Lecture 5:
E/R Diagrams and Constraints
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 exists
  – We discuss E/R diagrams

• 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 $Product \times Company$: 
Multiplicity of E/R Relations

- one-one:

- many-one

- many-many
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 (Q. how?)

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

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

[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]
Arrows in Multiway Relationships

Q: What does the arrow mean?

A: A given 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 app!
Design Principles: What’s Wrong?

Moral: pick the right kind of entities.
Moral: don’t complicate life more than it already is.
From E/R Diagrams to Relational Schema

- Entity set \(\rightarrow\) relation
- Relationship \(\rightarrow\) relation
Entity Set to Relation

Product(prod-ID, category, 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

DATA514 - Winter 2018
N-N Relationships to Relations

Orders\((\text{prod-ID, cust-ID, date})\)

Shipment\((\text{prod-ID, cust-ID, name, date})\)

Shipping-Co\((\text{name, address})\)

<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

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

- name
- address

Person

- ssn
- name

Store

Purchase(prod-ID, ssn, name)
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

Product

- name
- category
- price

isa

Subclasses:
- Software Product
  - platforms
- Educational Product
  - Age Group

DATA514 - Winter 2018
Subclasses to Relations

Other ways to convert are possible

DATA514 - Winter 2018
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
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?
What makes good schemas?

Why so many database tables???

I updated a schema once
It sucked
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
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

No formal way to specify multiple keys in E/R diagrams

Underline:

Product

name

category

price

Person

address

name

ssn
Single Value Constraints

makes

vs.

makes
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>Gizmo</td>
<td>Gadget</td>
<td>40</td>
</tr>
</tbody>
</table>
Other Keys

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
Foreign Key Constraints

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
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  
)

<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>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Snap</td>
<td>Camera</td>
</tr>
<tr>
<td>EasyShoot</td>
<td>Camera</td>
</tr>
</tbody>
</table>
Constraints on Attributes and Tuples

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

• Constraints on tuples
  - **CHECK** condition
Constraints on Attributes and Tuples

CREATE TABLE Product (
    productID CHAR(10),
    name CHAR(30),
    category VARCHAR(20),
    price INT CHECK (price > 0),
    PRIMARY KEY (productID))
CREATE TABLE Product (  
    productID CHAR(10),  
    name CHAR(30),  
    category VARCHAR(20)  
    CHECK (category in ('toy','gadget','apparel')),  
    price INT CHECK (price > 0),  
    PRIMARY KEY (productID))
CREATE TABLE Product (  
  productID CHAR(10),  
  name CHAR(30) NOT NULL,  
  category VARCHAR(20)  
    CHECK (category in ('toy', 'gadget', 'apparel')),  
  price INT CHECK (price > 0),  
  PRIMARY KEY (productID))
Constraints on Attributes and Tuples

```sql
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))
```
Constraints on Attributes and Tuples

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

```sql
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
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

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
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:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]

\[ A_1 \ldots A_n \text{ determines } B_1 \ldots B_m \]
Functional Dependencies (FDs)

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

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

\[
(t.A_1 = t'.A_1 \land \ldots \land t.A_m = t'.A_m \Rightarrow t.B_1 = t'.B_1 \land \ldots \land t.B_n = t'.B_n )
\]
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 → 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
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?
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:

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

Then this FD also holds:

- name, category → price

If we find out from application domain that a relation satisfies some FDs, it doesn’t mean that we found all the FDs that it satisfies! There could be more FDs implied by the ones we have.
Closure of a set of Attributes

**Given** a set of attributes \( A_1, \ldots, A_n \)

The closure, \( \{A_1, \ldots, A_n\}^+ \) = the set of attributes B s.t. \( A_1, \ldots, A_n \rightarrow B \)

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

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

\(X = \{A_1, \ldots, A_n\}\).

Repeat until \(X\) doesn’t change do:

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

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

Hence: name, category \(\rightarrow\) color, department, price
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 \( \{A, B\}^+ \)  \( X = \{A, B, \} \)

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

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

In class:

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

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

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

In class:

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

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

\[
\begin{align*}
\text{Compute } \{A, B\}^+ & \quad X = \{A, B, C, D, E\} \\
\text{Compute } \{A, F\}^+ & \quad X = \{A, F, B, C, D, E\}
\end{align*}
\]

What is the key of \( R \)?

DATA514 - Winter 2018
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:

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

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 only 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) + = { 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

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

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

The only key is: \{SSN, PhoneNumber\}
Hence \(SSN \rightarrow Name, City\) is a “bad” dependency

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?

SSN → Name, City

DATA514 - Winter 2018
Example BCNF Decomposition

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

SSN → name, age
age → hairColor

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

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

- SSN → name, age
- age → hairColor

**Iteration 1:**

**Person:** $SSN^+ = SSN, name, age, hairColor$

**Decompose into:**

- $P(SSN, name, age, hairColor)$
- $Phone(SSN, phoneNumber)$

Find $X$ s.t.: $X \neq X^+$ and $X^+ \neq \text{[all attributes]}$
Find $X$ s.t.: $X \neq X^+$ and $X^+ \neq \{\text{all attributes}\}$

Example BCNF Decomposition

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

- $\text{SSN} \rightarrow \text{name, age}$
- $\text{age} \rightarrow \text{hairColor}$

**Iteration 1:**
- **Person:** $\text{SSN}^+ = \text{SSN, name, age, hairColor}$
- Decompose into:
  - $P(\text{SSN, name, age, hairColor})$
  - $\text{Phone}(\text{SSN, phoneNumber})$

**Iteration 2:**
- **P:** $\text{age}^+ = \text{age, hairColor}$
- Decompose:
  - $\text{People}(\text{SSN, name, age})$
  - $\text{Hair}(\text{age, hairColor})$
  - $\text{Phone}(\text{SSN, phoneNumber})$
Find $X$ s.t.: $X \neq X^+$ and $X^+ \neq$ [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:
- $People(SSN, name, age)$
- $Hair(age, hairColor)$
- $Phone(SSN, phoneNumber)$

Note the keys!
Example: BCNF

A \rightarrow B
B \rightarrow C
Example: BCNF

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

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

\[ A^+ = ABC \neq ABCD \]
Example: BCNF

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

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

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

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

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

\[ R_1(A,B,C) \]
\[ B^+ = BC \neq ABC \]

\[ R_2(A,D) \]

\[ A \rightarrow B \]
\[ B \rightarrow C \]
Example: BCNF

What are the keys?

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

$A^+ = ABC \neq ABCD$

$R_1(A, B, C)$

$B^+ = BC \neq ABC$

$R_{11}(B, C)$

$R_{12}(A, B)$

$R_2(A, D)$

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>
<tr>
<td>Gizmo</td>
<td>19.99</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>

Name | Category
--- | ---
Gizmo | Gadget
OneClick | Camera
Gizmo | Camera

Price | Category
--- | ---
19.99 | Gadget
24.99 | Camera
19.99 | Camera

DATA514 - Winter 2018
Decomposition in General

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

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 \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
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: \( R \supseteq S1 \bowtie S2 \bowtie S3 \)
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

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

\[ R \text{ satisfies: } A \rightarrow B, \ B \rightarrow C, \ CD \rightarrow A \]

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

\[ \text{hence } R \subseteq S_1 \bowtie S_2 \bowtie S_3 \]

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 \(R\)?

\[ R \text{ must contain the following tuples:} \]

<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? 

\((a,d) \in S_1 = \Pi_{AD}(R)\)
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:

<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>
</tbody>
</table>

Why?

\( (a,d) \in S1 = \Pi_{AD}(R) \)

\( (a,c) \in S2 = \Pi_{BD}(R) \)
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: \( A \rightarrow B \), \( B \rightarrow C \), \( CD \rightarrow A \)

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

<table>
<thead>
<tr>
<th></th>
<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>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b2</td>
<td>c</td>
<td>d2</td>
<td></td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

Why?

\((a,d) \in S1 = \Pi_{AD}(R)\)

\((a,c) \in S2 = \Pi_{BD}(R)\)

\((b,c,d) \in S3 = \Pi_{BCD}(R)\)
The Chase Test for Lossless Join

Example from textbook Ch. 3.4.2

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

\[ R \text{ satisfies: } A \rightarrow B, \ B \rightarrow C, \ CD \rightarrow A \]

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

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:

“Chase” them (apply FDs):

\begin{tabular}{|c|c|c|c|}
  \hline
  A & B & C & D \\
  \hline
  a & b1 & c1 & d \\
  \hline
  a & b2 & c & d2 \\
  \hline
  a3 & b & c & d \\
  \hline
\end{tabular}

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

The Chase Test for Lossless Join

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

\[ R \text{ satisfies: } A \rightarrow B, B \rightarrow C, CD \rightarrow A \]

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

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:

"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>b1</td>
<td>c</td>
<td>d2</td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

<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>c</td>
<td>d</td>
</tr>
<tr>
<td>a3</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

Why?

\((a,d) \in S1 = \Pi_{AD}(R)\)
\((a,c) \in S2 = \Pi_{BD}(R)\)
\((b,c,d) \in S3 = \Pi_{BCD}(R)\)
The Chase Test for Lossless Join

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

$R$ 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:

```
A  B  C  D
a  b1 c1 d
a  b1 c d2
a3 b  c  d
```

"Chase" them (apply FDs):

```
A \rightarrow B
```

```
A  B  C  D
a  b1 c1 d
a  b1 c d2
a3 b  c  d
```

```
B \rightarrow C
```

```
A  B  C  D
a  b1 c1 d
a  b1 c d2
a3 b  c  d
```

```
CD \rightarrow A
```

```
A  B  C  D
a  b1 c1 d
a  b1 c d2
a3 b  c  d
```

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