Database Management Systems
CS 457

Lecture 9: Design Theory
Database Design Process

Conceptual Model:

Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical storage details
Physical Schema
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

<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 numbers?
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>

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**  
\[ A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n \text{ holds in } R \text{ if:} \]

\[ \forall t, t' \in R, \]

\[ (t.A_1 = t'.A_1 \land \ldots \land t.A_m = t'.A_m \rightarrow t.B_1 = t'.B_1 \land \ldots \land t.B_n = t'.B_n) \]

<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 \textit{holds, or does not hold} on an instance:

\begin{tabular}{|c|c|c|c|}
\hline
EmpID & Name & Phone & Position \\
\hline
E0045 & Smith & 1234 & Clerk \\
E3542 & Mike & 9876 & Salesrep \\
E1111 & Smith & 9876 & Salesrep \\
E9999 & Mary & 1234 & Lawyer \\
\hline
\end{tabular}

EmpID $\rightarrow$ Name, Phone, Position
Position $\rightarrow$ Phone
but not Phone $\rightarrow$ Position
Side Note

• Logical equivalence
  – (A -> B) means (Not A or B)
  • Truth table (discrete math class?)
    – A, B, A->B, Not A or B
    – T, T, T, T
    – T, F, F, F
    – F, T, T, T
    – F, F, T, T
# 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?

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>Black</td>
<td>Toys</td>
<td>99</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.
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, $\{A_1, \ldots, A_n\}^+ = \text{the set of attributes } B$

s.t. $A_1, \ldots, A_n \rightarrow B$

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

Closures:

\[
\begin{align*}
name^+ &= \{\text{name, color}\} \\
\{\text{name, category}\}^+ &= \{\text{name, category, color, department, price}\} \\
\text{color}^+ &= \{\text{color}\}
\end{align*}
\]
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 → color
2. category → department
3. color, category → price

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

Hence: name, category → color, department, price
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:

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

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

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

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>A</td>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>D</td>
</tr>
</tbody>
</table>

Step 1: Compute $X^+$, for every $X$:

- $A^+ = A$, $B^+ = BD$, $C^+ = C$, $D^+ = D$
- $AB^+ = ABCD$, $AC^+ = AC$, $AD^+ = ABCD$
- $BC^+ = BCD$, $BD^+ = BD$, $CD^+ = CD$
- $ABC^+ = ABD^+ = ACD^+ = ABCD$ (no need to compute— why ?)
- $BCD^+ = BCD$, $ABCD^+ = ABCD$

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

- $AB \rightarrow CD$, $AD \rightarrow BC$, $ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$
Keys

• A **superkey** is a set of attributes $A_1, \ldots, A_n$ s.t. for any other attribute $B$, we have $A_1, \ldots, A_n \rightarrow B$

• A **key** is a minimal superkey
  – A superkey and for which no subset is a superkey
Computing (Super)Keys

• For all sets $X$, compute $X^+$
  – “set”: a combination of elements from attributes

• 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)

name, category → price
category → color

What is the key?
Example

Product(name, price, category, color)

(name, category) → price
(category) → color

What is the key?

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

Hence (name, category) is a 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
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
IBM Research, 1970’s
Edgar Frank “Ted” Codd

"A Relational Model of Data for Large Shared Data Banks"
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:
$\forall X$, either $X^+ = X$ 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

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

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

**SSN → Name, City**

<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>
<tr>
<td>987-65-4321</td>
<td>908-555-1234</td>
</tr>
</tbody>
</table>
Find $X$ s.t.: $X \neq X^+$ and $X^+ \neq$ [all attributes]

Example BCNF Decomposition

$\text{Person(name, SSN, age, hairColor, phoneNumber)}$

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 ≠ 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)

What are the keys?
Example: BCNF

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

$A \rightarrow B$

$B \rightarrow C$

$R(A,B,C,D)$
Example: BCNF

Recall: find $X$ s.t. $X \subset X^+ \subset \text{[all-attrs]}$

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

$$A \rightarrow B$$
$$B \rightarrow C$$
Example: BCNF

R(A,B,C,D)

A⁺ = ABC ≠ ABCD
Example: BCNF

\[ R(A, B, C, D) \]
\[ A^+ = ABC \neq ABCD \]

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

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

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

$A^+ = ABC \neq ABCD$

$R_1(A, B, C)$

$B^+ = BC \neq ABC$

$R_2(A, D)$

$A \rightarrow B$

$B \rightarrow C$
What are the keys? 

$A \rightarrow B$

$B \rightarrow C$

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

$A^+ = ABC \neq ABCD$

$R_1(A,B,C)$

$B^+ = BC \neq ABC$

$R_{11}(B,C)$

$R_{12}(A,B)$

$R_2(A,D)$

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>

<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>
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

Let:

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

The decomposition is called \textbf{lossless} if \[ R = S_1 \bowtie S_2 \]

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

It follows that every BCNF decomposition is lossless
Schema Refinements = Normal Forms

- 1st Normal Form = all tables are flat
  - No nested attributes…
- 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
  - 3NF fixes that (is lossless and dependency-preserving), but some tables might not be in BCNF – i.e., they may have redundancy anomalies
How to split relations in SQL?
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.
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 x, Product 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 sell some products over 1000.
- For each customer, return all the high end stores that they visit.

```sql
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>Huston</td>
<td>Clob1…</td>
<td>Blob1…</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
<td>Clob2…</td>
<td>Blob2…</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
<td>Clob3…</td>
<td>Blob3…</td>
</tr>
<tr>
<td>432432</td>
<td>Ann</td>
<td>Portland</td>
<td>Clob4…</td>
<td>Blob4…</td>
</tr>
</tbody>
</table>

T2.SSN is a key \textit{and} a foreign key to T1.SSN. Same for T3.SSN

T1

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Huston</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
</tr>
</tbody>
</table>

T2

<table>
<thead>
<tr>
<th>SSN</th>
<th>Resume</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Clob1…</td>
</tr>
<tr>
<td>345345</td>
<td>Clob2…</td>
</tr>
</tbody>
</table>

T3

<table>
<thead>
<tr>
<th>SSN</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Blob1…</td>
</tr>
<tr>
<td>345345</td>
<td>Blob2…</td>
</tr>
</tbody>
</table>
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'

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

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

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

1. 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
...
Horizontal Partitioning

SELECT name
FROM Customers
WHERE city = ‘Seattle’

Which tables are inspected by the system?
-- All
Horizontal Partitioning Applications

- Performance optimization
  - Especially for data warehousing
  - E.g. one partition per month

- Distributed and parallel databases

- Data integration