Database Design and Implementation

CS 645

Schema Refinement
First Normal Form (1NF)

A schema is in 1NF if all tables are flat.
Schema design

Conceptual Model:

Relational Model:
plus FDs
(FD = Functional Dependency)

Normalization:
Eliminates anomalies
Data anomalies

When a database is poorly designed we get anomalies:

- **Redundancy**: data is repeated
- **Update** anomalies: need to change in several places
- **Delete** anomalies: may lose data when we don’t want
Recall set attributes (persons with several phones):

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>413-555-1234</td>
<td>Amherst</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>413-555-6543</td>
<td>Amherst</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

One person may have multiple phones, but lives in only one city

Primary key is thus (SSN, PhoneNumber)

The above is in 1NF, but what is the problem with this schema?
Relational schema design

Recall set attributes (persons with several phones):

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>413-555-1234</td>
<td>Amherst</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>413-555-6543</td>
<td>Amherst</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

Anomalies:
- Redundancy = repeat data
- Update anomalies = what if Fred moves to “Boston”?
- Deletion anomalies = what if Joe deletes his phone number?
  (what if Joe had only one phone #)
Relation decomposition

Break the relation into two:

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>PhoneNumber</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>413-555-1234</td>
<td>Amherst</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>413-555-6543</td>
<td>Amherst</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

Anomalies have gone:
- No more repeated data
- Easy to move Fred to “Boston” (how?)
- Easy to delete all Joe’s phone numbers (how?)
Relational schema design (logical design)

Main idea:
- Start with some relational schema
- Find out its functional dependencies (discussed next!)
- Use them to design a better relational schema
Functional Dependencies

- A form of constraint
- Hence, part of the schema
- Finding them is part of the database design
- Use them to normalize the relations


**Functional Dependencies (FDs)**

**Definition:**
If two tuples agree on the attributes \( A_1, A_2, ..., A_n \)
then they must also agree on the attributes \( B_1, B_2, ..., B_m \).

**Formally:**
\[ A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m \]

**Example:**
streetName \( \rightarrow \) zipCode
Example

A FD 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>
A FD **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>

**Position → Phone**
Example

A FD 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>

But not: Phone  →  Position
Example

FD’s are constraints:
• On some instances they hold
• On others they don’t

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
</tbody>
</table>

Does this instance satisfy all the FDs?
Example

FD’s are constraints:
- On some instances they hold
- On others they don’t

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Blue</td>
<td>Supplies</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?
FDs and anomalies

- Anomalies occur when certain “bad” FDs hold

- How do we know if a “bad” FD holds?
An interesting observation

If all these FDs are true:

- \( \text{name} \rightarrow \text{color} \)
- \( \text{category} ightarrow \text{department} \)
- \( \text{color, category} ightarrow \text{price} \)

Then this FD also holds:

\( \text{name, category} ightarrow \text{price} \)

Why??
Deriving all FDs: Armstrong’s Rules

**Splitting and combining**

\[ A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m \leftrightarrow \]

\[ A_1, A_2, ..., A_n \rightarrow B_1 \]

\[ A_1, A_2, ..., A_n \rightarrow B_2 \]

\[ ... \]

\[ A_1, A_2, ..., A_n \rightarrow B_m \]

**Trivial**

\[ A_1, A_2, ..., A_n \rightarrow A_i \]

**Transitive**

\[ A_1, ..., A_n \rightarrow B_1, ..., B_m \rightarrow A_1, ..., A_n \rightarrow C_1, ..., C_p \]

\[ B_1, ..., B_m \rightarrow C_1, ..., C_p \]
### Example (continued)

Start from the following FDs:

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

Infer the following FDs:

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category → name</td>
<td></td>
</tr>
<tr>
<td>5. name, category → color</td>
<td></td>
</tr>
<tr>
<td>6. name, category → category</td>
<td></td>
</tr>
<tr>
<td>7. name, category → color, category</td>
<td></td>
</tr>
<tr>
<td>8. name, category → price</td>
<td></td>
</tr>
</tbody>
</table>
Examples (continued)

Answers:

<table>
<thead>
<tr>
<th>Inferred FD</th>
<th>Which Rule did we apply?</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. name, category → name</td>
<td>Trivial</td>
</tr>
<tr>
<td>5. name, category → color</td>
<td>Transitivity on 4, 1</td>
</tr>
<tr>
<td>6. name, category → category</td>
<td>Trivial</td>
</tr>
<tr>
<td>7. name, category → color, category</td>
<td>Split/combine on 5, 6</td>
</tr>
<tr>
<td>8. name, category → price</td>
<td>Transitivity on 3, 7</td>
</tr>
</tbody>
</table>

THIS IS TOO HARD! Let’s see an easier way.
Deriving all FDs: Closure

**Given** a set of attributes $A_1, ..., A_n$

The closure, $\{A_1, ..., A_n\}^+ = \{B\}$, is the set of attributes $B$ such that $A_1, ..., A_n \rightarrow B$.

**Example:**
- name $\rightarrow$ color
- category $\rightarrow$ department
- price

**Closures:**
- $\text{name}^+ = \{\text{name, color}\}$
- $\{\text{name, category}\}^+ = \{\text{name, category, color, department, price}\}$
- $\text{color}^+ = \{\text{color}\}$
Closure algorithm

\[ X = \{ A_1, \ldots, A_n \}. \]

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

if \( B_1, \ldots, B_n \rightarrow C \) is a FD and \( B_1, \ldots, B_n \) are all in \( X \)
then add \( C \) to \( X \).

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

Hence: \( \text{name, category} \rightarrow \text{color, department, price} \)

Example:

name → color  
category → department  
color, category → price  
name → color  
category → department
Example

In class:

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

\[
\begin{array}{c}
A, B \rightarrow C \\
A, D \rightarrow E \\
B \rightarrow D \\
A, F \rightarrow B
\end{array}
\]

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

In class:

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

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

In class:

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

\[
\begin{array}{c|c}
A, B & C \\
A, D & E \\
B & D \\
A, F & B \\
\end{array}
\]

Compute \( \{A, B\}^+ \)  \quad X = \{A, B, C\}
Example

In class:

\[
\begin{array}{ccc}
(A, B, C, D, E, F) & A, B & \rightarrow C \\
& A, D & \rightarrow E \\
& B & \rightarrow D \\
& A, F & \rightarrow B \\
\end{array}
\]

Compute \{A, B\}^+ \quad X = \{A, B, C\}
Example

In class:

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

<table>
<thead>
<tr>
<th>Relation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B</td>
<td>C</td>
</tr>
<tr>
<td>A, D</td>
<td>E</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>A, F</td>
<td>B</td>
</tr>
</tbody>
</table>

Compute $\{A,B\}^+$  

$X = \{A, B, C, D\}$
Example

In class:

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

\[
\begin{array}{c|c}
A, B & \rightarrow C \\
A, D & \rightarrow E \\
B & \rightarrow D \\
A, F & \rightarrow B \\
\end{array}
\]

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

\[ X = \{A, B, C, D\} \]
Example

In class:

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

\[
\begin{array}{c|c}
A, B & \rightarrow C \\
A, D & \rightarrow E \\
B & \rightarrow D \\
A, F & \rightarrow B \\
\end{array}
\]

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

In class:

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

\[
\begin{array}{c}
A, B \rightarrow C \\
A, D \rightarrow E \\
B \rightarrow D \\
A, F \rightarrow B \\
\end{array}
\]

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

Compute \( \{A, F\}^+ \)  
\( X = \{A, F\} \)
Example

In class:

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

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

Compute \( \{A, F\}^+ \): \( X = \{A, F, B\} \)
Example

In class:

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

\[
\begin{array}{l}
A, B \rightarrow C \\
A, D \rightarrow E \\
B \rightarrow D \\
A, F \rightarrow B
\end{array}
\]

Compute \( \{A,B\}^+ \) \hspace{1cm} X = \{A, B, C, D, E\}

Compute \( \{A, F\}^+ \) \hspace{1cm} X = \{A, F, B, C, D, E\}
Why do we need closure?

- With closure we can find all FDs easily

- To check if $X \rightarrow A$
  - Compute $X^+$
  - Check if $A \in X^+$
A superkey is a set of attributes $A_1, ..., A_n$ s.t. for any other attribute $B$, we have $A_1, ..., A_n \rightarrow B$

A key is a minimal superkey

i.e., set of attributes which is a superkey and for which no subset is a superkey
Computing (super)keys

- Compute $X^+$ for all sets $X$
- If $X^+ =$ all attributes, then $X$ is a superkey
- List only the minimal $X$’s to get the keys
Example

Product(name, price, category, color)

name, category → price
category → color

What is the key?
Example

Product(name, price, category, color)

(name, category) + = {name, category, price, color}

Hence (name, category) is a key
Eliminating anomalies

Main idea:

\[ X \rightarrow A \text{ is OK if } X \text{ is a (super)key} \]

\[ X \rightarrow A \text{ is not OK otherwise} \]
What is the key? \{SSN, PhoneNumber\}

Hence \( SSN \rightarrow Name, City \) is a “bad” dependency
Boyce-Codd Normal Form (BCNF)

A simple condition for removing anomalies from relations:

A relation $R$ is in BCNF if:

If $A_1, \ldots, A_n \rightarrow B$ is a non-trivial dependency in $R$,
then $\{A_1, \ldots, A_n\}$ is a superkey for $R$

In other words: there are no “bad” FDs

Equivalently:
for all $X$, either $(X^+ = X)$ or $(X^+ = \text{all attributes})$
Example (revisited)

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>Amherst</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

Let’s check anomalies:
- Redundancy?
- Update?
- Delete?
Example decomposition

Person(name, SSN, age, hairColor, phoneNumber)

FD1: SSN → name, age
FD2: age → hairColor

Decompose into BCNF:
Example decomposition

Person(name, SSN, age, hairColor, phoneNumber)

| FD1: SSN → name, age |
| FD2: age → hairColor |

Decompose into BCNF:

What is the key? \{SSN, phoneNumber\}

But how to decompose?

Person(SSN, name, age)
Phone(SSN, hairColor, phoneNumber)

or

Person(SSN, name, age, hairColor)
Phone(SSN, phoneNumber)

SSN → name, age, hairColor

or ....
BCNF decomposition algorithm

BCNF_Decompose(R)

find X s.t.: X ≠ X⁺ ≠ [all attributes]

if (not found) then “R is in BCNF”

let Y = X⁺ - X
let Z = [all attributes] - X⁺
decompose R into R₁(X ∪ Y) and R₂(X ∪ Z)
continue to decompose recursively R₁ and R₂
Example

What are the keys?

What happens if in R we first pick B⁺? Or AB⁺?
Decompositions in general

\[ R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_p) \]

\[ R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \]

\[ R_2(A_1, \ldots, A_n, C_1, \ldots, C_p) \]

\[ R_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m \]

\[ R_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p \]
Theory of decomposition

Sometimes it is correct:

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

Lossless decomposition
## Incorrect decomposition

Sometimes it is not:

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>

Lossy decomposition

What’s incorrect??

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.99</td>
<td>Gadget</td>
</tr>
<tr>
<td>24.99</td>
<td>Camera</td>
</tr>
<tr>
<td>19.99</td>
<td>Camera</td>
</tr>
</tbody>
</table>
Decompositions in general

If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$

Then the decomposition is lossless

Note: don’t need $A_1, \ldots, A_n \rightarrow C_1, \ldots, C_p$

BCNF decomposition is always lossless. WHY?
General decomposition goals

1. Elimination of anomalies
2. Recoverability of information
   - Can we get the original relation back?
3. Preservation of dependencies
   - Want to enforce FDs without performing joins

Sometimes cannot decompose into BCNF without losing ability to check some FDs in single relation
BCNF and dependencies

<table>
<thead>
<tr>
<th>Unit</th>
<th>Company</th>
<th>Product</th>
</tr>
</thead>
</table>

So, there is a BCNF violation, and we decompose.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Company</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Unit</th>
<th>Product</th>
</tr>
</thead>
</table>

In BCNF we lose the FD

Unit → Company

Company, Product → Unit

No FDs

Unit → Company

Company, Product → Unit
A relation $R$ is in 3rd normal form if:
Whenever there is a nontrivial dependency $A_1, A_2, ..., A_n \rightarrow B$ for $R$, then $\{A_1, A_2, ..., A_n\}$ is a super-key for $R$, or $B$ is part of a key.

Tradeoffs:
**BCNF:** no anomalies, but may lose some FDs
**3NF:** keeps all FDs, but may have some anomalies