CS460: Intro to Database Systems

Class 6: Functional Dependencies

Instructor: Manos Athanassoulis

https://midas.bu.edu/classes/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 t₁, t₂ tuples : t₁.X = t₂.X, then t₁.Y = t₂.Y

“X → 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
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>115</td>
<td>6</td>
<td>Manos</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>Jonathan</td>
</tr>
</tbody>
</table>

which attribute determines which?

\[
\text{classID, Semester} \rightarrow \text{Instructor}
\]

\[
\text{studentID} \rightarrow \text{Semester}
\]

\[
\text{studentID, classID} \rightarrow \text{Semester}
\]
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 $r$ violates $f$
(2) we *cannot* determine if $f$ holds

“$K \rightarrow$ 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 → C, D

C, D are independently determined by A,B so, we can split: A, B → C and A, B → D

it does not work vice versa

we cannot infer: A → C, D or B → 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 *trivial* if A⊆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 \textbf{might be} valid
(2) infer that an FD is \textbf{definitely invalid}

\textbf{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>

We do not know!

name → department
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 $X \subseteq \{A_1, \ldots, A_n\}$

$A_1, \ldots, A_n \rightarrow X$

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

Examples
A $\rightarrow$ B and B $\rightarrow$ C then A $\rightarrow$ C
A $\rightarrow$ B, C and B, C $\rightarrow$ D then A $\rightarrow$ D
Union and Decomposition

rules that follow from AA

**Union**

if X → Y and X → Z then X → Y, Z

**Decomposition**

if X → Y, Z then X → Y and X → 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 → color
(2) category → department
(3) color, category → price

can we infer: name, category → color

(i) by reflexivity:
(5) name, category → name

(ii) transitivity to (5), (1)

name, category → 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 → color
(2) category → department
(3) color, category → price

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

let $X=\{A_1, ..., A_n\}$

closure = $X$

UNTIL closure does not change REPEAT:

IF $B_1, ..., B_m \rightarrow C$ AND

$B_1, ..., B_m$ are all in closure

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

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

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

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

\[
\begin{align*}
\{A, B\} & \rightarrow \{C\} \\
\{A, D\} & \rightarrow \{E\} \\
\{B\} & \rightarrow \{D\} \\
\{A, F\} & \rightarrow \{B\} \\
\{A, B\}^+ & \rightarrow \{A, B, C\} \\
\{A, F\}^+ & \rightarrow \{A, F, B, C, D, E\}
\end{align*}
\]
Calculating Attribute Closure

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

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

\[
\{A,B\}^+ = \{A,B,C,D,E\} \\
\{A,F\}^+ = \{A,F,B,C,D,E\}
\]
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:
A, B → C
A, C → B

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 \textit{minimal cover of $F$}

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

\textbf{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 $F^+$
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:

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

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 detect that a table is “bad”?*