Database Design Process

Conceptual Model:

Relational Model: Tables + constraints
And also functional dep.

Normalization: Eliminates anomalies

Conceptual Schema

Physical Schema

Database Systems
CSE 414

Lectures 18-19: Design Theory
(Ch. 3.1, 3.3-4)

Relational Schema Design

What makes good schemas?

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

Relational Schema Design

Relation Decomposition

Break the relation into two:

Anomalies have gone:
- No more repeated data
- Easy to move Fred to "Bellevue" (how?)
- Easy to delete all Joe's phone numbers (how?)

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?

CSE 414 - Spring 2017
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
\[ A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m \]
then they must also agree on the attributes
Formally:
\[ A_1, A_2, ..., A_n \text{ determines } B_1, B_2, ..., B_m \]

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 \( \rightarrow \) Name, Phone, Position
Position \( \rightarrow \) Phone
but not Phone \( \rightarrow \) 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 \( \rightarrow \) Phone

But not Phone \( \rightarrow \) Position
Example

Do all the FDs hold on this instance?

<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

name $\rightarrow$ color
category $\rightarrow$ department
color, category $\rightarrow$ 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>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:

name $\rightarrow$ color
category $\rightarrow$ department
color, category $\rightarrow$ price

Then this FD also holds:

name, category $\rightarrow$ 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**, $\{A_1, \ldots, A_n\}^+$ = the set of attributes $B$ s.t. $A_1, \ldots, A_n \rightarrow B$

Example:

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

Closures:

name$^+$ = \{name, color\}

{name, category}$^+$ = \{name, category, color, department, price\}

color$^+$ = \{color\}

Closure Algorithm

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

Repeat until $X$ doesn’t change do:

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

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:

(name, category)$^+$ = \{name, category, color, department, price\}

Hence: name, category $\rightarrow$ color, department, price
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\}

Example

Compute \{A,B\} + X = \{A, B, C, D, E\}
Compute \{A,F\} + X = \{A, F, B, C, D, E\}

Practice at Home

Find all FD's implied by:

\[ A, B \rightarrow C \]
\[ A, D \rightarrow B \]
\[ B \rightarrow D \]

Step 1: Compute X^+ for every X:
\[ A^+ = A \]
\[ B^+ = BD \]
\[ C^+ = C \]
\[ D^+ = D \]
\[ AB^+ = ABCD \]
\[ AC^+ = AC \]
\[ AD^+ = ABCD \]
\[ BC^+ = BCD \]
\[ BD^+ = BD \]
\[ CD^+ = CD \]
\[ ABC^+ = ABCD \]
\[ AD^+ = ABCD \]
\[ CD^+ = CD \]
Step 2: Enumerate all FD's X \rightarrow Y s.t. Y \subseteq X^+ and X - Y = \emptyset:
\[ AB \rightarrow CD, AD \rightarrow BC, ABC \rightarrow D, ABD \rightarrow C, ACD \rightarrow B \]

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\}

Practice at Home

What is a key of R?

Find all FD's implied by:

\[ A, B \rightarrow C \]
\[ A, D \rightarrow B \]
\[ B \rightarrow D \]

Step 1: Compute X^+ for every X:
\[ A^+ = A \]
\[ B^+ = BD \]
\[ C^+ = C \]
\[ D^+ = D \]
\[ AB^+ = ABCD \]
\[ AC^+ = AC \]
\[ AD^+ = ABCD \]
\[ BC^+ = BCD \]
\[ BD^+ = BD \]
\[ CD^+ = CD \]
\[ ABC^+ = ABCD \]
\[ AD^+ = ABCD \]
\[ CD^+ = CD \]
Step 2: Enumerate all FD's X \rightarrow Y s.t. Y \subseteq X^+ and X - Y = \emptyset:
\[ AB \rightarrow CD, AD \rightarrow BC, ABC \rightarrow D, ABD \rightarrow C, ACD \rightarrow B \]

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

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

or

\[
\begin{align*}
AB & \rightarrow C \\
BC & \rightarrow A \\
B & \rightarrow AC
\end{align*}
\]

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

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:

$\forall X$, either $X^+ = X$ or $X^+ = \text{[all attributes]}$
**Example BCNF Decomposition**

<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>

Let's check anomalies:
- Redundancy?
- Update?
- Delete?

**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)

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

Find X s.t.: X ≠ X' and X' ≠ [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)

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

Find X s.t.: X ≠ X’ and X’ ≠ [all attributes]

R(A,B,C,D)

Recall: find X s.t. X ≤ X’ ≤ [all-attrs]

R(A,B,C,D)

A → B
B → C

A → B
B → C

A → B
B → C

A → B
B → C

R(A,B,C,D)

A* = ABC ≠ ABCD

R1(A,B,C)

R2(A,D)

R(A,B,C,D)

A* = ABC ≠ ABCD

R1(A,B,C)

B* = BC ≠ ABC

R2(A,D)
Example: BCNF

R(A, B, C, D)

A \rightarrow B
B \rightarrow C

A^* = ABC ≠ ABCD

R_1(A, B, C)
B^* = BC ≠ ABC

R_2(A, B)

What are the keys?

What happens if in R we first pick B^*? Or AB^*?

Decompositions in General

R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_p)

S_1(A_1, ..., A_n, B_1, ..., B_m)
S_2(A_1, ..., A_n, C_1, ..., C_p)

S_1 = projection of R on A_1, ..., A_n, B_1, ..., B_m
S_2 = projection of R on A_1, ..., A_n, C_1, ..., C_p

Lossless Decomposition

R(A, B, C, D)

A \rightarrow B
B \rightarrow C

A^* = ABC ≠ ABCD

R_1(A, B, C)
B^* = BC ≠ ABC

R_2(A, B)

What are the keys?

What happens if in R we first pick B^*? Or AB^*?

Lossy Decomposition

R(A, B, C, D)

A \rightarrow B
B \rightarrow C

A^* = ABC ≠ ABCD

R_1(A, B, C)
B^* = BC ≠ ABC

R_2(A, B)

What are the keys?

What happens if in R we first pick B^*? Or AB^*?

Lossless Decomposition

Name | Price | Category
--- | --- | ---
Gizmo | 19.99 | Gadget
OneClick | 24.99 | Camera
Gizmo | 19.99 | Camera

Lossy Decomposition

Name | Price | Category
--- | --- | ---
Gizmo | 19.99 | Gadget
OneClick | 24.99 | Camera
Gizmo | 19.99 | Camera

Lossless Decomposition

Name | Price | Category
--- | --- | ---
Gizmo | 19.99 | Gadget
OneClick | 24.99 | Camera
Gizmo | 19.99 | Camera

Lossy Decomposition

Name | Price | Category
--- | --- | ---
Gizmo | 19.99 | Gadget
OneClick | 24.99 | Camera
Gizmo | 19.99 | Camera

Decomposition in General

R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_p)

S_1(A_1, ..., A_n, B_1, ..., B_m)
S_2(A_1, ..., A_n, C_1, ..., C_p)

Let:
S_1 = projection of R on A_1, ..., A_n, B_1, ..., B_m
S_2 = projection of R on A_1, ..., A_n, C_1, ..., C_p

The decomposition is called lossless if R = S_1 ∪ S_2

Fact: If A_1, ..., A_n \rightarrow B_1, ..., B_m then the decomposition is lossless

It follows that every BCNF 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

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

natural join all =

projection onto $(A,B)$ does not include $(0,0)!$