Introduction to Data Management
CSE 414

Unit 6: Conceptual Design
E/R Diagrams
Integrity Constraints
BCNF

(3 lectures)
Introduction to Data Management
CSE 414

Design Theory and BCNF
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>
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>

<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>
</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:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]
**Functional Dependencies (FDs)**

**Definition** \( A_1, ..., A_m \rightarrow B_1, ..., B_n \) holds in \( R \) if:

\[
\forall t, t' \in R,
(t.A_1 = t'.A_1 \land ... \land t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \land ... \land t.B_n = t'.B_n)
\]

<table>
<thead>
<tr>
<th>( R )</th>
<th>( A_1 )</th>
<th>( ... )</th>
<th>( A_m )</th>
<th>( B_1 )</th>
<th>( ... )</th>
<th>( B_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t' )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if \( t, t' \) agree here then \( t, t' \) agree here
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 ➔ 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  →  Position
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>99</td>
</tr>
</tbody>
</table>

Do all the FDs hold on this instance?
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?
Buzzwords

- 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, we are stating a constraint on $R$
Why bother with FDs?

Anomalies:
- **Redundancy** = repeat data
- **Update anomalies** = what if Fred moves to “Bellevue”?
- **Deletion anomalies** = what if Joe deletes his phone number?
An Interesting Observation

If all these FDs are true:

\[
\begin{align*}
\text{Name} & \rightarrow \text{Color} \\
\text{Category} & \rightarrow \text{Department} \\
\text{Color, Category} & \rightarrow \text{Price}
\end{align*}
\]

Then this FD also holds:

\[
\text{Name, Category} \rightarrow \text{Price}
\]
An Interesting Observation

If all these FDs are true:

- name → color
- category → department
- color, category → price

Then this FD also holds:

- name, category → price
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** is the set of attributes $B$, notated $\{A_1, \ldots, A_n\}^+$, s.t. $A_1, \ldots, A_n \rightarrow B$

**Example:**

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

**Closures:**

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

$color^+ = \{\text{color}\}$
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

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

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

\begin{itemize}
  \item 1. name \( \rightarrow \) color
  \item 2. category \( \rightarrow \) department
  \item 3. color, category \( \rightarrow \) price
\end{itemize}

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

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

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

\[
\text{if } B_1, \ldots, B_n \rightarrow C \text{ is a FD and } B_1, \ldots, B_n \text{ are all in } X
\]

then add \( C \) to \( X \).

Example:

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

\( \{ \text{name, category} \}^+ = \{ \text{name, category, color, department, price} \} \)
Closure Algorithm

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

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

\[
\text{if } B_1, \ldots, B_n \rightarrow C \text{ is a FD and } B_1, \ldots, B_n \text{ are all in } X
\]

\[
\text{then add } C \text{ to } X.
\]

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:

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

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

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

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

In class:

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

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

\[
\text{Compute } \{A,B\}^+ \quad X = \{A, B, C, D, E\}
\]

\[
\text{Compute } \{A, F\}^+ \quad X = \{A, F\}
\]
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, C, D, E\} \quad \}

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

CSE 414 – Autumn 2018
Example

In class:

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

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

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

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

What is the key of R?
Practice at Home

Find all FD’s implied by:

\[
\begin{align*}
A, B & \rightarrow C \\
A, D & \rightarrow B \\
B & \rightarrow D
\end{align*}
\]
Practice at Home

Find all FD’s implied by:

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

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

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

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

\[
\begin{align*}
AB & \rightarrow CD, \quad AD \rightarrow BC, \quad ABC \rightarrow D, \quad ABD \rightarrow C, \quad ACD \rightarrow B
\end{align*}
\]
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
  – 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
Example

Product(name, price, category, color)

name, category \rightarrow price
category \rightarrow color

What is the key ?
Example

Product(name, price, category, color)

name, category $\rightarrow$ price

category $\rightarrow$ color

What is the key?

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

Hence (name, category) is a 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?

\[(\text{name, category}) + = \{ \text{name, category, price, color} \}\]
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 distinct keys.
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 distinct keys

A → B
B → C
C → A

or

AB → C
BC → A

or

A → BC
B → 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
Edgar Frank “Ted” Codd

"A Relational Model of Data for Large Shared Data Banks"
If 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:

$$\forall X, \text{ either } X^+ = X \text{ (i.e., } X \text{ is not in any FDs) }$$
$$\text{ or } X^+ = \text{ [all attributes]} \text{ (computed using FDs)}$$
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 \(SSN \rightarrow \text{Name, City}\) is a “bad” dependency

In other words:
\(SSN^+ = \text{SSN, Name, City}\) 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 \rightarrow 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?
Find $X$ s.t.: $X \neq X^+$ and $X^+ \neq \{\text{all attributes}\}$

**Example BCNF Decomposition**

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

- SSN $\rightarrow$ name, age
- 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)
Find X s.t.: $X \neq X^+$ and $X^+ \neq \{\text{all attributes}\}$

## 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)
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]

Note the keys!
A \rightarrow B
B \rightarrow C
Recall: find $X$ s.t. $X \subsetneq X^+ \subsetneq \{\text{all-attrs}\}$

Example: BCNF

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

$A \rightarrow B$

$B \rightarrow C$
Example: BCNF

R(A,B,C,D)

A⁺ = ABC ≠ ABCD
Example: BCNF

R(A,B,C,D)

A → B
B → C

R(A,B,C,D)
A⁺ = ABC ≠ ABCD

R₁(A,B,C)

R₂(A,D)
Example: BCNF

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

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

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

\[ B^+ = BC \neq ABC \]

\[ R_2(A,D) \]
Example: BCNF

R(A,B,C,D)

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

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

R(A,B,C,D)

R_1(A,B,C)

\[ B^+ = BC \neq ABC \]

R_11(B,C)

R_12(A,B)

R_2(A,D)

What are the keys?
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
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

• Compare:
  – A **function** computes a value from other values, but does not keep track of changes to the inputs
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 x, Product y
    WHERE   x.product = y.pname
```
We Use a View Like Any Table

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

```
SELECT DISTINCT u.customer, u.store
FROM Purchase u, StorePrice 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)
  – Indexes *are* materialized views

• A key component of physical tuning of databases is the selection of materialized views and indexes
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>

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

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
Vertical Partitioning

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

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'

Original query:
SELECT T1.address
FROM T1, T2, T3
WHERE T1.name = 'Sue'
    AND T1.SSN = T2.SSN
    AND T1.SSN = T3.SSN
Vertical Partitioning

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'

Final query:
SELECT T1.address
FROM T1
WHERE T1.name = 'Sue'

Modified query:
SELECT T1.address
FROM T1, T2, T3
WHERE T1.name = 'Sue'
   AND T1.ssn = T2.ssn
   AND T1.ssn = T3.ssn
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
Horizontal Partitioning

Customers

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Houston</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
</tr>
<tr>
<td>234234</td>
<td>Ann</td>
<td>Portland</td>
</tr>
<tr>
<td></td>
<td>Frank</td>
<td>Calgary</td>
</tr>
<tr>
<td></td>
<td>Jean</td>
<td>Montreal</td>
</tr>
</tbody>
</table>

CustomersInHouston

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Houston</td>
</tr>
</tbody>
</table>

CustomersInSeattle

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
</tr>
</tbody>
</table>
Horizontal Partitioning

CREATE VIEW Customers AS
CustomersInHouston
UNION ALL
CustomersInSeattle
UNION ALL
...
Which tables are inspected by the system?
Horizontal Partitioning

Better: remove `CustomerInHouston.city` etc

```
CREATE VIEW Customers AS
(SELECT SSN, name, 'Houston' as city
FROM CustomersInHouston)
UNION ALL
(SELECT SSN, name, 'Seattle' as city
FROM CustomersInSeattle)
UNION ALL
... . . .
```
Horizontal Partitioning

```
SELECT name
FROM Customers
WHERE city = 'Seattle'
```

```
SELECT name
FROM CustomersInSeattle
```
Horizontal Partitioning Applications

• **Performance optimization**
  – Especially for data warehousing
  – E.g., one partition per month
  – E.g., archived applications and active applications

• **Distributed and parallel databases**

• **Data integration**
Conclusion

• Poor schemas can lead to performance inefficiencies

• E/R diagrams are means to structurally visualize and design relational schemas

• Normalization is a principled way of converting schemas into a form that avoid such problems

• BCNF is one of the most widely used normalized form in practice