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

(3 lectures)
Introduction to Data Management
CSE 344

E/R Diagrams
Class Overview

• Unit 1: Intro
• Unit 2: Relational Data Models and Query Languages
• Unit 3: Non-relational data
• Unit 4: RDMBS internals and query optimization
• Unit 5: Parallel query processing
• Unit 6: DBMS usability, conceptual design
  – E/R diagrams
  – Schema normalization
• Unit 7: Transactions
• Unit 8: Advanced topics (time permitting)
Database Design

What it is:
• Starting from scratch, design the database schema: relation, attributes, keys, foreign keys, constraints etc

Why it’s hard
• The database will be in operation for a very long time (years). Updating the schema while in production is very expensive (why?)
Database Design

• Consider issues such as:
  – What entities to model
  – How entities are related
  – What constraints exist in the domain

• Several formalisms exist
  – We discuss E/R diagrams
  – UML, model-driven architecture

• Reading: Sec. 4.1-4.6
Database Design Process

Conceptual Model:

Relational Model:
Tables + constraints
And also functional dep.

Normalization:
Eliminates anomalies

Conceptual Schema

Physical storage details
Physical Schema
Entity / Relationship Diagrams

- Entity set = a class
  - An entity = an object
- Attribute
- Relationship
Keys in E/R Diagrams

- Every entity set must have a key

Diagram:
- Product
  - name
  - price
What is a Relation?

• A mathematical definition:
  – if A, B are sets, then a relation R is a subset of $A \times B$

• $A=\{1,2,3\}$, $B=\{a,b,c,d\}$,
  $A \times B = \{(1,a),(1,b), \ldots, (3,d)\}$
  $R = \{(1,a), (1,c), (3,b)\}$

• makes is a subset of $Product \times Company$: 

![Diagram showing the relationship between Product, makes, and Company]
Multiplicity of E/R Relations

- one-one:

- many-one

- many-many
What does this say?
Attributes on Relationships

Person

name
address

Buys

date

Product

name
price

What does this say?
Multi-way Relationships

How do we model a purchase relationship between buyers, products and stores?

Can still model as a mathematical set (How?)

As a set of triples $\subseteq \text{Person} \times \text{Product} \times \text{Store}$
Q: What does the arrow mean?

A: Any person buys a given product from at most one store.

[Fine print: Arrow pointing to E means that if we select one entity from each of the other entity sets in the relationship, those entities are related to at most one entity in E]
Q: What does the arrow mean?

A: Any person buys a given product from at most one store AND every store sells to every person at most one product.
Converting Multi-way Relationships to Binary

Arrows go in which direction?
Converting Multi-way Relationships to Binary

Make sure you understand why!
3. Design Principles

What’s wrong?

Moral: Be faithful to the specifications of the application!
Design Principles: What’s Wrong?

Moral: pick the right kind of entities.
Design Principles: What’s Wrong?

Moral: don’t complicate life more than it already is.
From E/R Diagrams to Relational Schema

- Entity set → relation
- Relationship → relation
Product \((\text{prod-ID}, \text{category}, \text{price})\)

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>category</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Camera</td>
<td>99.99</td>
</tr>
<tr>
<td>Pokemn19</td>
<td>Toy</td>
<td>29.99</td>
</tr>
</tbody>
</table>
N-N Relationships to Relations

Represent this in relations
Orders\((prod-ID, cust-ID, date)\)

Shipment\((prod-ID, cust-ID, name, date)\)

Shipping-Co\((name, address)\)

<table>
<thead>
<tr>
<th>prod-ID</th>
<th>cust-ID</th>
<th>name</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>UPS</td>
<td>4/10/2011</td>
</tr>
<tr>
<td>Gizmo55</td>
<td>Joe12</td>
<td>FEDEX</td>
<td>4/9/2011</td>
</tr>
</tbody>
</table>
N-1 Relationships to Relations

Orders: prod-ID, cust-ID, date
Shipment: date
Shipping-Co: name, address

Represent this in relations
Orders (prod-ID, cust-ID, date1, name, date2)
Shipping-Co (name, address)

Remember: no separate relations for many-one relationship
Multi-way Relationships to Relations

Product

prod-ID
price

Purchase

Purchase\((\text{prod-ID}, \text{ssn}, \text{name})\)

Person

ssn
name

Store

name
address

Try this at home!
Modeling Subclasses

Some objects in a class may be special
- define a new class
- better: define a subclass

Products

- Software products
- Educational products

So --- we define subclasses in E/R
Subclasses to Relations

Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Price</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>99</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>49</td>
<td>photo</td>
</tr>
<tr>
<td>Toy</td>
<td>39</td>
<td>gadget</td>
</tr>
</tbody>
</table>

Software Product

<table>
<thead>
<tr>
<th>Name</th>
<th>platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>unix</td>
</tr>
</tbody>
</table>

Educational Product

<table>
<thead>
<tr>
<th>Name</th>
<th>Age Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>toddler</td>
</tr>
<tr>
<td>Toy</td>
<td>retired</td>
</tr>
</tbody>
</table>

Other ways to convert are possible

CSE 344 - 2017au
Modeling Union Types with Subclasses

Say: each piece of furniture is owned either by a person or by a company
Modeling Union Types with Subclasses

Say: each piece of furniture is owned either by a person or by a company

Solution 1. Acceptable but imperfect (What’s wrong ?)
Modeling Union Types with Subclasses

Solution 2: better, more laborious

Diagram:

- Owner
  - isa Person
  - isa Company
  - ownedBy FurniturePiece
Weak Entity Sets

Entity sets are weak when their key comes from other classes to which they are related.

Team(sport, number, universityName)
University(name)
What Are the Keys of R?

- A
- B
- R
- H
- T
- C
- V
- D
- E
- Q
- W
- F
- U
- L
- G
- Z
- K
Introduction to Data Management
CSE 344

Integrity Constraints
Integrity Constraints Motivation

An integrity constraint is a condition specified on a database schema that restricts the data that can be stored in an instance of the database.

- ICs help prevent entry of incorrect information
- How? DBMS enforces integrity constraints
  - Allows only legal database instances (i.e., those that satisfy all constraints) to exist
  - Ensures that all necessary checks are always performed and avoids duplicating the verification logic in each application
Constraints in E/R Diagrams

Finding constraints is part of the modeling process. Commonly used constraints:

**Keys**: social security number uniquely identifies a person.

**Single-value constraints**: a person can have only one father.

**Referential integrity constraints**: if you work for a company, it must exist in the database.

**Other constraints**: peoples’ ages are between 0 and 150.
Keys in E/R Diagrams

Underline:

No formal way to specify multiple keys in E/R diagrams
Single Value Constraints
Referential Integrity Constraints

Each product made by at most one company. Some products made by no company.

Each product made by exactly one company.
Q: What does this mean?
A: A Company entity cannot be connected by relationship to more than 99 Product entities
Constraints in SQL:

- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints: assertions

The more complex the constraint, the harder it is to check and to enforce.
Key Constraints

Product(name, category)

CREATE TABLE Product (  
    name CHAR(30) PRIMARY KEY,  
    category VARCHAR(20))

OR:

CREATE TABLE Product (  
    name CHAR(30),  
    category VARCHAR(20),  
    PRIMARY KEY (name))
Keys with Multiple Attributes

Product(name, category, price)

```
CREATE TABLE Product (  
    name CHAR(30),  
    category VARCHAR(20),  
    price INT,  
    PRIMARY KEY (name, category))
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>10</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>20</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Photo</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Gadget</td>
<td>40</td>
</tr>
</tbody>
</table>
CREATE TABLE Product (  
productID CHAR(10),
name CHAR(30),
category VARCHAR(20),
price INT,
PRIMARY KEY (productID),
UNIQUE (name, category))

There is at most one PRIMARY KEY; there can be many UNIQUE
CREATE TABLE Purchase (  
  prodName CHAR(30)  
  REFERENCES Product(name),  
  date DATETIME)  

prodName is a foreign key to Product(name)  
name must be a key in Product  

Referential integrity constraints  
May write just Product if name is PK
Foreign Key Constraints

• Example with multi-attribute primary key

CREATE TABLE Purchase (
    prodName CHAR(30),
    category VARCHAR(20),
    date DATETIME,
    FOREIGN KEY (prodName, category)
    REFERENCES Product(name, category)
)

• (name, category) must be a KEY in Product
What happens when data changes?

Types of updates:
- In Purchase: insert/update
- In Product: delete/update

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
What happens when data changes?

- SQL has three policies for maintaining referential integrity:
  - **NO ACTION** reject violating modifications (default)
  - **CASCADE** after delete/update do delete/update
  - **SET NULL** set foreign-key field to NULL
  - **SET DEFAULT** set foreign-key field to default value
    - need to be declared with column, e.g.,
      CREATE TABLE Product (pid INT DEFAULT 42)
Maintaining Referential Integrity

CREATE TABLE Purchase (  
prodName CHAR(30),  
category VARCHAR(20),  
date DATETIME,  
FOREIGN KEY (prodName, category)  
REFERENCES Product(name, category)  
ON UPDATE CASCADE  
ON DELETE SET NULL  
)
Constraints on Attributes and Tuples

• Constraints on attributes:
  - NOT NULL
  - CHECK condition
  -- obvious meaning...

• Constraints on tuples
  - CHECK condition
  -- any condition !
Constraints on Attributes and Tuples

CREATE TABLE R (  
    A int NOT NULL,  
    B int CHECK (B > 50 and B < 100),  
    C varchar(20),  
    D int,  
    CHECK (C >= 'd' or D > 0))
CREATE TABLE Product (  
  productID CHAR(10),  
  name CHAR(30),  
  category VARCHAR(20),  
  price INT CHECK (price > 0),  
  PRIMARY KEY (productID),  
  UNIQUE (name, category))
CREATE TABLE Purchase (prodName CHAR(30)
CHECK (prodName IN (SELECT Product.name
FROM Product)),
date DATETIME NOT NULL)

What does this constraint do?

What is the difference from Foreign-Key?
General Assertions

CREATE ASSERTION myAssert CHECK (NOT EXISTS (
    SELECT Product.name
    FROM Product, Purchase
    WHERE Product.name = Purchase.prodName
    GROUP BY Product.name
    HAVING count(*) > 200)
)

But most DBMSs do not implement assertions
Because it is hard to support them efficiently
Instead, they provide triggers
Introduction to Data Management
CSE 344

Design Theory and BCNF
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>
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
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**  
\( A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n \) holds in R if:

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

<table>
<thead>
<tr>
<th>( R )</th>
<th>( A_1 )</th>
<th>( \ldots )</th>
<th>( A_m )</th>
<th>( B_1 )</th>
<th>( \ldots )</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>

\( \text{if } t, t' \text{ agree here then } t, t' \text{ agree here} \)
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 → 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
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>

name → color
category → department
color, category → price
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?

<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?
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:

\[
\begin{align*}
\text{name}^+ &= \{\text{name, color}\} \\
\{\text{name, category}\}^+ &= \{\text{name, category, color, department, price}\} \\
\text{color}^+ &= \{\text{color}\}
\end{align*}
\]
Closure Algorithm

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

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

\[
\text{if } B_1, \ldots, B_n \rightarrow C \text{ is a FD and } \quad \text{B}_1, \ldots, B_n \text{ are all in } X \\
\text{then add } C \text{ to } X.
\]

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:

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

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

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

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

In class:

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

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

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 \( \{A,B\}^+ \)

\[ X = \{ A, B, C, D, E \} \]

Compute \( \{A, F\}^+ \)

\[ X = \{ A, F, B, C, D, E \} \]
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 the key of \( R \)?
Find all FD’s implied by:

- A, B → C
- A, D → B
- B → D
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 $X$:

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

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 $A_1, \ldots, A_n$ 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 $X^+$
- 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)

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

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 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

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

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
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:

\[ \forall X, \text{ either } X^+ = X \text{ or } X^+ = [\text{all attributes}] \]
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

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
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)
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 $X^+ \neq [\text{all attributes}]$
Example BCNF Decomposition

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

Iteration 1: \textbf{Person}: SSN+ = SSN, name, age, hairColor
Decompose into: \textbf{P}(SSN, name, age, hairColor)
  \textbf{Phone}(SSN, phoneNumber)

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

Note the keys!

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

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

$A \rightarrow B$

$B \rightarrow C$
Example: BCNF

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

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

[Diagram showing dependencies: A → B, B → C]
Example: BCNF

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

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

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

\[ R_2(A,D) \]

A \rightarrow B
B \rightarrow C
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
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 1:**

<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>
<tr>
<td>Gizmo</td>
<td>19.99</td>
</tr>
</tbody>
</table>

**Table 2:**

<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>

**Diagram:**

- Name: Gizmo
  - Price: 19.99
  - Category: Gadget

- Name: OneClick
  - Price: 24.99
  - Category: Camera

- Name: Gizmo
  - Price: 19.99
  - Category: Camera
**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>

CSE 344 - 2017au
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:
\[ S_1 = \text{projection of } R \text{ on } A_1, \ldots, A_n, B_1, \ldots, B_m \]
\[ S_2 = \text{projection of } R \text{ on } A_1, \ldots, A_n, C_1, \ldots, C_p \]

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

It follows that every BCNF decomposition is lossless
Testing for Lossless Join

If we decompose $R$ into $\Pi_{S_1}(R), \Pi_{S_2}(R), \Pi_{S_3}(R), \ldots$ Is it true that $S_1 \bowtie S_2 \bowtie S_3 \bowtie \ldots = R$?

That is true if we can show that:

$R \subseteq S_1 \bowtie S_2 \bowtie S_3 \bowtie \ldots$ always holds (why?)

$R \supseteq S_1 \bowtie S_2 \bowtie S_3 \bowtie \ldots$ neet to check
The Chase Test for Lossless Join

R(A,B,C,D) = S1(A,D) \Join S2(A,C) \Join S3(B,C,D)
R satisfies: A \rightarrow B, B \rightarrow C, CD \rightarrow A

S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R),
hence \ R \subseteq S1 \Join S2 \Join S3

Need to check: \ R \supseteq S1 \Join S2 \Join S3
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D)
R satisfies: A \rightarrow B, B \rightarrow C, CD \rightarrow A

S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R),

hence R \subseteq S1 \bowtie S2 \bowtie S3

Need to check: R \supseteq S1 \bowtie S2 \bowtie S3

Suppose (a,b,c,d) \in S1 \bowtie S2 \bowtie S3 Is it also in R?
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]

\( R \) satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\[ S1 = \Pi_{AD}(R), \quad S2 = \Pi_{AC}(R), \quad S3 = \Pi_{BCD}(R), \]

hence \( R \subseteq S1 \bowtie S2 \bowtie S3 \)

Need to check: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)

Suppose \((a,b,c,d) \in S1 \bowtie S2 \bowtie S3\) Is it also in \( R \)?

\( R \) must contain the following tuples:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
</tr>
</tbody>
</table>

Why?

\[(a,d) \in S1 = \Pi_{AD}(R)\]
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

\[ R(A, B, C, D) = S_1(A, D) \bowtie S_2(A, C) \bowtie S_3(B, C, D) \]

\( R \) satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\( S_1 = \Pi_{AD}(R), \quad S_2 = \Pi_{AC}(R), \quad S_3 = \Pi_{BCD}(R), \)

\( R \subseteq S_1 \bowtie S_2 \bowtie S_3 \)

Need to check: \( R \supseteq S_1 \bowtie S_2 \bowtie S_3 \)

Suppose \((a, b, c, d) \in S_1 \bowtie S_2 \bowtie S_3\) Is it also in \( R \)?

\( R \) must contain the following tuples:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
<td>(a,d) ( \in S_1 = \Pi_{AD}(R) )</td>
</tr>
<tr>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
<td>(a,c) ( \in S_2 = \Pi_{BD}(R) )</td>
</tr>
</tbody>
</table>
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

\[ R(A,B,C,D) = S_1(A,D) \bowtie S_2(A,C) \bowtie S_3(B,C,D) \]

\( R \) satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\( S_1 = \Pi_{AD}(R), S_2 = \Pi_{AC}(R), S_3 = \Pi_{BCD}(R), \)

hence \( R \subseteq S_1 \bowtie S_2 \bowtie S_3 \)

Need to check: \( R \supseteq S_1 \bowtie S_2 \bowtie S_3 \)

Suppose \((a,b,c,d) \in S_1 \bowtie S_2 \bowtie S_3\) Is it also in \( R \)?

\( R \) must contain the following tuples:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Why?

- \((a,d) \in S_1 = \Pi_{AD}(R)\)
- \((a,c) \in S_2 = \Pi_{BD}(R)\)
- \((b,c,d) \in S_3 = \Pi_{BCD}(R)\)
Example from textbook Ch. 3.4.2

The Chase Test for Lossless Join

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]

R satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\[ S1 = \Pi_{AD}(R), \quad S2 = \Pi_{AC}(R), \quad S3 = \Pi_{BCD}(R), \]

hence \( R \subseteq S1 \bowtie S2 \bowtie S3 \)

Need to check: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)

Suppose \( (a,b,c,d) \in S1 \bowtie S2 \bowtie S3 \) Is it also in R?

R must contain the following tuples:

“Chase” them (apply FDs):

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b1</td>
<td>c1</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Why?

- \((a,d) \in S1 = \Pi_{AD}(R)\)
- \((a,c) \in S2 = \Pi_{BD}(R)\)
- \((b,c,d) \in S3 = \Pi_{BCD}(R)\)
Example from textbook Ch. 3.4.2

**The Chase Test for Lossless Join**

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]

R satisfies: \( A \rightarrow B \), \( B \rightarrow C \), \( CD \rightarrow A \)

\[ S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R), \]
hence \( R \subseteq S1 \bowtie S2 \bowtie S3 \)

Need to check: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)

Suppose \((a,b,c,d) \in S1 \bowtie S2 \bowtie S3\) Is it also in R?

R must contain the following tuples:

```
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D \\
\hline
a & b1 & c1 & d \\
\hline
\end{array}
```

```
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D \\
\hline
a & b2 & c & d2 \\
\hline
\end{array}
```

```
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D \\
\hline
a3 & b & c & d \\
\hline
\end{array}
```

```
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D \\
\hline
a & b1 & c & d \\
\hline
\end{array}
```

```
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D \\
\hline
a & b1 & c & d2 \\
\hline
\end{array}
```

```
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D \\
\hline
a3 & b & c & d \\
\hline
\end{array}
```

```
\begin{array}{|c|c|c|c|}
\hline
A & B & C & D \\
\hline
(a,d) & \in S1 = \Pi_{AD}(R) \\
\hline
(a,c) & \in S2 = \Pi_{BD}(R) \\
\hline
(b,c,d) & \in S3 = \Pi_{BCD}(R) \\
\hline
\end{array}
```

“Chase” them (apply FDs):
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

\[ R(A,B,C,D) = S1(A,D) \bowtie S2(A,C) \bowtie S3(B,C,D) \]

R satisfies: \( A \rightarrow B, B \rightarrow C, CD \rightarrow A \)

\( S1 = \Pi_{AD}(R), S2 = \Pi_{AC}(R), S3 = \Pi_{BCD}(R), \)

hence \( R \subseteq S1 \bowtie S2 \bowtie S3 \)

Need to check: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)

Suppose \((a,b,c,d) \in S1 \bowtie S2 \bowtie S3\) Is it also in R?

R must contain the following tuples:

```
Why ?
(a,d) \in S1 = \Pi_{AD}(R)
(a,c) \in S2 = \Pi_{BD}(R)
(b,c,d) \in S3 = \Pi_{BCD}(R)
```

“Chase” them (apply FDs):

Hence R contains \((a,b,c,d)\)
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
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 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'
```
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
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>
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?
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)
Horizontal Partitioning

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

```sql
SELECT name
FROM CustomersInSeattle
```

CustomersInHouston(ssn,name,city)
CustomersInSeattle(ssn,name,city)
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