

# Design Theory for Relational Databases

csc343, Winter 2017

Diane Horton  
University of Toronto

Originally based on slides by Jeff Ullman

# Introduction

- ◆ There are always many different schemas for a given set of data.
- ◆ E.g., you could combine or divide tables.
- ◆ How do you pick a schema?  
Which is better?  
What does “better” mean?
- ◆ Fortunately, there are some principles to guide us.

# Database Design Theory

- ◆ It allows us to improve a schema systematically.
- ◆ General idea:
  - ◆ Express constraints on the relationships between attributes
  - ◆ Use these to decompose the relations
- ◆ Ultimately, get a schema that is in a “normal form” that guarantees good properties.
- ◆ “Normal” in the sense of conforming to a standard.
- ◆ The process of converting a schema to a normal form is called **normalization**.

# Part I:

# Functional Dependency Theory

# A poorly designed table

part	manufacturer	manAddress	seller	sellerAddress	price
1983	Hammers `R Us	99 Pinecrest	ABC	1229 Bloor W	5.59
8624	Lee Valley	102 Vaughn	ABC	1229 Bloor W	23.99
9141	Hammers `R Us	99 Pinecrest	ABC	1229 Bloor W	12.50
1983	Hammers `R Us	99 Pinecrest	Walmart	5289 St Clair W	4.99

- ◆ In any domain, there are relationships between attribute values.
- ◆ Perhaps:
  - ◆ Every part has 1 manufacturer
  - ◆ Every manufacture has 1 address
  - ◆ Every seller has 1 address
- ◆ If so, this table will have redundant data.

# Principle: Avoid redundancy

Redundant data can lead to anomalies.

part	manufacturer	manAddress	seller	sellerAddress	price
1983	Hammers `R Us	99 Pinecrest	ABC	1229 Bloor W	5.59
8624	Lee Valley	102 Vaughn	ABC	1229 Bloor W	23.99
9141	Hammers `R Us	99 Pinecrest	ABC	1229 Bloor W	12.50
1983	Hammers `R Us	99 Pinecrest	Walmart	5289 St Clair W	4.99

- **Update anomaly**: if Hammers `R Us moves and we update only one tuple, the data is inconsistent.
- **Deletion anomaly**: If ABC stops selling part 8624 and Lee Valley makes only that one part, we lose track of its address.

# Definition of FD

- ◆ Suppose  $R$  is a relation, and  $X$  and  $Y$  are subsets of the attributes of  $R$ .
- ◆  $X \rightarrow Y$  asserts that:
  - ◆ If two tuples agree on all the attributes in set  $X$ , they must also agree on all the attributes in set  $Y$ .
- ◆ We say that " $X \rightarrow Y$  holds in  $R$ ", or " $X$  functionally determines  $Y$ ."
- ◆ An FD constrains what can go in a relation.

# More formally...

$A \rightarrow B$  means:

$\forall$  tuples  $t_1, t_2$ ,

$$(t_1[A] = t_2[A]) \Rightarrow (t_1[B] = t_2[B])$$

Or equivalently:

$\neg \exists$  tuples  $t_1, t_2$  such that

$$(t_1[A] = t_2[A]) \wedge (t_1[B] \neq t_2[B])$$



# Generalization to multiple attributes

$A_1 A_2 \dots A_m \rightarrow B_1 B_2 \dots B_n$  means:

$\forall$  tuples  $t_1, t_2,$

$(t_1[A_1] = t_2[A_1] \wedge \dots \wedge t_1[A_m] = t_2[A_m]) \Rightarrow$

$(t_1[B_1] = t_2[B_1] \wedge \dots \wedge t_1[B_n] = t_2[B_n])$

Or equivalently:

$\neg \exists$  tuples  $t_1, t_2$  such that

$(t_1[A_1] = t_2[A_1] \wedge \dots \wedge t_1[A_m] = t_2[A_m]) \wedge$   
 $\neg (t_1[B_1] = t_2[B_1] \wedge \dots \wedge t_1[B_n] = t_2[B_n])$

# Why “functional dependency”?

- ◆ “dependency” because the value of  $Y$  depends on the value of  $X$ .
- ◆ “functional” because there is a mathematical function that takes a value for  $X$  and gives a *unique* value for  $Y$ .
- ◆ (It’s not a typical function; just a lookup.)

# Equivalent sets of FDs

- ◆ When we write a set of FDs, we mean that all of them hold.
- ◆ We can very often rewrite sets of FDs in equivalent ways.
- ◆ When we say  $S_1$  is equivalent to  $S_2$  we mean that:
  - ◆  $S_1$  holds in a relation iff  $S_2$  does.

# Splitting rules for FDs

- ◆ Can we split the RHS of an FD and get multiple, equivalent FDs?
- ◆ Can we split the LHS of an FD and get multiple, equivalent FDs?

# Coincidence or FD?

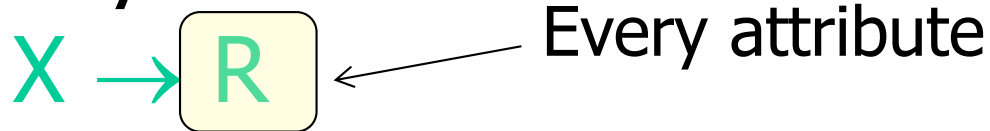
- ◆ An FD is an assertion about *every* instance of the relation.
- ◆ You can't know it holds just by looking at one instance.
- ◆ You must use knowledge of the domain to determine whether an FD holds.

# FDs are closely related to keys

- ◆ Suppose  $K$  is a set of attributes for relation  $R$ .
- ◆ Our old definition of superkey:  
a set of attributes for which no two rows can have the same values.
- ◆ A claim about FDs:  
 $K$  is a *superkey* for  $R$  iff  
 $K$  functionally determines all of  $R$ .

# FDs are a generalization of keys

- ◆ key:



- ◆ Functional dependency:



- ◆ An FD can be more subtle.

# Inferring FDs

- ◆ Given a set of FDs, we can often infer further FDs.
- ◆ This will be handy when we apply FDs to the problem of database design.
- ◆ Big task: given a set of FDs,  
infer *every* other FD that must also hold.
- ◆ Simpler task: given a set of FDs,  
check whether *a given* FD must also hold.



# Examples

- ◆ If  $A \rightarrow B$  and  $B \rightarrow C$  hold,  
must  $A \rightarrow C$  hold?
- ◆ If  $A \rightarrow H$ ,  $C \rightarrow F$ , and  $FG \rightarrow AD$  hold,  
must  $FA \rightarrow D$  hold?  
must  $CG \rightarrow FH$  hold?
- ◆ If  $H \rightarrow GD$ ,  $HD \rightarrow CE$ , and  $BD \rightarrow A$  hold,  
must  $EH \rightarrow C$  hold?
- ◆ Aside: we are not generating new FDs,  
but testing a specific possible one.

# Method 1: Prove an FD follows using first principles

- ◆ You can prove it by referring back to
  - ▶ The FDs that you know hold, and
  - ▶ The definition of functional dependency.
- ◆ But the Closure Test is easier.

# Method 2: Prove an FD follows using the Closure Test

- ◆ Assume you know the values of the LHS attributes, and figure out everything else that is determined.
- ◆ If it includes the RHS attributes, then you know that  $\text{LHS} \rightarrow \text{RHS}$
- ◆ This is called the closure test.

*Y is a set of attributes, S is a set of FDs.  
Return the closure of Y under S.*

Attribute\_closure(Y, S):

Initialize  $Y^+$  to Y

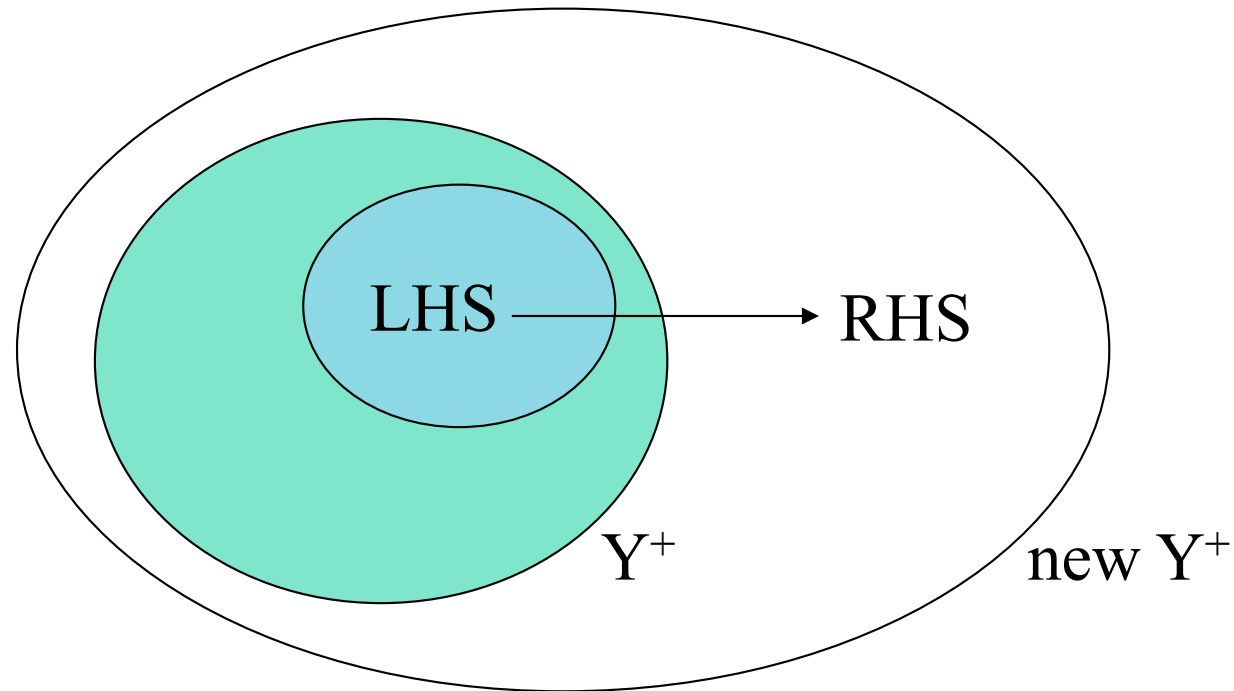
Repeat until no more changes occur:

    If there is an FD LHS  $\rightarrow$  RHS in S  
    such that LHS is in  $Y^+$ :

        Add RHS to  $Y^+$

Return  $Y^+$

# Visualizing attribute closure



If  $\text{LHS}$  is in  $Y^+$  and  $\text{LHS} \rightarrow \text{RHS}$  holds, we can add  $\text{RHS}$  to  $Y^+$

*S is a set of FDs; LHS  $\rightarrow$  RHS is a single FD.  
Return true iff LHS  $\rightarrow$  RHS follows from S.*

FD\_follows(S, LHS  $\rightarrow$  RHS):  
     $Y^+ = \text{Attribute\_closure}(\text{LHS}, S)$   
    return (RHS is in  $Y^+$ )

# Projecting FDs

- ◆ Later, we will learn how to **normalize** a schema by decomposing relations.

This is the whole point of this theory.

- ◆ We will need to know what FDs hold in the new, smaller, relations.

We must **project** our FDs onto the attributes of our new relations.

# Example

$R(A_1, \dots, A_n)$  Set of attributes:  $A$

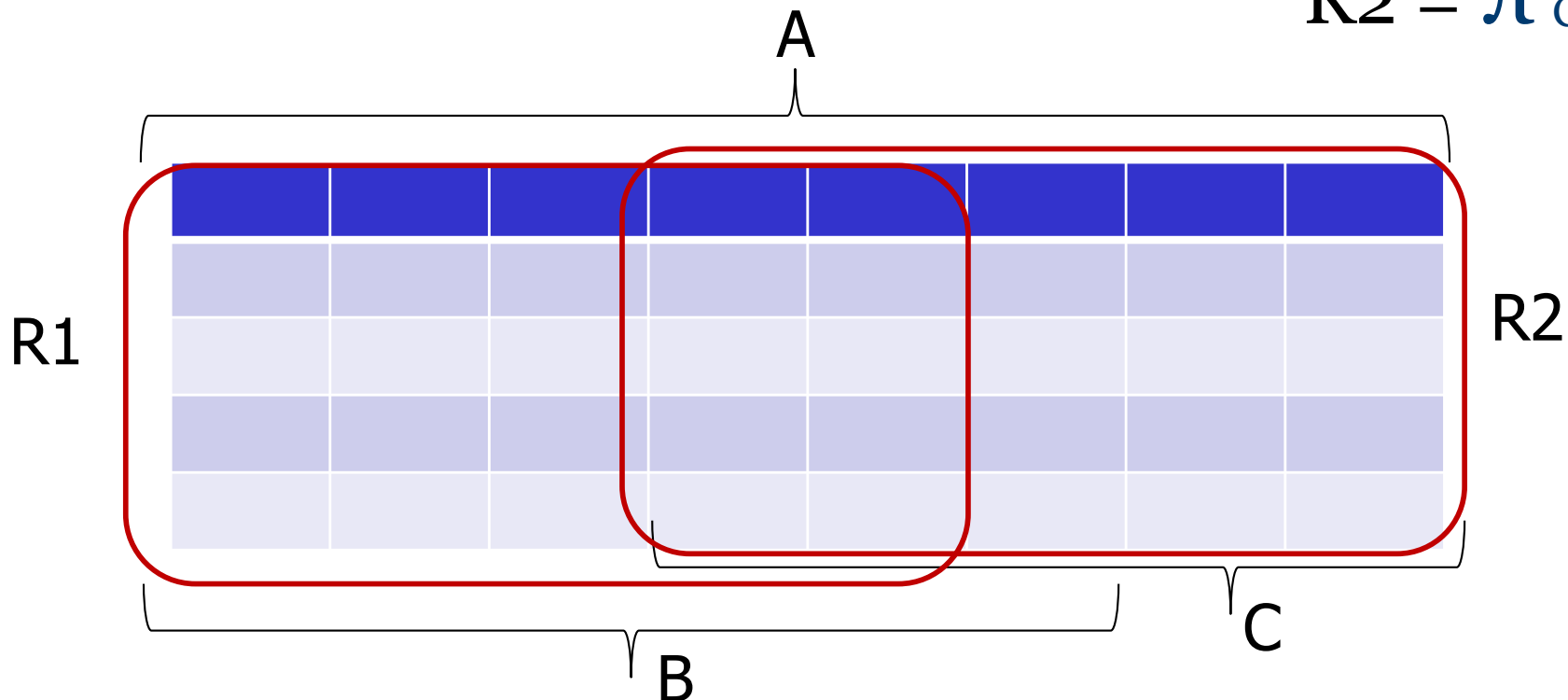
Decompose into:

- $R_1(B_1, \dots, B_k)$  Set of attributes:  $B$ , and
- $R_2(C_1, \dots, C_m)$  Set of attributes:  $C$

$$B \cup C = A, \quad R_1 \bowtie R_2 = R$$

$$R_1 = \pi_B(R)$$

$$R_2 = \pi_C(R)$$





*S is a set of FDs; L is a set of attributes.*

*Return the projection of S onto L:*

*all FDs that follow from S and involve only attributes from L.*

Project(S, L):

Initialize T to {}.

For each subset X of L:

Compute  $X^+$  *Close X and see what we get.*

For every attribute A in  $X^+$ :

If A is in L:  *$X \rightarrow A$  is only relevant if A is in L (we know X is).*

add  $X \rightarrow A$  to T.

Return T.

# A few speed-ups

- ◆ No need to add  $X \rightarrow A$  if  $A$  is in  $X$  itself. It's a trivial FD.
- ◆ These subsets of  $X$  won't yield anything, so no need to compute their closures:
  - ◆ the empty set
  - ◆ the set of all attributes
- ◆ Neither are big savings, but ...

# A big speed-up

- ◆ If we find  $X^+ = \text{all attributes}$ , we can ignore any superset of  $X$ .
  - ◆ It can only give use “weaker” FDs (with more on the LHS).
- ◆ This is a big time saver!

# Projection is expensive

- ◆ Even with these speed-ups, projection is still expensive.
- ◆ Suppose  $R_1$  has  $n$  attributes.  
How many subsets of  $R_1$  are there?

# Minimal Basis

- ◆ We saw earlier that we can very often rewrite sets of FDs in equivalent ways.
- ◆ Example:  $S_1 = \{A \rightarrow BC\}$  is equivalent to  $S_2 = \{A \rightarrow B, A \rightarrow C\}$ .
- ◆ Given a set of FDs  $S$ , we may want to find a **minimal basis**: A set of FDs that is equivalent, but has
  - ◆ no redundant FDs, and
  - ◆ no FDs with unnecessary attributes on the LHS.

*S is a set of FDs. Return a minimal basis for S.*

Minimal\_basis(S):

1. Split the RHS of each FD
2. For each FD  $X \rightarrow Y$  where  $|X| \geq 2$ :  
If you can remove an attribute from  $X$   
and get an FD that follows from S:  
Do so! (It's a stronger FD.)
3. For each FD  $f$ :  
If  $S - \{f\}$  implies  $f$ :  
Remove  $f$  from S.

# Some comments on minimal basis

- ◆ Often there are multiple possible results.  
Depends on the order in which you consider the possible simplifications.
- ◆ After you identify a redundant FD, you must not use it when computing subsequent closures.

## ... and less intuitive

- ◆ When you are computing closures to decide whether the LHS of an FD

$$X \rightarrow Y$$

can be simplified, continue to use that FD.

- ◆ You must do (2) and (3) in that order.  
Otherwise, must repeat until no changes occur.