CSC 261/461 – Database Systems
Lecture 10

Fall 2017
Announcement

• No class/quiz on next Monday!
1. Database Design

2. Normal forms & functional dependencies

3. Finding functional dependencies

4. Closures, superkeys & keys
Design Theory

• Design theory is about how to represent your data to avoid anomalies.

• Achieved by Data Normalization, a process of analyzing a relation to ensure that it is well formed.

• Normalization involves decomposing relations with anomalies to produce smaller well structured relations.

• If a relation is normalized (or well formed), rows can be inserted, deleted and modified without creating anomalies.
Normalization Example

• \((\text{Student ID}) \rightarrow (\text{Student Name}, \text{DormName}, \text{DormCost})\)

• However, if
  
  – \((\text{DormName}) \rightarrow (\text{DormCost})\)

Then, DormCost should be put into its own relation, resulting in:

\((\text{Student ID}) \rightarrow (\text{Student Name}, \text{DormName})\)

\((\text{DormName}) \rightarrow (\text{DormCost})\)
Normalization Example

• \((\text{AttorneyID}, \text{ClientID}) \rightarrow (\text{ClientName}, \text{MeetingDate}, \text{Duration})\)

• However, if \(- \text{ClientID} \rightarrow \text{ClientName}\)

• Then: \text{ClientName} should be in its own relation:

• \((\text{AttorneyID}, \text{ClientID}) \rightarrow (\text{MeetingDate}, \text{Duration})\)
• \((\text{ClientID}) \rightarrow (\text{ClientName})\)
Normal Forms

• 1\textsuperscript{st} Normal Form (1NF) = All tables are flat

• 2\textsuperscript{nd} Normal Form = disused

• 3\textsuperscript{rd} Normal Form (3NF)

• Boyce-Codd Normal Form (BCNF)

• 4\textsuperscript{th} and 5\textsuperscript{th} Normal Forms = see text books

DB designs based on \textit{functional dependencies}, intended to prevent data \textit{anomalies}
Normalization Steps

1. **Table with multivalued attributes**
   - Remove multivalued attributes
   - **First Normal Form (1NF)**

2. **First Normal Form (1NF)**
   - Remove Partial Dependencies
   - **Second Normal Form (2NF)**

3. **Second Normal Form (2NF)**
   - Remove Transitive Dependencies
   - **Third Normal Form (3NF)**

CSC 261, Fall 2017, UR
### 1st Normal Form (1NF)

#### Violates 1NF.

<table>
<thead>
<tr>
<th>Student</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>{CS145,CS229}</td>
</tr>
<tr>
<td>Joe</td>
<td>{CS145,CS106}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### In 1st NF

<table>
<thead>
<tr>
<th>Student</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS145</td>
</tr>
<tr>
<td>Mary</td>
<td>CS229</td>
</tr>
<tr>
<td>Joe</td>
<td>CS145</td>
</tr>
<tr>
<td>Joe</td>
<td>CS106</td>
</tr>
</tbody>
</table>

**1NF Constraint:** Types must be atomic!
Data Anomalies & Constraints
### Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes **anomalies**:

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CSC261</td>
<td>101</td>
</tr>
<tr>
<td>Joe</td>
<td>CSC261</td>
<td>101</td>
</tr>
<tr>
<td>Sam</td>
<td>CSC261</td>
<td>101</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

If every course is in only one room, contains **redundant** information!
Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes *anomalies*:

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CSC261</td>
<td>101</td>
</tr>
<tr>
<td>Joe</td>
<td>CSC261</td>
<td>703</td>
</tr>
<tr>
<td>Sam</td>
<td>CSC261</td>
<td>101</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

If we update the room number for one tuple, we get inconsistent data = an *update anomaly*.
A poorly designed database causes anomalies:

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>..</td>
<td>..</td>
<td>.</td>
</tr>
</tbody>
</table>

If everyone drops the class, we lose what room the class is in! = a delete anomaly
Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes anomalies:

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CSC261</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CSC261</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CSC261</td>
<td>B01</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

Similarly, we can’t reserve a room without students = an insert anomaly
Constraints Prevent (some) Anomalies in the Data

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CSC261</td>
</tr>
<tr>
<td>Joe</td>
<td>CSC261</td>
</tr>
<tr>
<td>Sam</td>
<td>CSC261</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSC261</td>
<td>101</td>
</tr>
<tr>
<td>CSC257</td>
<td>601</td>
</tr>
</tbody>
</table>

Today: develop theory to understand why this design may be better and how to find this decomposition...
Functional Dependencies
• A **relationship** between attributes where one attribute (or group of attributes) **determines** the value of another attribute (or group of attributes) in the same table.

• **Example:**

  The price of one cookie and the quantity of cookies can determine the price of a box of cookies. (assuming each box has 10 cookies).

  \[(\text{cookie\_price, quantity}) \rightarrow \text{box\_price}\]
Candidate Keys/Primary Keys and Functional Dependencies

• By definition:
• A candidate key of a relation functionally determines all other non key attributes in the row.

• Implies:
• A primary key of a relation functionally determines all other non key attributes in the row.

EmployeeID \rightarrow (EmployeeName, EmpPhone)
**Functional Dependency**

**Def:** Let $A, B$ be sets of attributes. We write $A \rightarrow B$ or say $A$ functionally determines $B$ if, for any tuples $t_1$ and $t_2$:

$$t_1[A] = t_2[A] \implies t_1[B] = t_2[B]$$

and we call $A \rightarrow B$ a **functional dependency**

---

*A-*$B$ **means that**

“*whenever two tuples agree on $A$ then they agree on $B$.”*
Defn (again):
Given attribute sets \( A = \{A_1, \ldots, A_m\} \) and \( B = \{B_1, \ldots, B_n\} \) in \( R \),
A Picture Of FDs

<table>
<thead>
<tr>
<th>A₁</th>
<th>...</th>
<th>Aᵣ</th>
<th>B₁</th>
<th>...</th>
<th>Bᵣ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Defn (again):
Given attribute sets \( \text{A} = \{A₁, \ldots, Aᵣ\} \) and \( \text{B} = \{B₁, \ldots, Bᵣ\} \) in \( \mathbb{R} \),

The *functional dependency* \( \text{A} \rightarrow \text{B} \) on \( \mathbb{R} \) holds if for *any* \( tᵢ, tⱼ \) in \( \mathbb{R} \):
### A Picture Of FDs

**Defn (again):**
Given attribute sets \( A=\{A_1,\ldots,A_m\} \) and \( B=\{B_1,\ldots,B_n\} \) in \( R \),

The **functional dependency** \( A \rightarrow B \) on \( R \) holds if for any \( t_i, t_j \) in \( R \):

\[
t_i[A_1] = t_j[A_1] \text{ AND } t_i[A_2] = t_j[A_2] \text{ AND } \ldots \text{ AND } t_i[A_m] = t_j[A_m]
\]
Defn (again): Given attribute sets $A = \{A_1, \ldots, A_m\}$ and $B = \{B_1, \ldots, B_n\}$ in $R$, the functional dependency $A \rightarrow B$ on $R$ holds if for any $t_i, t_j$ in $R$:

\[
\begin{align*}
\text{if } & t_i[A_1] = t_j[A_1] \text{ AND } t_i[A_2] = t_j[A_2] \text{ AND } \\
& \ldots \text{ AND } t_i[A_m] = t_j[A_m],
\end{align*}
\]

\[
\begin{align*}
\text{then } & t_i[B_1] = t_j[B_1] \text{ AND } t_i[B_2] = t_j[B_2] \text{ AND } \\
& \ldots \text{ AND } t_i[B_n] = t_j[B_n].
\end{align*}
\]
FDs for Relational Schema Design

• High-level idea: why do we care about FDs?

1. Start with some relational schema

2. Find out its functional dependencies (FDs)

3. Use these to design a better schema
   • One which minimizes the possibility of anomalies
A **functional dependency** is a form of constraint

- Holds on some instances not others.

- Part of the schema, helps define a valid *instance*.

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

Note: The FD \{Course\} -> \{Room\} **holds on this instance**

Recall: an *instance* of a schema is a multiset of tuples conforming to that schema, *i.e. a table*
Note that:

- You can check if an FD is **violated** by examining a single instance;

- However, you **cannot prove** that an FD is part of the schema by examining a single instance.
  
  - *This would require checking every valid instance*

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

However, cannot *prove* that the FD \{Course\} -> \{Room\} is **part of the schema**
More Examples

An FD is a constraint which **holds**, or **does not hold** on an instance:

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

CSC 261, Fall 2017, UR
## More Examples

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

\{\text{Position}\} \rightarrow \{\text{Phone}\}
<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

but not \{Phone\} → \{Position\}
Find at least *three* FDs which hold on this instance:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>
FINDING FUNCTIONAL DEPENDENCIES
What you will learn about in this section

1. “Good” vs. “Bad” FDs: Intuition
2. Finding FDs
3. Closures
We can start to develop a notion of **good** vs. **bad** FDs:

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

Intuitively:

- EmpID -> Name, Phone, Position is **“good FD”**
  - *Minimal redundancy, less possibility of anomalies*
“Good” vs. “Bad” FDs

We can start to develop a notion of **good** vs. **bad** FDs:

<table>
<thead>
<tr>
<th>EmpID</th>
<th>Name</th>
<th>Phone</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0045</td>
<td>Smith</td>
<td>1234</td>
<td>Clerk</td>
</tr>
<tr>
<td>E3542</td>
<td>Mike</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E1111</td>
<td>Smith</td>
<td>9876</td>
<td>Salesrep</td>
</tr>
<tr>
<td>E9999</td>
<td>Mary</td>
<td>1234</td>
<td>Lawyer</td>
</tr>
</tbody>
</table>

Intuitively:

- EmpID -> Name, Phone, Position is **“good FD”**
- But Position -> Phone is a **“bad FD”**
  - **Redundancy!**
  - **Possibility of data anomalies**
“Good” vs. “Bad” FDs

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CS145</td>
<td>B01</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

Given a set of FDs (from user) our goal is to:
1. Find all FDs, and
2. Eliminate the “Bad Ones”.

Returning to our original example... can you see how the “bad FD” {Course} -> {Room} could lead to an:
- Update Anomaly
- Insert Anomaly
- Delete Anomaly
- ...

CSC 261, Fall 2017, UR
• High-level idea: **why do we care about FDs?**

1. Start with some relational *schema*

2. Find out its *functional dependencies (FDs)*

3. **Use these to design a better schema**
   1. One which minimizes possibility of anomalies
Finding Functional Dependencies

• There can be a very large number of FDs…
  – How to find them all efficiently?

• We can’t necessarily show that any FD will hold on all instances…
  – How to do this?

We will start with this problem:
Given a set of FDs, F, what other FDs must hold?
Finding Functional Dependencies

Equivalent to asking: Given a set of FDs, $F = \{f_1, \ldots, f_n\}$, does an FD $g$ hold?

Inference problem: How do we decide?
Finding Functional Dependencies

Example:

<table>
<thead>
<tr>
<th>Name</th>
<th>Color</th>
<th>Category</th>
<th>Dep</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Green</td>
<td>Gadget</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Widget</td>
<td>Black</td>
<td>Gadget</td>
<td>Toys</td>
<td>59</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Green</td>
<td>Whatsit</td>
<td>Garden</td>
<td>99</td>
</tr>
</tbody>
</table>

Given the provided FDs, we can see that \{Name, Category\} → \{Price\} must also hold on any instance...

Which / how many other FDs do?!?
Equivalent to asking: Given a set of FDs, F = \{f_1, \ldots, f_n\}, does an FD g hold?

Inference problem: How do we decide?

Answer: Three simple rules called Armstrong’s Rules.

1. Split/Combine,
2. Reduction, and
3. Transitivity... ideas by picture
1. Split/Combine (Decomposition & Union Rule)

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$\ldots$</th>
<th>$A_m$</th>
<th>$B_1$</th>
<th>$\ldots$</th>
<th>$B_n$</th>
</tr>
</thead>
</table>

$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$
1. Split/Combine (Decomposition & Union Rule)

\[
A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n
\]

... is equivalent to the following \( n \) FDs...

\[
A_1, \ldots, A_m \rightarrow B_i \text{ for } i=1, \ldots, n
\]
1. Split/Combine (Decomposition & Union Rule)

And vice-versa, $A_1, ..., A_m \rightarrow B_i$ for $i=1, ..., n$

... is equivalent to ...

$A_1, ..., A_m \rightarrow B_1, ..., B_n$
2. Reduction/Trivial (Reflexive Rule)

\[ A_1, \ldots, A_m \rightarrow A_j \] for any \( j = 1, \ldots, m \)
3. Transitive Rule

\[ A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n \text{ and } B_1, \ldots, B_n \rightarrow C_1, \ldots, C_k \]
3. Transitive Rule

\[ A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n \text{ and } B_1, \ldots, B_n \rightarrow C_1, \ldots, C_k \]

implies

\[ A_1, \ldots, A_m \rightarrow C_1, \ldots, C_k \]
### Augmentation Rule

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>$\ldots$</th>
<th>$A_m$</th>
<th>$B_1$</th>
<th>$\ldots$</th>
<th>$B_n$</th>
<th></th>
</tr>
</thead>
</table>

$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ implies
### Augmentation Rule

The Augmentation Rule states that if

\[ X_1, A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n \]

implies

\[ X_1, A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n \]

This can be represented in a table as follows:

<table>
<thead>
<tr>
<th>(X_1)</th>
<th>(A_1)</th>
<th>(\ldots)</th>
<th>(A_m)</th>
<th>(B_1)</th>
<th>(\ldots)</th>
<th>(B_n)</th>
</tr>
</thead>
</table>

*Note: The table above illustrates the Augmentation Rule.*
Finding Functional Dependencies

Example:

<table>
<thead>
<tr>
<th>Name</th>
<th>Color</th>
<th>Category</th>
<th>Dep</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Green</td>
<td>Gadget</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Widget</td>
<td>Black</td>
<td>Gadget</td>
<td>Toys</td>
<td>59</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Green</td>
<td>Whatsit</td>
<td>Garden</td>
<td>99</td>
</tr>
</tbody>
</table>

Provided FDs:
1. {Name} → {Color}
2. {Category} → {Department}
3. {Color, Category} → {Price}

Which / how many other FDs hold?
Finding Functional Dependencies

Example:

**Inferred FDs:**

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Rule used</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. {Name, Category} -&gt; {Name}</td>
<td>?</td>
</tr>
<tr>
<td>5. {Name, Category} -&gt; {Color}</td>
<td>?</td>
</tr>
<tr>
<td>6. {Name, Category} -&gt; {Category}</td>
<td>?</td>
</tr>
<tr>
<td>7. {Name, Category} -&gt; {Color, Category}</td>
<td>?</td>
</tr>
<tr>
<td>8. {Name, Category} -&gt; {Price}</td>
<td>?</td>
</tr>
</tbody>
</table>

**Provided FDs:**

1. {Name} -> {Color}
2. {Category} -> {Dept.}
3. {Color, Category} -> {Price}

Which / how many other FDs hold?
Finding Functional Dependencies

Example:

Inferred FDs:

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Rule used</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. {Name, Category} -&gt; {Name}</td>
<td>Trivial</td>
</tr>
<tr>
<td>5. {Name, Category} -&gt; {Color}</td>
<td>Transitive (4 -&gt; 1)</td>
</tr>
<tr>
<td>6. {Name, Category} -&gt; {Category}</td>
<td>Trivial</td>
</tr>
<tr>
<td>7. {Name, Category} -&gt; {Color, Category}</td>
<td>Split/combine (5 + 6)</td>
</tr>
<tr>
<td>8. {Name, Category} -&gt; {Price}</td>
<td>Transitive (7 -&gt; 3)</td>
</tr>
</tbody>
</table>

Provided FDs:

1. {Name} \(\rightarrow\) {Color}
2. {Category} \(\rightarrow\) {Dept.}
3. {Color, Category} \(\rightarrow\) {Price}

Can we find an algorithmic way to do this?
Closures
Closure of a set of Attributes

Given a set of attributes $A_1, ..., A_n$ and a set of FDs $F$: Then the closure, $\{A_1, ..., A_n\}^+$ is the set of attributes $B$ s.t. $\{A_1, ..., A_n\} \rightarrow B$

Example: $F = \{\text{name} \rightarrow \{\text{color}\}, \{\text{category}\} \rightarrow \{\text{department}\}, \{\text{color, category}\} \rightarrow \{\text{price}\}\}$

Example Closures:

$\{\text{name}\}^+ = \{\text{name, color}\}$
$\{\text{name, category}\}^+ = \{\text{name, category, color, dept, price}\}$
$\{\text{color}\}^+ = \{\text{color}\}$
Start with $X = \{A_1, \ldots, A_n\}$ and set of FDs $F$.

Repeat until $X$ doesn’t change; do:

1. if $\{B_1, \ldots, B_n\} \rightarrow C$ is in $F$
2. and $\{B_1, \ldots, B_n\} \subseteq X$

then add $C$ to $X$.

Return $X$ as $X^+$
Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$, FDs $F$.

Repeat until $X$ doesn’t change;
do:
  if $\{B_1, \ldots, B_n\} \rightarrow C$ is in $F$ and $\{B_1, \ldots, B_n\} \subseteq X$:
    then add $C$ to $X$.

Return $X$ as $X^+$

$F = \{\text{name} \rightarrow \text{color}\}$

$\{\text{category} \rightarrow \text{dept}\}$

$\{\text{color, category} \rightarrow \text{price}\}$

$\{\text{name, category}\}^+ = \{\text{name, category}\}$
Closure Algorithm

Start with \( X = \{A_1, \ldots, A_n\} \), FDs \( F \).

Repeat until \( X \) doesn’t change;

\[
\text{do:} \\
\quad \text{if } \{B_1, \ldots, B_n\} \rightarrow C \text{ is in } F \text{ and } \{B_1, \ldots, B_n\} \subseteq X: \\
\quad \quad \text{then add } C \text{ to } X.
\]

Return \( X \) as \( X^+ \)

\[
F = \{\text{name} \rightarrow \{\text{color}\}, \text{category} \rightarrow \{\text{dept}\}, \{\text{color, category}\} \rightarrow \{\text{price}\}\}
\]

\[
\{\text{name, category}\}^+ = \{\text{name, category}\}
\]

\[
\{\text{name, category}\}^+ = \{\text{name, category, color}\}
\]
Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$, FDs $F$.
Repeat until $X$ doesn’t change;
do:
\[ \text{if } \{B_1, \ldots, B_n\} \rightarrow C \text{ is in } F \text{ and } \{B_1, \ldots, B_n\} \subseteq X: \]
\[ \text{then add } C \text{ to } X. \]
Return $X$ as $X^+$

\[ F = \{\text{name} \rightarrow \{\text{color}\}\} \]
\[ \{\text{category}\} \rightarrow \{\text{dept}\}\]
\[ \{\text{color, category}\} \rightarrow \{\text{price}\}\]
Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$, FDs $F$.

Repeat until $X$ doesn’t change;

do:
    if \{B_1, \ldots, B_n\} $\rightarrow$ C is in $F$ and \{B_1, \ldots, B_n\} $\subseteq$ X:
        then add C to X.

Return $X$ as $X^+$

$F =$

\{name\} $\rightarrow$ \{color\}

\{category\} $\rightarrow$ \{dept\}

\{color, category\} $\rightarrow$ \{price\}

\{name, category\}$^+$ = \{name, category\}

\{name, category\}$^+$ = \{name, category, color\}

\{name, category\}$^+$ = \{name, category, color, dept\}

\{name, category\}$^+$ = \{name, category, color, dept, price\}
EXAMPLE

\[ R(A, B, C, D, E, F) \]

\[ \begin{align*}
\{A, B\} & \rightarrow \{C\} \\
\{A, D\} & \rightarrow \{E\} \\
\{B\} & \rightarrow \{D\} \\
\{A, F\} & \rightarrow \{B\}
\end{align*} \]

Compute \( \{A, B\}^+ = \{A, B, \} \)

Compute \( \{A, F\}^+ = \{A, F, \} \)
**EXAMPLE**

\[ R(A, B, C, D, E, F) \]

- \( \{A, B\} \rightarrow \{C\} \)
- \( \{A, D\} \rightarrow \{E\} \)
- \( \{B\} \rightarrow \{D\} \)
- \( \{A, F\} \rightarrow \{B\} \)

Compute \( \{A, B\}^+ = \{A, B, C, D\} \)

Compute \( \{A, F\}^+ = \{A, F, B\} \)
EXAMPLE

Compute \( \{A,B\}^+ = \{A, B, C, D, E\} \)

Compute \( \{A, F\}^+ = \{A, B, C, D, E, F\} \)
3. CLOSURES, SUPERKEYS & KEYS
What you will learn about in this section

1. Closures
2. Superkeys & Keys
Why Do We Need the Closure?

• With closure we can find all FD’s easily

• To check if $X \rightarrow A$

  1. Compute $X^+$

  2. Check if $A \in X^+$

Note here that $X$ is a set of attributes, but $A$ is a single attribute. Why does considering FDs of this form suffice?

Recall the Split/combine rule:

$X \rightarrow A_1, \ldots, X \rightarrow A_n$

implies

$X \rightarrow \{A_1, \ldots, A_n\}$
Using Closure to Infer ALL FDs

Step 1: Compute $X^+$, for every set of attributes $X$:

$\{A\}^+ = \{A\}$
$\{B\}^+ = \{B, D\}$
$\{C\}^+ = \{C\}$
$\{D\}^+ = \{D\}$
$\{A, B\}^+ = \{A, B, C, D\}$
$\{A, C\}^+ = \{A, C\}$
$\{A, D\}^+ = \{A, B, C, D\}$
$\{A, B, C\}^+ = \{A, B, D\}^+ = \{A, C, D\}^+ = \{A, B, C, D\}$
$\{B, C, D\}^+ = \{B, C, D\}$
$\{A, B, C, D\}^+ = \{A, B, C, D\}$

Example:
Given $F = \{A, B \rightarrow C, A, D \rightarrow B, B \rightarrow D\}$

No need to compute these - why?
Using Closure to Infer ALL FDs

Example:

Given F =

{A, B} → C
{A, D} → B
{B} → D

Step 1: Compute $X^+$, for every set of attributes $X$:

\[
\begin{align*}
{A}^+ &= \{A\}, \quad {B}^+ = \{B, D\}, \quad {C}^+ = \{C\}, \quad {D}^+ = \{D\},
\{A, B\}^+ = \{A, B, C, D\}, \quad \{A, C\}^+ = \{A, C\},
\{A, D\}^+ = \{A, B, C, D\}, \quad \{A, B, C\}^+ = \{A, B, D\}^+ = \{A, B, C, D\},
\{A, C, D\}^+ = \{A, B, C, D\}, \quad \{B, C, D\}^+ = \{B, C, D\},
\{A, B, C, D\}^+ = \{A, B, C, D\}.
\end{align*}
\]

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

\[
\begin{align*}
\{A, B\} &\rightarrow \{C, D\}, \quad \{A, D\} \rightarrow \{B, C\},
\{A, B, C\} \rightarrow \{B, C\}, \quad \{A, B, D\} \rightarrow \{C\},
\{A, C, D\} \rightarrow \{B\}.
\end{align*}
\]
Using Closure to Infer ALL FDs

Step 1: Compute $X^+$, for every set of attributes $X$:

\[
egin{align*}
\{A\}^+ &= \{A\}, \quad \{B\}^+ = \{B,D\}, \quad \{C\}^+ = \{C\}, \quad \{D\}^+ = \{D\}, \quad \{A,B\}^+ = \{A,B,C,D\}, \quad \{A,C\}^+ = \{A,C\}, \\
\{A,D\}^+ &= \{A,B,C,D\}, \quad \{A,B,C\}^+ = \{A,B,D\}^+ = \{A,C,D\}^+ = \{A,B,C,D\}, \quad \{B,C,D\}^+ = \{B,C,D\}, \\
\{A,B,C,D\}^+ &= \{A,B,C,D\}
\end{align*}
\]

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

\[
\begin{align*}
\{A,B\} &\rightarrow \{C,D\}, \quad \{A,D\} \rightarrow \{B,C\}, \\
\{A,B,C\} &\rightarrow \{D\}, \quad \{A,B,D\} \rightarrow \{C\}, \\
\{A,C,D\} &\rightarrow \{B\}
\end{align*}
\]

Example:

Given $F =$

\[
\begin{align*}
\{A,B\} &\rightarrow C \\
\{A,D\} &\rightarrow B \\
\{B\} &\rightarrow D
\end{align*}
\]

"$Y$ is in the closure of $X$"
Using Closure to Infer ALL FDs

Step 1: Compute $X^+$, for every set of attributes $X$:

<table>
<thead>
<tr>
<th>$X$</th>
<th>$X^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${A}$</td>
<td>${A}$</td>
</tr>
<tr>
<td>${B}$</td>
<td>${B,D}$</td>
</tr>
<tr>
<td>${C}$</td>
<td>${C}$</td>
</tr>
<tr>
<td>${D}$</td>
<td>${D}$</td>
</tr>
<tr>
<td>${A,B}$</td>
<td>${A,B,C,D}$</td>
</tr>
<tr>
<td>${A,C}$</td>
<td>${A,C}$</td>
</tr>
<tr>
<td>${A,D}$</td>
<td>${A,B,C,D}$</td>
</tr>
<tr>
<td>${B}$</td>
<td>${B,D}$</td>
</tr>
<tr>
<td>${C}$</td>
<td>${C}$</td>
</tr>
<tr>
<td>${D}$</td>
<td>${D}$</td>
</tr>
<tr>
<td>${A,B,C}$</td>
<td>${A,B,C,D}$</td>
</tr>
<tr>
<td>${A,B,D}$</td>
<td>${A,B,C,D}$</td>
</tr>
<tr>
<td>${A,C,D}$</td>
<td>${A,B,C,D}$</td>
</tr>
<tr>
<td>${B,C,D}$</td>
<td>${B,C,D}$</td>
</tr>
<tr>
<td>${A,B,C,D}$</td>
<td>${A,B,C,D}$</td>
</tr>
</tbody>
</table>

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${A,B}$</td>
<td>${C,D}$</td>
</tr>
<tr>
<td>${A,D}$</td>
<td>${B,C}$</td>
</tr>
<tr>
<td>${A,B,C}$</td>
<td>${D}$</td>
</tr>
<tr>
<td>${A,B,D}$</td>
<td>${C}$</td>
</tr>
<tr>
<td>${A,C,D}$</td>
<td>${B}$</td>
</tr>
</tbody>
</table>

Example:

Given $F = \{\{A,B\} \rightarrow C, \{A,D\} \rightarrow B, \{B\} \rightarrow D\}$

The FD $X \rightarrow Y$ is non-trivial.
Superkeys and Keys
A **superkey** is a set of attributes $A_1, \ldots, A_n$ s.t. for *any other* attribute $B$ in $R$, we have $\{A_1, \ldots, A_n\} \rightarrow B$.

A **key** is a *minimal* superkey.

I.e. all attributes are *functionally determined* by a superkey.

Meaning that no subset of a key is also a superkey.
Finding Keys and Superkeys

• For each set of attributes $X$

1. Compute $X^+$

2. If $X^+ = \text{set of all attributes}$ then $X$ is a superkey

3. If $X$ is minimal, then it is a key

Do we need to check all sets of attributes? Which sets?
Example of Finding Keys

Product(name, price, category, color)

{name, category} → price
{category} → color

What is a key?
Example of Keys

Product(name, price, category, color)

{name, category} → price
{category} → color

{name, category}⁺ = {name, price, category, color}
= the set of all attributes
⇒ this is a superkey
⇒ this is a key, since neither name nor category alone is a superkey
Acknowledgement

• Some of the slides in this presentation are taken from the slides provided by the authors.
• Many of these slides are taken from cs145 course offered by Stanford University.
• Thanks to YouTube, especially to Dr. Daniel Soper for his useful videos.