

# Advanced Relational Design

## Closure of a Set of FDs

The closure  $F^+$  of  $F$  is the set of all FDs logically implied by  $F$ . We can use a set of inference rules known as **Armstrong's Axioms** to derive new FDs.

- ▶ **Reflexivity**. If  $Y \subseteq X$ , then  $X \rightarrow Y$
- ▶ **Augmentation**. If  $X \rightarrow Y$  holds, then  $XZ \rightarrow YZ$
- ▶ **Transitivity**. If  $X \rightarrow Y$  holds and  $Y \rightarrow Z$  holds, then  $X \rightarrow Z$  holds

Note that  $XY$  is shorthand for  $X \cup Y$ .

Armstrong's axioms are **sound** because they do not produce new FDs that don't hold, and **complete** because applying them repeatedly finds  $F^+$ , i.e., all FDs that are logically implied by  $F$ .

Note that  $F^+$  includes **all** FDs, including trivial FDs.

# Algorithm for Finding $F^+$

Apply Armstrong's Axioms repeatedly.

- ▶ Let  $F^+ = F$
- ▶ repeat:
  - ▶ for each FD  $f$  in  $F^+$ :
    - ▶ add results of applying reflexivity and augmentation rules on  $f$  to  $F^+$
  - ▶ for each pair of FDs  $X \rightarrow Y$  and  $Y \rightarrow Z$  in  $F^+$ :
    - ▶ add  $X \rightarrow Z$  to  $F^+$
- ▶ until  $F^+$  does not change any further

This algorithm is instructive, but tedious and expensive and mainly for conceptual understanding. Important concept: two sets of FDs are equivalent if they imply the same closure set of FDs.

# Attribute Closure

The set of attributes functionally determined by  $X$  under  $F$  is the **attribute closure** of  $X$  under  $F$ , denoted  $X^+$ .

## Algorithm 15.1 Determining $X^+$ , the closure of $X$ under $F$

**Input:** A set  $F$  of FDs on relation schema  $R$ , and a set of attributes  $X \subseteq R$

- ▶  $X^+ := X$
- ▶ **repeat:**
  - ▶  $oldX^+ := X^+$
  - ▶ **for each** functional dependency  $Y \rightarrow Z$ :
    - ▶ if  $Y \subseteq X^+$  then  $X^+ := X^+ \cup Z$
- ▶ **until**  $X^+ = oldX^+$

# Uses of Attribute Closure

- ▶ Test whether  $X$  is a superkey of  $R$ .
  - ▶ If  $X^+$  contains all attributes in  $R$ , then  $X$  is a superkey of  $R$ .
- ▶ Checking whether an FD  $X \rightarrow Y$  holds on  $R$  under  $F$ .
  - ▶ Compute  $X^+$ . If  $Y \subseteq X^+$  then  $X \rightarrow Y$  holds.
  - ▶ This is equivalent to saying that  $X \rightarrow Y$  is in  $F^+$ 
    - ▶ Note: you never actually need to compute  $F^+$ , you just need to be able to determine if some FD is in  $F^+$ .
- ▶ An alternate way to compute  $F^+$ .
  - ▶ For each  $Z \subseteq R$ , compute  $Z^+$ 
    - ▶ For each  $S \subseteq Z^+$  add FD  $Z \rightarrow S$  to  $F^+$

## Superkey Test

Given  $R(A, B, C, G, H, I)$  and  $F = \{$

$$A \rightarrow B,$$

$$A \rightarrow C,$$

$$CG \rightarrow H,$$

$$CG \rightarrow I,$$

$$B \rightarrow H$$

}

compute  $\{AG\}^+$ .

The first time we execute the outer loop:

- ▶  $A \rightarrow B$  adds  $B$  to  $\{AG\}^+$ , making  $\{AG\}^+ = \{ABG\}$ .
- ▶  $A \rightarrow C$  adds  $C$ , making  $\{AG\}^+ = \{ABCG\}$ .
- ▶  $CG \rightarrow H$  adds  $H$ , making  $\{AG\}^+ = \{ABCGH\}$ .
- ▶  $CG \rightarrow I$  adds  $I$ , making  $\{AG\}^+ = \{ABCGHI\}$ .

Since  $\{AG\}^+$  now includes all attributes in  $R$ , the second iteration of the outer loop adds no new attributes, so the algorithm terminates. Since  $\{AG\}^+$  includes all the attributes of  $R$ ,  $\{AG\}$  is a superkey of  $R$ .

# Finding A Candidate Key

**Algorithm 15.2(a):** Finding a key  $K$  for  $R$  given a set  $F$  of functional dependencies on  $R$

**Input:** A relation  $R$  and FDs  $F$  on  $R$

1. set  $K := R$
2. **for each** attribute  $a$  in  $K$ :
  - ▶ compute  $(K - \{a\})^+$  with respect to  $F$
  - ▶ **if**  $R \subseteq (K - \{a\})^+$ , **then** set  $K := K - \{a\}$

This algorithm finds a single candidate key depending on the order in which attributes are removed.

## BCNF Example 2

Given TEACH(Student, Course, Instructor) and

- ▶ FD1: {Student, Course} → Instructor
- ▶ FD2: Instructor → Course.

FD2 violates BCNF. There are three possible BCNF decompositions:

1. R1( Student , Instructor ) and R2( Student , Course )
2. R1( Instructor , Course ) and R2( Student , Course )
3. R1( Instructor , Course ) and R2( Instructor , Student )

All three decompositions lose FD1. Which decompositions are good?

# Desirable Properties of Decompositions

A decomposition of  $R$  into  $R_1$  and  $R_2$  must preserve attributes, that is,  $R = R_1 \cup R_2$ . We'd also like:

1. Dependency preservation (desired), and
2. Non-additive (lossless) joins (required).

Dependencies can be preserved in all 3NF decompositions, but not in all BCNF decompositions. **In all decompositions we must have non-additive join property.**

## Non-Additive Join Test

A Decomposition  $D = \{R_1, R_2\}$  of  $R$  has the lossless (nonadditive) join property with respect to FDs  $F$  on  $R$  if and only if either

- ▶ The FD  $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$  is in  $F^+$ , or
- ▶ The FD  $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$  is in  $F^+$

Important note: the non-additive join property assumes that **no null values are allowed for join attributes**.

The non-additive join test finds the right attribute to connect the two schemas, the attribute on which you would join the decomposed schemas to recover the original schema.

Remember how to test if  $X \rightarrow Y$  is in  $F^+$ ?

- ▶  $Y$  is in  $X^+$  under  $F$ . – use the attribute closure algorithm (Algorithm 15.1).

# Test of Decomposition # 1

For

1.  $R1(\underline{\text{Student}}, \underline{\text{Instructor}})$  and  $R2(\underline{\text{Student}}, \underline{\text{Course}})$
2.  $(R_1 \cap R_2) = \text{Student}$
3.  $(R_1 - R_2) = \text{Instructor}$
4.  $(R_2 - R_1) = \text{Course}$

So either

- ▶  $\text{Student} \rightarrow \text{Instructor}$ , or
- ▶  $\text{Student} \rightarrow \text{Course}$

must be in  $F^+$ . But they aren't.

# Visualizing Nonadditive Join

Say we have a relation state  $r(R) =$

student	course	instructor
Narayan	Database	Mark
Narayan	Operating Systems	Ammar
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe

# Bad Decomposition

$r(R1) =$

student	instructor
Narayan	Ammar
Narayan	Mark
Smith	Ammar
Smith	Navathe
Smith	Schulman
Wallace	Ahamad
Wallace	Mark
Wong	Omiecinski
Zelaya	Navathe

$r(R2) =$

student	course
Narayan	Database
Narayan	Operating Systems
Smith	Database
Smith	Operating Systems
Smith	Theory
Wallace	Database
Wallace	Operating Systems
Wong	Database
Zelaya	Database

We would join on student and end up with ...

## Join with Spurious Tuples

student	course	instructor
Narayan	Database	Ammar
Narayan	Database	Mark
Narayan	Operating Systems	Ammar
Narayan	Operating Systems	Mark
Smith	Database	Ammar
Smith	Database	Navathe

... and 13 more tuples, which is way more tuples than the original relation due to spurious tuples, so the join is not non-additive.

Lost the association between Instructor and Course. E.g., Mark does not teach Operating Systems.

## Test of Decomposition # 2

For

1.  $R1(\underline{\text{Instructor}}, \text{Course})$  and  $R2(\underline{\text{Student}}, \underline{\text{Course}})$
2.  $(R_1 \cap R_2) = \text{Course}$
3.  $(R_1 - R_2) = \text{Instructor}$
4.  $(R_2 - R_1) = \text{Student}$

So either

- ▶  $\text{Course} \rightarrow \text{Instructor}$ , or
- ▶  $\text{Course} \rightarrow \text{Student}$

must be in  $F^+$ . But they aren't.

# Test of Decomposition # 3

For

1.  $R1(\underline{\text{Instructor}}, \text{Course})$  and  $R2(\underline{\text{Instructor}}, \underline{\text{Student}})$
2.  $(R_1 \cap R_2) = \text{Instructor}$
3.  $(R_1 - R_2) = \text{Course}$
4.  $(R_2 - R_1) = \text{Student}$

So either

- ▶ Instructor  $\rightarrow$  Course, or
- ▶ Instructor  $\rightarrow$  Student

must be in  $F^+$ . Instructor  $\rightarrow$  Course is in  $F^+$ , so this decomposition is the right one.

# Schema Decomposition

A decomposition of a relation  $R$  into  $R_1$  and  $R_2$  can be defined as:

- ▶  $R_1 = \pi_A(R)$
- ▶  $R_2 = \pi_B(R)$

Where  $R = A \cup B$

To find the functional dependencies that hold on  $R_1$  and  $R_2$  we project the functional dependencies that hold on  $R$  into sets of FDs for  $R_1$  and  $R_2$ .

# Minimal Cover Sets of FDs

A set of FDs  $F$  is a minimal cover set if removing any FD changes  $F^+$ .  
To transform  $F$  into a minimal cover set:

- ▶ while there is an FD  $F$  in  $F$  that is implied by other FDs in  $F$ :
  - ▶ remove  $F$  from  $F$
- ▶ repeat
  - ▶ for each FD  $Y \rightarrow B$  in  $F$  with two or more attributes in  $Y$ :
    - ▶ let  $Z$  be  $Y$  minus one attribute in  $Y$
    - ▶ if  $Z \rightarrow B$  follows from the FDs in  $F$  (including  $Y \rightarrow B$ ), then replace  $Y \rightarrow B$  with  $Z \rightarrow B$
- ▶ until no more changes to  $F$  can be made

# Projection of FDs

**Input:** A relation  $R$ , a relation  $R_1$  computed by the projection  $\pi_L(R)$ , and a set of FDs  $S$  that hold on  $R$ .

1. **set**  $T = \{\}$  (the empty set)
2. **for each** subset of attributes  $X$  in  $R_1$ :
  - ▶ compute  $X^+$  with respect to  $S$ . Note that there may be attributes in  $X^+$  that are in  $R$  but not in  $R_1$ .
  - ▶ Add to  $T$  nontrivial FDs  $X \rightarrow A$  for which  $A$  is in  $X^+$  and  $R_1$ .
3. Optional: transform  $T$  into a minimal cover set of FDs.

**Output:**  $T$ , a (minimal) set of functional dependencies that hold on  $R_1$

## Bottom-Up Design Approaches

Bottom-up approaches start with one **universal relation** which contains all attributes in the database. 3NF or BCNF relation schemas are **synthesized** from this universal relation schema.

- ▶ Algorithm 15.4 synthesizes universal relation  $R$  into 3NF schemas that have the nonadditive join property **and** preserve dependencies.
- ▶ Algorithm 15.5 converts universal relation  $R$  into BCNF schemas that have the nonadditive join property (but not necessarily preserving dependencies) by iterative decomposition.

In this class you only need to know Algorithm 15.5, BCNF decomposition.

## Informal 3NF Synthesis

Informally, Algorithm 15.4 for 3NF synthesis does this:

1. Find a minimal cover set of FDs for  $R$ .
2. For each FD in the minimal cover create a relation schema with each attribute in the FD. The left-hand side of the FD is the key.
3. If none of the schemas above contains a key of  $R$ , create one more relation schema with attributes that form a key of  $R$  (the previously created schemas will contain foreign keys to this relation schema).
4. Eliminate redundant schemas.

Easy to understand conceptually, but many details which we don't require you to know.

## Informal BCNF Decomposition

Before diving into the much simpler BCNF decomposition algorithm, here's an informal description of the process it follows.

Let

- ▶  $R$  be a relation schema not in BCNF,
- ▶  $X \subseteq R$ , and
- ▶  $X \rightarrow A$  be the FD that violates BCNF.

Decompose  $R$  into

- ▶  $R - A$ , and
- ▶  $XA$

If either of these relations is not in BCNF, repeat the process.

# BCNF Decomposition Algorithm

**Algorithm 15.5:** Relational Decomposition into BCNF with Nonadditive Join Property

**Input:** A universal relation  $R$  and a set of FDs  $F$  on  $R$

1. **set**  $D := \{R\}$
2. **while** there is a relation schema  $Q$  in  $D$  that is not in BCNF:
  - ▶ choose a relation schema  $Q$  in  $D$  that is not in BCNF
  - ▶ find a functional dependency  $X \rightarrow Y$  in  $Q$  that violates BCNF
  - ▶ replace  $Q$  in  $D$  by two schemas  $(Q - X^+ + X)$  and  $X^+$
  - ▶ project the functional dependencies from  $Q$  into the new schemas.

**Output:**  $D$ , a set of relation schemas in BCNF with the non-additive join property such that  $D = \bigcup_1^n D_i$

Note that each schema has its own set of functional dependencies, so each decomposition results in the loss of one schema from  $D$  along with its functional dependencies, and the addition of two new schemas each with their own sets of functional dependencies.