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 236805
• 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: \((I, \Sigma)\) 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>Ahuva</td>
<td>PL</td>
<td>81</td>
</tr>
<tr>
<td>Alon</td>
<td>PL</td>
<td>80</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>PL</td>
<td>DC</td>
</tr>
<tr>
<td>PL</td>
<td>Keren</td>
</tr>
</tbody>
</table>

Functional Dependency: every student gets a unique grade per course

Integrity Constraints 1

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

Ahuva

We can’t always enforce. Why?

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>PL</td>
<td>DC</td>
</tr>
<tr>
<td>PL</td>
<td>Keren</td>
</tr>
</tbody>
</table>

Functional Dependency: every student gets a unique grade per course

Integrity Constraints 2

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

Ahuva

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>PL</td>
<td>DC</td>
</tr>
<tr>
<td>PL</td>
<td>Keren</td>
</tr>
</tbody>
</table>

Functional Dependency: every student gets a unique grade per course

Integrity Constraints 2

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

Ahuva

Alon
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>Embassy</th>
<th>Studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>country</td>
<td>host</td>
</tr>
<tr>
<td>France</td>
<td>Israel</td>
</tr>
<tr>
<td>USA</td>
<td>Israel</td>
</tr>
<tr>
<td>Israel</td>
<td>France</td>
</tr>
<tr>
<td>USA</td>
<td>France</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>Embassy</th>
<th>In “normal form”?</th>
</tr>
</thead>
<tbody>
<tr>
<td>country</td>
<td>host</td>
</tr>
<tr>
<td>France</td>
<td>Israel</td>
</tr>
<tr>
<td>USA</td>
<td>Israel</td>
</tr>
<tr>
<td>Israel</td>
<td>France</td>
</tr>
<tr>
<td>USA</td>
<td>France</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
  - 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>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>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 **price**

The attribute set **os** determines the attribute **vendor**

The attribute set **disk** determines the attribute **headq**

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>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>90046</td>
</tr>
</tbody>
</table>

The attribute set **state** determines the attribute **zip**

The attribute set **city** determines the attribute **street**

The attribute set **zip** determines the attribute **state**

Outline

- Introduction
- Functional Dependencies
  - Definitions
  - Armstrong’s Axioms
  - Algorithms
- Other Types of Constraints
  - Multivalued Dependencies
  - Inclusion Dependencies
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 U Y
  – XX = X
  – XY = YX = YXX

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)
- \textbf{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 $y \notin X$ 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...**

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

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

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

- $F = \{\text{name} \rightarrow \text{vendor}, \text{vendor} \rightarrow \text{headq}\}$
  - $F \models \text{name} \rightarrow \text{headq}$
  - $F \models \text{name}, \text{vendor} \rightarrow \text{headq}$
  - $F \models \text{name}, \text{vendor} \rightarrow \text{vendor}$
- $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$
  - $F \models A \rightarrow A$
  - $F \models A \rightarrow B$
  - $F \models A \rightarrow C$
  - $F \models A \rightarrow ABC$

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:
  – $G^* = F^*$
  – 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 $X_B \rightarrow A$ 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 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

- \( R(A,B,C,D,E,F) \quad \{A \rightarrow BC, CD \rightarrow EF\} \)
- 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 \( \text{stmt} \) 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?

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 \( X \rightarrow Z \) |

Example Revisited

| 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 \( X \rightarrow Z \) |

\( R(A,B,C,D,E,F) \) (\( A \rightarrow BC \), \( CD \rightarrow EF \)): prove \( AD \rightarrow F \)

– \( A \rightarrow BC \) implies \( A \rightarrow C \)
  • Reflexivity, Transitivity
– \( A \rightarrow C \) implies that \( AD \rightarrow CD \)
  • Augmentation
– \( AD \rightarrow CD \) and \( CD \rightarrow EF \) imply \( AD \rightarrow EF \)
  • Transitivity
– \( AD \rightarrow EF \) implies \( AD \rightarrow E \)
  • Reflexivity, Transitivity
Provable Rules

**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 \( X \rightarrow Z \)

- **Union**: If \( X \rightarrow Y \) and \( X \rightarrow Z \) then \( X \rightarrow YZ \)
  - \( X \rightarrow Y \) implies \( XZ \rightarrow YZ \) (augmentation)
  - \( X \rightarrow Z \) implies \( X \rightarrow XZ \) (augmentation), same as \( X \rightarrow XZ \)
  - \( X \rightarrow XZ \) and \( XZ \rightarrow YZ \) implies \( X \rightarrow YZ \) (transitivity)

- **Decomposition**: If \( X \rightarrow YZ \) then \( X \rightarrow Y \)
  - Reflexivity & transitivity

Entailment vs. Provable

- **Recall**: \( F \models X \rightarrow Y \) denotes that \( X \rightarrow Y \) is **entailed** from \( F \)
  - Whenever \( F \) holds, so does \( X \rightarrow Y \)
- **By** \( F \vdash X \rightarrow Y \) we denote that \( X \rightarrow Y \) is **provable** from \( F \) using Armstrong’s axioms
  - There is proof starting w/ \( F \) ending w/ \( X \rightarrow Y \)
- **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

Soundness and Completeness

- **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 \( X \rightarrow Z \)

**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
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 $F \vDash X \rightarrow Y$
- 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 $F \vDash X \rightarrow Y$
- We need to prove that $Y \subseteq X^*$
- Suppose, by way of contradiction, that $Y \not\subseteq X^*$
- Assuming $Y \not\subseteq X^*$, we construct a relation $R$ s.t.:
  - $R$ violates $X \rightarrow Y$ (Claim 1, Claim 2)
  - $R \not\vdash F$ (Claim 3)
  - This contradicts $F \vDash X \rightarrow Y$
- Conclusion $Y \subseteq X^*$
Proof of Completeness (3)

- **Construction:**
  - Let $X'$ be the set of attributes that are not in $X^*$
  - Observe that $Y \cap X' \neq \emptyset$ (our assumption)
  - Construct a relation $R$ with two tuples $t$ and $u$:
    - $t[X'] = u[X'] = (0, 0, 0, 1, 1, 1, 1)$
    - $t[X'] = (1, 1, 1, 1, 1, 1, 1)$
    - $u[X'] = (2, 2, 2, 2, 2, 2, 2)$

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 $Y \cap X' \neq \emptyset$
Proof of Completeness (6)

- **Claim 3:** \( R \) satisfies \( F \)

  - Proof:
    1. Let \( Z \rightarrow W \) be an FD in \( F \); we need to prove that \( R \) satisfies \( Z \rightarrow W \)
    2. If \( Z \subseteq X^* \) then \( u \) and \( t \) disagree on \( Z \), and we are done; so suppose that \( Z \nsubseteq X^* \)
    3. Then \( F \vdash X \rightarrow Z \) (union), hence \( F \vdash X \rightarrow W \) (transitivity), hence \( F \vdash X \rightarrow A \) for every \( A \in W \) (decomposition)
    4. We conclude that \( W \subseteq X^* \)
    5. 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:
  1. \( X^* = \{ A \mid F \models X \rightarrow A \} = \{ A \mid (X \rightarrow A) \in F^* \} \)
  2. \( X^+ = \{ A \mid F \vdash X \rightarrow A \} = \{ A \mid (X \rightarrow A) \in F^* \} \)
- Simple claim: \( Y \subseteq X^* \) iff \( F \vdash X \rightarrow Y \)

Outline

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

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 → Y
Goal: Determine whether F ⊨ X → Y

Key Generation
Given:
- A set F of FDs
Goal: Find a key

Equivalence Testing
Given:
- Two sets F and G of FDs
Goal: Determine whether F= rename G

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 → Z in F) {
      if (Y ⊆ V) {
        V := V ∪ Z
      }
    }
    return V
  }
}

Example:
F = {AB → C, A → B, BC → D, CE → F}
X = {A}

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 F 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

Given: A set $F$ of FDs
An FD $X \rightarrow Y$

Goal: Determine whether $F \models X \rightarrow Y$

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

Equivalence Testing

Given: $F$ and $G$ of FDs

Goal: Determine whether $F \equiv G$

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

Key Generation

Given: A set of FDs $F$

Goal: Find a key

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

Example:
$$\begin{array}{c|c|c|c}
  A & B & C \\
  \hline
  \text{F} & \text{R} & \text{IL} \\
  \hline
  A & B & C \\
  B & C & B \\
  B & C & C \\
  B & C & B \\
  B & C & C \\
  \hline
  (B)
\end{array}$$
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 \( A \in (K \setminus \{A_i\})^* \) then
      \[ K \subseteq (K \setminus \{A_i\})^* \]
      and then
      \[ \{A_j, \ldots, A_n\} = K^* \subseteq ((K \setminus \{A_i\})^* = (K \setminus \{A_i\})^* \]

Proof of Correctness (2)

- Let \( Q \) be the returned \( K \)
- **Claim 2**: \( Q \) is minimal
  - Proof: by way of contradiction
    - Suppose \( Q' \subseteq Q \) 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 \( Q(A) \subseteq K(A) \)
    - But then, \( Q(A) \) is a superkey, and so \( K(A) \) is a superkey, and in particular \( A \in (K(A))^* \)
    - So \( A \) should have been removed!

Outline

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

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

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
### 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 Eran</td>
<td></td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>04-111-1111</td>
<td>PL Keren</td>
<td></td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>052-111-1111</td>
<td>PL Eran</td>
<td></td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>052-111-1111</td>
<td>PL Keren</td>
<td></td>
</tr>
<tr>
<td>Amir</td>
<td>IE</td>
<td>04-222-2222</td>
<td>PL Eran</td>
<td></td>
</tr>
<tr>
<td>Amir</td>
<td>IE</td>
<td>04-222-2222</td>
<td>PL Keren</td>
<td></td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>04-333-3333</td>
<td>AI Shaul</td>
<td></td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>054-333-3333</td>
<td>AI Shaul</td>
<td></td>
</tr>
</tbody>
</table>

Why is this table “badly” designed?

Are there any FDs?

student→faculty  student→phone  student→course

### 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]
    - $t_3[s \setminus (XY)] = t_1[s \setminus (XY)]$
    - $t_3[Y] = t_2[Y]

### 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 Eran</td>
<td></td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>04-111-1111</td>
<td>PL Keren</td>
<td></td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>052-111-1111</td>
<td>PL Eran</td>
<td></td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>052-111-1111</td>
<td>PL Keren</td>
<td></td>
</tr>
<tr>
<td>Amir</td>
<td>IE</td>
<td>04-222-2222</td>
<td>PL Eran</td>
<td></td>
</tr>
<tr>
<td>Amir</td>
<td>IE</td>
<td>04-222-2222</td>
<td>PL Keren</td>
<td></td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>04-333-3333</td>
<td>AI Shaul</td>
<td></td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>054-333-3333</td>
<td>AI Shaul</td>
<td></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</th>
<th>Posting</th>
<th>Likes</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>faculty</td>
<td>id</td>
</tr>
<tr>
<td>Alma</td>
<td>CS</td>
<td>23</td>
</tr>
<tr>
<td>Amir</td>
<td>CS</td>
<td>45</td>
</tr>
<tr>
<td>Ahuva</td>
<td>EE</td>
<td>79</td>
</tr>
</tbody>
</table>

Likes[student] $\subseteq$ Student[name]
Likes[posting] $\subseteq$ Posting[id]
Posting[owner] $\subseteq$ Student[name]

<table>
<thead>
<tr>
<th>Grad</th>
<th>StudentGrant</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>faculty</td>
</tr>
<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>

<table>
<thead>
<tr>
<th>name</th>
<th>faculty</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>

StudentGrant[prof,student] $\subseteq$ Grad[advisor,name]

A prof. receives a grant for a student only if she advises that student.
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 \( R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m] \)
  - \( R \) 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?
  \( \text{Grad}[\text{name}] \subseteq \text{StudentGrant}[\text{student}] \)
- 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 \( R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_n] \) 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 \( R[X] \subseteq T[Z] \)