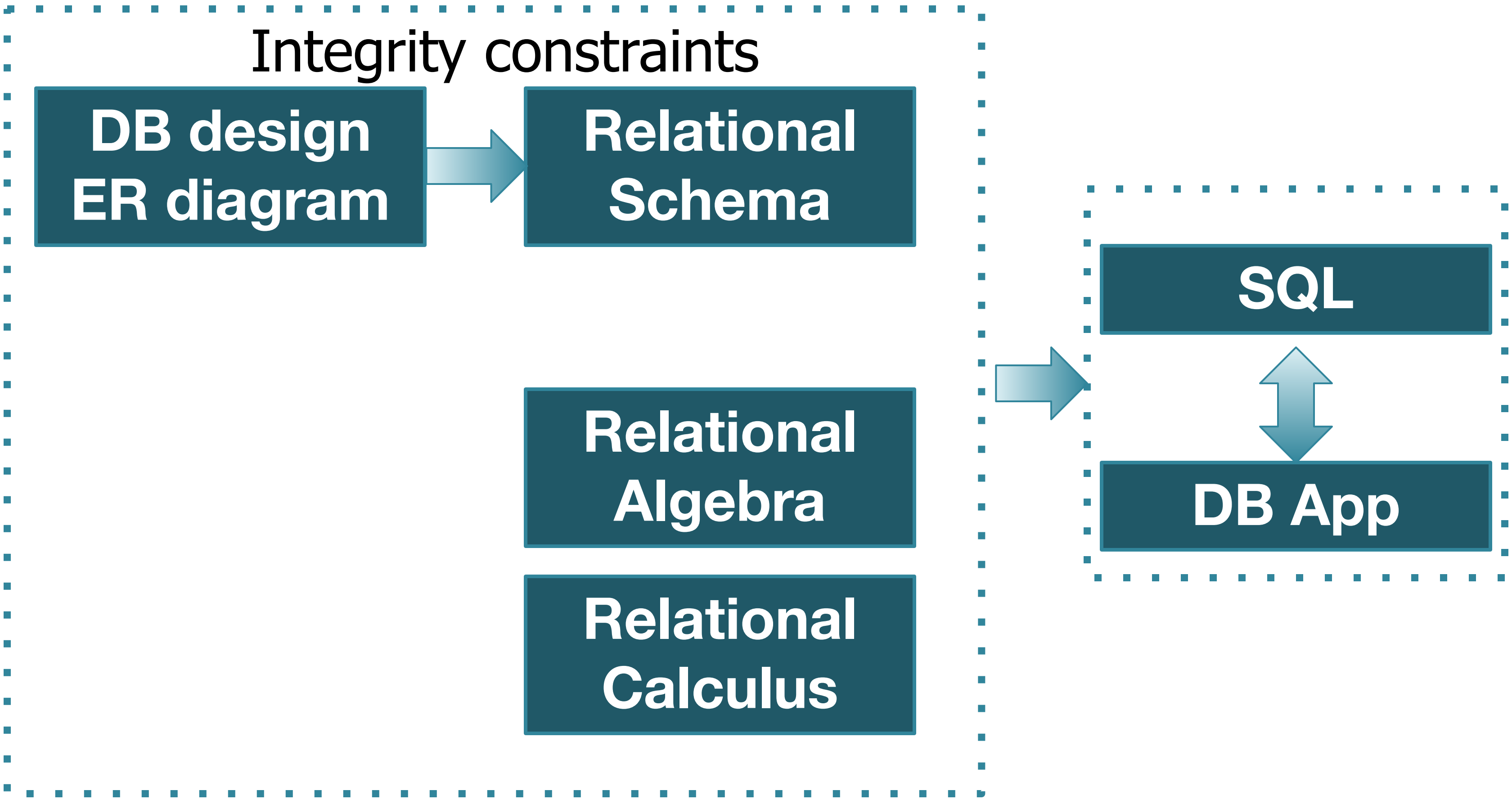
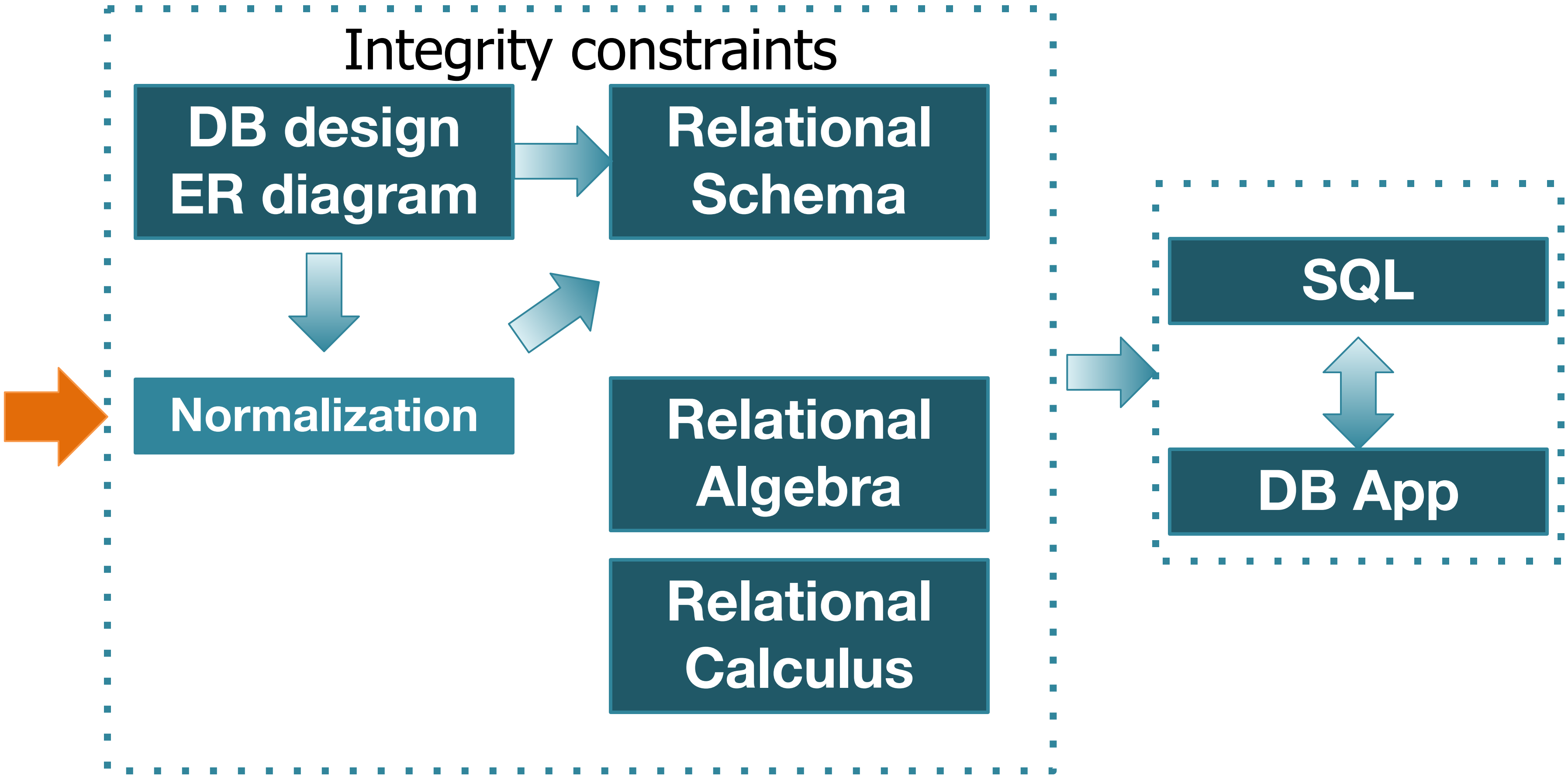
Normalization using Schema Refinement

Chapter 19

Review


Integrity constraints
DB design
ER diagram
Relational
Schema
Relational
Algebra
Relational
Calculus
SQL
DB App

Today



Integrity constraints
DB design ER diagram
Relational Schema
Normalization
Relational Algebra
Relational Calculus
SQL
DB App

Form/Spreadsheet

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
			Paint	blue	$10
			Flowers	pink	$3
2	Beanery	A2, A3	Dynamite	boom	$8

(Note: ACME is a fictional corporation from the Road Runner and Wile E. Coyote series)

Problems with the above table?

Form/Spreadsheet

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$8
			Paint	blue	$10
			Flowers	pink	$3
2	Beanery	A2, A3	Dynamite	boom	$8

Bad Table!

- (Supplier ID, item) appears to be the key, but Supplier ID is NULL in many places – assumed to be copied from the prior non-null entry – ordering matters.
- Addresses appear to be multi-valued
- Redundancy in (Item, Desc)

Normalization

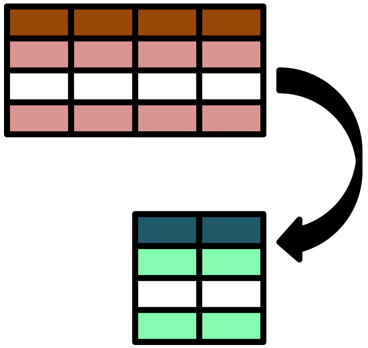
Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
			Paint	blue	$10
			Flowers	pink	$3
2	Beanery	A2, A3	Dynamite	boom	$8

Going to a proper set of tables is called "normalization”.

- avoid redundancy of data
- capture the dependencies inherent in the data

- We will always start with one giant table and then "normalize it" into multiple tables, ala, Project 1

Goal



- Design ‘good’ tables
  - What is good?
  - How to fix bad tables?
- In short:

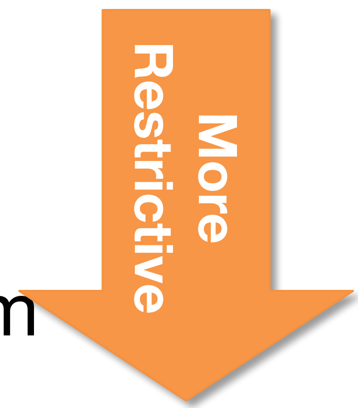
> We want tables where the attributes depend on the primary key, on the whole key, and nothing but the key.

Two Approaches to Normalization

- Approach 1 (you did this in Project 1):
  - Create an ER model and then map to tables. Should result in good (normalized) tables (very manual)
- Approach 2 [Today]:
  - State dependencies between attributes of tables
  - Map dependencies to tables. Can be done automatically!

Normal Forms

- Guarantees that certain problems won’t occur & obeys certain rules:
  - 1 NF : Starting point
  - 2 NF : Historical
  - 3 NF : …
  - BCNF : Boyce-Codd Normal Form
  - 4NF: Use lossless decompositions for multi-valued dependencies



1st Normal Form – First Step

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
			Paint	blue	$10
			Flowers	pink	$3
2	Beanery	A2, A3	Dynamite	boom	$8

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Dynamite	boom	$8
2	Beanery	A3	Dynamite	boom	$8

1st Normal Form – First Step

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Dynamite	boom	$8
2	Beanery	A3	Dynamite	boom	$8

- Each value in table is single-valued
- Each row contains all the relevant data

We now have a relational table.
Rows can be reordered, all rows independent.

1st Normal Form: Redundancy remains however

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Dynamite	boom	$8
2	Beanery	A3	Dynamite	boom	$8

Redundancy bad for changing DBs

- Space inefficient – same thing stored multiple times
- Makes for messy update process
  - **Update anomalies:** Changing address of a suppler requires changing multiple rows
  - **Insertion anomalies:** Inserting a supplier requires inserting NULL or other values in unrelated columns
  - **Deletion anomalies:** Deleting an item requires care. It could end up deleting a Supplier as well.

Dealing with Redundancy

- Normalize tables further
- ER Diagramming and translation to Relational model did that
- But ER diagramming and translation seems a bit ad hoc. Can the method be formalized?
- We will learn another trick today: using functional dependencies more to normalize tables

Example Normalization using ER approach

- **ER Approach:** (1) Supplier Entity, (2) Item Entity, (3) Supplier-Item Relationship with Price as an attribute

Supplier ID	Supplier Name	Item	Desc	Price
1	Acme	Dynamite	boom	$7
1	Acme	Paint	blue	$10
1	Acme	Flowers	pink	$3
2	Beanery	Dynamite	boom	$8
2	Beanery	Dynamite	boom	$8

Supplier ID	Supplier Name
1	Acme
2	Beanery

Item	Desc
Dynamite	boom
Paint	blue
Flowers	pink

Supp ID	Item	Price
1	Dynamite	$7
1	Paint	$10
1	Flowers	$3
2	Dynamite	$8

Alternative Way: Use Functional Dependencies

Supplier ID	Supplier Name	Item	Desc	Price
1	Acme	Dynamite	boom	$7
1	Acme	Paint	blue	$10
1	Acme	Flowers	pink	$3
2	Beanery	Dynamite	boom	$8
2	Beanery	Dynamite	boom	$8

Supplier ID	Supplier Name
1	Acme
2	Beanery

Item	Desc
Dynamite	boom
Paint	blue
Flowers	pink

Supp ID	Item	Price
1	Dynamite	$7
1	Paint	$10
1	Flowers	$3
2	Dynamite	$8

Use key constraints among attributes as the starting point

- Supplier ID → Supplier Name
- Item → Desc
- Supplier ID, Item → Price

Functional Dependencies (FD)

- FD captures dependency between attributes.
- Notation: $X \rightarrow Y$
- Read as: X functionally determines Y
- i.e., *Y depends on X* or *for a given X, there is one Y*

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Dynamite	boom	$8
2	Beanery	A3	Dynamite	boom	$8

E.g.: Supplier ID $\rightarrow$ Supplier Name

FD: Definition

- Notation: $X \rightarrow Y$
- **Informally:** Given a specific X, there is one Y value.
- **Formally**: A form of Integrity Constraint

D: $X \rightarrow Y$ X and Y subsets of a relation R's attributes.

Given tuples t1 and t2 in relation instance r of R:

$$\pi_X(t1) = \pi_X(t2) \Rightarrow \pi_y(t1) = \pi_y(t2)$$

(Supplier ID, Item) $\rightarrow$ Price

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Dynamite	boom	$8
2	Beanery	A3	Dynamite	boom	$8

Question?

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Dynamite	boom	$8
2	Beanery	A3	Dynamite	boom	$8

Which of the following FDs are definitely wrong?

- Item $\rightarrow$ Desc
- Item $\rightarrow$ Price

A. Yes/Yes B. Yes/No C. No/Yes D No/No

FD: Example

FDs capture dependencies among attributes

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Dynamite	boom	$8
2	Beanery	A3	Dynamite	boom	$8

- Supplier ID → Supplier Name
- Item → Desc
- Supplier ID, Item → Price

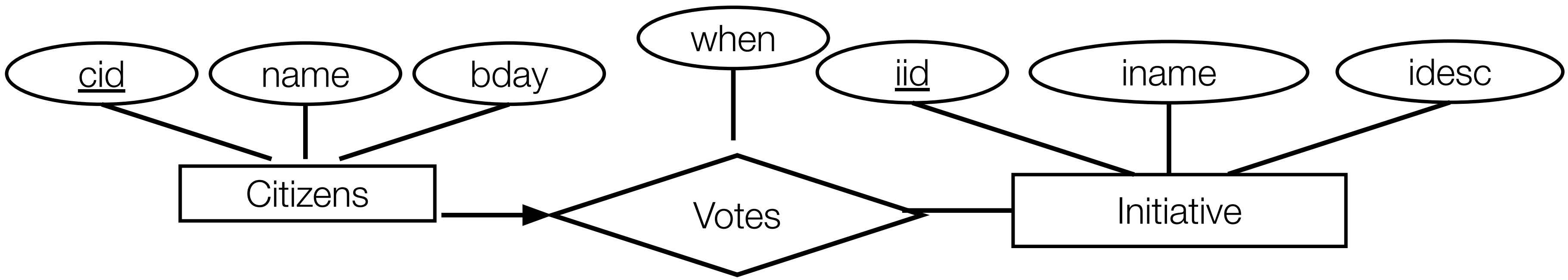
FD: Example

FDs capture dependencies among attributes

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Dynamite	boom	$8
2	Beanery	A3	Dynamite	boom	$8

- Supplier ID → Supplier Name
- Item → Desc
- Supplier ID, Item → Price

Another Example



What are the FDs among attributes in the above diagram? Try it out.

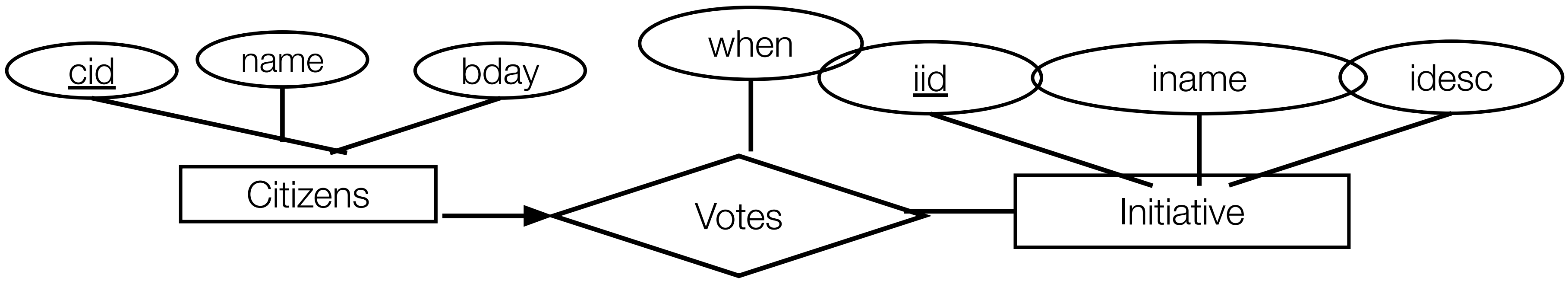
cid ⟶ name, bday

iid ⟶ iname, idesc   (iid ⟶ iname; iid ⟶ idesc)

cid, iid ⟶ when.   (even if there was no arrow)

cid ⟶ iid

Another Example



What are the FDs for the attributes in the above diagram?

cid ⟶ name, bday

iid ⟶ iname, idesc

cid, iid ⟶ when

cid ⟶ iid

A logically equivalent answer:

cid ⟶cid, name, bday, when, iid, iname, idesc *(i.e., everything)*

iid ⟶ iid, iname, idesc

More on FDs

- An FD is a statement about all allowable relations.
  - Based only on application semantics, not a table instance

Primary Key IC: special case of FD

- Primary key attributes → All other attributes

Basic Normalization

- Map FDs to tables

Supplier ID	Supplier Name	Item	Desc	Price
1	Acme	Dynamite	boom	$7
1	Acme	Paint	blue	$10
1	Acme	Flowers	pink	$3
2	Beanery	Dynamite	boom	$8
2	Beanery	Dynamite	boom	$8

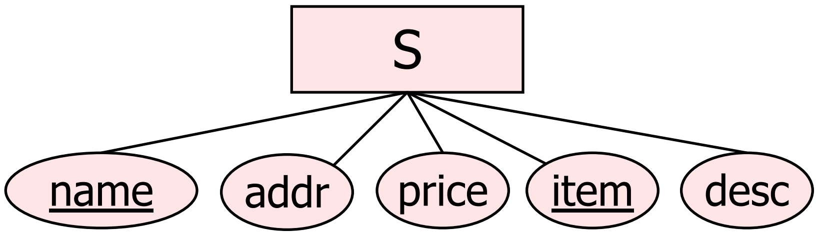
Supplier ID	Supplier Name
1	Acme
2	Beanery

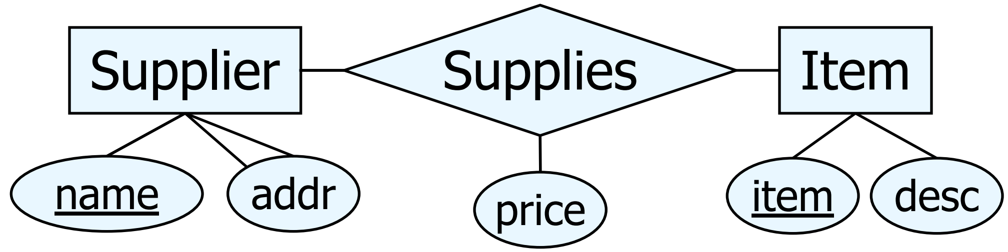
Item	Desc
Dynamite	boom
Paint	blue
Flowers	pink

Supp ID	Item	Price
1	Dynamite	$7
1	Paint	$10
1	Flowers	$3
2	Dynamite	$8

- Supplier ID → Supplier Name
- Item → Desc
- Supplier ID, Item → Price

Example: Constraints on Entity Set





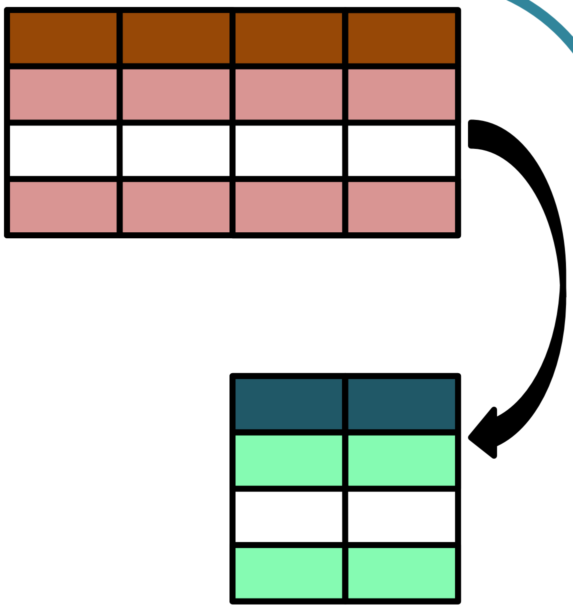
- S(name, item, desc, addr, price)
- FD: {n,i} → {n,i,d,a,p}
- Additional dependencies:
  - FD: {n} → {a}
  - FD: {i} → {d}
- Decompose to: NA, ID, INP

- Resulting Tables:
  - Supplier(name, addr)
    - FD: {n} → {n, a}
  - Item (item, desc)
    - FD: {i} → {i, d}
  - Supplies(name, item, price)
    - FD: {n,i} → {n, i, p}

ER design is subjective and can have many E + Rs
FDs: More systematic

High -Level Goal

- Given a relation and FDs:
  - R(sid, sname, …)
  - FDs (sid →…, iid → …)
- Algorithm that generates
  - ‘good’ schemas



Concept of Closure

Given:

- A base set of “facts”
- A set of derivation rules

Closure is the set of all derivable facts

E.g.

Facts: $a < a+1$ for all natural numbers a

Derivation rule: transitivity

Closure = ??

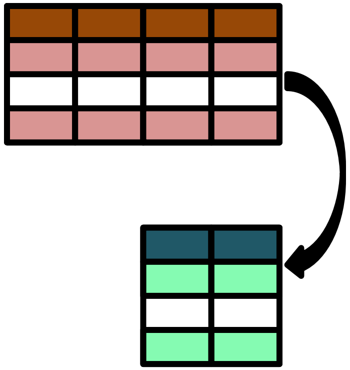
Implied FDs

- F+: Closure of F = Set of all valid FDs

F:

Supplier ID → Supplier Name
Item → Desc
Supplier ID, Item → Price

- Many other dependencies in the closure F+, e.g.,
  - Supplier ID, Item → Desc
  - Supplier ID → Supplier ID
  - Supplier ID → Supplier ID, Supplier Name



- How to derive F+: Armstrong's Axioms!

First Armstrong Axioms

- Axiom#1: Reflexive Property
- Given attribute sets X and Y:
  - Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$

- Example: (Supplier ID, Item#) → Item#
- In the example, X is [Supplier ID, Item#]. Y is [Item#]
- Given left side, there is a unique value for right side
- This is called a trivial dependency
  - E.g., X → **X**

Armstrong's Inference Axioms

- Armstrong’s Axioms (X, Y, Z are sets of attributes):
  - Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$ (trivial dependency)
  - Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

    e.g. ename → ejob, ejob → esal; ⇒ ename → esal
- Additional useful rules (derivable):
  - Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Deriving Union Rule from Axioms

- Prove: if $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$
- Proof:

- Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$ (trivial dependency)
- Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
- Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Deriving Union Rule from Axioms

- Prove: if $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$
- Proof:
  1. $X \rightarrow Y$ (given)
  2. $X \rightarrow Z$ (given)
  3. $XX \rightarrow XZ$ or $X \rightarrow XZ$ (augmentation of 2)
  4. $XZ \rightarrow YZ$ (augmentation of 1)
  5. $X \rightarrow YZ$ (transitivity of 3 and 4)
- Possible to derive the decomposition rule from the basic Armstrong rules

Question?

Given the FD

$X \rightarrow A$, where A includes all attributes of a table R, you can deduce that:

A. X is primary key
B. X is candidate key
C. X is superkey
D. Cannot say for sure: could be any (or none) of the above

Closure

- F+: Closure of F = Set of all FDs that can be derived from F using Armstrong's axioms
- E.g., F = {X → Y, Y → Z}
- F+ = {X → Y, X → Z, (original)

  X → X, Y → Y, Z → Z, XY → X, XY → XY, … (reflexive)

  X → Z (transitivity),

  X → YZ (union), …}

Armstrong's Axioms: Sound and Complete

- F*: All FDs that are implied by F
- F+: All FDs that can be generated from F by applying Armstrong's Axioms
- Soundness: F+ is a subset of F*
- Completeness: F* is a subset of F+
- Armstrong's Axioms can be shown to be both sound and complete

Solution to Redundancy: Decomposition

- Split a large relation to smaller ones to eliminate redundancies

Supplier ID	Supplier Name	Supplier Address	Item	Desc	Price
1	Acme	A1	Dynamite	boom	$7
1	Acme	A1	Paint	blue	$10
1	Acme	A1	Flowers	pink	$3
2	Beanery	A2	Dynamite	boom	$8
2	Beanery	A3	Dynamite	boom	$8

?

?

Solution to Redundancy: Decomposition

Two key goals of decomposition:

Must have
- Lossless Join: Can we reconstruct the original relation from instances of the decomposed relations?

Good to have
- Dependency Preservation: Avoid having to join decomposed relations to check dependencies

- Downside of decomposition:
  - Some queries become more expensive (more joins)

Lossless Join Decompositions

- Given Relation R, FDs F: Say, R decomposed to X, Y
- Decomposition of R into X,Y is Lossless-Join if joining back X and Y always gives R, i.e.,

$\pi_X(r) \bowtie \pi_Y(r) = r$ for **every** instance r of R

- Project 1: Part 4 checked if part 2 satisfied lossless-join!
- The following decomposition is **not** lossless-join:

A	B	C
1	2	3
4	5	6
7	2	8

r

A	B
1	2
4	5
7	2

$\pi_{AB}(r)$

B	C
2	3
5	6
2	8

$\pi_{BC}(r)$

$\bowtie$

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Lossless Join (cont.)

- Relation R, FDs F; Decomposed to X, Y
  - Test: lossless-join w.r.t. F if and only if F^+ contains:

$$X \cap Y \rightarrow X, \quad \text{or} \quad X \cap Y \rightarrow Y$$

i.e. attributes common to X and Y contain a key for either X or Y

Lossless join decomposition is always required!

Dependency Preserving Decomposition

- **Informally**: We don’t want the original FDs to span two tables.
- R has a dependency-preserving decomposition to X, Y

  if $F^+ = (F_x \cup F_y)^+$
- Note: F not necessarily = $F_x \cup F_y$

- Example:
- R (sailor, boat, date) F: {D → S, D → B}
- Consider decomp. to X (sailor, boat) Y (boat, date) and dependencies F_Y: {D → B}.
- To enforce D → S, must join X and Y (expensive)

The above is not dependency preserving

Decomposition: Example

X

ssn	cid	grade
123	413	A
123	415	B
234	211	A

Y

ssn	name	addr
123	Smith	Main
234	Jones	Huron

Given the dependencies below:
Does $X \cap Y \rightarrow X$? Does $X \cap Y \rightarrow Y$?

---

ssn	cid	grade	name	addr
123	413	A	Smith	Main
123	415	B	Smith	Main
234	211	A	Jones	Huron

ssn → name, address
(assigned to Y after decomposition)
ssn, cid → grade

Decomposition: Example

X

ssn	cid	grade
123	413	A
123	415	B
234	211	A

ssn, cid → grade

Y

ssn	name	addr
123	Smith	Main
234	Jones	Huron

ssn → name, address

ssn	cid	grade	name	addr
123	413	A	Smith	Main
123	415	B	Smith	Main
234	211	A	Jones	Huron

ssn → name, address
ssn, cid → grade

Is X, Y decomposition dependency preserving?
Does it satisfy Lossless-join?

Example continued

- Is it dependency preserving?
  - Yes! Joins are not required to capture all the original dependencies
- Does decomposition have lossless-join property?
  - Check if one of the following is true.
    - $X \cap Y \rightarrow X$, i.e., ssn → ssn, cid, grade
    - $X \cap Y \rightarrow Y$, i.e., ssn → ssn, name, addr

Yes, it has the lossless-join property! Second one is true.

This decomposition is lossless and dependency preserving!

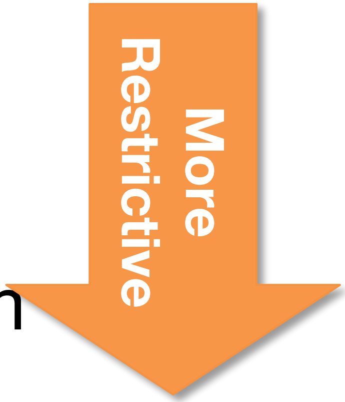
Question?

Suppose you have a choice between Lossless Join and Dependency Preservation. That is, you find you can get at most one of these properties.
Which would you prefer?

A. Lossless Join
B. Dependency Preservation
C. Either: it doesn't matter
D. Depends on the specific case

Normal Forms

- Guarantees that certain problems won’t occur & obeys certain rules:
  - 1 NF : No set-valued attrs
  - 2 NF : Historical
  - 3 NF : …
  - BCNF : Boyce-Codd Normal Form
  - 4NF: Use lossless decompositions for multi-valued dependencies



Boyce-Codd Normal Form (BCNF)

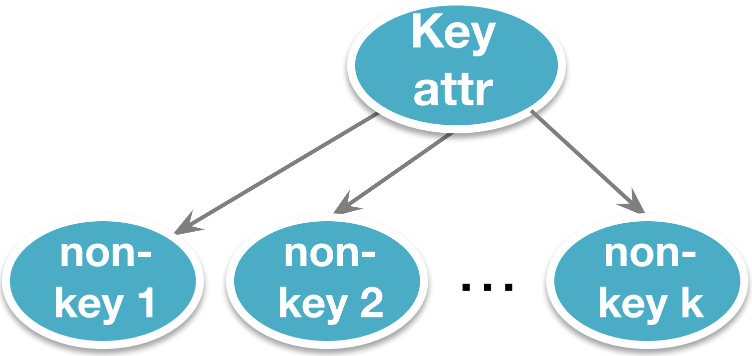
- Rel. R with FDs F is in BCNF if, for all $X \rightarrow A$ in F^+
  - $A \subseteq X$ (trivial FD), or
  - X is a super key

X:subset of attributes

A: single attribute

i.e., all non-trivial FDs over R are due to keys.

- No redundancy in R (at least none that FDs detect)
- Most often desired normal form



Question?

- Consider a relation in BCNF and FD: $X \rightarrow A$,
  Two tuples have the same X value
- Can the y values be different?

X	Y	A
x	y1	a
x	?	a

A. Yes

B. No

3NF

- Relation R with FDs F is in 3NF if, for all $X \rightarrow A$ in F^+
  - $A \subseteq X$ (trivial dependency) or
  - X is a super key or
  - A is part of some (minimal) key for R **(prime attribute)**

  Minimality of a key (i.e. a candidate key) is crucial!

X:subset of attributes

A: single attribute

- BCNF implies 3NF, but 3NF does not imply BCNF

3NF: Example

- e.g. Reserves(Sailor, Boat, Date, CreditCard)
  - SBD -> SBDC, S -> C (not 3NF)

    Why? SBD is the only key, S not a key, C not a key
  - If **additionally** C -> S, then CBD -> SBDC (i.e., CBD is also a key). → Now in 3NF!
  - Note redundancy in (S, C); 3NF permits this
  - Compromise used when BCNF not achievable, or performance considerations

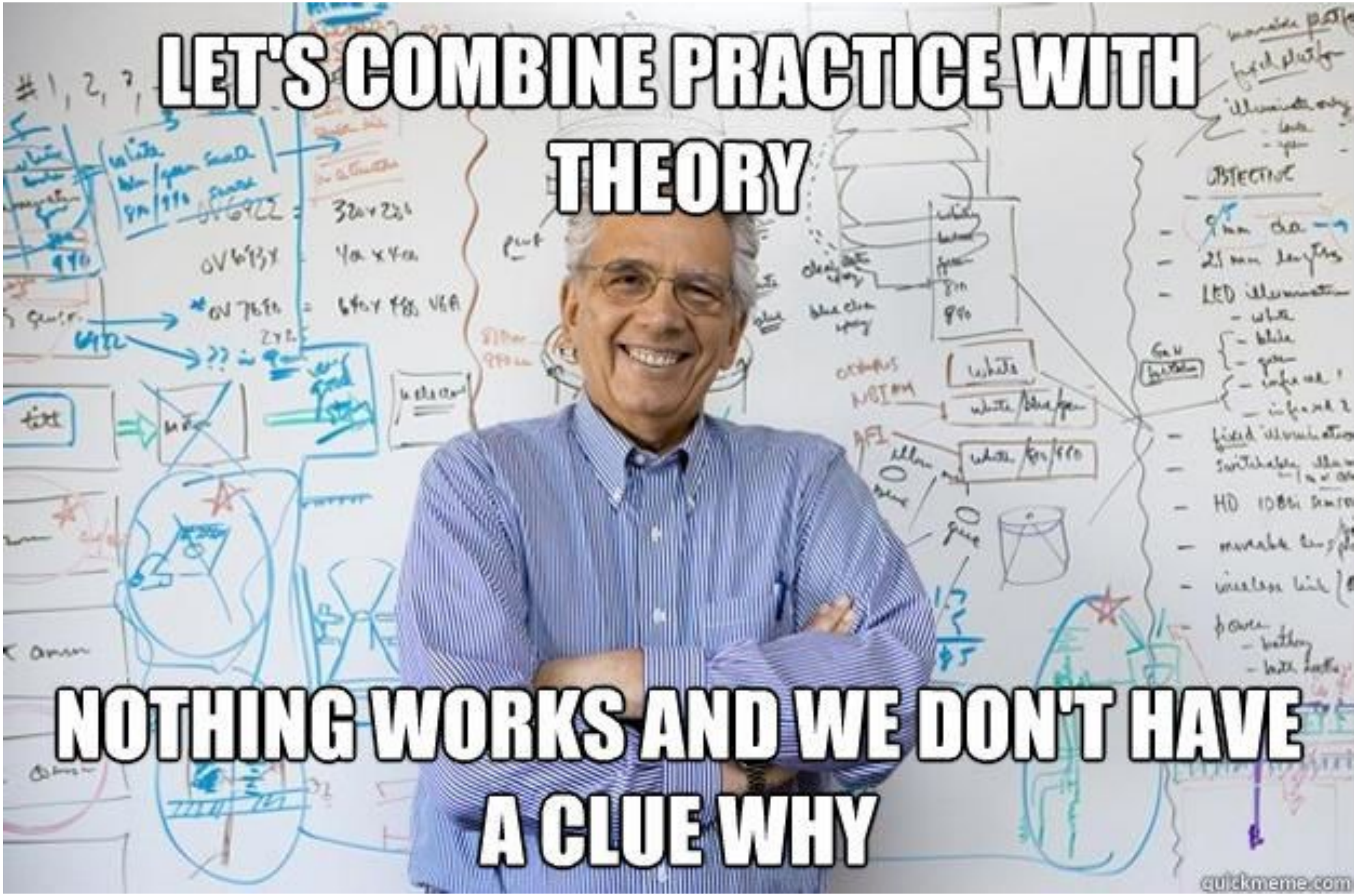
Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations is always possible.

Relation R with FDs F is in 3NF if, for all $X \rightarrow A$ in F^+

- $A \subseteq X$ (trivial dependency) or
- X is a super key or
- A is part of some (minimal) key for R **(prime attribute)**

Minimality of a key (i.e, not a super key) is crucial!

Time to practice ☺️



Exercise 1: BCNF or 3NF?

- Relation R=(A,B,C,D,E)
- FDs:

  $A \rightarrow BC$

  $CD \rightarrow E$

  $B \rightarrow D$

  $E \rightarrow A$

A is a (candidate) Key.
E is also a key.
CD is also a key
BC is also a key

- Is R in BCNF?
- Is R in 3NF?

Hint:
Use Armstrong's Axioms

- Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$ (trivial dependency)
- Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
- Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Exercise 1: BCNF or 3NF?

- **Keys:**
  - A, E, CD, BC
- **Is R in BCNF?**
  - No, because of B->D
- **Is R in 3NF?**
  - Yes

Keys found by repeatedly applying Armstrong's axioms

- Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$ (trivial dependency)
- Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
- Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Exercise 2: FDs & Normal Forms

Suppose you are given the following relation R: ABCDEF

$BC \rightarrow D$

$CD \rightarrow B$

$D \rightarrow E$

$ACD \rightarrow F$

1. Find the keys of R
2. List all of the above FDs that violate BCNF
3. List all of the above FDs that violate 3NF

Exercise 2: Solution

1. All possible keys: ACD, ABC

2. Violates BCNF:

   BC → D, CD → B, D → E

3. Violates 3NF

   D → E

== FDs: ==
BC → D
CD → B
D → E
ACD → F

Decomposition into BCNF

High-Level Algorithm

Input: a relation R with FDs F

1. Identify if any FDs violate BCNF (How?)
   - If X → Y violates BCNF, decompose R into **R - Y and XY**
2. Repeat for every X → Y that violates BCNF.

Output: a collection of relations that are in BCNF

- Does this algorithm provide a lossless join decomposition?
  - Yes! Notice that X is a key for the relation XY
- Several dependencies may cause violation of BCNF. The order in which we "deal with" them could lead to very different sets of relations!

Algorithm for BCNF (relation R, FDs F)

done = false;

result = {R};

compute F^+;

while (not done) do

    if $\exists\ R_i \in$ result and R_i is not in BCNF

      let $\alpha \rightarrow \beta$ be a nontrivial FD that holds in R_i

            such that $(\alpha \rightarrow R_i) \notin F^+$ and $\alpha \cap \beta = \emptyset$;

      result=(result$-R_i$) $\cup$ $(R_i - \beta)$ $\cup$ (α, β) ;

    else done=true ;

Exercise 3: Fix R to be in BCNF

- Relation R=(A,B,C,D,E)
- FDs: A $\rightarrow$ BC, CD $\rightarrow$ E, B $\rightarrow$ D, E $\rightarrow$ A
- Keys: A, BC, CD, E
- B $\rightarrow$ D violates BCNF
- Decompose R into:
  - R1=(A,B,C,E) R2=(B,D)
- Is this decomposition lossless join?
  - Yes! R1∩R2=B and B $\rightarrow$ R2
- Is it dependency preserving?
  - F1: A $\rightarrow$ BCE, E $\rightarrow$ A, BC -> AE
  - F2: B $\rightarrow$ D
  - No! CD $\rightarrow$ E is not in $(F1 \cup F2)^+$

In this case, leave it in 3NF

Exercise 4

Suppose you are given the following relation R: ABCDEF

- BC → D
- CD → B
- D → E
- ACD → F

Now decompose R into R1 and R2,
Do each of them satisfy lossless join property?

- R1: ACDF, R2: ABCDE
- R1: BCD, R2: ABEF

Exercise 4: Solution

- R1: ACDF, R2: ABCDE

  It satisfies lossless-join

  Attributes common: ACD - it is a key

- R1: BCD, R2: ABEF

  It does not satisfy lossless-join

  Attributes common: B - not a key

Schema Refinement

- Suppose you are given the following schema.
- IS (<u>item</u>, name, desc, loc, price)
  S (<u>name</u>, addr)

- You are asked to further normalize it assuming that supplier keeps all items of the same name in the same location. i.e., Add an FD:

  name → loc

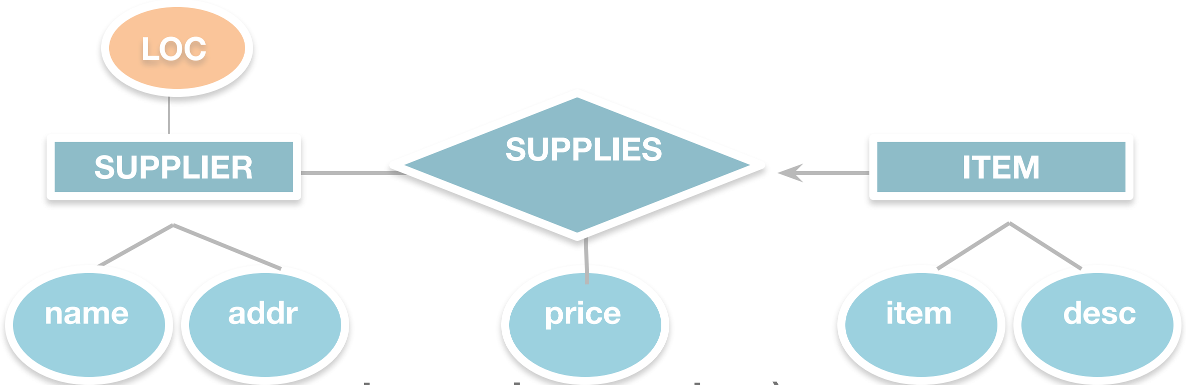
Schema Refinement

IS (<u>item</u>, name, desc, loc, price)
S (<u>name</u>, addr)

FDs = {i → ndlp, n → la}

- IS is not in BCNF, due to n → l
- Break it up: IS(<u>i</u>,n,d,p), Loc(<u>n</u>,l)
- S(<u>n</u>,a) remains unchanged
- Now notice same key for S and Loc, so merge
- Loc (<u>name</u>, addr, loc)

Schema Refinement



- IS (item, name, desc, loc, price)
  S (name, addr)

- A supplier keeps all items of the same name in the same location FD: name → loc

```
Solution:
IS (item, name, desc, price)
Loc (name, addr, loc)
```

Refined Schema

Normalization Summary

- Bad schemas lead to redundancy
  - Redundant storage, update, insert, and delete anomaly
- To "correct" bad schemas: decompose relations
  - Must be a lossless-join decomposition
  - Would like dependency preserving decompositions
- Desired Normal Forms
  - BCNF: allow only super-key functional dependencies
  - 3NF: allow dependencies with prime attributes on the RHS
    - Allows a limited form of redundancy
    - Trades off performance (avoid joins) for redundancy