Database Management Systems
Course 236363

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
- 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, Σ) where I is a database instance and Σ is a set of integrity constraints; alas, I violates Σ
  - (Slides from “Uncertainty in Databases,” Advanced Topics 236605)
Consistent Query Answering

**Functionality:**
- Every student gets a unique grade per course.

**Integrity Constraints**

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

- Ahuva
- Alon

We can't always enforce. Why?

Use 4: Constraints for Schema Design

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

Example of Schema Design

**Embassy**
- **Country**
  - France
  - USA
  - Israel

**Host City**
- Tel Aviv
- Paris

**City Population**
- 400,000
- 2,200,000

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)

Normal Forms

**In “normal form”?

**In some “normal form”?

- Not in “normal form”

- In “normal form”?
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>Karen</td>
</tr>
<tr>
<td>Alma</td>
<td>052-311-1111</td>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>Alma</td>
<td>052-311-1111</td>
<td>PL</td>
<td>Karen</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>Karen</td>
</tr>
<tr>
<td>Ahuva</td>
<td>04-313-3333</td>
<td>PL</td>
<td>Shaul</td>
</tr>
<tr>
<td>Ahuva</td>
<td>054-333-3333</td>
<td>PL</td>
<td>Shaul</td>
</tr>
</tbody>
</table>

Outline

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

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 (≤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>price</th>
<th>vendor</th>
<th>headq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Galaxy S6</td>
<td>Android</td>
<td>32</td>
<td>Samsung</td>
<td>Suwon, South Korea</td>
</tr>
<tr>
<td>Galaxy S6</td>
<td>Android</td>
<td>64</td>
<td>Samsung</td>
<td>Suwon, South Korea</td>
</tr>
<tr>
<td>Galaxy Note 5</td>
<td>Android</td>
<td>32</td>
<td>Samsung</td>
<td>Suwon, South Korea</td>
</tr>
<tr>
<td>iPhone 6</td>
<td>iOS</td>
<td>16</td>
<td>Apple</td>
<td>Cupertino, CA, USA</td>
</tr>
<tr>
<td>iPhone 6</td>
<td>iOS</td>
<td>128</td>
<td>Apple</td>
<td>Cupertino, CA, USA</td>
</tr>
<tr>
<td>Nexus 6p</td>
<td>Android</td>
<td>128</td>
<td>Google</td>
<td>MV, CA, USA</td>
</tr>
<tr>
<td>Nexus 6p</td>
<td>Android</td>
<td>900</td>
<td>Google</td>
<td>MV, CA, USA</td>
</tr>
</tbody>
</table>

The attribute set \{name, os\} determines the attribute price
The attribute set \{os, vendor\} determines the attribute headq

Example: US Addresses

<table>
<thead>
<tr>
<th>US Locations</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 \{state, city\} determines the attribute street
The attribute set \{state\} determines the attribute zip
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 X ∪ Y
  – XX = X
  – XY = YX = YYXX

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 Y ⊈ X 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>

Faculty symbol name dean
CS Computer Science Itai Yavneh
EE Electrical Engineering Ariel Orda
IE Industrial Engineering Avishai Mandelbaum

• f₁ = [symbol → name, dean, name → symbol, dean, dean → name, symbol]
• f₂ = [symbol → name, name → dean, dean → symbol]
• f₃ = [symbol → name, name → symbol, dean → symbol, symbol → 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

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

- $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$
  - $F \models A \rightarrow A$
  - $F \models A \rightarrow B$
  - $F \models A \rightarrow C$
  - $F \models A \rightarrow ABC$

### 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

### 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:
  - $G^* \models F^*$
  - 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 $G \models X \rightarrow A$.
  - All attributes are required: no FD $X \rightarrow A$ in $G$ is such that $G \not\models X \rightarrow A$.
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

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

Example

- \( R(A,B,C,D,E,F) \) \{\( A \rightarrow BC, CD \rightarrow EF \)\}
- Prove that \( AD \rightarrow F \) holds
  - \( A \rightarrow BC \) implies \( A \rightarrow C \)
  - \( A \rightarrow C \) implies that \( AD \rightarrow CD \)
  - \( AD \rightarrow CD \) and \( CD \rightarrow EF \) imply \( AD \rightarrow EF \)
  - \( AD \rightarrow EF \) implies \( AD \rightarrow E \)

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 \( stmt \) is:
  - A sequence of rule applications over the facts inferred so far
    - Each application infers new facts
  - starting with what is known
  - 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 an easy example of a sound (not necessarily complete) proof system?
  - Can you give an easy example of a complete (not necessarily sound) proof system?

Example Revisited

| Reflexivity: If Y ⊆ X then X → Y |
| Augmentation: If X → Y then XZ → YZ |
| Transitivity: If X → Y and Y → Z then X → Z |

Armstrong’s Axioms

| Reflexivity: If Y ⊆ X then X → Y |
| Augmentation: If X → Y then XZ → YZ |
| Transitivity: If X → Y and Y → Z then X → Z |

Provable Rules

| Armstrong’s Axioms |
| Reflexivity: If Y ⊆ X then X → Y |
| Augmentation: If X → Y then XZ → YZ |
| Transitivity: If X → Y and Y → Z then X → Z |

- Union: If X → Y and X → Z then X → YZ
  - X → Y implies X2 → YZ (augmentation)
  - X → Z implies X → YZ (augmentation); same as X → YZ
  - X → XZ and XZ → YZ implies X → YZ (transitivity)
- Decomposition: If X → YZ then X → Y
  - Reflexivity & transitivity

Entailment vs. Provable

- Recall: F |= X → Y denotes that X → Y is entailed from F
  - Whenever F holds, so does X → Y
- By F |= X → Y we denote that X → Y is provable from F using Armstrong’s axioms
  - There is proof starting w/ F ending w/ X → Y
- Example: F={A → B, BC → D}
  - Clearly, F |= AC → D is true
  - But is F |= AC → D true?
    - If so, a proof is required

Soundness and Completeness

| Reflexivity: If Y ⊆ X then X → Y |
| Augmentation: If X → Y then X2 → YZ |
| Transitivity: If X → Y and Y → Z then X → Z |

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 |= X → Y ⇔ F ⊢ X → 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
  – For complete formality, use induction
• 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 \( X^\ast \) the set \( \{ A \mid F \vdash X \rightarrow A \} \)
  – We will show that \( Y \subseteq X^\ast \)
  – Why is it enough? Since 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 \( Y \subseteq X \vdash \{ A \mid F \vdash X \rightarrow A \} \)

• Conclusion \( Y \subseteq X^\ast \)

Proof of Completeness (3)

• Construction:
  – Let \( X^\ast \) be the set of attributes that are not in \( X \vdash \)
  – Suppose, by way of contradiction, that \( Y \nsubseteq X^\ast \)
  – Assuming \( Y \nsubseteq X^\ast \), we construct a relation \( R \) s.t.:
    – \( R \) violates \( X \rightarrow Y \) (Claim 1, Claim 2)
    – \( R \) violates \( F \) (Claim 3)
    – This contradicts \( F \models X \rightarrow Y \)

Proof of Completeness (4)

• Claim 1: \( X \nsubseteq X^\ast \)
  – Proof: apply \textit{reflexivity} to each \( A \in X \)

Proof of Completeness (5)

• Claim 2: \( R \) violates \( X \rightarrow Y \)
  – Proof:
    • \( t \) and \( u \) agree on \( X \), due to \textit{Claim 1}
    • \( t \) and \( u \) disagree on \( Y \), since \( Y \nsubseteq X^\ast \)
Computing the Closure of an Attribute Set

Outline

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

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, \( O(nm) \) time
  - Can be improved to run in time \( O(|X|+m) \)

Given: Goal:  
\[ \text{Closure Computation} \]
\[
\begin{array}{ll}
\text{Given:} & \text{Goal:} \\
\text{A set of FDs} & \text{Compute } \mathcal{X}^+ \\
\text{A set of attributes} & \\
\end{array}
\]

Some observations

- The closure \( \mathcal{F}^+ \) of \( F \) is the set of all the FDs entailed by \( F \)
- The closure \( \mathcal{F}^* \) of \( F \) is the set of all the FDs provable from \( F \)
- Notation:
  - \( \mathcal{F}^+ = \{ A \mid F \vdash X \rightarrow A \} = \{ A \mid (X \rightarrow A) \in \mathcal{F}^+ \} \)
  - \( \mathcal{F}^* = \{ A \mid F \vdash X \rightarrow A \} = \{ A \mid (X \rightarrow A) \in \mathcal{F}^* \} \)
- Simple claim: \( Y \subseteq \mathcal{X}^+ \iff F \vdash X \rightarrow Y \)

Outline

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

Computational Problems

Closure Computation

Given: Goal:
\[ \begin{array}{ll}
\text{Given:} & \text{Goal:} \\
\text{A set of FDs} & \text{Compute } \mathcal{X}^+ \\
\text{A set of attributes} & \\
\end{array} \]

Entailment Testing

Given: Goal:
\[ \begin{array}{ll}
\text{Given:} & \text{Goal:} \\
\text{A set of FDs} & \text{Determine whether } F \vdash X \rightarrow Y \\
\text{An FD } X \rightarrow Y & \\
\end{array} \]

Key Generation

Given: Goal:
\[ \begin{array}{ll}
\text{Given:} & \text{Goal:} \\
\text{A set of FDs} & \text{Find a key} \\
\end{array} \]

Equivalence Testing

Given: Goal:
\[ \begin{array}{ll}
\text{Given:} & \text{Goal:} \\
\text{Sets } F \text{ and } G \text{ of FDs} & \text{Determine whether } F = G^* \\
\end{array} \]

Computing the Closure of an Attribute Set

Given:
\[ \begin{array}{ll}
\text{Closure} & \{ X, F \} \\
\text{V := X} & \\
\text{while (V changes) } \{ \\
\text{for all } (Y \rightarrow Z \text{ in } F) \{ \\
\text{if } (Y \subseteq V) \text{ } \\
\text{V := V U Z} & \\
\} & \\
\} & \\
\text{return } V & \\
\end{array} \]

Example:
\[ F = \{ AB \rightarrow C, A \rightarrow B, BC \rightarrow D, CE \rightarrow F \} \]
\[ X = \{ A \} \]
\[ \begin{array}{|c|c|}
\hline
\text{A} & \text{X} \\
\hline
\text{A} & \{\text{A}\} \\
\text{B} & \{\text{A,}\text{B}\} \\
\text{C} & \{\text{A,}\text{B,}\text{C}\} \\
\text{E} & \text{null} \\
\text{A} & \text{null} \\
\text{B} & \text{null} \\
\text{C} & \text{null} \\
\hline
\end{array} \]
Implication Testing

Given:
• A set F of FDs
• An FD X → Y

Determine whether $F \models X \rightarrow Y$

IsImplied(X, Y, F) {
  if ($Y \subseteq \text{Closure}(X, F)$) return true
  else return false
}

Equivalence Testing

Given:
• Sets F and G of FDs

Determine whether $F + G = G + F$

IsEquiv(F, G) {
  for all $X \rightarrow Y$ in F
    if (!IsImplied(X, Y, G)) return false
  for all $X \rightarrow Y$ in G
    if (!IsImplied(X, Y, F)) return false
  return true
}

Key Generation

Given:
• A set F of FDs

Find a key

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

Example:
R(A, B, C)
F = {B \rightarrow A, AB \rightarrow C}
K
A
B
C
K \setminus A
K \setminus B
K \setminus C
(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,
      – $K^i = \{A_i, ..., A_n\}$
    • 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
      $\{A_1, ..., A_n\} = K^i \subseteq (K \setminus \{A_i\})^*$

Proof of Correctness (2)

• Let Q be the returned K
• CLAIM 2: Q is minimal
  – Proof: by way of contradiction
    • Suppose Q' ⊆ Q is a superkey, and let A ∈ Q \ Q'
    • Then Q \{A\} is a superkey (why?)
    • In the i'th iteration of handling A, we have Q ⊂ K
      (since we only delete from K), so Q \{A\} ∊ K \{A\}
    • But then, Q \{A\} is a superkey, and so K \{A\} is a superkey, and in particular A ∈ (K \{A\})^*
    • So A should have been removed!
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)

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>Amer</td>
<td>IE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>Amer</td>
<td>IE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Keren</td>
</tr>
<tr>
<td>Amer</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 → faculty  student → phone  student → course

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>Amer</td>
<td>IE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Eran</td>
</tr>
<tr>
<td>Amer</td>
<td>IE</td>
<td>04-222-2222</td>
<td>PL</td>
<td>Keren</td>
</tr>
<tr>
<td>Amer</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>

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

Example of Inclusion Dependencies

<table>
<thead>
<tr>
<th>Student name</th>
<th>Faculty</th>
<th>Posting id</th>
<th>Owner name</th>
<th>Likes posting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>CS</td>
<td>23</td>
<td>Alma</td>
<td>45</td>
</tr>
<tr>
<td>Amir</td>
<td>CS</td>
<td>45</td>
<td>Amir</td>
<td>76</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>76</td>
<td>Ahuva</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>79</td>
<td>Ahuva</td>
<td>76</td>
</tr>
</tbody>
</table>

Example of Inclusion Dependencies

<table>
<thead>
<tr>
<th>Grad name</th>
<th>Faculty</th>
<th>Advisor name</th>
<th>StudentGrant prof</th>
<th>student sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>CS</td>
<td>Anna</td>
<td>Anna</td>
<td>1000</td>
</tr>
<tr>
<td>Amir</td>
<td>CS</td>
<td>Anna</td>
<td>Ahmed</td>
<td>1500</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>Ahmed</td>
<td>Ahuva</td>
<td></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
  \[ R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m] \]
  where:
  - \( R \) and \( S \) are relation names in \( S \)
  - \( A_1, \ldots, A_m \) are distinct attributes of \( R \)
  - \( B_1, \ldots, B_m \) are distinct attributes of \( S \)
- Semantics: \( \pi_{A_1, \ldots, A_m}(R) \subseteq \pi_{B_1, \ldots, B_m}(S) \)

Examples

- What is the meaning of the following IND?
  \( \text{Grad}[\text{name}] \subseteq \text{StudentGrant}[	ext{student}] \)

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

Sound 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 \( R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m] \) then for every sequence \( i_1, \ldots, i_m \) of distinct indices in \( \{1, \ldots, m\} \) we have \( R[A_{i_1}, \ldots, A_{i_m}] \subseteq S[B_{i_1}, \ldots, B_{i_m}] \)
  - Transitivity: If \( R[X] \subseteq S[Y] \) and \( S[Y] \subseteq T[Z] \) then \( R[X] \subseteq T[Z] \)