Part VII

Schema Normalization
In the database design process, we tried to produce good relational schemata (e.g., by merging relations, slide 76).

→ But what is “good,” after all?

Let us consider an example:

<table>
<thead>
<tr>
<th>StudID</th>
<th>Name</th>
<th>Address</th>
<th>Seminar Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>08-15</td>
<td>John Doe</td>
<td>74 Main St</td>
<td>Databases</td>
</tr>
<tr>
<td>08-15</td>
<td>John Doe</td>
<td>74 Main St</td>
<td>Systems Design</td>
</tr>
<tr>
<td>47-11</td>
<td>Mary Jane</td>
<td>8 Summer St</td>
<td>Data Mining</td>
</tr>
<tr>
<td>12-34</td>
<td>Dave Kent</td>
<td>19 Church St</td>
<td>Databases</td>
</tr>
<tr>
<td>12-34</td>
<td>Dave Kent</td>
<td>19 Church St</td>
<td>Statistics</td>
</tr>
<tr>
<td>12-34</td>
<td>Dave Kent</td>
<td>19 Church St</td>
<td>Multimedia</td>
</tr>
</tbody>
</table>
Obviously, this is **not** an example of a “good” relational schema.

→ **Redundant** information may lead to problems during **updates**:

**Update Anomaly**

If a student changes his address, several rows have to be updated.

**Insert Anomaly**

What if a student is not enrolled to any seminar?

→ Null value in column *SeminarTopic*?

   (→ may be problematic since *SeminarTopic* is part of a key)

→ To enroll a student to a course: overwrite null value (if student is not enrolled to any course) or create new tuple (otherwise)?

**Delete Anomaly**

Conversely, to un-register a student from a course, we might now either have to create a null value or delete an entire row.
Decomposed Schema

Those anomalies can be avoided by decomposing the table:

<table>
<thead>
<tr>
<th>StudID</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>08-15</td>
<td>John Doe</td>
<td>74 Main St</td>
</tr>
<tr>
<td>47-11</td>
<td>Mary Jane</td>
<td>8 Summer St</td>
</tr>
<tr>
<td>12-34</td>
<td>Dave Kent</td>
<td>19 Church St</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StudID</th>
<th>SeminarTopic</th>
</tr>
</thead>
<tbody>
<tr>
<td>08-15</td>
<td>Databases</td>
</tr>
<tr>
<td>08-15</td>
<td>Systems Design</td>
</tr>
<tr>
<td>47-11</td>
<td>Data Mining</td>
</tr>
<tr>
<td>12-34</td>
<td>Databases</td>
</tr>
<tr>
<td>12-34</td>
<td>Statistics</td>
</tr>
<tr>
<td>12-34</td>
<td>Multimedia</td>
</tr>
</tbody>
</table>

No redundancy exists in this representation any more.
The previous example might seem silly. But what about this one:

Real-world constraints:

- Each student may take only one exam with any particular professor.
- For any course, all exams are done by the same professor.
Ternary relationship set $\rightarrow$ ternary relation:

<table>
<thead>
<tr>
<th>Student</th>
<th>Professor</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td>Prof. Smart</td>
<td>Information Systems</td>
</tr>
<tr>
<td>Dave Kent</td>
<td>Prof. Smart</td>
<td>Information Systems</td>
</tr>
<tr>
<td>John Doe</td>
<td>Prof. Clever</td>
<td>Computer Architecture</td>
</tr>
<tr>
<td>Mary Jane</td>
<td>Prof. Bright</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>John Doe</td>
<td>Prof. Bright</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>Dave Kent</td>
<td>Prof. Bright</td>
<td>Software Engineering</td>
</tr>
</tbody>
</table>

- The association *Course* $\rightarrow$ *Professor* occurs multiple times.
- Decomposition without that redundancy?
Both examples contained instance of **functional dependencies**, e.g.,

\[ \text{Course} \rightarrow \text{Professor} \]

We say that

“\textit{Course (functionally) determines Professor.}”

meaning that when two tuples \( t_1 \) and \( t_2 \) agree on their \textit{Course} values, they **must** also contain the same \textit{Professor} value.
For this chapter, we’ll simplify our notation a bit.

- We use **single capital letters** $A, B, C, \ldots$ for **attribute names**.
- We use a short-hand notation for **sets of attributes**:

$$ABC \overset{\text{def}}{=} \{A, B, C\}.$$

A **functional dependency (FD)** $A_1 \ldots A_n \rightarrow B_1 \ldots B_m$ on a relation schema $\text{sch}(R)$ describes a **constraint** that, for every instance $R$:

$$t.A_1 = s.A_1 \land \cdots \land t.A_n = s.A_n \Rightarrow t.B_1 = s.B_1 \land \cdots \land t.B_m = s.B_m.$$

→ A functional dependency is a constraint over **one** relation. $A_1, \ldots, A_n, B_1, \ldots, B_m$ must all be in $\text{sch}(R)$. 
Functional Dependencies ↔ Keys

Functional dependencies are a generalization of key constraints:

\[ A_1, \ldots, A_n \text{ is a key of relation } R(A_1, \ldots, A_n, B_1, \ldots, B_m). \]

\[ \iff \]

\[ A_1 \ldots A_n \rightarrow B_1 \ldots B_m \text{ holds.} \]

Conversely, functional dependencies can be explained with keys.

\[ A_1 \ldots A_n \rightarrow B_1 \ldots B_m \text{ holds for } R. \]

\[ \iff \]

\[ A_1, \ldots, A_n \text{ is a key of } \pi_{A_1,\ldots,A_n,B_1,\ldots,B_m}(R). \]

→ Functional dependencies are “partial keys”.

→ A goal of this chapter is to turn FDs into real keys, because key constraints can easily be enforced by a DBMS.
**Functional Dependencies**

- **Functional dependencies in Students?**

<table>
<thead>
<tr>
<th>StudID</th>
<th>Name</th>
<th>Address</th>
<th>SeminarTopic</th>
</tr>
</thead>
<tbody>
<tr>
<td>08-15</td>
<td>John Doe</td>
<td>74 Main St</td>
<td>Databases</td>
</tr>
<tr>
<td>08-15</td>
<td>John Doe</td>
<td>74 Main St</td>
<td>Systems Design</td>
</tr>
<tr>
<td>47-11</td>
<td>Mary Jane</td>
<td>8 Summer St</td>
<td>Data Mining</td>
</tr>
<tr>
<td>12-34</td>
<td>Dave Kent</td>
<td>19 Church St</td>
<td>Databases</td>
</tr>
<tr>
<td>12-34</td>
<td>Dave Kent</td>
<td>19 Church St</td>
<td>Statistics</td>
</tr>
<tr>
<td>12-34</td>
<td>Dave Kent</td>
<td>19 Church St</td>
<td>Multimedia</td>
</tr>
</tbody>
</table>

- **Functional dependencies in the TakesExam example?**
A functional dependency with \( m \) attributes on the right-hand side

\[
A_1 \ldots A_n \rightarrow B_1 \ldots B_m
\]

is equivalent to the \( m \) functional dependencies

\[
A_1 \ldots A_n \rightarrow B_1 \\
\vdots \\
A_1 \ldots A_n \rightarrow B_m
\]

Often, functional dependencies imply one another.

→ We say that a set of FDs \( \mathcal{F} \) entails another FD \( f \) if the FDs in \( \mathcal{F} \) guarantee that \( f \) holds as well.

→ If a set of FDs \( \mathcal{F}_1 \) entails all FDs in the set \( \mathcal{F}_2 \), we say that \( \mathcal{F}_1 \) is a cover of \( \mathcal{F}_2 \); \( \mathcal{F}_1 \) covers (all FDs in) \( \mathcal{F}_2 \).
Reasoning over Functional Dependencies

Intuitively, we want to (re-)write relational schemas such that

- **redundancy is minimized** (and thus also update anomalies) and
- the system can still guarantee the **same integrity constraints**.

Functional dependencies allow us to **reason** over the latter.

**E.g.,**

- Given two schemas \( S_1 \) and \( S_2 \) and their associated sets of FDs \( F_1 \) and \( F_2 \), are \( F_1 \) and \( F_2 \) “equivalent”?

**Equivalence of two sets of functional dependencies:**

- We say that two sets of FDs \( F_1 \) and \( F_2 \) are **equivalent** \( (F_1 \equiv F_2) \) when \( F_1 \) entails all FDs in \( F_2 \) and vice versa.
Closure of a Set of Functional Dependencies

Given a set of functional dependencies \( \mathcal{F} \), the set of all functional dependencies entailed by \( \mathcal{F} \) is called the **closure of \( \mathcal{F} \)**, denoted \( \mathcal{F}^+ \):

\[
\mathcal{F}^+ := \{ \alpha \rightarrow \beta \mid \alpha \rightarrow \beta \text{ entailed by } \mathcal{F} \}.
\]

Closures can be used to express **equivalence** of sets of FDs:

\[
\mathcal{F}_1 \equiv \mathcal{F}_2 \iff \mathcal{F}_1^+ = \mathcal{F}_2^+.
\]

If there is a way to **compute** \( \mathcal{F}^+ \) for a given \( \mathcal{F} \), we can test:

- whether a given FD \( \alpha \rightarrow \beta \) is entailed by \( \mathcal{F} \) \( (\sim \alpha \rightarrow \beta \in \mathcal{F}^+) \)
- whether two sets of FDs, \( \mathcal{F}_1 \) and \( \mathcal{F}_2 \), are equivalent.

\(^{12}\)Let \( \alpha, \beta, \ldots \) denote sets of attributes.
Armstrong Axioms

\( \mathcal{F}^+ \) can be computed from \( \mathcal{F} \) by repeatedly applying the so-called **Armstrong axioms** to the FDs in \( \mathcal{F} \):

- **Reflexivity**: ("trivial functional dependencies")
  
  If \( \beta \subseteq \alpha \) then \( \alpha \rightarrow \beta \).

- **Augmentation**:
  
  If \( \alpha \rightarrow \beta \) then \( \alpha \gamma \rightarrow \beta \gamma \).

- **Transitivity**:
  
  If \( \alpha \rightarrow \beta \) and \( \beta \rightarrow \gamma \) then \( \alpha \rightarrow \gamma \).

It can be shown that the three Armstrong axioms are **sound** and **complete**: exactly the FDs in \( \mathcal{F}^+ \) can be generated from those in \( \mathcal{F} \).
Building the full $\mathcal{F}^+$ for an entailment test can be very expensive:
- The size of $\mathcal{F}^+$ can be exponential in the size of $\mathcal{F}$.
- Blindly applying the three Armstrong axioms to FDs in $\mathcal{F}$ can be very inefficient.

A better strategy is to focus on the particular FD of interest.

**Idea:**
- Given a set of attributes $\alpha$, compute the attribute closure $\alpha^+_F$:
  \[
  \alpha^+_F = \{ X \mid \alpha \rightarrow X \in \mathcal{F}^+ \} 
  \]
- Testing $\alpha \rightarrow \beta \in \mathcal{F}^+$ then means testing $\beta \in \alpha^+_F$. 

The attribute closure $\alpha^+_F$ can be computed as follows:

1. **Algorithm**: AttributeClosure
   
   **Input**: $\alpha$ (a set of attributes); $F$ (a set of FDs $\alpha_i \rightarrow \beta_i$)
   
   **Output**: $\alpha^+_F$ (all attributes functionally determined by $\alpha$ in $F^+$)

2. $x \leftarrow \alpha$

3. repeat

4. $x' \leftarrow x$

5. foreach $\alpha_i \rightarrow \beta_i \in F$ do

6. if $\alpha_i \subseteq x$ then

7. $x \leftarrow x \cup \beta_i$

8. until $x' = x$

9. return $x$
Example

Given

\[ \mathcal{F} = \{ AB \rightarrow C, D \rightarrow E, AE \rightarrow G, GD \rightarrow H, ID \rightarrow J \} \]

for a relation \( R \), \( \text{sch}(R) = ABCDEFGHIJ \).

- \( ABD \rightarrow GH \) entailed by \( \mathcal{F} \)?
- \( ABD \rightarrow HJ \) entailed by \( \mathcal{F} \)?
\( \mathcal{F}^+ \) is the **maximal cover** for \( \mathcal{F} \).

\[ \rightarrow \quad \mathcal{F}^+ \text{ (even } \mathcal{F} \text{) can be large and contain many redundant FDs. This makes } \mathcal{F}^+ \text{ a poor basis to study a relational schema.} \]

**Thus:** Construct a **minimal cover** \( \mathcal{F}^- \) such that

- \( \mathcal{F}^- \equiv \mathcal{F}, \text{ i.e., } (\mathcal{F}^-)^+ = \mathcal{F}^+ \).
- In \( \alpha \rightarrow \beta \in \mathcal{F}^- \), no attributes are redundant (in \( \alpha \) or \( \beta \)):
  
  \[
  \forall A \in \alpha : (\mathcal{F}^- - \{\alpha \rightarrow \beta\} \cup \{\alpha - A \rightarrow \beta\}) \not\equiv \mathcal{F}^- \\
  \forall B \in \beta : (\mathcal{F}^- - \{\alpha \rightarrow \beta\} \cup \{\alpha \rightarrow (\beta - B)\}) \not\equiv \mathcal{F}^- 
  \]

- Every left-hand side in \( \mathcal{F}^- \) occurs only once.
  
  This is trivial to achieve, since \( \{\alpha \rightarrow \beta, \alpha \rightarrow \gamma\} \equiv \{\alpha \rightarrow \beta \gamma\} \).
Constructing a Minimal Cover

To construct the minimal cover \( \mathcal{F}^- \):

1. **Minimize left-hand sides:**
   
   1. \( \mathcal{F}^- \leftarrow \mathcal{F} \);
   
   2. foreach \( \alpha \to \beta \in \mathcal{F}^- \) do
      
      3. foreach \( A \in \alpha \) do
         
         4. if \( \beta \subseteq (\alpha - A)_{\mathcal{F}^-}^+ \) then  
            
            A redundant in \( \alpha \)? Remove it.
         
         5. \( \mathcal{F}^- \leftarrow \mathcal{F}^- - \{ \alpha \to \beta \} \cup \{ (\alpha - A) \to \beta \} \);

2. **Minimize right-hand sides:**
   
   1. foreach \( \alpha \to \beta \in \mathcal{F}^- \) do
      
      2. foreach \( B \in \beta \) do
         
         3. \( \mathcal{F}' \leftarrow \mathcal{F}^- - \{ \alpha \to \beta \} \cup \{ \alpha \to (\beta - B) \} \);
         
         4. if \( B \in \alpha_{\mathcal{F}'}^+ \) then  
            
            B redundant in \( \beta \)? Remove it.
         
         5. \( \mathcal{F}^- \leftarrow \mathcal{F}' \);

© Jens Teubner · Information Systems · Summer 2013
Constructing a Minimal Cover

3. **Eliminate** all \( \alpha \rightarrow \emptyset \) (which might have resulted from 2).

4. **Combine** all \( \alpha \rightarrow \beta \), \( \alpha \rightarrow \gamma \) into \( \alpha \rightarrow \beta \gamma \).

---

**Minimal cover for the following FDs?**

\[
\begin{align*}
ABH & \rightarrow C \\
F & \rightarrow A \\
C & \rightarrow E \\
E & \rightarrow F \\
A & \rightarrow D \\
F & \rightarrow D \\
BGH & \rightarrow F \\
BH & \rightarrow E
\end{align*}
\]

1. **minimize left-hand sides:**

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

2. **minimize right-hand sides:**

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

3. **Eliminate and combine:**

\[
\begin{align*}
BH & \rightarrow C \\
F & \rightarrow A \\
C & \rightarrow E \\
E & \rightarrow F \\
A & \rightarrow D
\end{align*}
\]
Normal forms try to avoid the anomalies that we discussed earlier.

Codd originally proposed three normal forms (each stricter than the previous one):

- **First normal form (1NF)**
- **Second normal form (2NF)**
- **Third normal form (3NF)**

Later, Boyce and Codd added the

- **Boyce-Codd normal form (BCNF)**

Toward the end of this chapter, we will briefly talk also about the

- **Fourth normal form (4NF)**.
The first normal form states that **all attribute values must be atomic.**

That is, relations like

<table>
<thead>
<tr>
<th>StudID</th>
<th>Name</th>
<th>Address</th>
<th>SeminarTopic</th>
</tr>
</thead>
<tbody>
<tr>
<td>08-15</td>
<td>John Doe</td>
<td>74 Main St</td>
<td>{Databases, Systems Design}</td>
</tr>
<tr>
<td>47-11</td>
<td>Mary Jane</td>
<td>8 Summer St</td>
<td>{Data Mining}</td>
</tr>
<tr>
<td>12-34</td>
<td>Dave Kent</td>
<td>19 Church St</td>
<td>{Databases, Statistics, Multimedia}</td>
</tr>
</tbody>
</table>

are not allowed.

→ This characteristic is already implied by our definition of a relation. Likewise, nested tables (↗ slide 90) are not allowed in 1NF relations.
Given a schema $sch(R)$ and a set of FDs $\mathcal{F}$, $sch(R)$ is in Boyce-Codd Normal Form (BCNF) if, for every $\alpha \rightarrow A \in \mathcal{F}^+$ any of the following is true:

- $A \in \alpha$ (i.e., this is a trivial FD)
- $\alpha$ contains a key (or: “$\alpha$ is a superkey”)

**Example:** Consider a relation

$Courses(CourseNo, Title, InstrName, Phone)$

with the FDs

- $CourseNo \rightarrow Title, InstrName, Phone$
- $InstrName \rightarrow Phone$

This relation is **not** in BCNF, because in $InstrName \rightarrow Phone$, the left-hand side is not a key of the entire relation and the FD is not trivial.
Boyce-Codd Normal Form (BCNF)

A BCNF schema can have more than one key. E.g.,

- \( \text{sch}(R) = ABCD \),
- \( F = \{ AB \rightarrow CD, AC \rightarrow BD \} \).

This relation is in BCNF, because the left-hand side of each of the two FDs in \( F \) is a key.

BCNF prevents all of the anomalies that we saw earlier in this chapter.

→ By ensuring BCNF in our database designs, we can produce “good” relational schemas.

A beauty of BCNF is that its FDs can easily be checked by a database system.

→ Only need to mark left-hand sides as key in the relational schema.
Third Normal Form (3NF)

Given a schema \( \text{sch}(R) \) and a set of FDs \( \mathcal{F} \), \( \text{sch}(R) \) is in **third normal form (3NF)** if, for every \( \alpha \rightarrow A \in \mathcal{F}^+ \) any of the following is true:

- \( A \in \alpha \) (i.e., this is a trivial FD)
- \( \alpha \) contains a key (or: “\( \alpha \) is a superkey”)
- \( A \in \kappa \) for some key \( \kappa \subseteq \text{sch}(R) \).

Observe how the third case relaxes BCNF.

\[ \rightarrow \text{The } \text{TakesExam}(\text{Student}, \text{Professor}, \text{Course}) \text{ relation on slide 215 is in 3NF:} \]

\[ \text{Student, Professor } \rightarrow \text{ Course} \]

\[ \text{Course } \rightarrow \text{ Professor} \]

\[ \rightarrow \text{But TakesExam is not in BCNF.} \]
Third Normal Form (3NF)

Obviously, the additional condition allows some redundancy.

→ What is the merit of that condition then?

**Answer:**

1. There is none. 3NF was discovered “accidentally” in the search for BCNF.

2. As we shall see, relational schemas can always be converted into 3NF form losslessly, while in some cases this is not true for BCNF.

**Note:**

- We will not discuss 2NF in this course. It is of no practical use today and only exists for historical reasons.
As illustrated by example on slide 214, redundancy can be eliminated by **decomposing** a schema into a collection of schemas:

$$(\text{sch}(R), \mathcal{F}) \sim (\text{sch}(R_1), \mathcal{F}_1), \dots, (\text{sch}(R_n), \mathcal{F}_n).$$

The corresponding relations can be obtained by **projecting** on columns of the original relation:

$$R_i = \pi_{\text{sch}(R_i)} R.$$

While decomposing a schema, we do **not** want to **lose information**.
A decomposition is lossless if the original relation can be reconstructed from the decomposed tables:

$$R = R_1 \bowtie \cdots \bowtie R_n.$$

For binary decompositions, losslessness is guaranteed if any of the following is true:

- $$(\text{sch}(R_1) \cap \text{sch}(R_2)) \rightarrow \text{sch}(R_1) \in \mathcal{F}^+$$
- $$(\text{sch}(R_1) \cap \text{sch}(R_2)) \rightarrow \text{sch}(R_2) \in \mathcal{F}^+$$

“The decomposition is guaranteed to be lossless if the intersection of attributes of the new tables is a key of at least one of the two relations.”
Dependency-Preserving Decompositions

For a lossless decomposition of $R$, it would always be possible to **re-construct** $R$ and check the original set of FDs $\mathcal{F}$ over the re-constructed table.

→ But re-construction is **expensive**.

→ We’d rather like to guarantee that FDs $\mathcal{F}_1, \ldots, \mathcal{F}_n$ over decomposed tables $R_1, \ldots, R_n$ **entail all** FDs in $\mathcal{F}$.

A decomposition is **dependency-preserving** if

$$\mathcal{F}_1 \cup \cdots \cup \mathcal{F}_n \equiv \mathcal{F}.$$
Consider a zip code directory

\[ \text{ZipCodes}(\text{Street}, \text{City}, \text{State}, \text{ZipCode}) \],

where

\[ \text{ZipCode} \rightarrow \text{City}, \text{State} \]
\[ \text{Street}, \text{City}, \text{State} \rightarrow \text{ZipCode} \].

A **lossless decomposition** would be

\[ \text{Streets}(\text{ZipCode}, \text{Street}) \]
\[ \text{Cities}(\text{ZipCode}, \text{City}, \text{State}) \].

However, the FD \( \text{Street}, \text{City}, \text{State} \rightarrow \text{ZipCode} \) cannot be assigned to either of the two relations. This decomposition is **not** dependency-preserving.
Algorithm for BCNF Decomposition

BCNF can be obtained by repeatedly decomposing a table along an FD that violates BCNF:

1. **Algorithm**: BCNFDecomposition
   
   **Input**: \((\text{sch}(R), \mathcal{F})\)
   
   **Output**: Schema \(\{(\text{sch}(R_1), \mathcal{F}_1), \ldots, (\text{sch}(R_n), \mathcal{F}_n)\}\) in BCNF

2. \(\text{Decomposed} \leftarrow \{(\text{sch}(R), \mathcal{F})\}\);

3. while \(\exists (\text{sch}(S), \mathcal{F}_S) \in \text{Decomposed}\) that is not in BCNF do
   
   4. Let \(\alpha \rightarrow \beta\) be an FD in \(\mathcal{F}_S\) that violates BCNF;
   
   5. Decompose \(S\) into \(S_1(\alpha\beta)\) and \(S_2((S - \beta) \cup \alpha)\);

6. **return** \(\text{Decomposed}\);
Example

Consider

\[ R(ABCDEFGH) \]

with

\[ ABH \rightarrow C \]
\[ A \rightarrow DE \]
\[ BGH \rightarrow F \]
\[ F \rightarrow ADH \]
\[ BH \rightarrow GE \]
Properties of BCNF Decomposition

Algorithm $BCNFDecomposition$ always yields a **lossless decomposition**.

- Attribute set $\alpha$ is contained in $S_1$ and $S_2$ (line 5).
- $\alpha \rightarrow \beta \in F_S$ (line 4), so $\alpha \rightarrow \text{sch}(S_1)$.

We already saw that BCNF decomposition is **not always dependency-preserving**.

BCNF decomposition is **not deterministic**. Different choices of FDs in line 4 might lead to different decompositions.

→ Those different decompositions might even preserve more or less dependencies!
The **3NF synthesis algorithm** produces a 3NF schema that is always **lossless** and **dependency-preserving**:

1. Compute the **minimal cover** $\mathcal{F}^-$ of the given set of FDs $\mathcal{F}$.
2. For each $\alpha \rightarrow \beta \in \mathcal{F}^-$ create a table $R_\alpha(\alpha\beta)$.
3. If **none** of the constructed tables from step 2 contains a key of the original relation $R$, add one relation $R_\kappa(\kappa)$, where $\kappa$ is a (candidate) key in $R$.
4. Steps 2 and 3 might lead to relations $R_1$ and $R_2$, where $R_1$ is contained in $R_2$. If this is the case, discard $R_1$. 
Given a table \( R(ABCDEFGH) \) with the FDs

\[
\begin{align*}
ABH & \rightarrow C \\
A & \rightarrow DE \\
BGH & \rightarrow F \\
F & \rightarrow ADH \\
BH & \rightarrow GE
\end{align*}
\]

determine a corresponding 3NF schema.
Example (cont.)

1. Minimal cover: Eliminate \( \alpha \rightarrow \emptyset \) (nothing to do).

2. Combine: \( BH \rightarrow CFGE \quad A \rightarrow DE F \quad AH \rightarrow \).

3. Create a table for each FD:
   - \( R_1 \): \( BH \cup CFGE \)
   - \( R_2 \): \( A \cup DE \)
   - \( R_3 \): \( F \cup AH \)

4. Any \( R_i \) contains a key of \( R \)? \( \rightarrow \) Yes. (\( BH \rightarrow F \); \( F \rightarrow A \); all attributes of \( R \) are in \( R_1 \), \( R_2 \), \( R_3 \))

4. Redundant relations? \( \rightarrow \) No, we are done.
Normal forms are increasingly restrictive.

→ In particular, every BCNF relation is also 3NF.

- Our decomposition algorithms produce lossless decompositions.
  → It is always possible to losslessly transform a relation into 1NF, 2NF, 3NF, BCNF.

- BCNF decomposition might not be dependency-preserving. Preservation of dependencies can only be guaranteed up to 3NF.
BCNF vs. 3NF

BCNF decomposition is **non-deterministic**.

→ Some decompositions might be **dependency-preserving**, some might not.

**Decomposition strategy:**

1. Establish 3NF schema (through synthesis; dependency preservation guaranteed).

2. Decompose resulting schema to obtain BCNF.

→ This strategy typically leads to “good” (dependency-preserving if possible) BCNF decompositions.
Fourth Normal Form (4NF)

Not all redundancies can be explained through functional dependencies.

<table>
<thead>
<tr>
<th>ISBN</th>
<th>Author</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>3486598341</td>
<td>Kemper</td>
<td>Databases</td>
</tr>
<tr>
<td>3486598341</td>
<td>Kemper</td>
<td>Computer Science</td>
</tr>
<tr>
<td>3486598341</td>
<td>Eickler</td>
<td>Databases</td>
</tr>
<tr>
<td>3486598341</td>
<td>Eickler</td>
<td>Computer Science</td>
</tr>
<tr>
<td>0321268458</td>
<td>Kifer</td>
<td>Databases</td>
</tr>
<tr>
<td>0321268458</td>
<td>Bernstein</td>
<td>Databases</td>
</tr>
<tr>
<td>0321268458</td>
<td>Lewis</td>
<td>Databases</td>
</tr>
</tbody>
</table>

→ There is no clear association between authors and keywords, and no functional dependencies exist for this table.

→ This relation is in BCNF!
Observe that the relation satisfies the following property:

\[ Books = \pi_{ISBN,Author}(Books) \Join \pi_{ISBN,Keyword}(Books) . \]

A join dependency, written as

\[ sch(R) = \alpha \Join \beta , \]

is a constraint specifying that, for any legal instance of \( R \),

\[ R = \pi_\alpha(R) \Join \pi_\beta(R) . \]
Fourth Normal Form (4NF)

Given a schema $\text{sch}(R)$ and a set of join and dependencies $\mathcal{J}$ and $\mathcal{F}$, $\text{sch}(R)$ is in **fourth normal form (4NF)** if, for every join dependency $\text{sch}(R) = \alpha \Join \beta$ entailed by $\mathcal{F}$ and $\mathcal{J}$, either of the following is true:

- The join dependency is trivial, *i.e.*, $\alpha \subseteq \beta$.
- $\alpha \cap \beta$ contains a key of $R$ (or: “$\alpha$ is a superkey of $R$”).

(Relation *Books* is not in 4NF, because *ISBN* is not a key.)

**4NF relations are also BCNF:**

- Suppose $\text{sch}(R)$ with $\alpha \rightarrow \beta$ is in 4NF (and $\alpha \cap \beta = \emptyset$).
- Then, $R = \pi_{\alpha\beta}(R) \Join \pi_{\text{sch}(R) - \beta}(R)$ (↑ slide 238).
- Thus, $\alpha\beta \cap (\text{sch}(R) - \beta) = \alpha$ is a superkey of $R$ (4NF property).
- BCNF requirement satisfied. ✓
Join dependencies are also called **multi-valued dependencies**. The MVD

$$\alpha \rightarrow \beta$$

is another notation for the join dependency

$$sch(R) = \alpha\beta \Join \alpha(sch(R) - \beta)$$.

Intuitively,

"**The set of values in columns β associated with every α is independent of all other columns.**"

Note:

- **MVDs always come in pairs.** If $\alpha \rightarrow \beta$ holds, then $\alpha \rightarrow (sch(R) - \beta)$ automatically holds as well.
Decomposing a schema

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

into

\[ R_1(A_1, \ldots, A_n, B_1, \ldots, B_m) \quad \text{and} \quad R_2(A_1, \ldots, A_n, C_1, \ldots, C_k) \]

is **lossless** if and only if (↗ slide 238)

\[ A_1, \ldots, A_n \rightarrow B_1, \ldots B_m \quad \text{(or} \quad A_1, \ldots, A_n \rightarrow C_1, \ldots B_k) \]

**Thus:** (intuition for obtaining 4NF)

- Whenever there is a lossless (non-trivial) decomposition, decompose.