Lecture 6: Integrity Constraints
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
Why is it important to model and understand constraints?

• Application coherence/safety
• Efficiency
• Inconsistency management
  ▪ Advanced course 236605
• Principles of schema design
  ▪ Next lecture
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.

\[
\begin{align*}
R(A, B) & \bowtie S(B, C) \bowtie T(C, D) \leq 1M \\
R(A, B) & \bowtie S(B, C) \bowtie T(C, D) \leq 1000 \\
\end{align*}
\]

- 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

### Grades

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>PL</td>
<td>90</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
<td>86</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
<td>81</td>
</tr>
</tbody>
</table>

### Courses

<table>
<thead>
<tr>
<th>course</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>DC</td>
<td>Keren</td>
</tr>
</tbody>
</table>

#### Database $D$

**Functional Dependency:**

Every student gets a unique grade per course.

**Integrity Constraints $\Sigma$**

We can’t always enforce. Why?

```sql
SELECT student
FROM Grades G, Courses C
WHERE G.grade >= 85 AND
  G.course = C.course AND
  C.lecturer = 'Eran'
```

Ahuva

Alon

?
Consistent Query Answering

### Grades

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>PL</td>
<td>90</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
<td>86</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
<td>81</td>
</tr>
</tbody>
</table>

### Courses

<table>
<thead>
<tr>
<th>course</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>DC</td>
<td>Keren</td>
</tr>
</tbody>
</table>

**Database D**

**Functional Dependency:**
every student gets a unique grade per course

**Integrity Constraints** $\Sigma$

SELECT student FROM Grades G, Courses C WHERE G.grade >= 87 AND G.course = C.course AND C.lecturer = 'Eran'

- Ahuva
- Alon
**Consistent Query Answering**

### Database $D$

#### Grades

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahuva</td>
<td>PL</td>
<td>90</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
<td>86</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
<td>81</td>
</tr>
</tbody>
</table>

#### Courses

<table>
<thead>
<tr>
<th>course</th>
<th>lecturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>DC</td>
<td>Keren</td>
</tr>
</tbody>
</table>

**Functional Dependency:**
every student gets a unique grade per course

**Integrity Constraints $\Sigma$**

```
SELECT student
FROM Grades G, Courses C
WHERE G.grade >= 80 AND
    G.course = C.course AND
    C.lecturer='Eran'
```
• Interestingly, the motivation to inventing some popular types of constraints was to define what “good schemas” should avoid!
Example of Schema Design

### Embassy

<table>
<thead>
<tr>
<th>country</th>
<th>host</th>
<th>city</th>
<th>cityPopulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Israel</td>
<td>Tel Aviv</td>
<td>400,000</td>
</tr>
<tr>
<td>USA</td>
<td>Israel</td>
<td>Tel Aviv</td>
<td>400,000</td>
</tr>
<tr>
<td>Israel</td>
<td>France</td>
<td>Paris</td>
<td>2,200,000</td>
</tr>
<tr>
<td>USA</td>
<td>France</td>
<td>Paris</td>
<td>2,200,000</td>
</tr>
</tbody>
</table>

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 used nulls)

### Studies

<table>
<thead>
<tr>
<th>student</th>
<th>course</th>
<th>credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>DB</td>
<td>3</td>
</tr>
<tr>
<td>Alma</td>
<td>PL</td>
<td>2</td>
</tr>
<tr>
<td>Avia</td>
<td>DB</td>
<td>3</td>
</tr>
<tr>
<td>Amir</td>
<td>DB</td>
<td>3</td>
</tr>
<tr>
<td>Amir</td>
<td>PL</td>
<td>2</td>
</tr>
</tbody>
</table>
### Normal Forms

#### Embassy

<table>
<thead>
<tr>
<th>country</th>
<th>host</th>
<th>city</th>
<th>cityPopulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Israel</td>
<td>Tel Aviv</td>
<td>400,000</td>
</tr>
<tr>
<td>USA</td>
<td>Israel</td>
<td>Tel Aviv</td>
<td>400,000</td>
</tr>
<tr>
<td>Israel</td>
<td>France</td>
<td>Paris</td>
<td>2,200,000</td>
</tr>
<tr>
<td>USA</td>
<td>France</td>
<td>Paris</td>
<td>2,200,000</td>
</tr>
</tbody>
</table>

### CountryCity

<table>
<thead>
<tr>
<th>country</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>Israel</td>
<td>Tel Aviv</td>
</tr>
<tr>
<td>France</td>
<td>Paris</td>
</tr>
<tr>
<td>USA</td>
<td>NYC</td>
</tr>
<tr>
<td>UK</td>
<td>London</td>
</tr>
</tbody>
</table>

### CityPopulation

<table>
<thead>
<tr>
<th>city</th>
<th>population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tel Aviv</td>
<td>400,000</td>
</tr>
<tr>
<td>Paris</td>
<td>2,200,000</td>
</tr>
<tr>
<td>NYC</td>
<td>8,400,000</td>
</tr>
<tr>
<td>London</td>
<td>8,500,000</td>
</tr>
</tbody>
</table>

### In some “formal form”

<table>
<thead>
<tr>
<th>country</th>
<th>host</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Israel</td>
<td>Tel Aviv</td>
</tr>
<tr>
<td>USA</td>
<td>Israel</td>
<td>Tel Aviv</td>
</tr>
<tr>
<td>Israel</td>
<td>France</td>
<td>Paris</td>
</tr>
<tr>
<td>USA</td>
<td>France</td>
<td>Paris</td>
</tr>
</tbody>
</table>
## Another Bad Schema

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

• Introduction

• Functional Dependencies
  ▪ Definitions
  ▪ Armstrong’s Axioms
  ▪ Algorithms

• Other Types of Constraints
  ▪ Multivalued Dependencies
  ▪ Inclusion Dependencies

• Anti-Monotonicity
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)
## Smartphone Store

### Smartphone

<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>630</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>128</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>635</td>
<td>Google</td>
<td>MV, CA, USA</td>
</tr>
<tr>
<td>Nexus 6p</td>
<td>Android</td>
<td>128</td>
<td>900</td>
<td>Google</td>
<td>MV, CA, USA</td>
</tr>
</tbody>
</table>

The attribute set **name** determines the attribute **os**.

The attribute set **disk** determines the attribute **price**.

The attribute set **name** determines the attribute **vendor**.

The attribute set **vendor** determines the attribute **headq**.
### US Locations

<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>350 Fifth Avenue</td>
<td>10118</td>
</tr>
<tr>
<td>Hollywood Sign</td>
<td>CA</td>
<td>Los Angeles</td>
<td>4059 Mt Lee Dr.</td>
<td>90068</td>
</tr>
</tbody>
</table>

The attribute set \( \text{state} \) \( \text{city} \) \( \text{street} \) \( \text{zip} \) determines the attribute \( \text{zip} \) \( \text{street} \) \( \text{city} \) \( \text{state} \).
• 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 consider *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 XUY
  – XX = X
  – XY = YX = YYXX
From now on, we will assume the schema $s$ without mentioning it explicitly.

A *Functional Dependency* (FD) is an expression $X \rightarrow Y$ where $X$ and $Y$ are sets of attributes.

- Examples:
  - $\text{name, disk} \rightarrow \text{price, os, vendor}$
  - $\text{name} \rightarrow \text{os, vendor}$
  - $\text{country, city, street} \rightarrow \text{zip}$
  - $\text{zip} \rightarrow \text{country}$
Semantics of an FD

• A relation $R$ satisfies the FD $X \rightarrow Y$ if:
  for all tuples $t$ and $u$ in $R$, if $t$ and $u$ agree on $X$ then they also agree on $Y$

• Mathematically:
  $$t[X] = u[X] \implies t[Y] = u[Y]$$

• 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 $X \rightarrow Y$ is trivial if and only if $Y \subseteq X$

– 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 $Y \not\subseteq X$ then we get a violation of $X \rightarrow Y$
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>
Problem: No Unique Representation...

**Faculty**

<table>
<thead>
<tr>
<th>symbol</th>
<th>name</th>
<th>dean</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>Computer Science</td>
<td>Irad Yavneh</td>
</tr>
<tr>
<td>EE</td>
<td>Electrical Engineering</td>
<td>Ariel Orda</td>
</tr>
<tr>
<td>IE</td>
<td>Industrial Engineering</td>
<td>Avishai Mandelbaum</td>
</tr>
</tbody>
</table>

- $F_1 = \{\text{symbol} \rightarrow \text{name}, \text{name} \rightarrow \text{symbol}, \text{dean} \rightarrow \text{name}, \text{symbol}\}$
- $F_2 = \{\text{symbol} \rightarrow \text{name}, \text{name} \rightarrow \text{dean}, \text{dean} \rightarrow \text{symbol}\}$
- $F_3 = \{\text{symbol} \rightarrow \text{name}, \text{name} \rightarrow \text{symbol}, \text{dean} \rightarrow \text{symbol}, \text{symbol} \rightarrow \text{dean}\}$

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

• \( F = \{\text{name} \rightarrow \text{vendor}, \text{vendor} \rightarrow \text{headq}\} \)
  - \( F \models \text{name} \rightarrow \text{headq} \)
  - \( F \models \text{name, vendor} \rightarrow \text{headq} \)
  - \( F \models \text{name, vendor} \rightarrow \text{vendor} \)

• \( F = \{\text{A} \rightarrow \text{B}, \text{B} \rightarrow \text{C}, \text{C} \rightarrow \text{A}\} \)
  - \( F \models \text{A} \rightarrow \text{A} \)
  - \( F \models \text{A} \rightarrow \text{B} \)
  - \( F \models \text{A} \rightarrow \text{C} \)
  - \( F \models \text{A} \rightarrow \text{ABC} \)
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 $F$

$$F^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}$$

• Observations:
  
  – $F \subseteq F^+$
  
  – $(F^+)^+ = F^+$
  
  – $F^+$ contains every trivial FD
• 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.
• 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 $G \{X \rightarrow A\} \vdash X \rightarrow A$
  - All attributes are required: no FD $XB \rightarrow A$ in $G$ is such that $G \vdash X \rightarrow A$
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”? 
• 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 $F \models X \rightarrow Y$ we need to consider every possible relation that satisfies $F$, and check whether $X \rightarrow Y$ holds.

• But so far, for each such 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.
A proof system is a collection of rules/patterns of the form “if you know x then infer y”

A proof of a statement stmt is a sequence of rule applications (each adding new facts), starting with what is known and ending with 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

**Reflexivity:** If $Y \subseteq X$ then $X \rightarrow Y$

**Augmentation:** If $X \rightarrow Y$ then $XZ \rightarrow YZ$

**Transitivity:** If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$
RR(C, T, H, R, S, G)

\[ F = \{ C \rightarrow T, HR \rightarrow C, HT \rightarrow R, CS \rightarrow G, HS \rightarrow R \} \]

1. \( HS \rightarrow R \in F \)
2. \( HS \rightarrow HR \quad 1 \quad (A2) \)
3. \( HS \rightarrow C \quad 2 \quad HR \rightarrow C, \quad (A3) \)
4. \( HS \rightarrow T \quad 3, \quad C \rightarrow T, \quad (A3) \)
5. \( HS \rightarrow CS \quad 3, \quad (A2) \)
6. \( HS \rightarrow G \quad 5, \quad CS \rightarrow G, \quad (A3) \)

Conclusion: \( F \models HS \rightarrow G \)

And, HS is a key
### Armstrong’s Axioms

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reflexivity</strong></td>
<td>If $Y \subseteq X$ then $X \rightarrow Y$</td>
</tr>
<tr>
<td><strong>Augmentation</strong></td>
<td>If $X \rightarrow Y$ then $XZ \rightarrow YZ$</td>
</tr>
<tr>
<td><strong>Transitivity</strong></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 $F \vdash X \rightarrow Y$ 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 $F \vdash AC \rightarrow D$ true?
    • If so, a proof is required
**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
  \[
  F \models X \rightarrow Y \iff F \vdash X \rightarrow Y
  \]
- Hence, Armstrong’s axioms fully capture the implication dependencies among FDs
• 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 $F \vdash X \rightarrow Y$

• Proof:
  – Denote by $X^\vdash$ the set $\{A \mid F \vdash X \rightarrow A\}$
  – We will show that $Y \subseteq X^\vdash$
  – Then $X \rightarrow Y$ is proved by repeatedly using union
    • Recall – we showed that union is provable
  – ... and we are done
Proof of Completeness (2)

- We assume that $F \models X \rightarrow Y$
- We need to prove that $Y \subseteq X^\vdash = \{ A \mid F \vdash X \rightarrow A \}$
- Suppose, by way of contradiction, that $Y \not\subseteq X^\vdash$
- Assuming $Y \not\subseteq X^\vdash$, we construct a relation $R$ such that:
  - $R$ violates $X \rightarrow Y$ (Claim 1, Claim 2)
  - $R \models F$ (Claim 3)
  - This contradicts $F \models X \rightarrow Y$
- Conclusion $Y \subseteq X^\vdash$
Construction:
- Let $X^c$ be the set of attributes that are not in $X^+$
- Observe that $Y \cap X^c \neq \emptyset$
- Construct a relation $R$ with two tuples $t$ and $u$:
  - $t[X^+] = u[X^+] = (0, \ldots, 0)$
  - $t[X^c] = (1, \ldots, 1)$
  - $u[X^c] = (2, \ldots, 2)$
Proof of Completeness (4)

- **Claim 1**: $X \subseteq X^\dagger$
  
  - Proof: apply reflexivity to each $A \in X$
• **Claim 2:** \( R \) violates \( X \rightarrow Y \)

  – Proof:
  
  • \( t \) and \( u \) agree on \( X \), due to **Claim 1**
  
  • \( t \) and \( u \) disagree on \( Y \), since \( Y \cap X^c \neq \emptyset \)
**Claim 3**: $R$ satisfies $F$

- **Proof:**
  
  - Let $Z \rightarrow W$ be an FD in $F$; we need to prove that $R$ satisfies $Z \rightarrow W$
  
  - If $Z \not\subseteq X^\vdash$ then $u$ and $t$ disagree on $Z$, and we are done; so suppose that $Z \subseteq X^\vdash$

  - Then $F \vdash X \rightarrow Z$ (union), hence $F \vdash X \rightarrow W$ (transitivity), hence $F \vdash X \rightarrow A$ for every $A \in W$ (reflexivity and transitivity)

  - We conclude that $W \subseteq X^\vdash$

  - Hence, $u$ and $t$ agree on $W$, and $R$ satisfies $Z \rightarrow W$
Some observations

• The *closure* of $F$, denoted $F^+$, is the set of all the FDs *entailed* by $F$

• The *closure* of $F$, denoted $F^+$, is the set of all the FDs *provable* from $F$

• For all $F$, $X$, $Y$:
  - $X^+ = \{ A \mid F \models X \rightarrow A \} = \{ A \mid (X \rightarrow A) \in F^+ \}$
  - $X^+ = \{ A \mid F \vdash X \rightarrow A \} = \{ A \mid (X \rightarrow A) \in F^+ \}$

• **Simple lemma:** $Y \subseteq X^+$ iff $F \vdash X \rightarrow Y$
Outline

• Introduction
• Functional Dependencies
  ▪ Definitions
  ▪ Armstrong’s Axioms
  ▪ Algorithms
• Other Types of Constraints
  ▪ Multivalued Dependencies
  ▪ Inclusion Dependencies
• Anti-Monotonicity
## Computational Problems

### Closure Computation

<table>
<thead>
<tr>
<th><strong>Given:</strong></th>
<th><strong>Goal:</strong></th>
</tr>
</thead>
</table>
| • A set $F$ of FDs  
• A set $X$ of attributes | Compute $X^+$ |

### Entailment Testing

<table>
<thead>
<tr>
<th><strong>Given:</strong></th>
<th><strong>Goal:</strong></th>
</tr>
</thead>
</table>
| • A set $F$ of FDs  
• An FD $X \rightarrow Y$ | Determine whether $F \models X \rightarrow Y$ |

### Key Generation

<table>
<thead>
<tr>
<th><strong>Given:</strong></th>
<th><strong>Goal:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• A set $F$ of FDs</td>
<td>Find a key</td>
</tr>
</tbody>
</table>

### Equivalence Testing

<table>
<thead>
<tr>
<th><strong>Given:</strong></th>
<th><strong>Goal:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sets $F$ and $G$ of FDs</td>
<td>Determine whether $F^+ = G^+$</td>
</tr>
</tbody>
</table>

---

*Recall: we always assume an underlying relation schema!*
Computing the Closure of an Attribute Set

Closure($X, F$) {
    $V := X$
    while ($V$ changes) {
        for all ($Y \rightarrow Z$ in $F$) {
            if ($Y \subseteq V$) $V := V \cup Z$
        }
    }
    return $V$
}

Example:
$F = \{AB \rightarrow C, A \rightarrow B, BC \rightarrow D, CE \rightarrow F\}$
$X = \{A\}$

<table>
<thead>
<tr>
<th>$Y \rightarrow Z$</th>
<th>$V$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>${A}$</td>
</tr>
<tr>
<td>$AB \rightarrow C$</td>
<td>${A}$</td>
</tr>
<tr>
<td>$A \rightarrow B$</td>
<td>${A, B}$</td>
</tr>
<tr>
<td>$BC \rightarrow D$</td>
<td>${A, B}$</td>
</tr>
<tr>
<td>$CE \rightarrow F$</td>
<td>${A, B}$</td>
</tr>
<tr>
<td>$AB \rightarrow C$</td>
<td>${A, B, C}$</td>
</tr>
<tr>
<td>$BC \rightarrow D$</td>
<td>${A, B, C, D}$</td>
</tr>
<tr>
<td>$CE \rightarrow F$</td>
<td>${A, B, C, D}$</td>
</tr>
</tbody>
</table>

$\{A, B, C, D\}$
What is \( \{DE\}_F^+ \)

\[ F=\{ A \rightarrow B , \quad EBD \rightarrow CE , \]
\[ IA \rightarrow D , \quad BE \rightarrow I , \quad D \rightarrow E , \quad E \rightarrow GA \} \]

What is \( \{ DE \}_F^+ \)
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, \( O(nm) \) time
- Can be improved to run in time \( O(|X|+m) \)
  - [Beeri & Bernstein, 1979]
### Implication Testing

<table>
<thead>
<tr>
<th><strong>Given:</strong></th>
<th><strong>Goal:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• A set $F$ of FDs</td>
<td>Determine whether $F \models X \rightarrow Y$</td>
</tr>
<tr>
<td>• An FD $X \rightarrow Y$</td>
<td></td>
</tr>
</tbody>
</table>

```plaintext
IsImplied(X,Y,F) {
  if ($Y \subseteq \text{Closure}(X,F)$) return true
  else return false
}
// used simple lemma
```
Equivalence Testing

Given: \( \text{Sets } F \text{ and } G \text{ of FDs} \)

Goal: \( \text{Determine whether } F^+ = G^+ \)

\[ \text{IsEquiv}(F,G) \{ \]
\[ \quad \text{for all } X \rightarrow Y \text{ in } F \]
\[ \quad \quad \text{if } (\neg \text{IsImplied}(X,Y,G)) \quad \text{return false} \]
\[ \quad \text{for all } X \rightarrow Y \text{ in } G \]
\[ \quad \quad \text{if } (\neg \text{IsImplied}(X,Y,F)) \quad \text{return false} \]
\[ \quad \text{return true} \]
\[ \} \]
Key Generation

Given:

- A set F of FDs

Goal:

Find a key

FindKey(F, R(A₁,...,Aₙ)) {
  K = {A₁,...,Aₙ}
  for (i=1,...,n) {
    if ( Aᵢ ∈ Closure(K\{Aᵢ}, F) )
      K := K\{Aᵢ}
  }
  return K
}

Example:

R(A,B,C)
F={B→A, AB→C}

<table>
<thead>
<tr>
<th>K</th>
<th>Aᵢ</th>
<th>K\Aᵢ</th>
</tr>
</thead>
<tbody>
<tr>
<td>A,B,C</td>
<td>A</td>
<td>B,C</td>
</tr>
<tr>
<td>B,C</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B,C</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

{B}
**Proof of Correctness (1)**

- **Claim 1**: Throughout the execution, \( K \) is always a superkey
  - Proof: Induction on iteration 
    - Induction hypothesis: at start of iteration \( i \),
      - \( K^+ = \{A_1, \ldots, A_n\} \)
    - Basis (\( i=1 \)): Initial \( K \) contains all attributes
    - Inductive step: If \( A_i \in (K \setminus \{A_i\})^+ \) then
      \[
      K \subseteq (K \setminus \{A_i\})^+
      \]
      and then
      \[
      \{A_1, \ldots, A_n\} = K^+ \subseteq ((K \setminus \{A_i\})^+)^+ = (K \setminus \{A_i\})^+
      \]
      This is \( K \) for the next iteration as \( K := K \setminus \{A_i\} \)
• Let $Q$ be the returned $K$

• **Claim 2:** $Q$ is minimal
  
  – Proof: by way of contradiction

  • Suppose that $Q' \not\subseteq Q$ is a superkey, and let $A_i \in Q \setminus Q'$
  
  • Then $Q \setminus \{A_i\}$ is a superkey (why?)

  • Consider the $i$’th iteration handling $A_i$: we have $Q \subseteq K$ (since we only delete things from $K$), and so, $Q \setminus \{A_i\} \subseteq K \setminus \{A_i\}$

  • But then, $Q \setminus \{A_i\}$ is a superkey, and so $K \setminus \{A_i\}$ is a superkey, and in particular $A_i \in (K \setminus \{A_i\})^+$

  • So $A_i$ should have been removed!
• 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)
• 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>04-111-1111</td>
<td>PL</td>
<td>Keren</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>IE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>Amir</td>
<td>IE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Keren</td>
</tr>
<tr>
<td>Amir</td>
<td>IE</td>
<td>04-222-2222</td>
<td>AI</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?

student $\rightarrow$ faculty  
student $\rightarrow$ phone  
student $\rightarrow$ course
Multivalued Dependency

- Let $s$ be a relation schema.
- A multivalued dependency (MVD) has the form $X\rightarrow\rightarrow Y$ where $X$ and $Y$ are disjoint sets of attributes.
- A relation $R$ satisfies $X\rightarrow\rightarrow Y$ 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
    - $t_3[X]=t_1[X]=t_2[X]$
    - $t_3[s\setminus(XY)]=t_1[s\setminus(XY)]$
    - $t_3[Y]=t_2[Y]$

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>CS</td>
<td>04-111-1111</td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>052-111-1111</td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>052-111-1111</td>
</tr>
</tbody>
</table>
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>Eran</td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>04-111-1111</td>
<td>PL</td>
<td>Keren</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>IE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>Amir</td>
<td>IE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Keren</td>
</tr>
<tr>
<td>Amir</td>
<td>IE</td>
<td>04-222-2222</td>
<td>AI</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>054-333-333</td>
<td>AI</td>
<td>Shaul</td>
</tr>
</tbody>
</table>

student → phone  student→ course
Some Properties (Exercise / Assignment)

• Every FD is an MVD
• If $X \rightarrow Y$ then $X \rightarrow s \setminus (XY)$
• An MVD $X \rightarrow Y$ is trivial (always holds) if and only if $Y = \emptyset$ or $Y = s \setminus X$
• If $X$, $Y$, $Z$ are pairwise disjoint, then $X \rightarrow Y$ and $Y \rightarrow Z$ imply $X \rightarrow Z$
Outline

• Introduction

• Functional Dependencies
  ▪ Definitions
  ▪ Armstrong’s Axioms
  ▪ Algorithms

• Other Types of Constraints
  ▪ Multivalued Dependencies
  ▪ Inclusion Dependencies

• Anti-Monotonicity
### Example of Inclusion Dependencies

#### Student

<table>
<thead>
<tr>
<th>name</th>
<th>Faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>CS</td>
</tr>
<tr>
<td>Amir</td>
<td>CS</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
</tr>
</tbody>
</table>

#### Posting

<table>
<thead>
<tr>
<th>id</th>
<th>owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Alma</td>
</tr>
<tr>
<td>45</td>
<td>Amir</td>
</tr>
<tr>
<td>76</td>
<td>Ahuva</td>
</tr>
<tr>
<td>79</td>
<td>Ahuva</td>
</tr>
</tbody>
</table>

#### Likes

<table>
<thead>
<tr>
<th>student</th>
<th>posting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>45</td>
</tr>
<tr>
<td>Alma</td>
<td>76</td>
</tr>
<tr>
<td>Ahuva</td>
<td>23</td>
</tr>
<tr>
<td>Amir</td>
<td>76</td>
</tr>
</tbody>
</table>

**Constraints:**

- \( \text{Likes}[	ext{student}] \subseteq \text{Student}[	ext{name}] \)
- \( \text{Likes}[	ext{posting}] \subseteq \text{Posting}[	ext{id}] \)
- \( \text{Posting}[	ext{owner}] \subseteq \text{Student}[	ext{name}] \)

#### Grad

<table>
<thead>
<tr>
<th>name</th>
<th>faculty</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>CS</td>
<td>Anna</td>
</tr>
<tr>
<td>Amir</td>
<td>CS</td>
<td>Anna</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>Ahmed</td>
</tr>
</tbody>
</table>

#### StudentGrant

<table>
<thead>
<tr>
<th>prof</th>
<th>student</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Amir</td>
<td>1000</td>
</tr>
<tr>
<td>Ahmed</td>
<td>Ahuva</td>
<td>1500</td>
</tr>
</tbody>
</table>

**Constraint:**

- \( \text{StudentGrant}[	ext{prof},\text{student}] \subseteq \text{Grad}[	ext{advisor},\text{name}] \)

*Note: A prof. receives a grant for a student only if she advises that student.*
• Let $S$ be a relational schema
  – Recall: $S$ consists of several relation schemas

• An *Inclusion Dependency* (IND) has the following form $R[A_1,...,A_m] \subseteq S[B_1,...,B_m]$
  where:
  – $R$ and $S$ are relation names in $S$
  – $A_1,...,A_m$ are distinct attributes of $R$
  – $B_1,...,B_m$ are distinct attributes of $S$

• Semantics: $\pi_{A_1,...,A_m}(R) \subseteq \pi_{B_1,...,B_m}(S)$
Examples

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

• 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 not covered):
  
  - **Reflexivity**: $R[X] \subseteq R[X]$
  
  - **Projection**: If $R[A_1,...,A_m] \subseteq S[B_1,...,B_m]$ 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 $S[Y] \subseteq T[Z]$ 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
- An MVD
- An IND