

## Lecture 24: Decomposition and Synthesis

### Relation decomposition properties, “bottom-up” design

- attribute preservation
- dependency preservation
- lossless (non-additive) join property
- minimal cover of functional dependency set
- schema synthesis by decomposing the “universal relation”

“*Solve et Coagula*” - Latin motto of the alchemists

Today's material is from the beginning of [E&N Chapter 11].

## Attribute Preservation

We want to keep all the attributes, so we say a decomposition  $\mathbf{D}$  of  $\mathbf{R}$  has the *attribute preservation* property iff

$$\mathbf{R}_1 \cup \dots \cup \mathbf{R}_n = \mathbf{R}$$

ie, each attribute of  $\mathbf{R}$  is in at least one of the components  $\mathbf{R}_i$

## Decomposition

Decomposition is our only weapon in the war against unnormalised relations.

We have observed that some decompositions are kind of “bad”.

But it's all been a bit vague so far, it's time to get serious!

A *decomposition* of a relation schema  $\mathbf{R}$  is a set of relation schemas

$$\mathbf{D} = \{\mathbf{R}_1, \dots, \mathbf{R}_n\}$$

where each  $\mathbf{R}_i \subseteq \mathbf{R}$ .

## Dependency Projection

Given one schema  $\mathbf{R}_i$  from a decomposition, some functional dependencies will make sense for it, and others will not because some of the attributes are not there.

We want a way to pick out the ones that do make sense.

Recall that a projection  $\pi_X$  applies to a tuple or a whole relation, and just keeps the values for attributes in  $X$ .

Here is a similar idea for functional dependencies.

$$\pi_{\mathbf{R}_i}(F) = \{X \longrightarrow Y \in F \mid X \cup Y \subseteq \mathbf{R}_i\}$$

ie, the functional dependencies from  $F$  that only mention attributes in  $\mathbf{R}_i$

## Dependency Preservation

We have talked about “losing” functional dependencies when they don’t make sense for any of the decomposed relations.

ie  $X \longrightarrow Y \in F$  but  $X \longrightarrow Y \notin \pi_{R_i}(F)$  for any  $i$

Is it a problem when this happens?

Its not really the actual set  $F$  of functional dependencies we want to preserve, but its closure  $F^+$

So the decomposition does not need to “have” the *same* functional dependencies, just an equivalent set.

## Surprising Preservation?

This example shows that equivalent sets of functional dependencies are not necessarily preserved by the same decompositions.

$$\begin{aligned} R &= \{A, B, C\}, D = \{\{A, B\}, \{A, C\}\} \\ F &= \{A \longrightarrow BC\} \equiv \{A \longrightarrow B, A \longrightarrow C\} = G \end{aligned}$$

Now,  $F$  is not preserved, because

$$\pi_{A,B}(F) \cup \pi_{A,C}(F) = \{\} \cup \{\} = \{\} \neq F$$

But  $G$  is

$$\pi_{A,B}(G) \cup \pi_{A,C}(G) = \{A \longrightarrow B\} \cup \{A \longrightarrow C\} = G \equiv F$$

## Dependency Preservation

A decomposition  $D$  of  $R$  preserves dependencies iff

$$(\pi_{R_1}(F) \cup \dots \cup \pi_{R_n}(F))^+ = F^+$$

That is, the set of functional dependencies that make sense for some  $R_i$  is equivalent to the original set  $F$ .

## Minimal Cover

(A little rewind here to [E&N §10.2.4])

- a set  $F$  of functional dependencies covers another one  $G$  iff  $G^+ \subseteq F^+$
- so  $F$  and  $G$  are equivalent iff they cover one another
- a *minimal cover* of  $F$  is a cover which
  - ① (isn't a *proper* cover at all, but is actually equivalent!)
  - ② each dependency has only a single attribute on the RHS
  - ③ each dependency has as few attributes on the LHS as possible whilst maintaining  $G \equiv F$
  - ④ removing any dependency would lose  $G \equiv F$
- you can obtain a minimal cover for given  $F$  by just *doing* these three things (2,3,4)

## Working with Functional Dependencies

[E&N Chapter 11] is full of algorithms for working with functional dependencies. You might need them one day for a really big project.

But, in the labs next week, the tasks can be achieved by just having a look and a think.

eg, the set  $\{A \longrightarrow B, B \longrightarrow C, A \longrightarrow C\}$  can clearly be reduced to  $\{A \longrightarrow B, B \longrightarrow C\}$ , because we can recover  $A \longrightarrow C$  by the transitive deduction rule.

eg,  $\{AB \longrightarrow C, B \longrightarrow A\}$  can be reduced to  $\{B \longrightarrow C, B \longrightarrow A\}$ , because  $B \longrightarrow C$  can be derived from the first set.

## Lossless Join Property

(Also called non-additive join in [E&N])

When we have a decomposition  $\mathbf{D} = \{\mathbf{R}_1, \dots, \mathbf{R}_n\}$  of  $\mathbf{R}$ , and a relation  $R$  which is an instance of  $\mathbf{R}$ , we will write  $R_i$  for  $\pi_{\mathbf{R}_i}(R)$ .

A decomposition  $\mathbf{D} = \{\mathbf{R}_1, \dots, \mathbf{R}_n\}$  of a relation schema  $\mathbf{R}$  is *lossless* with respect to functional dependencies  $F$  iff for every state  $R$  satisfying  $F$

$$R_1 * \dots * R_n = R$$

## Different Minimal Covers?

There is not generally a unique minimal cover for a given set of FD's.

Will they always give the same answer about the highest normal form for a relation?

Yes, the *closure* determines what's dependent on what, the keys, and thus the non-prime attributes.

So, any equivalent set of FD's will give the same answers.

## We know when we haven't got it, but . . .

Recall the 3 decomposition choices for the  $\{student, course, instructor\}$  example in Lecture 23?

We showed that the first two were *not* lossless, by giving example relation states where the natural join of the decomposition had garbage in it.

We *think* the third one was lossless, but can not prove it yet.

## Testing the Lossless Join Property

[E&N Algorithm 11.1] takes any decomposition and functional dependencies, and tells you whether it is lossless.

But [E&N §11.1.4] gives a much simpler test for *binary* decompositions, which we mentioned in the previous lecture.

This is useful because we usually obtain a decomposition by several steps of splitting relations in two.

## Property NJB

### Nonadditive Join for Binary Decompositions

- the “join attributes” are  $R_1 \cap R_2$
- its a key for  $R_i$  if this determines the rest of the attributes
- the closure  $F^+$  of the functional dependencies  $F$  tells us what determines what
- the rest of the relation is either  $R_1 - R_2$  or  $R_2 - R_1$
- so, the binary decomposition  $\{R_1, R_2\}$  is lossless (non-additive) iff

$$(R_1 \cap R_2) \longrightarrow (R_1 - R_2) \in F^+ \text{ or}$$

$$(R_1 \cap R_2) \longrightarrow (R_2 - R_1) \in F^+$$

## Join on a Key

$R_1$		$R_2$	
course	instructor	student	instructor
Philosophy	Hegel	u123456	Hegel
Philosophy	Descartes	u234567	Descartes
Database	O'Keefe	u234567	O'Keefe

- join is on **instructor**
- but  $FD2 : instructor \longrightarrow course$
- so its a key in  $R_1$

See [E&N §11.1.4]

In general, if the join attributes are a key in one of the decomposition tables, we are laughing, because the other table “knows” what its tuples will join up with.

## “Synthesis” of Database Schemas

So far, we have assumed we have a database schema already, and we want to get its relation schemas into higher normal forms.

[E&N §11.2] contains algorithms that take

- a set of functional dependencies  $F$
- a set of attributes  $R$

and produce a database schema.

This is called “synthesis” but actually they start by putting all the attributes into a “universal relation”, then decomposing that.

These techniques are not used much in practice, because there is no obvious way of collecting all the functional dependencies.

## Synthesis Algorithm Properties

The main features that distinguish these algorithms are

- whether the resulting decomposition is lossless
- whether or not they preserve dependencies
- highest normal form they can guarantee

We will just look at Algorithm 11.4 [E&N §11.2.3] which

- is lossless
- preserves dependencies
- yields relation schemas in at least (EN)3NF

## Properties of this Algorithm

- preserves attributes
  - if there is no  $X \longrightarrow A$  in the minimal cover  $G$ , then  $A$  is in every key of  $\mathbf{R}$  and thus in the “extra” relation
- preserves dependencies
  - every functional dependency  $X_i \longrightarrow Y_i$  in  $H$  becomes  $\mathbf{R}_i$  with the attributes of  $X_i$  and  $Y_i$
  - therefore  $X_i \longrightarrow Y_i \in \pi_{\mathbf{R}_i}(H)$
  - therefore  $\pi_{\mathbf{R}_1}(H) \cup \dots \cup \pi_{\mathbf{R}_n}(H) = H$
  - and  $H$  is equivalent to  $F$
- each  $\mathbf{R}_i$  is in EN3NF
  - (very roughly)
  - if  $K \longrightarrow A$  is not full, we get  $K \supset X \longrightarrow A$  and  $X \longrightarrow A$  should be in the minimal cover, but isn't
  - if  $K \longrightarrow A$  is transitive, then we get  $K \longrightarrow X \longrightarrow A$  and again  $X \longrightarrow A$  should have been in the minimal cover
- lossless join (read the book if you want to know why!)

## A Synthesis Algorithm

Input is attributes  $\mathbf{R}$  and functional dependencies  $F$

- 1 obtain a minimal cover  $G$  of  $F$
- 2 obtain an even cooler kind of cover,  $H$  with minimum number of maximally strong functional dependencies, by
  - 1 replacing  $X \longrightarrow Y$  and  $X \longrightarrow Z$  by  $X \longrightarrow YZ$  as often as possible
- 3 for each  $X \longrightarrow Y$  in  $H$ , create a table with attributes  $X \cup Y$  and primary key  $X$
- 4 if no key of  $\mathbf{R}$  is present in any relation schema, add one as a relation
- 5 remove any relation schemas which are contained by others, ie if  $\mathbf{R}_i \subseteq \mathbf{R}_j$ , remove  $\mathbf{R}_i$

## A few final thoughts on normal forms

[E&N 11.2.4]

- NULL's stuff everything up [E&N 11.2.4]
  - if NULL values are allowed, then that attribute isn't suitable as part of a primary key
  - when NULL's occur in join attributes, the joined result will be less than the undecomposed “original”
- there are 4th and 5th normal forms [E&N 11.4, 5] and more
  - they are based on different kinds of dependencies: “multi-valued” and “join” dependencies respectively
  - they are a more theoretical than practical (“normalization into 5NF is very rarely done in practice” [E&N §11.4])
- **these ideas should really sink in when you do next weeks lab exercises!**