CS460: Intro to Database Systems

Class 6: *Functional Dependencies*

Instructor: Manos Athanassoulis

https://bu-disc.github.io/CS460/
Review: Database Design

Requirements Analysis

user needs; what must database do?

Conceptual Design

high level description (often done w/ ER model)

Logical Design

translate ER into DBMS data model

Schema Refinement

consistency, normalization

Physical Design

indexes, disk layout
Review: Database Design

Requirements Analysis
user needs; what must database do?

Conceptual Design
high level description (often done w/ ER model)

Logical Design
translate ER into DBMS data model

Schema Refinement
consistency, normalization

Physical Design
indexes, disk layout
Why schema refinement

what is a bad schema?

*a schema with redundancy*

why?

redundant storage & insert/update/delete anomalies

how to fix it?

*normalize* the schema by decomposing normal forms: BCNF, 3NF, … [next time]
Motivating Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>987-00-8761</td>
<td>John</td>
<td>65K</td>
<td>857-555-1234</td>
</tr>
<tr>
<td>987-00-8761</td>
<td>John</td>
<td>65K</td>
<td>857-555-8800</td>
</tr>
<tr>
<td>123-00-9876</td>
<td>Anna</td>
<td>80K</td>
<td>617-555-9876</td>
</tr>
<tr>
<td>787-00-4321</td>
<td>Kurt</td>
<td>25K</td>
<td>617-555-3761</td>
</tr>
</tbody>
</table>

primary key?  (SSN,Telephone)

problems of the schema?
Motivating Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>987-00-8761</td>
<td>John</td>
<td>65K</td>
<td>857-555-1234</td>
</tr>
<tr>
<td>987-00-8761</td>
<td>John</td>
<td>65K</td>
<td>857-555-8800</td>
</tr>
<tr>
<td>123-00-9876</td>
<td>Anna</td>
<td>80K</td>
<td>617-555-9876</td>
</tr>
<tr>
<td>787-00-4321</td>
<td>Kurt</td>
<td>25K</td>
<td>617-555-3761</td>
</tr>
</tbody>
</table>

Problems

Storage
Update
Insert
Delete
Motivating Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>987-00-8761</td>
<td>John</td>
<td>65K</td>
<td>857-555-1234</td>
</tr>
<tr>
<td>987-00-8761</td>
<td>John</td>
<td>65K</td>
<td>857-555-8800</td>
</tr>
<tr>
<td>123-00-9876</td>
<td>Anna</td>
<td>80K</td>
<td>617-555-9876</td>
</tr>
<tr>
<td>787-00-4321</td>
<td>Kurt</td>
<td>25K</td>
<td>617-555-3761</td>
</tr>
</tbody>
</table>

Problems

Storage: store *Salary* multiple times
Update: change John’s salary?
Insert: how to store someone with *no phone*?
Delete: how to delete Kurt’s phone?
Solution: Decomposition

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>987-00-8761</td>
<td>John</td>
<td>65K</td>
<td>857-555-1234</td>
</tr>
<tr>
<td>987-00-8761</td>
<td>John</td>
<td>65K</td>
<td>857-555-8800</td>
</tr>
<tr>
<td>123-00-9876</td>
<td>Anna</td>
<td>80K</td>
<td>617-555-9876</td>
</tr>
<tr>
<td>787-00-4321</td>
<td>Kurt</td>
<td>25K</td>
<td>617-555-3761</td>
</tr>
</tbody>
</table>

**can decomposition cause problems?**

**how to find good decompositions?**
FUNCTIONAL DEPENDENCIES
Functional Dependencies

**Definition**

Functional Dependencies (FDs) : form of constraint  
“generalized keys”

let $X, Y$ nonempty sets of attributes of relation $R$  
let $t_1, t_2$ tuples : if $t_1.X = t_2.X$, then $t_1.Y = t_2.Y$

“$X \rightarrow Y$”: ”$X$ (functionally) determines $Y$”

an FD comes from the application (not the data)  
an FD cannot be inferred (only validated)
Functional Dependencies

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>987-00-8761</td>
<td>John</td>
<td>65K</td>
<td>857-555-1234</td>
</tr>
<tr>
<td>987-00-8761</td>
<td>John</td>
<td>65K</td>
<td>857-555-8800</td>
</tr>
<tr>
<td>123-00-9876</td>
<td>Anna</td>
<td>80K</td>
<td>617-555-9876</td>
</tr>
<tr>
<td>787-00-4321</td>
<td>Kurt</td>
<td>25K</td>
<td>617-555-3761</td>
</tr>
</tbody>
</table>

which attribute determines which?

- **SSN ➔ Telephone**
- **SSN ➔ Name, Salary**
- **SSN, Salary ➔ Name**

they both *might* be true

actually, the second can be inferred if the first is true! (more on that later)
### FD: Example 3

<table>
<thead>
<tr>
<th>studentID</th>
<th>classID</th>
<th>Semester</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>15</td>
<td>2</td>
<td>Mark</td>
</tr>
<tr>
<td>0043</td>
<td>15</td>
<td>1</td>
<td>Evimaria</td>
</tr>
<tr>
<td>4322</td>
<td>15</td>
<td>2</td>
<td>Mark</td>
</tr>
<tr>
<td>9876</td>
<td>175</td>
<td>4</td>
<td>Dora</td>
</tr>
<tr>
<td>1211</td>
<td>177</td>
<td>4</td>
<td>Manos</td>
</tr>
<tr>
<td>0043</td>
<td>154</td>
<td>2</td>
<td>Abraham</td>
</tr>
</tbody>
</table>

Which attribute determines which?

- `classID, Semester → Instructor`
- `studentID → Semester`
- `studentID, classID → Semester`
Reasoning about FDs

an FD holds for all allowable relations (legal)
identified based on semantics of application

given an instance r of R and an FD f:
(1) we can check whether the instance r violates FD f
(2) we cannot determine if f holds

“K → all attributes of R” then K is a superkey for R
(does not require K to be minimal)
remember: in order to be a candidate key minimality is required

FDs are a generalization of keys
Reasoning about FDs (Splitting)

assume $A, B \rightarrow C, D$

$C, D$ are *independently determined* by $A, B$
so, we can split: $A, B \rightarrow C$ and $A, B \rightarrow D$

it does **not** work vice versa
we **cannot** infer: $A \rightarrow C, D$ or $B \rightarrow C, D$
Trivial FDs

for every relation

A → A
A, B, C → A
these are not informative!

in general an FD X → A is called \textit{trivial} if A \subseteq X

it always holds!
Identifying FDs

FD comes from the application (domain) property of app semantics (not of instance) cannot infer from an instance

given a set of tuples (instance r), we can:
(1) confirm that an FD might be valid
(2) infer that an FD is definitely invalid

but we cannot prove that an FD is valid
# FD: Example 3

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>price</th>
<th>department</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>smartphone</td>
<td>black</td>
<td>600</td>
<td>phones</td>
</tr>
<tr>
<td>Lenovo Yoga</td>
<td>laptop</td>
<td>grey</td>
<td>800</td>
<td>computers</td>
</tr>
<tr>
<td>unifi</td>
<td>networking</td>
<td>white</td>
<td>150</td>
<td>computers</td>
</tr>
<tr>
<td>unifi</td>
<td>cables</td>
<td>white</td>
<td>10</td>
<td>stationary</td>
</tr>
<tr>
<td>OnePlus</td>
<td>smartphone</td>
<td>silver</td>
<td>450</td>
<td>phones</td>
</tr>
</tbody>
</table>

- **name** ➔ **department**

- **name, category** ➔ **department**

*maybe! we do not know!*
Why use FDs?

the capture (and generalize) key constraints

offer more integrity constraints

help us detect redundancies
tell us how to normalize

it is the principled way to solve the redundancy problem
More on: Reasoning for FD

when a set of FD holds over a relation

more FD can be inferred

Armstrong’s Axioms
Axiom 1: Reflexivity

for every subset $X \subseteq \{A_1, ..., A_n\}$

$$A_1, ..., A_n \rightarrow X$$

**Examples**

- $A, B \rightarrow B$
- $A, B, C \rightarrow B, C$
- $A, B, C \rightarrow A, B, C$
Axiom 2: Augmentation

for any attribute sets $X$, $Y$, $Z$
if $X \rightarrow Y$, then $X, Z \rightarrow Y, Z$

Examples

$A \rightarrow B$ then $A, C \rightarrow B, C$

$A, B \rightarrow C$ then $A, B, C \rightarrow C$

(here $X=A,B$ and $Y=Z=C$)
Axiom 3: Transitivity

for any attribute sets X, Y, Z
if X → Y and Y → Z then X → Z

Examples
A → B and B → C then A → C
A → B, C and B, C → D then A → D
Union and Decomposition

rules that follow from AA

Union

if $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow Y, Z$

Decomposition

if $X \rightarrow Y, Z$ then $X \rightarrow Y$ and $X \rightarrow Z$
Applying AA

### Product

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>price</th>
<th>department</th>
</tr>
</thead>
</table>

**we know:**

1. name → color
2. category → department
3. color, category → price

**can we infer:** name, category → price

(i) augmentation to (1):
   - (4) name, category → color, category

(ii) transitivity to (4), (3)
   - name, category → price
Applying AA

Product

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>price</th>
<th>department</th>
</tr>
</thead>
</table>

we know:

(1) name $\rightarrow$ color
(2) category $\rightarrow$ department
(3) color, category $\rightarrow$ price

can we infer: name, category $\rightarrow$ color

(i) by reflexivity:
   (5) name, category $\rightarrow$ name
(ii) transitivity to (5), (1)
     name, category $\rightarrow$ color
FD Closure

how can we find all FD?

**FD Closure**

if $F$ is a set of FD, the closure $F^+$ is the set of all FDs logically implied by $F$

*Using Armstrong Axioms we can find $F^+$*

- **sound**: any generated FD belongs to $F^+$
- **complete**: repeated application of AA generates $F^+$
Attribute Closure

X an attribute set, the closure $X^+$ is the set of all attributes $B : X \rightarrow B$

in other words:
attribute closure of X is the set of all attributes that “are (functionally) determined by X”
Applying AA

Product

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>price</th>
<th>department</th>
</tr>
</thead>
</table>

we know:

(1)  name $\rightarrow$ color
(2)  category $\rightarrow$ department
(3)  color, category $\rightarrow$ price

Attribute closure:

(i)  Closure of name
     $\{\text{name}\}^+ = \{\text{name}, \text{color}\}$

(i)  Closure of name, category
     $\{\text{name}, \text{category}\}^+ = \{\text{name}, \text{color}, \text{category}, \text{department}, \text{price}\}$
Calculating Attribute Closure

let \( X = \{ A_1, \ldots, A_n \} \)

\[
\text{closure} = X
\]

\textbf{UNTIL} closure does not change \textbf{REPEAT:}

\textbf{IF} \( B_1, \ldots, B_m \rightarrow C \) \textbf{AND}

\( B_1, \ldots, B_m \) are all in closure

\textbf{THEN} add \( C \) to closure
Calculating Attribute Closure

Example: $R(A,B,C,D,E,F)$

- $A, B \rightarrow C$
- $A, D \rightarrow E$
- $B \rightarrow D$
- $A, F \rightarrow B$

$\{A,B\}^+$
$\{A,F\}^+$
Calculating Attribute Closure

Example: \( R(A,B,C,D,E,F) \)

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

\( \{A, B\} \)

\( \{A, B, C\} \)

\( \{A, B, C, D\} \)

\( \{A, B, C, D, E\} \)

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

\( \{A, F\}^+ \)
Calculating Attribute Closure

Example: $R(A,B,C,D,E,F)$

- $A, B \rightarrow C$  \quad \{A,B,C\}$
- $A, D \rightarrow E$  \quad \{A,B,C,D\}$
- $B \rightarrow D$  \quad \{A,B,C,D,E\}$
- $A, F \rightarrow B$  \quad \{A,F\}$

$\{A,B\}^+$  \quad \{A,F,B\}$

$\{A,F\}^+$  \quad \{A,F,B,C\}$

\{A,F,B,C,D\}$

\{A,F,B,C,D,E\}$
Calculating Attribute Closure

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 \\
\{A,B\}^+ & = \{A,B,C,D,E\} \\
\{A,F\}^+ & = \{A,F,B,C,D,E\}
\end{align*}
Why calculate attribute closure?

for “does $X \rightarrow Y$ hold” questions check if $Y \subseteq X^+$

To compute the closure $F^+$ of FDs
(i) for each subset of attributes $X$, compute $X^+$
(ii) for each subset of attributes $Y \subseteq X^+$, output the FD $X \rightarrow Y$

*why do we need the FD closure?*

*to decide on decomposition (next time)*
FD and Keys

*in terms of relational model*

**superkey**: a set of attributes such that:
no two distinct tuples can have same values in all key fields

*in terms of FD*

**superkey**: a set of attributes $A_1, A_2, \ldots, A_n$ such that
for any attribute $B$: $A_1, A_2, \ldots, A_n \rightarrow B$

**key (or candidate key)**: requires minimality
what if we have multiple candidate keys?
- we specify one to be the **primary key**
Computing (Super)Keys

(1) compute $X^+$ for all sets of attributes $X$

(2) if $X^+$=all attributes, then $X$ is a superkey
   why?
      - because then "X determines `all attributes`"

(3) if, also, no subset of $X$ is superkey
    then $X$ is also a key
Example

Product

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>price</th>
</tr>
</thead>
</table>

we know:

(1) name → color
(2) color, category → price

Superkeys:

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

Keys:

{name, category}
Can we have more than 1 key?

what about the relation \( R \ (A,B,C) \) with:

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

which are the keys?

\( \{A, B\} \) and \( \{A, C\} \) are both minimal
Should we use all FDs?

given a set of FDs $F$ we have discussed about $F^+$

the useful info is in the minimal cover of $F$

“the smallest subset of FDs $S$: $S^+ = F^+$“

Formally: minimal cover $S$ for a set of FDs $F$:

(1) $S^+ = F^+$

(2) RHS of each FD in $S$ is a single attribute

(3) if we remove any FD from $S$ or remove any attribute from its LHS the closure is not $F^+$
F closure: $F^+$

Initial Set of FDs: $F$

Minimal Cover: $S$

$S^+ = F^+$

*here, the subset notation is meant as “simpler set of FDs”*
Example of Minimal Cover

\[ R(C, S, J, D, P, Q, V) \]

key \( C \) (\( C^+ = \{C, S, J, D, P, Q, V\} \))

\( J, P \rightarrow C \)

\( S, D \rightarrow P \)

\( J \rightarrow S \)

Minimal cover:

\( J, P \rightarrow C \)

\( S, D \rightarrow P \)

\( J \rightarrow S \)

\( C \rightarrow C, C \rightarrow S, C \rightarrow J, C \rightarrow D, C \rightarrow P, C \rightarrow Q, C \rightarrow V \)

(1) put FDs in standard form
   single attribute on the RHS using decomposition

(2) minimize the LHS
   check if by removing attr. equivalence is preserved

(3) delete redundant FDs
Example of Minimal Cover

\( R(C, S, J, D, P, Q, V) \)

key C (\( C^+ = \{C, S, J, D, P, Q, V\} \))

J, P → C

S, D → P

J → S

Minimal cover:

J, P → C

S, D → P

J → S

C → J, C → P, C → Q, C → V

(1) put FDs in standard form
   single attribute on the RHS using decomposition

(2) minimize the LHS
   check if by removing attr. equivalence is preserved

(3) delete redundant FDs

This is useful to decide how to solve the problem of redundancy (decomposition)!

More on that next time!!
Summary

Functional Dependencies and (Super)Keys

Reasoning with FDs:
(1) given a set of FDs, infer all implied FDs
(2) given a set of attributes X, infer all attributes that are functionally determined by X

Next: how to use FDs to detect that a table is “bad”?