Database Management Systems
Course 236363

Lecture 6:
Integrity Constraints

Why is it important to model and understand constraints?
- Application coherence/safety
- Efficiency
- Inconsistency management
  - Advanced course 236605
  - Principles of schema design
  - Next lecture

Database Constraints (Dependencies)
- Definition: properties that DB instances should satisfy beyond conforming to the schema structure
- There are various types of constraints, each with its designated
  - Language (how do rules look like?)
  - Semantics (what do rules mean?)
- In this lecture, we will learn constraint languages, discuss their semantics and discuss reasoning over them

Use 1: Constraints for Application Coherence
- The “obvious” application of constraints is software safety: DBMS assures that, whatever app developers/users do, DB always satisfies specified constraints
- Database constraints reduce (but typically not eliminate) responsibility of custom code to verify integrity

Use 2: Constraints for Efficiency
- Knowing that constraints are satisfied can significantly help query planning
  - In addition, joins are commonly via keys; so designated structure/indices can be built

Use 3: Constraints for Handling Inconsistency
- An inconsistent database contains inconsistent (or impossible) information
  - Two students have the same ID
  - A student gets credit for the same course twice
  - A student takes a non-existing course
  - A student gets a grade but missing an assignment
- Modeling: \((I, \Sigma)\) where \(I\) is a database instance and \(\Sigma\) is a set of integrity constraints; alas, \(I\) violates \(\Sigma\)
  - (Slides from “Uncertainty in Databases,” Advanced Topics 236605)
Consistent Query Answering

Function Dependency:
every student gets a unique grade per course

Integrity Constraints:

\[
\text{SELECT student FROM Grades G, Courses C WHERE G.grade } \geq 85 \text{ AND G.course } = \text{C.course AND C.lecturer='Eran'}
\]

Example of Schema Design

Population repeated for every city! Why is it bad?
- Redundancy – we store more bits than needed
- We can get inconsistencies
- We may not be able to store some information (or be forced to use nulls)

Consistent Query Answering

Use 4: Constraints for Schema Design

- Interestingly, the motivation to inventing some popular types of constraints was to define what “good schemas” should avoid!

Normal Forms

In some “normal form”
Functional Dependencies (FDs)

- **Functional Dependency** is the most studied type of database constraint
- Most famous special case: *keys*
  - SQL distinguishes between two types of key constraints: primary key (\(\leq 1\) allowed), and uniqueness (as many as you want)
  - A primary key cannot be NULL, and it typically has a more efficient index (determines tuple physical sorting)

Example: Smartphone Store

<table>
<thead>
<tr>
<th>name</th>
<th>os</th>
<th>disk</th>
<th>price</th>
<th>vendor</th>
<th>headq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Galaxy S6</td>
<td>Android</td>
<td>32</td>
<td>550</td>
<td>Samsung</td>
<td>Suwon, South Korea</td>
</tr>
<tr>
<td>Galaxy S6</td>
<td>Android</td>
<td>64</td>
<td>700</td>
<td>Samsung</td>
<td>Suwon, South Korea</td>
</tr>
<tr>
<td>Galaxy Note 5</td>
<td>Android</td>
<td>32</td>
<td>610</td>
<td>Samsung</td>
<td>Suwon, South Korea</td>
</tr>
<tr>
<td>iPhone 6</td>
<td>iOS</td>
<td>16</td>
<td>595</td>
<td>Apple</td>
<td>Cupertino, CA, USA</td>
</tr>
<tr>
<td>iPhone 6</td>
<td>iOS</td>
<td>64</td>
<td>700</td>
<td>Apple</td>
<td>Cupertino, CA, USA</td>
</tr>
<tr>
<td>Nexus 6p</td>
<td>Android</td>
<td>32</td>
<td>615</td>
<td>Google</td>
<td>Mountain View, CA, USA</td>
</tr>
<tr>
<td>Nexus 6p</td>
<td>Android</td>
<td>128</td>
<td>900</td>
<td>Google</td>
<td>Mountain View, CA, USA</td>
</tr>
</tbody>
</table>

The attribute set \(\text{state}, \text{zip}\) determines the attribute \(\text{city}\)

The attribute set \(\text{street}\) determines the attribute \(\text{state}\)

Example: US Addresses

<table>
<thead>
<tr>
<th>name</th>
<th>state</th>
<th>city</th>
<th>street</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>White House</td>
<td>DC</td>
<td>Washington</td>
<td>1600 Pennsylvania Ave NW</td>
<td>20500</td>
</tr>
<tr>
<td>Wall Street</td>
<td>NY</td>
<td>New York</td>
<td>11 Wall St.</td>
<td>10005</td>
</tr>
<tr>
<td>Empire State B</td>
<td>NY</td>
<td>New York</td>
<td>290 Fifth Avenue</td>
<td>10118</td>
</tr>
<tr>
<td>Hollywood Sign</td>
<td>CA</td>
<td>Los Angeles</td>
<td>4059 N Lee Dr.</td>
<td>90068</td>
</tr>
</tbody>
</table>

The attribute set \(\text{state}, \text{city}\) determines the attribute \(\text{street}\)

The attribute set \(\text{zip}\) determines the attribute \(\text{state}\)

Outline

- Introduction
  - Functional Dependencies
    - Definitions
    - Armstrong’s Axioms
    - Algorithms
  - Other Types of Constraints
    - Multivalued Dependencies
    - Inclusion Dependencies
  - Anti-Monotonicity
Notation

- In the case of FDs, we restrict to a single relation schema
- We write an attribute set as a sequence of attribute names (not set notation {…})
  - name, os, disk, price
- An attribute set is denoted by a capital letter from the end of the Latin alphabet
  - X, Y, Z
- Concatenation stands for union
  - XY stands for X U Y
  - XX = X
  - XY = YX = Y Y X X

Functional Dependency

- From now on, we will assume the schema s without mentioning it explicitly
- A Functional Dependency (FD) is an expression X → Y where X and Y are sets of attributes
  - Examples:
    - name, disk → price, os, vendor
    - name → os, vendor
    - country, city, street → zip
    - zip → country

Semantics of an FD

- A relation R satisfies the FD X → Y if:
  - for all tuples t and u in R, if t and u agree on X then they also agree on Y
- Mathematically:
  - t[X] = u[X] ⇒ t[Y] = u[Y]
- A relation R satisfies a set F of FDs if R satisfies every FD in F

Trivial FDs

- An FD over is trivial if it holds in every relation (over the underlying schema)
- PROPOSITION: An FD X → Y is trivial if and only if Y ⊆ X
  - Proof:
    - The “if” direction is straightforward
    - For the “only if” direction, consider the instance I that contains two tuples that agree precisely on the attributes of X; if y3X then we get a violation of X → Y

Problem: No Unique Representation...

Can you express an FD stating that a column must contain a constant value (same across all tuples)?

<table>
<thead>
<tr>
<th>Faculty</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>AI</td>
</tr>
<tr>
<td>CS</td>
<td>DB</td>
</tr>
<tr>
<td>CS</td>
<td>PL</td>
</tr>
<tr>
<td>CS</td>
<td>OS</td>
</tr>
</tbody>
</table>

Answer: yes! S → faculty

They all mean precisely the same thing!
Entailed (Implied) FDs

- Let $F$ be a set of FDs
- An FD $X \rightarrow Y$ is entailed (or implied) by $F$ if for every relation $R$ over the schema, if $R$ satisfies $F$ then $R$ satisfies $X \rightarrow Y$
- Notation: $F \models X \rightarrow Y$

Examples of Entailment

- $F = \{\text{name} \rightarrow \text{vendor}, \text{vendor} \rightarrow \text{headq}\}$
  - $F \models \text{name} \rightarrow \text{headq}$
  - $F \models \text{name}, \text{vendor} \rightarrow \text{headq}$
  - $F \models \text{name}, \text{vendor} \rightarrow \text{vendor}$

Closure of an FD Set

- Let $F$ be a set of FDs
- The closure of $F$, denoted $F^*$, is the set of all the FDs entailed by $F$
- Observations:
  - $F \subseteq F^*$
  - $(F^*)^* = F^*$
  - $F^*$ contains every trivial FD

Closure of an Attribute Set

- Let $F$ be a set of FDs, and let $X$ be a set of attributes
- The closure of $X$ under $F$, denoted $X^*$ is the set of all the attributes $A$ such that $X \rightarrow A$ is implied by $F$
- Note: notation assumes that $F$ is known from the context

Observations

- For all $F$, $X$, $Y$:
  - $X^* = \{A \mid F \models X \rightarrow A\} = \{A \mid (X \rightarrow A) \in F^*\}$
  - $X \subseteq X^*$
  - $(X^*)^* = X^*$
  - If $X \subseteq Y$ then $X^* \subseteq Y^*$

Minimal Cover

- Let $F$ be a set of FDs
- A minimal cover (or minimal basis) for $F$ is a set $G$ of FDs with the following properties:
  - $G^* = F^*$
  - FDs in $G$ have a single attribute on the right hand side; that is, they have the form $X \rightarrow A$
  - All FDs are required: no FD $X \rightarrow A$ in $G$ is such that $G \setminus \{X \rightarrow A\} \models X \rightarrow A$
  - All attributes are required: no FD $XB \rightarrow A$ in $G$ is such that $G \setminus \{X \rightarrow A\} \models X \rightarrow A$
Example of Minimal Covers

\{(A \rightarrow BC, B \rightarrow AC, C \rightarrow AB, AB \rightarrow C, AC \rightarrow B)\}

- Minimal cover 1:
  \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}
- Minimal cover 2:
  \{C \rightarrow B, B \rightarrow A, A \rightarrow C\}
- Minimal cover 3:
  \{A \rightarrow B, B \rightarrow A, A \rightarrow C, C \rightarrow A\}
- Any more?
- In what sense is a minimal cover “minimal”?

Keys and Superkeys

- Assume \(s\) is our underlying relation schema
- A superkey is a set \(X\) of attributes such that \(X\) contains every attribute in \(s\)
- A key is a superkey \(X\) that does not contain any other superkey
  - That is, if \(Y \subseteq X\) then \(Y\) is not a superkey
- Later, we will see an efficient algorithm for finding a key

Outline

- Introduction
- Functional Dependencies
  - Definitions
  - Armstrong’s Axioms
  - Algorithms
- Other Types of Constraints
  - Multivalued Dependencies
  - Inclusion Dependencies
  - Anti-Monotonicity

Mechanically Proving FD Entailment

- Conceptually, to prove \(F \vdash X \rightarrow Y\) we need to consider every possible relation that satisfies \(F\), and check whether \(X \rightarrow Y\) holds
- But so far, for each such a proof we have found a finite argument
- Can we detect entailment algorithmically?
  - Yes! Using a proof system
    - Later, we will see an efficient (not just computable) proof procedure

Proof System

- A proof system is a collection of rules/patterns of the form “if you know \(x\) then infer \(y\)”
- A proof of a statement \(\text{stmt}\) is a sequence of rule applications (each adding new facts), starting with what is known and ending with \(\text{stmt}\)
- A proof system is:
  - Sound if every provable fact is correct
  - Complete if every correct fact is provable

Proof System for FDs

- Think of proof systems for inferring FDs from a known set of FDs... (“if you know some FDs, then you can infer a new FD”)
  - Can you give easy example of a sound (not necessarily complete) proof system?
  - Can you give an easy example of a complete (not necessarily sound) proof system?
Armstrong’s Axioms

<table>
<thead>
<tr>
<th>Property</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflexivity</td>
<td>If ( Y \subseteq X ) then ( X \rightarrow Y )</td>
</tr>
<tr>
<td>Augmentation</td>
<td>If ( X \rightarrow Y ) then ( XZ \rightarrow YZ )</td>
</tr>
<tr>
<td>Transitivity</td>
<td>If ( X \rightarrow Y ) and ( Y \rightarrow Z ) then ( X \rightarrow Z )</td>
</tr>
</tbody>
</table>

Provable Rules

Armstrong’s Axioms

<table>
<thead>
<tr>
<th>Property</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflexivity</td>
<td>If ( Y \subseteq X ) then ( X \rightarrow Y )</td>
</tr>
<tr>
<td>Augmentation</td>
<td>If ( X \rightarrow Y ) then ( XZ \rightarrow YZ )</td>
</tr>
<tr>
<td>Transitivity</td>
<td>If ( X \rightarrow Y ) and ( Y \rightarrow Z ) then ( X \rightarrow Z )</td>
</tr>
</tbody>
</table>

- Union: If \( X \rightarrow Y \) and \( X \rightarrow Z \) then \( X \rightarrow YZ \)
  - \( XZ \rightarrow YZ \) (augmentation)
  - \( X \rightarrow X \) (reflexivity)
  - \( XX \rightarrow XZ \) (augmentation); same as \( X \rightarrow XZ \)
  - \( X \rightarrow YZ \) (transitivity)
- Decomposition: If \( X \rightarrow YZ \) then \( X \rightarrow Y \)

Entailment vs. Provable

- Recall: \( F \models X \rightarrow Y \) denotes that \( X \rightarrow Y \) is entailed from \( F \)
- By \( F \models X \rightarrow Y \) we denote that \( X \rightarrow Y \) is provable from \( F \) using Armstrong’s axioms
- Example: \( F = \{ A \rightarrow B, BC \rightarrow D \} \)
  - Clearly, \( F \models AC \rightarrow D \) is true
  - But is \( F \models AC \rightarrow D \) true?
    - If so, a proof is required

Soundness and Completeness

<table>
<thead>
<tr>
<th>Property</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflexivity</td>
<td>If ( Y \subseteq X ) then ( X \rightarrow Y )</td>
</tr>
<tr>
<td>Augmentation</td>
<td>If ( X \rightarrow Y ) then ( XZ \rightarrow YZ )</td>
</tr>
<tr>
<td>Transitivity</td>
<td>If ( X \rightarrow Y ) and ( Y \rightarrow Z ) then ( X \rightarrow Z )</td>
</tr>
</tbody>
</table>

**THEOREM:** Armstrong’s axioms form a sound and complete proof system for FDs

- That is, every provable FD is correct, and every correct FD is provable
- That is, for all \( F, X, Y \) we have
  \[ F \models X \rightarrow Y \quad \iff \quad F \vdash X \rightarrow Y \]
- Hence, Armstrong’s axioms fully capture the implication dependencies among FDs

Proof

- We need to prove two things:
  1. Soundness
  2. Completeness

- Proving soundness is straightforward: the axioms are correct, so derived facts are correct, ... so end conclusions are correct
- Proving completeness is more involved

Proof of Completeness (1)

- We assume that \( F \models X \rightarrow Y \)
- We need to prove that \( F \vdash X \rightarrow Y \)

- Strategy:
  - Denote by \( X^* \) the set \( \{ A \mid F \models X \rightarrow A \} \)
  - We will show that \( Y \subseteq X^* \)
  - Then \( X \rightarrow Y \) is proved by repeatedly using union
    - Recall – we showed that union is provable
    - ... and we are done
Proof of Completeness (1)

- We assume that $F \vdash X \rightarrow Y$
- We need to prove that $Y \subseteq X^* = \{A | F \vdash X \rightarrow A\}$
- Let $X^*$ be the set of attributes that are not in $X$.
- Construct a relation $R$ with two tuples $t$ and $u$:
  - $t(X^*) = u(X^*) = (0, \ldots, 0)$
  - $t(X) = u(X) = (1, \ldots, 1)$
  - $u(X^*) = (2, \ldots, 2)$

Proof of Completeness (2)

- **CLAIM 1:** $X \subseteq X^*$
  - Proof: apply reflexivity to each $A \in X$

Proof of Completeness (3)

- Suppose, by way of contradiction, that $Y \not\subseteq X^*$
- **CLAIM 2:** $R$ violates $X \rightarrow Y$
  - Proof:
    - $t$ and $s$ agree on $X$, due to **CLAIM 1**
    - $t$ and $s$ disagree on $Y$, since $Y \cap X^* = \emptyset$

Proof of Completeness (4)

- Recall: we assumed that $F \vdash X \rightarrow Y$
- We have so far:
  - **CLAIM 2:** $R$ violates $X \rightarrow Y$
  - **CLAIM 3:** $R$ satisfies $F$
  - This is a contradiction to $F \vdash X \rightarrow Y$
- As required \( \square \)
Computational Problems

Closure Computation

Given:

• A set of FDs
• A set of attributes

Goal:

Compute \( \sigma' \)

Entailment Testing

Given:

• A set of FDs
• An FD \( \sigma \rightarrow \tau \)

Goal:

Determine whether \( \sigma \models \sigma \rightarrow \tau \)

Key Generation

Given:

• A set of FDs

Goal:

Find a key

Equivalence Testing

Given:

• A set of FDs
• A set of FDs

Goal:

Determine whether \( \sigma = \sigma' \)

Correctness and Running Time

- The proof of correctness is very similar to the proof of soundness & completeness of Armstrong's axioms (omitted)
- Running time:
  - Suppose that \( R \) contains \( n \) attributes
  - Let \( m \) be the total # of attribute occurrences in \( F \)
  - With reasonable data structures, \( O(pm) \) time
  - Can be improved to run in time \( O(|X|+m) \)
    - (Beeri & Bernstein, 1979)

Implication Testing

Given:

• A set of FDs

Goal:

Determine whether \( \sigma \models \sigma \rightarrow \tau \)

IsImplicated \( (X, Y, F) \) { 
  \[ \text{if } (Y \subseteq \text{Closure}(X, F)) \text{ return true} \]
  \[ \text{else return false} \]
}

Equivalence Testing

Given:

• A set of FDs
• A set of FDs

Goal:

Determine whether \( \sigma = \sigma' \)

IsEquiv \( (\sigma, \sigma') \) { 
  \[ \text{for all } X \rightarrow Y \text{ in } F \]
  \[ \text{if } (\text{IsImplicated}(X, Y, \sigma)) \text{ return false} \]
  \[ \text{for all } X \rightarrow Y \text{ in } G \]
  \[ \text{if } (\text{IsImplicated}(X, Y, F)) \text{ return false} \]
  \[ \text{return true} \]
}

Key Generation

Given:

• A set of FDs

Goal:

Find a key

FindKey \( F, R(A_1, \ldots, A_n) \) { 
  \[ K = \{ A_1, \ldots, A_n \} \]
  \[ \text{for } i = 1, \ldots, n \{ \]
  \[ \text{if } A_i \in \text{Closure}(K \setminus \{A_i\}, F) \]
  \[ \text{K} := K \setminus \{A_i\} \]
  \[ \text{return K} \]
}

Example:

\( F = \{ AB \rightarrow C, A \rightarrow B, BC \rightarrow D, CE \rightarrow F \} \)

\( R(A,B,C) \)

FindKey \( F, R(A,B,C) \) { 
  \[ K = \{ A_1, A_2 \} \]
  \[ \text{for } i = 1, \ldots, 3 \{ \]
  \[ \text{if } A_i \in \text{Closure}(K \setminus \{A_i\}, F) \]
  \[ \text{K} := K \setminus \{A_i\} \]
  \[ \text{return K} \]
}

Example:

\( F = \{ AB \rightarrow C, A \rightarrow B, BC \rightarrow D, CE \rightarrow F \} \)

\( R(A,B,C) \)

FindKey \( F, R(A,B,C) \) { 
  \[ K = \{ A_1, A_2 \} \]
  \[ \text{for } i = 1, \ldots, 3 \{ \]
  \[ \text{if } A_i \in \text{Closure}(K \setminus \{A_i\}, F) \]
  \[ \text{K} := K \setminus \{A_i\} \]
  \[ \text{return K} \]
}
Proof of Correctness (1)

• **CLAIM 1:** Throughout the execution, $K$ is always a superkey
  – Proof: Induction on iteration #
  • Basis: Initial $K$ contains all attributes
  • Inductive step: If $A \subseteq (K \setminus \{A\})^*$ then $\forall \subseteq (K \setminus \{A\})^*$
    and then $(A_1, \ldots, A_n) \subseteq \{K \setminus \{A\}^* \subseteq (K \setminus \{A\})^*

Proof of Correctness (2)

• Let $Q$ be the returned $K$
  • **CLAIM 2:** $Q$ is minimal
    – Proof: by way of contradiction
    • Suppose that $Q' \subseteq Q$ is a superkey, and let $A \not\subseteq Q' \subseteq Q$
    • Then $Q \setminus \{A\}$ is a superkey (why?)
    • Consider the $i$th iteration: we have $Q \subseteq K$ (since we only delete things from $K$), and so, $Q \setminus \{A\} \subseteq K\setminus \{A\}$
    • But then, $Q \setminus \{A\}$ is a superkey, and so $K \setminus \{A\}$ is a superkey, and in particular $A \subseteq (K \setminus \{A\})^*$
    • So $A$ should have been removed!

Outline

- Introduction
- Functional Dependencies
  - Definitions
  - Armstrong’s Axioms
  - Algorithms
- Other Types of Constraints
  - Multivalued Dependencies
  - Inclusion Dependencies
  - Anti-Monotonicity

Additional Types of Constraints

- So far we have been looking at functional dependencies, and the special cases of superkeys and keys
- Next, we consider two additional types:
  – Multivalued Dependency (MVD)
  – Inclusion Dependency (IND)

Outline

- Introduction
- Functional Dependencies
  - Definitions
  - Armstrong’s Axioms
  - Algorithms
- Other Types of Constraints
  - Multivalued Dependencies
  - Inclusion Dependencies
  - Anti-Monotonicity

Example of Multivalued Dependency

<table>
<thead>
<tr>
<th>Student</th>
<th>Faculty</th>
<th>Phone</th>
<th>Course</th>
<th>Lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>CS</td>
<td>04-111-1111</td>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>052-111-1111</td>
<td>PL</td>
<td>Keren</td>
</tr>
<tr>
<td>Amir</td>
<td>EE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Shaul</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>04-333-3333</td>
<td>AI</td>
<td>Shaul</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>054-333-3333</td>
<td>AI</td>
<td>Shaul</td>
</tr>
</tbody>
</table>

Why is this table “badly” designed?

Are there any FDs?
**Multivalued Dependency**

- Let $s$ be a relation schema.
- A multivalued dependency (MVD) has the form $X\rightarrow Y$ where $X$ and $Y$ are disjoint sets of attributes.
- A relation $R$ satisfies $X\rightarrow Y$ if:
  - Informally: for every two tuples that agree on $X$, swapping their $Y$ component doesn’t change $R$.
  - For every tuples $t_1$ and $t_2$, with $t_1[X]=t_2[X]$ there exists a tuple $t_3$, with:
    - $t_3[X]=t_1[X]=t_2[X]$.
    - $t_3[Y]=t_1[Y]=t_2[Y]$.

**Some Properties (Exercise / Assignment)**

- $X\rightarrow Y$ implies $X\rightarrow Y$.
- If $X\rightarrow Y$ then $X\rightarrow s’(XY)$.
- An MVD $X\rightarrow Y$ is trivial (always holds) if and only if $Y=\emptyset$ or $Y=s’X$.
- If $X$, $Y$, $Z$ are pairwise disjoint, then $X\rightarrow Y$ and $Y\rightarrow Z$ imply $X\rightarrow Z$.

**Example of Inclusion Dependencies**

<table>
<thead>
<tr>
<th>Student</th>
<th>Faculty</th>
<th>Posting</th>
<th>Likes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>CS</td>
<td>23</td>
<td>Alma</td>
</tr>
<tr>
<td>Amir</td>
<td>CS</td>
<td>45</td>
<td>Amir</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>76</td>
<td>Ahuva</td>
</tr>
<tr>
<td></td>
<td></td>
<td>79</td>
<td>Ahuva</td>
</tr>
</tbody>
</table>

**Definition of an Inclusion Constraint**

- Let $s$ be a relational schema.
- Recall: $s$ consists of several relation schemas.
- An Inclusion Dependency (IND) has the following form $R(A_1,...,A_m) \subseteq s(B_1,...,B_n)$.

**Outline**

- Introduction
- Functional Dependencies
  - Definitions
  - Armstrong’s Axioms
  - Algorithms
- Other Types of Constraints
  - Multivalued Dependencies
    - Inclusion Dependencies
  - Anti-Monotonicity

**Any Other MVDs?**

<table>
<thead>
<tr>
<th>student</th>
<th>faculty</th>
<th>phone</th>
<th>course</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>CS</td>
<td>04-111-1111</td>
<td>PL</td>
<td>Enan</td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>052-111-1111</td>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>052-111-1111</td>
<td>PL</td>
<td>Keren</td>
</tr>
<tr>
<td>Amir</td>
<td>EE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Enan</td>
</tr>
<tr>
<td>Amir</td>
<td>EE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>04-333-3333</td>
<td>AJ</td>
<td>Shaul</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>054-333-3333</td>
<td>AJ</td>
<td>Shaul</td>
</tr>
</tbody>
</table>

**StudentGrant**

- A prof. receives a grant for a student only if she advises that student.

<table>
<thead>
<tr>
<th>name</th>
<th>Faculty</th>
<th>student</th>
<th>posted</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>CS</td>
<td>Anna</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Amir</td>
<td>CS</td>
<td>Ahmed</td>
<td>1500</td>
<td>1500</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Examples

• What is the meaning of the following IND?
  Grad\(\text{name}\) \(\subseteq\) StudentGrant\(\text{student}\)

• What does the following mean about the binary relation \(R(A,B)\):
  \(R(A,B) \subseteq R(B,A)\)

Sounds and Complete System for INDs

• Like FDs, INDs have a simple sound and complete proof system (proof uncovered):
  – Reflexivity: \(R[X] \subseteq R[X]\)
  – Projection: If \(R[A_1,...,A_m] \subseteq S[B_1,...,B_n]\) then for every sequence \(i_1,...,i_k\) of distinct indices in \(\{1,...,m\}\) we have \(R[A_{i_1},...,A_{i_k}] \subseteq S[B_{i_1},...,B_{i_k}]\)
  – Transitivity: If \(R[X] \subseteq S[Y]\) and \(S[X] \subseteq T[Z]\) then \(R[X] \subseteq T[Z]\)

Outline

• Introduction
• Functional Dependencies
  § Definitions
  § Armstrong’s Axioms
  § Algorithms
• Other Types of Constraints
  § Multivalued Dependencies
  § Inclusion Dependencies
  ▶ Anti-Monotonicity

Anti-Monotonic Constraints

• Let \(S\) be a database schema
• Recall: \(I \subseteq J\) if for every relation name, the corresponding relation in \(I\) is a subset of the corresponding relation in \(J\)
• A constraint \(C\) (over \(S\)) is **monotonic** if for all instances \(I\) and \(J\) where \(I \subseteq J\), if \(I\) satisfies \(C\) then \(J\) satisfies \(C\)
• A constraint \(C\) is **anti-monotonic** if for all instances \(I\) and \(J\) where \(I \subseteq J\), if \(J\) satisfies \(C\) then \(I\) satisfies \(C\)

Which is Monotonic? Anti-Monotonic?

• An FD **No / Yes**
• An MVD **No / No**
• An IND **No / No**