Database Design Process

Conceptual Model:

Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical storage details
Physical Schema
What makes good schemas?
Relational Schema Design

<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>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</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).

What is the problem with this schema?
Relational Schema Design

Anomalies:
• Redundancy = repeat data
• Update anomalies = what if Fred moves to “Bellevue”?
• Deletion anomalies = what if Joe deletes his phone number?

<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>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

These can cause bugs!
Worry most about later two.
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>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

<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>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

Anomalies have gone:

- No more repeated data
- Easy to move Fred to “Bellevue” (how?)
- Easy to delete all Joe’s phone numbers (how?)
Relational Schema Design (or Logical Design)

How do we do this systematically?

• Start with some relational schema

• Find out its *functional dependencies* (FDs)

• Use FDs to *normalize* the relational schema
Functional Dependencies (FDs)

Definition

If two tuples agree on the attributes

\[ A_1, A_2, \ldots, A_n \]

then they must also agree on the attributes

\[ B_1, B_2, \ldots, B_m \]

Formally:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

\( A_1 \ldots A_n \) determines \( B_1 \ldots B_m \)
Functional Dependencies (FDs)

**Definition**  
FD $A_1, ..., A_m \rightarrow B_1, ..., B_n$ holds in $R$ if:
for every pair of tuples $t, t' \in R$,  
$(t.A_1 = t'.A_1$ and ... $t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1$ and ... $t.B_n = t'.B_n$)
Example

An 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>

EmpID → Name, Phone, Position
Position → Phone
but not Phone → Position
## Example

<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

<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 $\rightarrow$ Position
Example

Do all the FDs hold on this instance?

name → color
category → department
color, category → price

<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>
### Example

<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>49</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-supp.</td>
<td>59</td>
</tr>
</tbody>
</table>

What about this one?
Terminology

• FD holds or does not hold on an instance

• If we can be sure that every instance of $R$ will be one in which a given FD is true, then we say that $R$ satisfies the FD

• If we say that $R$ satisfies an FD $F$, we are stating a constraint on $R$ (part of schema)
An Interesting Observation

If all these FDs are true:

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

Then this FD also holds:

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

If we find out from application domain that a relation satisfies some FDs, it doesn’t mean that we found all the FDs that it satisfies! There could be more FDs implied by the ones we have.
Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$

The closure, $\{A_1, \ldots, A_n\}^+ = \{B \mid A_1, \ldots, A_n \rightarrow B\}$

Example:

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

Example:

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

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

Hence: name, category \( \rightarrow \) color, department, price
Example

In class:

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

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

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

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

In class:

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

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

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

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

What is a key of \( R \)?
Practice at Home

Find all FD’s implied by:

- A, B → C
- A, D → B
- B → D
Practice at Home

Find all FD’s implied by:

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

Step 1: Compute \( X^+ \), for every \( X \):

\[
\begin{align*}
A^+ &= A, \quad B^+ = BD, \quad C^+ = C, \quad D^+ = D \\
AB^+ &= ABCD, \quad AC^+ = AC, \quad AD^+ = ABCD, \\
&\hspace{1em} BC^+ = BCD, \quad BD^+ = BD, \quad CD^+ = CD \\
ABC^+ &= ABD^+ = ACD^+ = ABCD \text{ (no need to compute – why?)} \\
BCD^+ &= BCD, \quad ABCD^+ = ABCD
\end{align*}
\]

Step 2: Enumerate all FD’s \( X \rightarrow Y \), s.t. \( Y \subseteq X^+ \) and \( X \cap Y = \emptyset \):

\[
\begin{array}{c}
AB \rightarrow CD, \quad AD \rightarrow BC, \quad ABC \rightarrow D, \quad ABD \rightarrow C, \quad ACD \rightarrow B
\end{array}
\]
Keys

• 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
  – superkey and for which no subset is a superkey
Computing (Super)Keys

- For all sets $X$, compute $X^+$
- If $X^+ = [\text{all attributes}]$, then $X$ is a superkey
- Try only the minimal $X$’s to get the key
Example

Product(name, price, category, color)

\[
\begin{align*}
\text{name, category} & \rightarrow \text{price} \\
\text{category} & \rightarrow \text{color}
\end{align*}
\]

What is the key?

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

Hence \{name, category\} is a (super)key
Key or Keys?

Can we have more than one key?

Given $R(A,B,C)$ define FD’s s.t. there are two or more keys

A $\rightarrow$ B
B $\rightarrow$ C
C $\rightarrow$ A

or

AB $\rightarrow$ C
BC $\rightarrow$ A

or

A $\rightarrow$ BC
B $\rightarrow$ AC

what are the keys here?
Eliminating Anomalies

Main idea:

• $X \rightarrow A$ is OK if $X$ is a (super)key

• $X \rightarrow A$ is not OK otherwise
  – Need to decompose the table, but how?

Boyce-Codd Normal Form
Boyce-Codd Normal Form

Dr. Raymond F. Boyce
Boyce-Codd Normal Form

There are no “bad” FDs:

**Definition.** A relation $R$ is in BCNF if:
Whenever $X \rightarrow A$ is a non-trivial dependency, then $X$ is a superkey.

Equivalently:

**Definition.** A relation $R$ is in BCNF if:
\[
\forall X, \text{ either } X^+ = X \text{ or } X^+ = [\text{all attributes}]
\]
BCNF Decomposition Algorithm

Normalize(R)
find X s.t.: X ≠ X⁺ and X⁺ ≠ [all attributes]
if (not found) then “R is in BCNF”
let Y = X⁺ - X; Z = [all attributes] - X⁺
decompose R into R₁(X ∪ Y) and R₂(X ∪ Z)
Normalize(R₁); Normalize(R₂);
Example

<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>206-555-1234</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-6543</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-2121</td>
<td>Westfield</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>908-555-1234</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

The only key is: \{SSN, PhoneNumber\}
Hence SSN → Name, City is a “bad” dependency
In other words:
SSN+ = SSN, Name, City and is neither SSN nor All Attributes
Example BCNF Decomposition

<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>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>Westfield</td>
</tr>
</tbody>
</table>

SSN → Name, City

<table>
<thead>
<tr>
<th>SSN</th>
<th>PhoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>206-555-1234</td>
</tr>
<tr>
<td>123-45-6789</td>
<td>206-555-6543</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>908-555-2121</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>908-555-1234</td>
</tr>
</tbody>
</table>

Let's check anomalies:
- Redundancy ?
- Update ?
- Delete ?
Example BCNF Decomposition

```plaintext
Person(name, SSN, age, hairColor, phoneNumber)
    SSN → name, age
    age → hairColor
```

Find $X$ s.t.: $X \neq X^+$ and $X^+ \neq \{\text{all attributes}\}$
Example BCNF Decomposition

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

SSN → name, age
age → hairColor

Iteration 1: Person: SSN+ = SSN, name, age, hairColor
Decompose into: P(SSN, name, age, hairColor)
Phone(SSN, phoneNumber)
Example BCNF Decomposition

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

SSN → name, age
age → hairColor

Iteration 1: **Person**: SSN⁺ = SSN, name, age, hairColor
Decompose into: $P(\text{SSN}, \text{name}, \text{age}, \text{hairColor})$

Phone(\text{SSN}, \text{phoneNumber})

Iteration 2: **P**: age⁺ = age, hairColor
Decompose: People(\text{SSN}, \text{name}, \text{age})
Hair(\text{age}, \text{hairColor})

Phone(\text{SSN}, \text{phoneNumber})
Example BCNF Decomposition

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

SSN $\rightarrow$ name, age
age $\rightarrow$ hairColor

Iteration 1: **Person**: SSN$^+$ = SSN, name, age, hairColor
Decompose into: $P$(SSN, name, age, hairColor)
Phone(SSN, phoneNumber)

Iteration 2: **P**: age$^+$ = age, hairColor
Decompose: People(SSN, name, age)
Hair(age, hairColor)
Phone(SSN, phoneNumber)

Find X s.t.: $X \neq X^+$ and $X^+ \neq$ [all attributes]
Example: BCNF

R(A,B,C,D) → B
B → C
Example: BCNF

Recall: find $X$ s.t. $X \not\subseteq X^+ \not\subseteq [\text{all-attrs}]$
Example: BCNF

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

\[ A^+ = ABC \neq ABCD \]
Example: BCNF

\[ \text{R}(A,B,C,D) \]

\[ A^+ = ABC \neq ABCD \]

\[ \text{R}_1(A,B,C) \]

\[ \text{R}_2(A,D) \]
Example: BCNF

R(A,B,C,D)

R(A,B,C,D)
A⁺ = ABC ≠ ABCD

R₁(A,B,C)
B⁺ = BC ≠ ABC

R₂(A,D)

A → B
B → C
Example: BCNF

R(A,B,C,D)

A⁺ = ABC ≠ ABCD

R₁(A,B,C)

B⁺ = BC ≠ ABC

R₁₁(B,C)  R₁₂(A,B)

R₂(A,D)

What are the keys?

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

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

\[S_1(A_1, \ldots, A_n, B_1, \ldots, B_m)\]
\[S_2(A_1, \ldots, A_n, C_1, \ldots, C_p)\]

\[S_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m\]
\[S_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p\]
Lossless Decomposition

<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>

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

<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>
Lossy Decomposition

What is lossy here?

<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

<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>

CSE 414 - Spring 2017
Decomposition in General

Let:

- $S_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m$
- $S_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p$

The decomposition is called \textit{lossless} if $R = S_1 \bowtie S_2$

Fact: If $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$ then the decomposition is lossless
DBs and QM (off topic)

Lossless joins are related to quantum mechanics:

• Tables of measurement outcomes from one object
  – each measures two properties at once
  – e.g., \{height, hair color\}, \{hair color, weight\}, etc.

• Each 2-property table should be a projection of a table with all properties

• Somehow this does not happen for QM systems
measurement

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A'</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A'</th>
<th>B'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

natural join all =

<table>
<thead>
<tr>
<th>A</th>
<th>A'</th>
<th>B</th>
<th>B'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(projection onto (A,B) does not include (0,0)!)
Schema Refinements
= Normal Forms

• 1st Normal Form = all tables are flat (no list values)
• 2nd Normal Form = obsolete
• Boyce Codd Normal Form = no bad FDs
• 3rd Normal Form = see book
  – BCNF is lossless but can cause lose ability to check some FDs without a join (see book 3.4.4)
  – 3NF fixes that (is lossless and dependency-preserving), but some tables might not be in BCNF – i.e., they may have redundancy anomalies