Information Systems (Informationssysteme)

Jens Teubner, TU Dortmund
jens.teubner@cs.tu-dortmund.de

Summer 2018

Part VII

Schema Normalization

Motivation

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:

Students			
StudID	Name	Address	SeminarTopic
08-15	John Doe	74 Main St	Databases
08-15	John Doe	74 Main St	Systems Design
47-11	Mary Jane	8 Summer St	Data Mining
12-34	Dave Kent	19 Church St	Databases
12-34	Dave Kent	19 Church St	Statistics
12-34	Dave Kent	19 Church St	Multimedia

Update Anomalies

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:

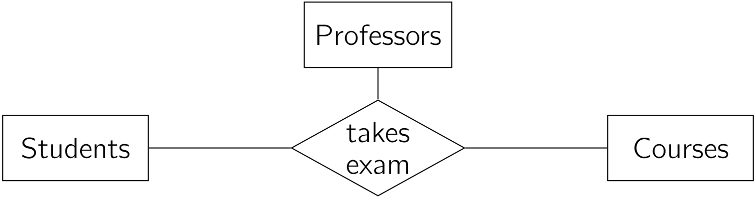
Students		
StudID	Name	Address
08-15	John Doe	74 Main St
47-11	Mary Jane	8 Summer St
12-34	Dave Kent	19 Church St

Students	
StudID	SeminarTopic
08-15	Databases
08-15	Systems Design
47-11	Data Mining
12-34	Databases
12-34	Statistics
12-34	Multimedia

No redundancy exists in this representation any more.

Anomalies: Another Example

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.

Anomalies: Another Example

Ternary relationship set $\rightarrow$ ternary relation:

TakesExam		
Student	Professor	Course
John Doe	Prof. Smart	Information Systems
Dave Kent	Prof. Smart	Information Systems
John Doe	Prof. Clever	Computer Architecture
Mary Jane	Prof. Bright	Software Engineering
John Doe	Prof. Bright	Software Engineering
Dave Kent	Prof. Bright	Software Engineering

- The association *Course* $\rightarrow$ *Professor* occurs multiple times.
- Decomposition without that redundancy?

Functional Dependencies

Both examples contained instance of **functional dependencies**, *e.g.*,

$$Course \rightarrow Professor \quad .$$

We say that

> *"Course (functionally) determines Professor."*

meaning that when two tuples t_1 and t_2 agree on their *Course* values, they **must** also contain the same *Professor* value.

Notation

For this chapter, we'll simplify our **notation** a bit.

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

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

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

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

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

Functional Dependencies $\leftrightarrow$ Keys

Functional dependencies are a generalization of **key constraints**:

$A_1, \ldots, A_n$ is a set of identifying attributes[11]
in relation $R(A_1, \ldots, A_n, B_1, \ldots, B_m)$.

$$\Leftrightarrow$$

$A_1 \ldots A_n \rightarrow B_1 \ldots B_m$ holds.

Conversely, functional dependencies can be explained with keys.

$A_1 \ldots A_n \rightarrow B_1 \ldots B_m$ holds for R.

$$\Leftrightarrow$$

$A_1, \ldots, A_n$ is a set of identifying attributes in $\pi_{A_1,\ldots,A_n,B_1,\ldots B_m}(R)$.

$\rightarrow$ Functional dependencies are **"partial keys"**.

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

[11] If the set is also minimal, $A_1, \ldots, A_n$ is a key (↗ slide 53).

Functional Dependencies

✎ **Functional dependencies in *Students*?**

Students			
StudID	Name	Address	SeminarTopic
08-15	John Doe	74 Main St	Databases
08-15	John Doe	74 Main St	Systems Design
47-11	Mary Jane	8 Summer St	Data Mining
12-34	Dave Kent	19 Church St	Databases
12-34	Dave Kent	19 Church St	Statistics
12-34	Dave Kent	19 Church St	Multimedia

✎ **Functional dependencies in the *TakesExam* example?**

Functional Dependencies, Entailment

A functional dependency with m attributes on the right-hand side

$$A_1 \dots A_n \rightarrow B_1 \dots B_m$$

is **equivalent** to the m functional dependencies

$$\begin{array}{ccc} A_1 \dots A_n & \rightarrow & B_1 \\ \vdots & & \vdots \\ A_1 \dots A_n & \rightarrow & B_m \end{array}$$

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 $\mathcal{F}_1$ and $\mathcal{F}_2$, are $\mathcal{F}_1$ and $\mathcal{F}_2$ "equivalent"?

Equivalence of two sets of functional dependencies:

- We say that two sets of FDs $\mathcal{F}_1$ and $\mathcal{F}_2$ are **equivalent** ($\mathcal{F}_1 \equiv \mathcal{F}_2$) when $\mathcal{F}_1$ entails all FDs in $\mathcal{F}_2$ 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 $\mathcal{F}^+$:[12]

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

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

$$\mathcal{F}_1 \equiv \mathcal{F}_2 \ \Leftrightarrow \ \mathcal{F}_1^+ = \mathcal{F}_2^+ \ .$$

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 $\mathcal{F}$ ($\leadsto \alpha \rightarrow \beta \overset{?}{\in} \mathcal{F}^+$)
- whether two sets of FDs, $\mathcal{F}_1$ and $\mathcal{F}_2$, are equivalent.

[12] Let α, β, ... 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 Amstrong axioms are **sound** and **complete**: exactly the FDs in $\mathcal{F}^+$ can be generated from those in $\mathcal{F}$.

Testing Entailment / Attribute Closure

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 α, compute the **attribute closure** $\alpha^+_{\mathcal{F}}$:

$$\alpha^+_{\mathcal{F}} = \left\{ X \mid \alpha \rightarrow X \in \mathcal{F}^+ \right\}$$

- Testing $\alpha \rightarrow \beta \stackrel{?}{\in} \mathcal{F}^+$ then means testing $\beta \stackrel{?}{\subseteq} \alpha^+_{\mathcal{F}}$.

Attribute Closure

The attribute closure $\alpha_{\mathcal{F}}^{+}$ can be computed as follows:

Algorithm: AttributeClosure

Input : α (a set of attributes); $\mathcal{F}$ (a set of FDs $\alpha_i \rightarrow \beta_i$)

Output: $\alpha_{\mathcal{F}}^{+}$ (all attributes functionally determined by α in $\mathcal{F}^{+}$)

```
x ← α;
repeat
    x' ← x;
    foreach α_i → β_i ∈ F do
        if α_i ⊆ x then
            x ← x ∪ β_i;
until x' = x;
return x;
```

Example

Given

$$\mathcal{F} = \{AB \to C, D \to E, AE \to G, GD \to H, ID \to J\}$$

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

- ✎ $ABD \to GH$ **entailed by $\mathcal{F}$?**
- ✎ $ABD \to HJ$ **entailed by $\mathcal{F}$?**

Minimal Cover

$\mathcal{F}^+$ is the **maximal cover** for $\mathcal{F}$.

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

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

1. $\mathcal{F}^- \equiv \mathcal{F}$, *i.e.*, $(\mathcal{F}^-)^+ = \mathcal{F}^+$.
2. All functional dependencies in $\mathcal{F}^-$ have the form $\alpha \rightarrow X$ (*i.e.*, the right side is a single attribute).
3. In $\alpha \rightarrow X \in \mathcal{F}^-$, no attributes in α are redundant:

$$\forall A \in \alpha : \left(\mathcal{F}^- - \{\alpha \rightarrow X\} \cup \{(\alpha - A) \rightarrow X\}\right) \not\equiv \mathcal{F}^- \quad .$$

4. No rule $\alpha \rightarrow X$ is redundant in $\mathcal{F}^-$:

$$\forall \alpha \rightarrow X \in \mathcal{F}^- : \left(\mathcal{F}^- - \{\alpha \rightarrow X\}\right) \not\equiv \mathcal{F}^- \quad .$$

Constructing a Minimal Cover

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

1. $\mathcal{F}^- \leftarrow \mathcal{F}$ where all functional dependencies are converted to have only **one attribute on the right side**.
2. **Remove redundant attributes** from the left-hand sides of functional dependencies in $\mathcal{F}^-$:

```
foreach α → X ∈ F⁻ do
    foreach A ∈ α do
        if X ∈ (α − A)⁺_{F⁻} then   ▷ A redundant in α? Remove it.
            F⁻ ← F⁻ − {α → X} ∪ {(α − A) → X};
```

3. **Remove redundant functional dependencies** from $\mathcal{F}^-$:

```
foreach α → X ∈ F⁻ do
    if (F⁻ − {α → X}) ≡ F⁻ then
        F⁻ ← F⁻ − {α → X} ;
```

Constructing a Minimal Cover

✎ **Minimal cover for the following FDs?**

$$ABH \rightarrow C \qquad F \rightarrow AD \qquad C \rightarrow E \qquad E \rightarrow F$$
$$A \rightarrow D \qquad BGH \rightarrow F \qquad BH \rightarrow E$$

Normal Forms

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)**.

First Normal Form

The first normal form states that **all attribute values must be atomic**.

That is, relations like

Students			
StudID	Name	Address	SeminarTopic
08-15	John Doe	74 Main St	{Databases, Systems Design}
47-11	Mary Jane	8 Summer St	{Data Mining}
12-34	Dave Kent	19 Church St	{Databases, Statistics, Multimedia}

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.

Boyce-Codd Normal Form (BCNF)

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)
- α contains a key (or: "α is a superkey")

Example: Consider a relation

$$Courses(\underline{CourseNo}, Title, InstrName, Phone)$$

with the FDs

$$\begin{aligned} CourseNo &\rightarrow Title, InstrName, Phone \\ InstrName &\rightarrow Phone \quad . \end{aligned}$$

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.*,

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

This relation **is** in BCNF, because the left-hand side of each of the two FDs in $\mathcal{F}$ 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 sch(R) and a set of FDs $\mathcal{F}$, sch(R) is in **third normal form (3NF)** if, for every $\alpha \to A \in \mathcal{F}^+$ any of the following is true:

- $A \in \alpha$ (*i.e.*, this is a trivial FD)
- α contains a key (or: "α is a superkey")
- $A \in \kappa$ for some key $\kappa \subseteq$ sch(R).

Observe how the third case **relaxes** BCNF.

→ The $\mathit{TakesExam}(\mathit{Student}, \mathit{Professor}, \mathit{Course})$ relation on slide 215 **is** in 3NF:

$$\begin{aligned} \mathit{Student}, \mathit{Professor} &\to \mathit{Course} \\ \mathit{Course} &\to \mathit{Professor} \quad . \end{aligned}$$

→ 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.

Schema Decomposition

As illustrated by example on slide 214, redundancy can be eliminated by **decomposing** a schema into a collection of schemas:

$$\big(\mathrm{sch}(R), \mathcal{F}\big) \;\rightsquigarrow\; \big(\mathrm{sch}(R_1), \mathcal{F}_1\big), \dots, \big(\mathrm{sch}(R_n), \mathcal{F}_n\big) \enspace .$$

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

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

While decomposing a schema, we do **not** want to **lose information**.

Lossless and Lossy Decompositions

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:

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

"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 $\mathcal{F}_1, \dots, \mathcal{F}_n$ over decomposed tables $R_1, \dots, R_n$ **entail all** FDs in $\mathcal{F}$.

A decomposition is **dependency-preserving** if

$$\mathcal{F}_1 \cup \dots \cup \mathcal{F}_n \equiv \mathcal{F} \ .$$

Example

Consider a zip code directory

$$ZipCodes(Street, City, State, ZipCode) \quad ,$$

where

$$\begin{aligned} ZipCode &\rightarrow City, State \\ Street, City, State &\rightarrow ZipCode \quad . \end{aligned}$$

A **lossless decomposition** would be

$$\begin{aligned} &Streets(ZipCode, Street) \\ &Cities(ZipCode, City, State) \quad . \end{aligned}$$

However, the FD $Street, City, State \rightarrow ZipCode$ cannot be assigned to either of the two relations. This decomposition is **not dependency-preserving**.

Decomposing A Schema

When decomposing a schema, we obtain schemas by **projecting** on columns of the original relation (↗ slide 237):

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

How do we obtain the corresponding functional dependencies?

$$\mathcal{F}_i := \pi_{\mathrm{sch}(R_i)} \mathcal{F} := \{\alpha \rightarrow \beta \mid \alpha \rightarrow \beta \in \mathcal{F}^+ \text{ and } \alpha\beta \subseteq \mathrm{sch}(R_i)\}$$

→ We call this the **projection** of the set $\mathcal{F}$ of functional dependencies on the set of attributes $\mathrm{sch}(R_i)$.

Algorithm for BCNF Decomposition

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

```
1 Algorithm: BCNFDecomposition
  Input   : (sch(R), F)
  Output: Schema {(sch(R_1), F_1), ..., (sch(R_n), F_n)} in BCNF
2 Decomposed ← {(sch(R), F)};
3 while ∃ (sch(S), F_S) ∈ Decomposed that is not in BCNF do
4     Let α → β be an FD in F_S that violates BCNF;
5     Decompose S into S_1(αβ) and S_2((S − β) ∪ α);
6 return Decomposed;
```

In line 5, use the projection mechanism on slide 241 to obtain the $\mathcal{F}_{S_i}$.

Example

Consider

$$R(ABCDEFGH)$$

with

$$\begin{aligned} ABH &\rightarrow C \\ A &\rightarrow DE \\ BGH &\rightarrow F \\ F &\rightarrow ADH \\ BH &\rightarrow GE \end{aligned}$$

Properties of BCNF Decomposition

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

- Attribute set α is contained in S_1 and S_2 (line 5).
- $\alpha \rightarrow \beta \in \mathcal{F}_S$ (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!

3NF Decomposition Through Schema Synthesis

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. **Merge** rules in $\mathcal{F}^-$ that have the same left-hand side ($\rightarrow \mathcal{G}$).
3. For each $\alpha \rightarrow \beta \in \mathcal{G}$ create a table $R_\alpha(\alpha\beta)$ and associate $\mathcal{F}_\alpha = \{\alpha \rightarrow \beta\}$ with it.
4. If **none** of the constructed tables from step 3 contains a key of the original relation R, add one relation $R_\kappa(\kappa)$, where κ is a (candidate) key in R. No functional dependencies are associated with R_κ.

Example

✎ **Given a table** $R(ABCDEFGH)$ **with the FDs**

$$\begin{array}{lllllllll} ABH & \rightarrow & C & A & \rightarrow & DE & BGH & \rightarrow & F \\ F & \rightarrow & ADH & BH & \rightarrow & GE & & & \end{array}$$

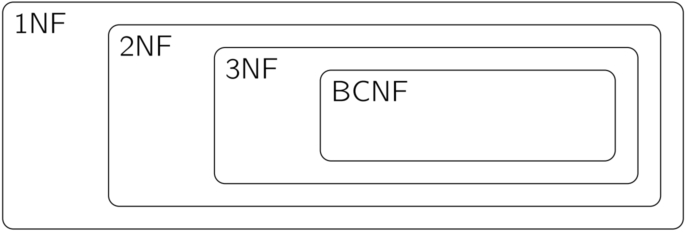
determine a corresponding 3NF schema.

Example (cont.)

Normal Forms

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.

Books		
ISBN	Author	Keyword
3486598341	Kemper	Databases
3486598341	Kemper	Computer Science
3486598341	Eickler	Databases
3486598341	Eickler	Computer Science
0321268458	Kifer	Databases
0321268458	Bernstein	Databases
0321268458	Lewis	Databases

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

→ This relation is in BCNF!

Join Dependencies

Observe that the relation satisfies the following property:

$$Books = \pi_{ISBN,Author}(Books) \bowtie \pi_{ISBN,Keyword}(Books) \enspace .$$

A **join dependency**, written as

$$sch(R) = \alpha \bowtie \beta \enspace ,$$

is a **constraint** specifying that, for any legal instance of R,

$$R = \pi_\alpha(R) \bowtie \pi_\beta(R) \enspace .$$

Fourth Normal Form (4NF)

Given a schema sch(R) and a set of join and dependencies $\mathcal{J}$ and $\mathcal{F}$, sch(R) is in **fourth normal form (4NF)** if, for every join dependency sch$(R) = \alpha \bowtie \beta$ 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: "α is a superkey of R").

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

4NF relations are also BCNF:

- Suppose sch(R) with $\alpha \rightarrow \beta$ is in 4NF (and $\alpha \cap \beta = \emptyset$).
- Then, $R = \pi_{\alpha\beta}(R) \bowtie \pi_{\text{sch}(R)-\beta}(R)$ (↗ slide 238).
- Thus, $\alpha\beta \cap (\text{sch}(R) - \beta) = \alpha$ is a superkey of R (4NF property).
- BCNF requirement satisfied. ✔

Multi-Valued Dependencies (MVDs)

Join dependencies are also called **multi-valued dependencies**.

The MVD

$$\alpha \twoheadrightarrow \beta$$

is another notation for the join dependency

$$sch(R) = \alpha\beta \bowtie \alpha(\text{sch}(R) - \beta) \ .$$

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 $\alpha \twoheadrightarrow \beta$ holds, then $\alpha \twoheadrightarrow (sch(R) - \beta)$ automatically holds as well.

Obtaining 4NF Schemas

Decomposing a schema

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

into

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

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

$$A_1, \ldots, A_n \twoheadrightarrow B_1, \ldots B_m \quad (\text{or } A_1, \ldots, A_n \twoheadrightarrow C_1, \ldots B_k) \ .$$

Thus: (intuition for obtaining 4NF)

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