Introduction to Data Management

Design Theory

Alyssa Pittman
Based on slides by Jonathan Leang, Dan Suciu, et al
Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle
Recap: Other Constraints

- **CHECK** (condition)
  - Single attribute
  - Single tuples

CREATE TABLE User (
    uid INT PRIMARY KEY,
    firstName TEXT,
    lastName TEXT,
    age INT **CHECK** (age > 12 AND age < 120),
    email TEXT,
    phone TEXT,
    **CHECK** (email IS NOT NULL OR phone IS NOT NULL)
);
Assertions

- Hard to support
- Usually impractical
- Usually not supported
  - Simulated with triggers

CREATE ASSERTION myAssert CHECK
  (NOT EXISTS (
    SELECT Product.name
    FROM Product, Purchase
    WHERE Product.name = Purchase.prodName
    GROUP BY Product.name
    HAVING count(*) > 200));
Triggers activate on a specified event

CREATE TRIGGER LowCredit ON Purchasing.PurchaseOrderHeader
AFTER INSERT AS

   IF (ROWCOUNT_BIG() = 0) RETURN;
   IF EXISTS (SELECT *
               FROM Purchasing.PurchaseOrderHeader AS p
               JOIN inserted AS i
               ON p.PurchaseOrderID = i.PurchaseOrderID
               JOIN Purchasing.Vendor AS v
               ON v.BusinessEntityID = p.VendorID
               WHERE v.CreditRating = 5 )
   BEGIN
      RAISERROR ('A vendor's credit rating is too low to accept new purchase orders.', 16, 1);
      ROLLBACK TRANSACTION;
   END;

GO

= you don't need to study this for the class
Recap

- **ER Diagrams**
  - Conceptual modeling
  - Rules of thumb for converting diagram into schema

```
CREATE TABLE Company (
    name VARCHAR(100) PRIMARY KEY,
    ...);
CREATE TABLE Product (
    name VARCHAR(100) PRIMARY KEY,
    cname VARCHAR(100) REFERENCES Company ...);
```
The Database Design Process

Conceptual Model

Relational Model
- + Schema
- + Constraints

Conceptual Schema
- + Normalization

Physical Schema
- + Partitioning
- + Indexing
Goals for Today

▪ Figure out the fundamentals of what makes a good schema
Outline

▪ Background
  • Anomalies, i.e. things we want to avoid
  • Functional Dependencies (FDs)
  • Closures and formal definitions of keys
▪ Normalization: BCNF Decomposition
▪ Losslessness
Think About This

Make a simple directory that can:

- Hold information about name, SSN, phone, and city
- Associate people with the city they live in
- Associate people with any phone numbers they have

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-9999</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-8888</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>415-555-7777</td>
<td>San Francisco</td>
</tr>
</tbody>
</table>

The above instance does the job, but are there issues?
Think About This

Make a simple directory that can:

• Hold information about name, SSN, phone, and city
• Associate **people** with the **city** they live in
• Associate **people** with any **phone numbers** they have

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-9999</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-8888</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>415-555-7777</td>
<td>San Francisco</td>
</tr>
</tbody>
</table>

**Anomalies:**

• **Redundancy** □ **Slow Update**
  • Change Fred’s city to Bellevue (two rows!)

• **Deletion Anomalies**
  • How to delete Joe’s phone without deleting Joe?
Think About This

Make a simple directory that can:
- Hold information about name, SSN, phone, and city
- Associate **people** with the **city** they live in
- Associate **people** with any **phone numbers** they have

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-9999</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-8888</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>415-555-7777</td>
<td>San Francisco</td>
</tr>
</tbody>
</table>

Anomalies:
- **Redundancy → Slow Update**
  - Change Fred’s city to Bellevue (two rows!)
- **Deletion Anomalies**
  - How to delete Joe’s phone without deleting Joe?
Think About This

Make a simple directory that can:
- Hold information about name, SSN, phone, and city
- Associate people with the city they live in
- Associate people with any phone numbers they have

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-9999</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-8888</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>415-555-7777</td>
<td>San Francisco</td>
</tr>
</tbody>
</table>

Anomalies:
- **Redundancy** □ Slow Update
  - Change Fred’s city to Bellevue (two rows!)
- **Deletion Anomalies**
  - How to delete Joe’s phone without deleting Joe?
Think About This

We can solve the anomalies by converting this

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Phone</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-9999</td>
<td>Seattle</td>
</tr>
<tr>
<td>Fred</td>
<td>123-45-6789</td>
<td>206-555-8888</td>
<td>Seattle</td>
</tr>
<tr>
<td>Joe</td>
<td>987-65-4321</td>
<td>415-555-7777</td>
<td>San Francisco</td>
</tr>
</tbody>
</table>

into this

<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>San Francisco</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>123-45-6789</td>
<td>206-555-9999</td>
</tr>
<tr>
<td>123-45-6789</td>
<td>206-555-8888</td>
</tr>
<tr>
<td>987-65-4321</td>
<td>415-555-7777</td>
</tr>
</tbody>
</table>

How can we systematically avoid anomalies?
Informal Design Guidelines

- Semantics of attributes should be self-evident
- Avoid redundant information in tuples
- Avoid NULL values in tuples
- Disallow the generation of “spurious” tuples
  - If certain tuples shouldn’t exist, don’t allow them
Database Design or Logical Design or Relational Schema Design is the process of organizing data into a database model. This is done by considering what data needs to be stored and the interrelationship of the data.
Database Design is about (1) characterizing data and (2) organizing data.
Database Design is about

(1) characterizing data and (2) organizing data
Database Design

Database Design is about
(1) characterizing data and (2) organizing data

How to talk about properties we know or see in the data
Data Interrelationships

How do we start talking about data interrelationships?

▪ What rules govern our data?
  • Domain knowledge
    • Dimension vs measure
  • Pattern analysis
Data Interrelationships

How do we start talking about data interrelationships?

▪ What rules govern our data?
  • Domain knowledge
    • Dimension vs measure
  • Pattern analysis

The rules that are known to us since we made them up or they correlate to things in the real world.
Data Interrelationships

How do we start talking about data interrelationships?

- What rules govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

The rules that are known to us since we made them up or they correlate to things in the real world

[ex] An engineer knows that a plane model determines the plane’s wingspan
How do we start talking about data interrelationships?

▪ What rules govern our data?
  • Domain knowledge
    • Dimension vs measure
  • Pattern analysis
Data Interrelationships

How do we start talking about data interrelationships?

- What rules govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

Rules that are found by finding correlations within the given data
How do we start talking about data interrelationships?

- What rules govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

Rules that are found by finding correlations within the given data:

- Data mining
- Knowledge Discovery in Databases (KDD)
Data Interrelationships

How do we start talking about data interrelationships?

▪ What **rules** govern our data?
  
  • Domain knowledge
  
    • Dimension vs measure
  
  • Pattern analysis
Data Interrelationships

How do we start talking about data interrelationships?

- What **rules** govern our data?
  - Domain knowledge
    - Dimension vs measure
  - Pattern analysis

![Diagram showing relationships between Identifiers, Dependent Variables, Correlation, and Dependency]
A *Functional Dependency* $A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$ holds in the relation $R$ if:

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

Informally, some attributes determine other attributes.

Warning! Dependency does not imply causation!
Fundamentals of FDs

Armstrong’s Axioms

▪ Axiom of **Reflexivity** *(Trivial FD)*

▪ Axiom of **Augmentation**

▪ Axiom of **Transitivity**
Fundamentals of FDs

Armstrong’s Axioms

- Axiom of Reflexivity (Trivial FD)
  \[
  \text{If } B \subseteq A \quad \text{then } A \rightarrow B
  \]

- Axiom of Augmentation

- Axiom of Transitivity
Fundamentals of FDs

Armstrong’s Axioms

▪ Axiom of Reflexivity (Trivial FD)
  If \( B \subseteq A \) then \( A \rightarrow B \)
  
  \([\text{ex}]\) \{\text{name}\} \subseteq \{\text{name, job}\} \text{ so } \{\text{name, job}\} \rightarrow \{\text{name}\}

▪ Axiom of Augmentation

▪ Axiom of Transitivity
Fundamentals of FDs

Armstrong’s Axioms

- **Axiom of Reflexivity (Trivial FD)**
  
  If \( B \subseteq A \) then \( A \rightarrow B \)

  \[
  \text{[ex]} \quad \{\text{name}\} \subseteq \{\text{name, job}\} \Rightarrow \{\text{name, job}\} \rightarrow \{\text{name}\}
  \]

- **Axiom of Augmentation**

  If \( A \rightarrow B \) then \( \forall C, AC \rightarrow BC \)

- **Axiom of Transitivity**
Fundamentals of FDs

Armstrong’s Axioms

- **Axiom of Reflexivity (Trivial FD)**
  \[
  \text{If } B \subseteq A \text{ then } A \rightarrow B
  \]
  
  \[\text{ex}] \quad \{\text{name}\} \subseteq \{\text{name, job}\} \text{ so } \{\text{name, job}\} \rightarrow \{\text{name}\}\]

- **Axiom of Augmentation**
  \[
  \text{If } A \rightarrow B \text{ then } \forall C, AC \rightarrow BC
  \]
  
  \[\text{ex}] \quad \{\text{ID}\} \rightarrow \{\text{name}\} \text{ so } \{\text{ID, job}\} \rightarrow \{\text{name, job}\}\]

- **Axiom of Transitivity**
Fundamentals of FDs

Armstrong’s Axioms

- **Axiom of Reflexivity (Trivial FD)**
  
  If $B \subseteq A$ then $A \rightarrow B$

  
  [ex]  
  \[
  \{\text{name}\} \subseteq \{\text{name, job}\} \text{ so } \{\text{name, job}\} \rightarrow \{\text{name}\}
  \]

- **Axiom of Augmentation**
  
  If $A \rightarrow B$ then $\forall C, AC \rightarrow BC$

  
  [ex]  
  \[
  \{\text{ID}\} \rightarrow \{\text{name}\} \text{ so } \{\text{ID, job}\} \rightarrow \{\text{name, job}\}
  \]

- **Axiom of Transitivity**
  
  If $A \rightarrow B$ and $B \rightarrow C$ then $A \rightarrow C$
Armstrong’s Axioms

- **Axiom of Reflexivity (Trivial FD)**
  \[
  \text{If } B \subseteq A \quad \text{then } A \rightarrow B
  \]
  
  [ex] \{name\} \subseteq \{name, job\} so \{name, job\} \rightarrow \{name\}

- **Axiom of Augmentation**
  \[
  \text{If } A \rightarrow B \quad \text{then } \forall C, AC \rightarrow BC
  \]
  
  [ex] \{ID\} \rightarrow \{name\} so \{ID, job\} \rightarrow \{name, job\}

- **Axiom of Transitivity**
  \[
  \text{If } A \rightarrow B \text{ and } B \rightarrow C \quad \text{then } A \rightarrow C
  \]
  
  [ex] \{ID\} \rightarrow \{name\} and \{name\} \rightarrow \{initials\}
  so \{ID\} \rightarrow \{initials\}
Fundamentals of FDs

Interesting Secondary Rules

- **Pseudo Transitivity**
  If \( A \rightarrow BC \) and \( C \rightarrow D \) then \( A \rightarrow BD \)

- **Extensivity**
  If \( A \rightarrow B \) then \( A \rightarrow AB \)
Fundamentals of FDs

Can I do this to FDs?

I only know \{ID\} \rightarrow \{name\}
So \{ID, \textit{hair color}\} \rightarrow \{name\}
Can I do this to FDs?

I only know \( \{ID\} \rightarrow \{name\} \)

So \( \{ID, \text{hair color}\} \rightarrow \{name\} \)

Yes!
Fundamentals of FDs

Can I do this to FDs?

I only know \{ID\} \rightarrow \{name\}
So \{ID, \textit{hair color}\} \rightarrow \{name\}

Yes!

Adding more attributes to the antecedent can never remove attributes in the consequent.
Fundamentals of FDs

What about this?

I only know \{ID\} \rightarrow \{name\}

So \{ID\} \rightarrow \{name, hair color\}
What about this?

I only know \( \{ID\} \rightarrow \{\text{name}\} \)

So \( \{ID\} \rightarrow \{\text{name, hair color}\} \)

No!
What about this?

I only know $\{ID\} \rightarrow \{name\}$

So $\{ID\} \rightarrow \{name, \text{hair color}\}$

No!

No way to use the axioms to introduce hair color to the consequent without also introducing it to the antecedent.
Finding Keys

All this talk about FDs sounds awfully similar to keys...
Closure

The **Closure** of the set \( \{A_1, \ldots, A_m\} \), written as \( \{A_1, \ldots, A_m\}^+ \), is the set of attributes \( B \) is such that \( A_1, \ldots, A_m \rightarrow B \).

A closure finds **everything a set of attributes determines**.

Closure (example)

Given the functional dependencies:

- \( SSN \rightarrow Name \)
- \( Name \rightarrow Initials \)

We can derive some closures:

- \( Name^+ = \)
- \( SSN^+ = \)
- \( Initials^+ = \)
- \( \{SSN, Initials\}^+ = \)
**Closure**

The **Closure** of the set \( \{A_1, \ldots, A_m\} \), written as \( \{A_1, \ldots, A_m\}^{+} \), is the set of attributes \( B \) is such that \( A_1, \ldots, A_m \rightarrow B \).

A closure finds **everything** a set of attributes determines.

---

**Closure (example)**

Given the functional dependencies:
- \( SSN \rightarrow Name \)
- \( Name \rightarrow Initials \)

We can derive some closures:
- \( Name^{+} = \{Name, Initials\} \)
- \( SSN^{+} = \)
- \( Initials^{+} = \)
- \( \{SSN, Initials\}^{+} = \)
Closure

The **Closure** of the set \( \{A_1, ..., A_m\} \), written as \( \{A_1, ..., A_m\}^+ \), is the set of attributes \( B \) is such that \( A_1, ..., A_m \rightarrow B \).

A closure finds **everything a set of attributes determines**.

**Closure (example)**

Given the functional dependencies:
- \( SSN \rightarrow Name \)
- \( Name \rightarrow Initials \)

We can derive some closures:
- \( Name^+ = \{Name, Initials\} \)
- \( SSN^+ = \{SSN, Name, Initials\} \)
- \( Initials^+ = \)
- \( \{SSN, Initials\}^+ = \)
Closure

The **Closure** of the set \( \{A_1, \ldots, A_m\} \), written as \( \{A_1, \ldots, A_m\}^+ \), is the set of attributes \( B \) is such that \( A_1, \ldots, A_m \rightarrow B \).

A closure finds **everything** a set of attributes determines.

---

**Closure (example)**

**Given the functional dependencies:**
- \( SSN \rightarrow Name \)
- \( Name \rightarrow Initials \)

**We can derive some closures:**
- \( Name^+ = \{Name, Initials\} \)
- \( SSN^+ = \{SSN, Name, Initials\} \)
- \( Initials^+ = \{Initials\} \)
- \( \{SSN, Initials\}^+ = \)
Closure

The **Closure** of the set $\{A_1, \ldots, A_m\}$, written as $\{A_1, \ldots, A_m\}^+$, is the set of attributes $B$ is such that $A_1, \ldots, A_m \rightarrow B$.

A closure finds **everything** a set of attributes determines.

**Closure (example)**

Given the functional dependencies:
- $SSN \rightarrow Name$
- $Name \rightarrow Initials$

We can derive some closures:
- $Name^+ = \{Name, Initials\}$
- $SSN^+ = \{SSN, Name, Initials\}$
- $Initials^+ = \{Initials\}$
- $\{SSN, Initials\}^+ = \{SSN, Name, Initials\}$
Closure

Closure Algorithm

\[ X = \{A_1, ..., A_m\} \]

**Repeat until \( X \) does not change:**

\textit{if} \( B_1, ..., B_n \rightarrow C \) is a FD \textbf{and} \( B_1, ..., B_n \in X \)

\textbf{then} \( X \leftarrow X \cup C \)

In practice:
Repeted use of transitivity
Closure

Closure Algorithm

Find the closure of \{A_1, ..., A_m\}

\[
X = \{A_1, ..., A_m\}
\]

Repeat until \(X\) does not change:

if \(B_1, ..., B_n \rightarrow C\) is a FD and \(B_1, ..., B_n \in X\)
then \(X \leftarrow X \cup C\)

In practice:
Repeated use of transitivity

If a FD applies, add the consequent to the answer
Let’s say we have the following relations and FDs:

Restaurants(rid, name, rating, popularity)

rid $\rightarrow$ name
rid $\rightarrow$ rating
rating $\rightarrow$ popularity

Compute $\{rid\}^+$
Closure Example

Let’s say we have the following relations and FDs:

Restaurants(rid, name, rating, popularity)

rid → name

rid → rating

rating → popularity

Compute \( \{rid\}^+ \)

\( \{rid\}^+ = \{rid, name, rating, popularity\} \)

it’s a key!
Finding Keys

What do FDs and Closures do for us?
- Characterize the interrelationships of data
- Able to find keys
Finding Keys

**Superkey**

A *Superkey* is a set of attributes $A_1, ..., A_n$ s.t. for any single attribute $B$:

$$A_1, ..., A_n \rightarrow B$$

In other words, for the set of all attributes $C$ in the relation $R$, the set

$\{A_1, ..., A_n\}$ is a superkey iff $\{A_1, ..., A_n\}^+ = C$

**Key**

A *Key* is a minimal superkey, i.e. no subset of a key is a superkey.

**Candidate Key**

When a relation has multiple keys, each key is a *Candidate Key*. 
Usefulness of Keys in Design

What intuitions do we get from data interrelationships?

- FDs that are not superkeys hint at redundancy
  - If a FD antecedent is not a superkey, we can remove redundant information, i.e. the FD consequent
Usefulness of Keys in Design

What intuitions do we get from data interrelationships?

- FDs that are not superkeys hint at redundancy
  - If a FD antecedent is not a superkey, we can remove redundant information, i.e. the FD consequent

- Rephrased
  - $A \rightarrow B$ is fine if $A$ is a superkey
  - Otherwise, we can extract $B$
Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

- rid △ name
- rid △ rating
- rating △ popularity

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
<th>popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
<td>OK</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco’s Tacos</td>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cery</td>
<td>5</td>
<td>High</td>
</tr>
</tbody>
</table>
Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

- rid → name
  - Fine because rid is a superkey
- rid → rating
- rating → popularity

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
<th>popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
<td>OK</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco’s Tacos</td>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cery</td>
<td>5</td>
<td>High</td>
</tr>
</tbody>
</table>
Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

- rid = name: Fine because rid is a superkey
- rid = rating
- rating = popularity

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
<th>popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
<td>OK</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco’s Tacos</td>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cery</td>
<td>5</td>
<td>High</td>
</tr>
</tbody>
</table>
Usefulness of Keys in Design

Restaurants(rid, name, rating, popularity)

rid ∨ name

Fine because rid is a superkey

rating ∨ popularity

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
<th>popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
<td>OK</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco’s Tacos</td>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cery</td>
<td>5</td>
<td>High</td>
</tr>
</tbody>
</table>

Redundancy!
Database Design is about

(1) characterizing data and (2) organizing data

How to talk about properties we know or see in the data
Database Design is about (1) characterizing data and (2) organizing data.

How to organize data to promote ease of use and efficiency.
Normal Forms

- **1NF** → Flat
- **2NF** → No partial FDs (obsolete)
- **3NF** → Preserve all FDs, but allow anomalies
- **BCNF** → No transitive FDs, but can lose FDs
- **4NF** → Considers multi-valued dependencies
- **5NF** → Considers join dependencies (hard to do)
Normal Forms

▪ **1NF** □ Flat
▪ **2NF** □ No partial FDs (obsolete)
▪ **3NF** □ Preserve all FDs, but allow anomalies
▪ **BCNF** □ No transitive FDs, but can lose FDs
▪ **4NF** □ Considers multi-valued dependencies
▪ **5NF** □ Considers join dependencies (hard to do)
**1NF**

A relation $R$ is in **First Normal Form** if all attribute values are atomic. Attribute values cannot be multivalued. Nested relations are not allowed.

We call data in 1NF “flat.”
A relation $R$ is in **Boyce-Codd Normal Form (BCNF)** if for every non-trivial dependency, $X \rightarrow A$, $X$ is a superkey.

Equivalently, a relation $R$ is in BCNF if $\forall X$ either $X^+ = X$ or $X^+ = C$ where $C$ is the set of all attributes in $R$. 


“Extracting” attributes can be done with decomposition (split the schema into smaller parts)

For this class, decomposition means the following:

\[ R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_k) \prec \begin{array}{l} R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \\ R_2(A_1, \ldots, A_n, C_1, \ldots, C_k) \end{array} \]
Decomposition

▪ “Extracting” attributes can be done with decomposition (split the schema into smaller parts)

▪ For this class, decomposition means the following:

\[
R(A_1, \ldots, A_n, B_1, \ldots, B_m, C_1, \ldots, C_k) \leftarrow R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \\
R_2(A_1, \ldots, A_n, C_1, \ldots, C_k)
\]

Some common attributes are present so we can rejoin data
BCNF Decomposition Algorithm

\[
\text{Normalize}(R) \\
\quad \mathcal{C} \leftarrow \text{the set of all attributes in } R \\
\text{find } X \text{ s.t. } X^+ \neq X \text{ and } X^+ \neq \mathcal{C} \\
\text{if } X \text{ is not found} \\
\text{then } "R \text{ is in BCNF}" \\
\text{else} \\
\quad \text{decompose } R \text{ into } R_1(X^+) \text{ and } R_2((\mathcal{C} - X^+) \cup X) \\
\text{Normalize}(R_1) \\
\text{Normalize}(R_2)
\]
**BCNF Decomposition Algorithm**

<table>
<thead>
<tr>
<th>Normalize($R$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$ ← the set of all attributes in $R$</td>
</tr>
<tr>
<td><strong>find</strong> $X$ s.t. $X^+ \neq X$ and $X^+ \neq C$</td>
</tr>
<tr>
<td><strong>if</strong> $X$ is not found</td>
</tr>
<tr>
<td><strong>then</strong> “$R$ is in BCNF”</td>
</tr>
<tr>
<td><strong>else</strong></td>
</tr>
<tr>
<td>decompose $R$ into $R_1(X^+)$ and $R_2((C - X^+) \cup X)$</td>
</tr>
<tr>
<td>Normalize($R_1$)</td>
</tr>
<tr>
<td>Normalize($R_2$)</td>
</tr>
</tbody>
</table>

Determine if $R$ is in BCNF already
### BCNF Decomposition Algorithm

**Normalize(R)**
- $\mathcal{C} \leftarrow$ the set of all attributes in $R$
- **find** $X$ s.t. $X^+ \neq X$ and $X^+ \neq \mathcal{C}$
- **if** $X$ is not found
  **then** “$R$ is in BCNF”
- **else**
  - decompose $R$ into $R_1(X^+)$ and $R_2((\mathcal{C} - X^+) \cup X)$
  - $\text{Normalize}(R_1)$
  - $\text{Normalize}(R_2)$

**Determine if R is in BCNF already**

**Decompose into a relation where X is a superkey**

**Decompose into a relation with X and attributes X cannot determine**
**BCNF Decomposition Example**

<table>
<thead>
<tr>
<th>Normalize($R$)</th>
<th>Restaurants($rid, name, rating, popularity, recommended$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C \leftarrow$ the set of all attributes in $R$</td>
<td>$rid \notin name, rating$</td>
</tr>
<tr>
<td><strong>find</strong> $X$ s.t. $X^+ \neq X$ and $X^+ \neq C$</td>
<td>rating $\notin popularity$</td>
</tr>
<tr>
<td>if $X$ is not found</td>
<td>popularity $\notin recommended$</td>
</tr>
<tr>
<td>then “$R$ is in BCNF”</td>
<td>else</td>
</tr>
<tr>
<td><strong>else</strong></td>
<td></td>
</tr>
<tr>
<td>decompose $R$ into $R_1(X^+)$ and $R_2((C - X^+) \cup X)$</td>
<td></td>
</tr>
<tr>
<td>Normalize($R_1$)</td>
<td></td>
</tr>
<tr>
<td>Normalize($R_2$)</td>
<td></td>
</tr>
</tbody>
</table>
### BCNF Decomposition Example

**Normalize**($R$)

$C \leftarrow$ the set of all attributes in $R$

*find $X$ s.t. $X^+ \neq X$ and $X^+ \neq C$*

*if $X$ is not found*

*then “$R$ is in BCNF”*

*else*

*decompose $R$ into $R_1(X^+)$ and $R_2((C - X^+) \cup X)$*

*Normalize($R_1$)*

*Normalize($R_2$)*

---

#### Restaurants

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco’s Tacos</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cery</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rating</th>
<th>popularity</th>
<th>recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>OK</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>High</td>
<td>yes</td>
</tr>
<tr>
<td>5</td>
<td>High</td>
<td>yes</td>
</tr>
</tbody>
</table>
### BCNF Decomposition Example

**Normalize** $(R)$

1. $C \leftarrow$ the set of all attributes in $R$
2. **find** $X$ s.t. $X^+ \neq X$ and $X^+ \neq C$
3. **if** $X$ is not found
4. **then** "$R$ is in BCNF"
5. **else**
   - **decompose** $R$ into $R_1(X^+)$ and $R_2((C - X^+) \cup X)$
   - **Normalize** $(R_1)$
   - **Normalize** $(R_2)$

#### Restaurants
- **rid** $\sqsubseteq$ name, rating, popularity, recommended
- **rating** $\sqsubseteq$ name, rating
- **popularity** $\sqsubseteq$ popularity
- **recommended** $\sqsubseteq$ recommended

<table>
<thead>
<tr>
<th>rid</th>
<th>name</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mee Sum Pastry</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Café on the Ave</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Guanaco’s Tacos</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Aladdin Gyro-Cery</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>rating</th>
<th>popularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>OK</td>
</tr>
<tr>
<td>4</td>
<td>High</td>
</tr>
<tr>
<td>5</td>
<td>High</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>popularity</th>
<th>rec</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>no</td>
</tr>
<tr>
<td>High</td>
<td>yes</td>
</tr>
</tbody>
</table>
Losslessness

**Definition**

**Lossless Decomposition** is a reversible decomposition, i.e. rejoining all decomposed relations will always result exactly with the original data.

This is the opposite of a **Lossy Decomposition**, an irreversible decomposition, where rejoining all decomposed relations may result something other than the original data, specifically with extra tuples.

This concept might be familiar if you have ever encountered lossless data compression (e.g. Huffman encoding or PNG) or lossy data compression (e.g. JPEG).
Is BCNF decomposition lossless?
Is BCNF decomposition lossless?

Yes!
Losslessness

Definition – Heath’s Theorem

Suppose we have the relation $R$ and three disjoint subsets of the attributes of $R$ we will write as $A_1, ..., A_n, B_1, ..., B_m$, and $C_1, ..., C_k$. Suppose we also have a FD that is $A_1, ..., A_n \rightarrow B_1, ..., B_m$.

Heath’s Theorem states that the decomposition of $R$ into $R_1(A_1, ..., A_n, B_1, ..., B_m)$ and $R_2(A_1, ..., A_n, C_1, ..., C_k)$ is lossless where $R_1$ and $R_2$ are the projections of $R$ on their respective attributes.

$$R(A_1, ..., A_n, B_1, ..., B_m, C_1, ..., C_k) \leftarrow R_1(A_1, ..., A_n, B_1, ..., B_m) \leftarrow R_2(A_1, ..., A_n, C_1, ..., C_k)$$

By reflection, the same decomposition of $R$ under the alternate FD $A_1, ..., A_n \rightarrow C_1, ..., C_k$ is also lossless.
On A Practical Note

- You may inherit a database that could be lossy. Before you use it, it may be worth your time to check if it is lossy.
- Full normalization is nice but can be inefficient
  - Denormalization  don’t normalize all the way
Takeaways

- We can characterize the relationships in our data using domain knowledge or pattern analysis.
- Functional dependencies give us ways of normalizing our data to avoid anomalies.