $S = \{R_E/2\}$
- $I_G = \{R_E(i, j) \mid \{i, j\} \in E \text{ and } i < j\}$
- $Q_k$ is a CQ with existential variables $X_1, \ldots, X_k$, and an atom $R_E(X_i, X_j)$ for every $i$ and $j$ with $1 \leq i < j \leq k$
Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, and $k$, construct:

- $S = \{R_E/2\}$
- $I_G = \{R_E(i, j) | \{i, j\} \in E \text{ and } i < j\}$
- $Q_k$ is a CQ with existential variables $X_1, \ldots, X_k$, and an atom $R_E(X_i, X_j)$ for every $i$ and $j$ with $1 \leq i < j \leq k$

For example, suppose that $G$ is the following graph:

```
  1  2
 / \ /
3---4
```
Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, and $k$, construct:

- $S = \{R_E/2\}$
- $I_G = \{R_E(i, j) \mid \{i, j\} \in E \text{ and } i < j\}$
- $Q_k$ is a CQ with existential variables $X_1, \ldots, X_k$, and an atom $R_E(X_i, X_j)$ for every $i$ and $j$ with $1 \leq i < j \leq k$

For example, suppose that $G$ is the following graph:

```
1 -- 2
 |   |
3 -- 4
```

$I_G = \begin{array}{c|c}
R_E & \\
\hline
1 & 3 \\
2 & 3 \\
2 & 4 \\
3 & 4 \\
\end{array}$

$Q_3 : \neg R_E(X_1, X_2), R_E(X_1, X_3), R(X_2, X_3)$
The reduction is correct since the following two are equivalent:

1. $G$ has a clique of size at least $k$
2. $Q_k(I_G) = \text{true}$
Correctness

- The reduction is correct since the following two are equivalent:
  1. $G$ has a clique of size at least $k$
  2. $Q_k(I_G) = \text{true}$

- Hence, determining whether $Q(I) = \text{true}$, given $S$, $Q$ and $I$, is NP-hard
  - Membership in NP is straightforward, hence, the problem is NP-complete
Correctness

- The reduction is correct since the following two are equivalent:
  1. $G$ has a clique of size at least $k$
  2. $Q_k(I_G) = \text{true}$

- Hence, determining whether $Q(I) = \text{true}$, given $S$, $Q$ and $I$, is NP-hard
  - Membership in NP is straightforward, hence, the problem is NP-complete

- Note: The schema $S$ does not depend on the input $(G, k)$, but the size of $Q$ is quadratic in $k$
What is the data complexity of answering a query in RA?
What is the data complexity of answering a query in RA?

- We consider the problem $P_{S,Q}$ of computing the answers for a query $Q$ in RA (Relational Algebra) over a given input instance $I$ over $S$. 
What is the data complexity of answering a query in RA?

- We consider the problem $P_{S,Q}$ of computing the answers for a query $Q$ in RA (Relational Algebra) over a given input instance $I$ over $S$.
- The naive way of straightforwardly executing $Q$ runs in polynomial time!
Data Complexity

What is the data complexity of answering a query in RA?

- We consider the problem $P_{S,Q}$ of computing the answers for a query $Q$ in RA (Relational Algebra) over a given input instance $I$ over $S$.
- The naive way of straightforwardly executing $Q$ runs in polynomial time!
- As a special case, CQ evaluation is in polynomial time under data complexity.
  - Note that data complexity is insensitive to the representation of the query.
Other yardsticks of efficiency are often used in database complexity; here are two examples:
Parameterized complexity is between data complexity and combined complexity;

- Query evaluation is *Fixed Parameter Tractable* if it can be evaluated in time $O(f(Q) \cdot p(I))$, where $f$ is any (computable) function of the query in $p(I)$ is polynomial in the size of $I$.
  - Our reduction from clique shows that CQ evaluation is not likely to be FPT.
A query (e.g., CQ) may be required to output an exponential number of results.

Hence, it makes no sense to require evaluation in polynomial time.

Input-output complexity measures the time as a function of both the input and the output.

- Polynomial total time: the running time is polynomial in the combined size of the input and the output.
- Polynomial delay: the answers are produced one by one, where the delay between every two answers is polynomial in the input only.
# Table of Contents

1. Database Model
2. Queries
3. Constraints
4. Complexity
5. References
References I


End of lecture 2

Essential Database Foundations