Lecture 6: Design Theory
Announcements

• Solutions to PS1 are posted online. Grades coming soon!

• Project part 1 is out.
  • Check your groups and let us know if you have any issues.
  • We have assigned people to groups that had only two members.

• Activities and Notebooks are there for your benefit!
Lecture 6: Design Theory I
Today’s Lecture

1. Normal forms & functional dependencies
   • ACTIVITY: Finding FDs

2. Finding functional dependencies

3. Closures, superkeys & keys
   • ACTIVITY: The key or a key?
1. Normal forms & functional dependencies
What you will learn about in this section

1. Overview of design theory & normal forms

2. Data anomalies & constraints

3. Functional dependencies

4. ACTIVITY: Finding FDs
Design Theory

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

• It is a mostly mechanical process
  • Tools can carry out routine portions

• *We have a notebook implementing all algorithms!*
  • We’ll play with it in the activities!
Normal Forms

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

• 2\textsuperscript{nd} Normal Form = disused

• Boyce-Codd Normal Form (BCNF)

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

• 4\textsuperscript{th} and 5\textsuperscript{th} Normal Forms = see text books
1st Normal Form (1NF)

Violates 1NF.

1NF Constraint: Types must be atomic!

<table>
<thead>
<tr>
<th>Student</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>{CS564,CS368}</td>
</tr>
<tr>
<td>Joe</td>
<td>{CS564,CS552}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS564</td>
</tr>
<tr>
<td>Mary</td>
<td>CS368</td>
</tr>
<tr>
<td>Joe</td>
<td>CS564</td>
</tr>
<tr>
<td>Joe</td>
<td>CS552</td>
</tr>
</tbody>
</table>

In 1st NF
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>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CS564</td>
<td>B01</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>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS564</td>
<td>C12</td>
</tr>
<tr>
<td>Sam</td>
<td>CS564</td>
<td>B01</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*. 
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>..</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>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CS564</td>
<td>B01</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>CS564</td>
</tr>
<tr>
<td>Joe</td>
<td>CS564</td>
</tr>
<tr>
<td>Sam</td>
<td>CS564</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>CS368</td>
<td>C12</td>
</tr>
</tbody>
</table>

Is this form better?

- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

Today: develop theory to understand why this design may be better and how to find this decomposition...

Functional Dependencies
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 $t_2$:

$$t_1[A] = t_2[A] \text{ implies } t_1[B] = t_2[B]$$

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

$A \rightarrow B$ means that

“whenever two tuples agree on A then they agree on B.”
A Picture Of FDs

Defn (again):

Given attribute sets $A=\{A_1, \ldots, A_m\}$ and $B = \{B_1, \ldots, B_n\}$ in $R$, 

| $A_1$ | $\ldots$ | $A_m$ | $B_1$ | $\ldots$ | $B_n$ |
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 $R$: 

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 $R$:

$t_i[A_1] = t_j[A_1]$ AND $t_i[A_2] = t_j[A_2]$ AND ... AND $t_i[A_m] = t_j[A_m]$

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>...</th>
<th>$A_m$</th>
<th>$B_1$</th>
<th>...</th>
<th>$B_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_i$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t_j$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 $R$:

If $t_i[A_1] = t_j[A_1]$ AND $t_i[A_2] = t_j[A_2]$ AND ... AND $t_i[A_m] = t_j[A_m]$

then $t_i[B_1] = t_j[B_1]$ AND $t_i[B_2] = t_j[B_2]$ AND ... AND $t_i[B_n] = t_j[B_n]$
FDs for Relational Schema Design

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

1. Start with some relational schema

2. Model its functional dependencies (FDs)

3. Use these to design a better schema
   1. 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*.

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

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

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

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>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

However, cannot **prove** that the FD \( \{ \text{Course} \} \rightarrow \{ \text{Room} \} \) 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>
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>

\{Position\} $\rightarrow$ \{Phone\}
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>

but not \{\text{Phone}\} \rightarrow \{\text{Position}\}
**ACTIVITY**

<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>

Find at least *three* FDs which are violated on this instance:

\[
\{ \} \to \{ \}
\]

\[
\{ \} \to \{ \}
\]

\[
\{ \} \to \{ \}
\]
2. Finding functional dependencies
What you will learn about in this section

1. “Good” vs. “Bad” FDs: Intuition
2. Finding FDs
3. Closures
4. ACTIVITY: Compute the closures
“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”
- 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>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Joe</td>
<td>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>Sam</td>
<td>CS564</td>
<td>B01</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

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

- Update Anomaly
- Insert Anomaly
- Delete Anomaly
- ...

Given a set of FDs (from user) our goal is to:
1. Find all FDs, and
2. Eliminate the “Bad Ones".
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
   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\} \rightarrow \{Price\} must also hold on any instance...
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?

Answer: Three simple rules called **Armstrong’s Rules**.

1. Split/Combine,
2. Reduction, and
3. Transitivity... *ideas by picture*
1. Split/Combine

<table>
<thead>
<tr>
<th>A_1</th>
<th>...</th>
<th>A_m</th>
<th>B_1</th>
<th>...</th>
<th>B_n</th>
</tr>
</thead>
</table>

A_1, ..., A_m \rightarrow B_1, ..., B_n
1. Split/Combine

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

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

\[
A_1, \ldots, A_m \rightarrow B_i \text{ for } i=1,\ldots,n
\]
1. Split/Combine

<table>
<thead>
<tr>
<th>A₁</th>
<th>...</th>
<th>Aₘ</th>
<th>B₁</th>
<th>...</th>
<th>Bₙ</th>
</tr>
</thead>
</table>

*And vice-versa, A₁,...,Aₘ → Bᵢ for i=1,...,n*

... is equivalent to ...

A₁, ..., Aₘ → B₁,...,Bₙ
2. Reduction/Trivial

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>...</th>
<th>$A_m$</th>
</tr>
</thead>
</table>

$A_1,\ldots,A_m \rightarrow A_j$ for any $j=1,\ldots,m$
### 3. Transitive Closure

<table>
<thead>
<tr>
<th></th>
<th>$A_1$</th>
<th>...</th>
<th>$A_m$</th>
<th>$B_1$</th>
<th>...</th>
<th>$B_n$</th>
<th>$C_1$</th>
<th>...</th>
<th>$C_k$</th>
</tr>
</thead>
</table>

$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ and

$B_1, \ldots, B_n \rightarrow C_1, \ldots, C_k$
### 3. Transitive Closure

A_1, ..., A_m \rightarrow B_1, ..., B_n and B_1, ..., B_n \rightarrow C_1, ..., C_k

implies

A_1, ..., A_m \rightarrow C_1, ..., C_k
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\} \rightarrow \{Color\}
2. \{Category\} \rightarrow \{Department\}
3. \{Color, Category\} \rightarrow \{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} \rightarrow {Name}</td>
<td>?</td>
</tr>
<tr>
<td>5. {Name, Category} \rightarrow {Color}</td>
<td>?</td>
</tr>
<tr>
<td>6. {Name, Category} \rightarrow {Category}</td>
<td>?</td>
</tr>
<tr>
<td>7. {Name, Category} \rightarrow {Color, Category}</td>
<td>?</td>
</tr>
<tr>
<td>8. {Name, Category} \rightarrow {Price}</td>
<td>?</td>
</tr>
</tbody>
</table>

Provided FDs:

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

Which / how many other FDs hold?
# Finding Functional Dependencies

## Example:

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Rule used</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. {Name, Category} \rightarrow {Name}</td>
<td>Trivial</td>
</tr>
<tr>
<td>5. {Name, Category} \rightarrow {Color}</td>
<td>Transitive (4 \rightarrow 1)</td>
</tr>
<tr>
<td>6. {Name, Category} \rightarrow {Category}</td>
<td>Trivial</td>
</tr>
<tr>
<td>7. {Name, Category} \rightarrow {Color, Category}</td>
<td>Split/combine (5 + 6)</td>
</tr>
<tr>
<td>8. {Name, Category} \rightarrow {Price}</td>
<td>Transitive (7 \rightarrow 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 $A_1, \ldots, A_n$ and a set of FDs $F$:
Then the closure, $\{A_1, \ldots, A_n\}^+$ is the set of attributes $B$ s.t. $\{A_1, \ldots, A_n\} \rightarrow B$

Example: $F = \{$

<table>
<thead>
<tr>
<th>Attribute</th>
<th>→</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>{name}</td>
<td></td>
<td>{color}</td>
</tr>
<tr>
<td>{category}</td>
<td></td>
<td>{department}</td>
</tr>
<tr>
<td>{color, category}</td>
<td></td>
<td>{price}</td>
</tr>
</tbody>
</table>

Example Closures:

<table>
<thead>
<tr>
<th>Attribute Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>{name}</td>
</tr>
<tr>
<td>{color}</td>
</tr>
<tr>
<td>{name, category}</td>
</tr>
<tr>
<td>{name, category, color, dept, price}</td>
</tr>
<tr>
<td>{color}</td>
</tr>
</tbody>
</table>
Closure Algorithm

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

Repeat until $X$ doesn’t change; do:

- if $\{B_1, \ldots, B_n\} \rightarrow C$ is entailed by $F$
  
  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 \( X^+ \)

\[ F = \{\text{name} \rightarrow \{\text{color}\}, \{\text{category} \rightarrow \{\text{dept}\}, \{\text{color, category} \rightarrow \{\text{price}\}} \right] \]
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 \( X^+ \).

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

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

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

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

Repeat until \( X \) doesn’t change; do:

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 \( X^+ \)

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

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

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

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

Start with $X = \{A_1, ..., A_n\}$, FDs $F$.
Repeat until $X$ doesn’t change; do:
  if $\{B_1, ..., B_n\} \rightarrow C$ is in $F$ and $\{B_1, ..., B_n\} \subseteq X$:
    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

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

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

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

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

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

1. Closures Pt. II

2. Superkeys & Keys

3. ACTIVITY: The key or a key?
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.

Recall the **Split/combine** rule:

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

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

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

Example:

Given $F =$ 

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

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

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

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

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

$\{A,B\} \rightarrow \{C,D\}$, $\{A,D\} \rightarrow \{B,C\}$,
$\{A,B,C\} \rightarrow \{D\}$, $\{A,B,D\} \rightarrow \{C\}$,
$\{A,C,D\} \rightarrow \{B\}$
Using Closure to Infer ALL FDs

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


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

$\{A,B\} \rightarrow \{C,D\}$, $\{A,D\} \rightarrow \{B,C\}$,
$\{A,B,C\} \rightarrow \{D\}$, $\{A,B,D\} \rightarrow \{C\}$,
$\{A,C,D\} \rightarrow \{B\}$.

Example:

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

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

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

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

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

- $\{A, B\} \rightarrow \{C, D\}$,
- $\{A, D\} \rightarrow \{B, C\}$,
- $\{A, B, C\} \rightarrow \{D\}$,
- $\{A, B, D\} \rightarrow \{C\}$,
- $\{A, C, D\} \rightarrow \{B\}$

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 $A_1, ..., A_n$ s.t. for any other attribute $B$ in $R$, we have $\{A_1, ..., A_n\} \rightarrow B$

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
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)

\{\text{name, category}\} \rightarrow \text{price}

\{\text{category}\} \rightarrow \text{color}

\{\text{name, category}\}^+ = \{\text{name, price, category, color}\}

= \text{the set of all attributes}

\Rightarrow \text{this is a superkey}

\Rightarrow \text{this is a key, since neither name nor category alone is a superkey}
Activity-6.ipynb