Database Management Systems
Course 236363
Lecture 6:
Integrity Constraints

Database Constraints (Dependencies)
• Definition: properties that DBs 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
• Principles of schema design

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
  ![Diagram]
• 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: \((D,\Sigma)\) where \(D\) is a database and \(\Sigma\) is a set of integrity constraints; alas, \(D\) violates \(\Sigma\)
  (Slides from “Uncertainty in Databases,” Advanced Topics 236605)
**Consistent Query Answering**

<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>

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

Integrity Constraints

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

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

```
SELECT student
FROM Grades G, Courses C
WHERE G.grade >= 80 AND G.course = C.course AND C.lecturer='Eran'
```

**Database D**

**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**

<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)

**Normal Forms**

<table>
<thead>
<tr>
<th>country</th>
<th>host</th>
<th>address</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>

In some “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>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>Ahuva</td>
<td>04-333-3333</td>
<td>Al</td>
<td>Shaul</td>
</tr>
<tr>
<td>Ahuva</td>
<td>054-333-3333</td>
<td>Al</td>
<td>Shaul</td>
</tr>
</tbody>
</table>

Outline

- Introduction
- Functional Dependencies
  - Definitions
  - Armstrong’s Axioms
  - Algorithms
- Other Types of Constraints [Complimentary]
  - 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 per relation), 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>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>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 determines the attribute:

- state
- zip
- city

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

The attribute set determines the attribute:

- state
- city
- street
- zip

The attribute set determines the attribute:

- state
- 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 = 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
- Notationally:
  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 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: tuples that agree on entire X also agree on every subset of X.
    - For the “only if” direction, consider the relation r that contains 2 tuples that agree precisely on 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 → course

<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 = (symbol → name dean, name → symbol dean, dean → name symbol)
- F_2 = (symbol → name dean, name → dean symbol)
- F_3 = (symbol → name, name → 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

- It is often convenient to work with a set of FDs that does not have any triviality/redundancy within; this is captured by the formal notion of a minimal cover
- Formally, a minimal cover (or minimal basis) of a set $F$ of FDs 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 $G \setminus X \rightarrow A$
  - All attributes are required: no FD $XB \rightarrow A$ in $G$ is such that $G \models X \rightarrow A$
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 of \(S\)
- 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

Outline

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

Example

- \(R(A,B,C,D,E,F) \quad \{A \rightarrow BC, \ CD \rightarrow EF\}\)
- Prove that \(AD \rightarrow F\) is entailed (holds in any consistent database)

  - \(A \rightarrow BC\) 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 \(AD \rightarrow F\)

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

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 over the facts inferred so far
    - Each application infers new facts
  - starting with what is known
  - 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 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?
• (Class worksheet)

Example Revisited

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 |

Entailment vs. Proof

• Recall: F ⊨ X ⟷ Y denotes that X ⟷ Y is entailed by 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 with F ending with 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 XZ ⟷ 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
• In notation, 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 $X^+$ the set $\{ A \mid F \vdash X \rightarrow A \}$
  - We will show that $Y \subseteq X^+$
  - 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 $Y \subseteq X^+$

  Proof:
  - Denote by $X^c$ the set $\{ A \mid F \vdash X \rightarrow A \}$
  - Suppose, by way of contradiction, that $Y \not\subseteq X^c$
  - This contradicts $F \models X \rightarrow Y$
  - Conclusion $Y \subseteq X^c$

Proof of Completeness (3)

- Construction:
  - Let $X^c$ be the set of attributes that are not in $X^+$
  - Suppose, by way of contradiction, that $Y \not\subseteq X^c$
  - Observe that $Y \cap X^c \neq \emptyset$ (our assumption)
  - Construct a relation $r$ with two tuples $t$ and $u$:
    - $t[X^c] = u[X^c] = (0, ..., 0)$
    - $t[X^c] = (1, ..., 1)$
    - $u[X^c] = (2, ..., 2)$

Proof of Completeness (4)

- **Claim 1**: $X \subseteq X^+$
  - Proof: apply *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 Claim 1
    - $t$ and $u$ disagree on $Y$, since $Y \cap X^c \neq \emptyset$
Proof of Completeness (6)

- **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 \( Z \subseteq X \) then \( u \) and \( t \) disagree on \( Z \), and we are done; so suppose that \( Z \not\subseteq X \)
    - Then \( F \vdash Z \rightarrow X \) (union), hence \( F \vdash X \rightarrow W \) (transitivity), hence \( F \vdash X \rightarrow A \) for every \( A \in W \) (decomposition)
    - We conclude that \( W \subseteq X \)
    - Hence, \( u \) and \( t \) agree on \( W \), and \( r \) satisfies \( Z \rightarrow W \)

Some observations

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

Outline

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

Computational Problems (1)

**Closure Computation**
- Given:
  - A set \( F \) of FDs
  - A set \( X \) of attributes
- Goal:
  - Compute \( X^+ \)

**Entailment Testing**
- Given:
  - A set \( F \) of FDs
  - An FD \( X \rightarrow Y \)
- Goal:
  - Determine whether \( F \vdash X \rightarrow Y \)

**Equivalence Testing**
- Given:
  - Sets \( F \) and \( G \) of FDs
- Goal:
  - Determine whether \( F = G^+ \)

Computational Problems (2)

**Key Generation**
- Given:
  - A set \( F \) of FDs
- Goal:
  - Find a key

**Minimal Covering**
- Given:
  - A set \( F \) of FDs
- Goal:
  - Compute a minimal cover of \( F \)

Computing the Closure of an Attribute Set

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

\[
\begin{array}{c|c}
Y \rightarrow Z & Y \\
\hline
AB \rightarrow C & \{A\} \\
A \rightarrow B & \{A,B\} \\
BC \rightarrow D & \{A,B\} \\
CE \rightarrow F & \{A,B,C\} \\
AB \rightarrow C & \{A,B,C\} \\
BC \rightarrow D & \{A,B,C,D\} \\
CE \rightarrow F & \{A,B,C,D\} \\
\end{array}
\]

- \( \{A,B,C,D\} \)
Correctness and Running Time

• Correctness is due to the completeness of Armstrong’s axioms: \( X \rightarrow Y \) is provable by applying the axioms if \( Y \subseteq \text{Closure}(X) \)
  – Formal proof omitted (out of scope)
• 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 \( O(|X|+m) \)

\[ \text{[Beeri & Bernstein, 1979]} \]

Equivalence Testing

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

Key Generation

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

\text{Example:}
\[ R(A,B,C) \]
\[ F = \{ B \rightarrow A, AB \rightarrow C \} \]
\[ \text{FindKey}(F,R(A,B,C)) \]
\[ K = \{ A, B, C \} \]
\[ K \backslash\{A\} = \{ B, C \} \]
\[ K \backslash\{B\} = \{ A, C \} \]
\[ K \backslash\{B\} = \{ A, C \} \]

Proof of Correctness (1)

• \textbf{CLAIM 1:} Throughout the execution, \( K \) is always a superkey
  – Proof: Induction on iteration #
    • Induction hypothesis: at start of iteration \( i \), \( K^* = \{ A_1,\ldots,A_i \} \)
    • Basis \((i=1)\): Initial \( K \) contains all attributes
    • Inductive step: If \( A_i \in (K\backslash\{A_i\})^+ \) then
      \[ K \subseteq (K\backslash\{A_i\})^+ \]
      and then
      \[ \{ A_1,\ldots,A_n \} = K^* \subseteq ((K\backslash\{A_i\})^+)^+ = (K\backslash\{A_i\})^+ \]

Proof of Correctness (2)

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

Key Generation

Given: A set \( F \) of FDs
Goal: Find a minimal cover of \( F \)

\[
\text{MinCover}(F) \begin{cases}
G := \{ X \rightarrow A \mid A \in Y \text{ for some } X \rightarrow Y \in F \} \\
\text{while}(G \text{ changes}) \\
\quad \text{if } G \text{ contains } X \rightarrow A \text{ s.t. } A \in \text{Closure}(X, G \setminus \{X \rightarrow A\}) \\
\quad \quad G := G \setminus \{X \rightarrow A\} \\
\quad \text{if } G \text{ contains } XB \rightarrow A \text{ s.t. } A \in \text{Closure}(X, G) \\
\quad \quad G := (G \setminus \{XB \rightarrow A\}) \cup \{X \rightarrow A\} \\
\end{cases}
\]

return \( G \)

Outline

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

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>Shaul</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>Shaul</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>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>04-333-3333</td>
<td>AI</td>
<td>Shaul</td>
</tr>
</tbody>
</table>

Why is this table “badly” designed? Are there any FDs?

The rest of the presentation is not in the official course material.

OPTIONAL MATERIAL
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
    - \( t_1[X] = t_1[X] = t_3[X] \)
    - \( t_1[X \setminus Y] = t_1[X \setminus Y] \)
    - \( t_1[Y] = t_2[Y] \)

Some Properties (Exercise / Assignment)

- Every FD is an MVD
- 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 \( Y = S \setminus X \)
- 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>Posting</th>
<th>Likes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alma</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>Amir</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grad</th>
<th>StudentGrant</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>

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 \( R[A_1,...,A_m] \subseteq S[B_1,...,B_m] \)
    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?
  Grad[name] \subseteq StudentGrant[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 $i_1,\ldots,i_k$ of distinct indices in $\{1,\ldots,m\}$ we have $R[A_{i_1},\ldots,A_{i_k}] \subseteq S[B_{i_1},\ldots,B_{i_k}]$
  – Transitivity: If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$ then $R[X] \subseteq T[Z]$