Introduction to Database Systems
CSE 414

Lecture 21: BCNF

What makes good schemas?

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 (in terms of # of attributes)
  - A 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 reducing to the minimal $X$'s to get the key

Relational Schema Design

SSN $\rightarrow$ Name, City

<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>

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

Relation Decomposition

Break the relation into two:

SSN $\rightarrow$ Name, City

<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>

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

There are no “bad” FDs:

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

Equivalently:

Definition. A relation $R$ is in BCNF if:
$orall X$, either $X^+ = X$ (i.e., $X$ is not in any FDs) or $X^+ = \{\text{all attributes}\}$ (computed using FDs)

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>

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>

Let’s check anomalies:

• Redundancy ?
• Update ?
• Delete ?

Find X s.t.: $X \neq X^*$ and $X^*$ # [all attributes]

Find X s.t.: $X \neq X^*$ and $X^*$ # [all attributes]

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

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

$SSN \rightarrow$ name, age, hairColor

age $\rightarrow$ hairColor
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)

Note the keys!
Example: BCNF

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

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

\[ R_1(A, B, C) \]

\[ R_2(A, D) \]

What happens if in \( R \) we first pick \( B^+ \)? Or \( AB^+ \)?

Decompositions in General

\[ R(A_1, A_2, A_3, ..., B_1, ..., B_m, C_1, ..., C_p) \]

\[ S_1 = \text{projection of } R \text{ on } A_1, ..., A_n, B_1, ..., B_m \]

\[ S_2 = \text{projection of } R \text{ on } A_1, ..., A_n, C_1, ..., C_p \]

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>

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>
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>

Decomposition in General

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

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

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 lossless if \( R = S_1 \Join S_2 \)

It follows that every BCNF decomposition is lossless

Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat
- 2nd Normal Form = obsolete
- Boyce Codd Normal Form = no bad FDs
- 3rd Normal Form = see book
  - BCNF is lossless but can cause loss of ability to check some FDs (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

Getting Practical

How to implement normalization in SQL

Motivation

- We learned about how to normalize tables to avoid anomalies
- How can we implement normalization in SQL if we can’t modify existing tables?  
  - This might be due to legacy applications that rely on previous schemas to run

Use Views!

- A view in SQL =  
  - A table computed from other tables, s.t., whenever the base tables are updated, the view is updated too
- More generally:  
  - A view is derived data that keeps track of changes in the original data
A Simple View

Create a view that returns for each store the prices of products purchased at that store.

CREATE VIEW StorePrice AS
SELECT DISTINCT x.store, y.price
FROM Purchase AS x, Product AS y
WHERE x.product = y.pname

This is like a new table
StorePrice(store, price)

We Use a View Like Any Table

- A "high end" store is a store that sells some products over 1000.
- For each customer, return all the high end stores that they visit.

SELECT DISTINCT u.customer, u.store
FROM Purchase AS u, StorePrice AS v
WHERE u.store = v.store
AND v.price > 1000

Types of Views

- **Virtual views**
  - Computed only on-demand – slow at runtime
  - Always up to date

- **Materialized views**
  - Pre-computed offline – fast at runtime
  - May have stale data (must recompute or update)

- A key component of database performance tuning is the selection of materialized and virtual views

Vertical Partitioning

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Resume</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Houston</td>
<td>Doc1...</td>
<td>JPG1...</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
<td>Doc2...</td>
<td>JPG2...</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
<td>Doc3...</td>
<td>JPG3...</td>
</tr>
<tr>
<td>432432</td>
<td>Ann</td>
<td>Portland</td>
<td>Doc4...</td>
<td>JPG4...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Resume</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Houston</td>
<td>Doc1...</td>
<td>JPG1...</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
<td>Doc2...</td>
<td>JPG2...</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
<td>Doc3...</td>
<td>JPG3...</td>
</tr>
<tr>
<td>432432</td>
<td>Ann</td>
<td>Portland</td>
<td>Doc4...</td>
<td>JPG4...</td>
</tr>
</tbody>
</table>

T2.SSN is a key and a foreign key to T1.SSN. Same for T3.SSN.

CREATE VIEW Resumes AS
SELECT T1.ssn, T1.name, T1.address,
T2.resume, T3.picture
FROM T1, T2, T3
WHERE T1.ssn = T2.ssn AND T1.ssn = T3.ssn

CREATE VIEW Resumes AS
SELECT T1.ssn, T1.name, T1.address,
T2.resume, T3.picture
FROM T1, T2, T3
WHERE T1.ssn = T2.ssn AND T1.ssn = T3.ssn

SELECT address
FROM Resumes
WHERE name = 'Sue'
Vertical Partitioning Applications

- **Advantages**
  - Speeds up queries that touch only a small fraction of columns
  - Single column can be compressed effectively, reducing disk I/O

- **Disadvantages**
  - Updates are expensive!
  - Need many joins to access many columns
  - Repeated key columns add overhead