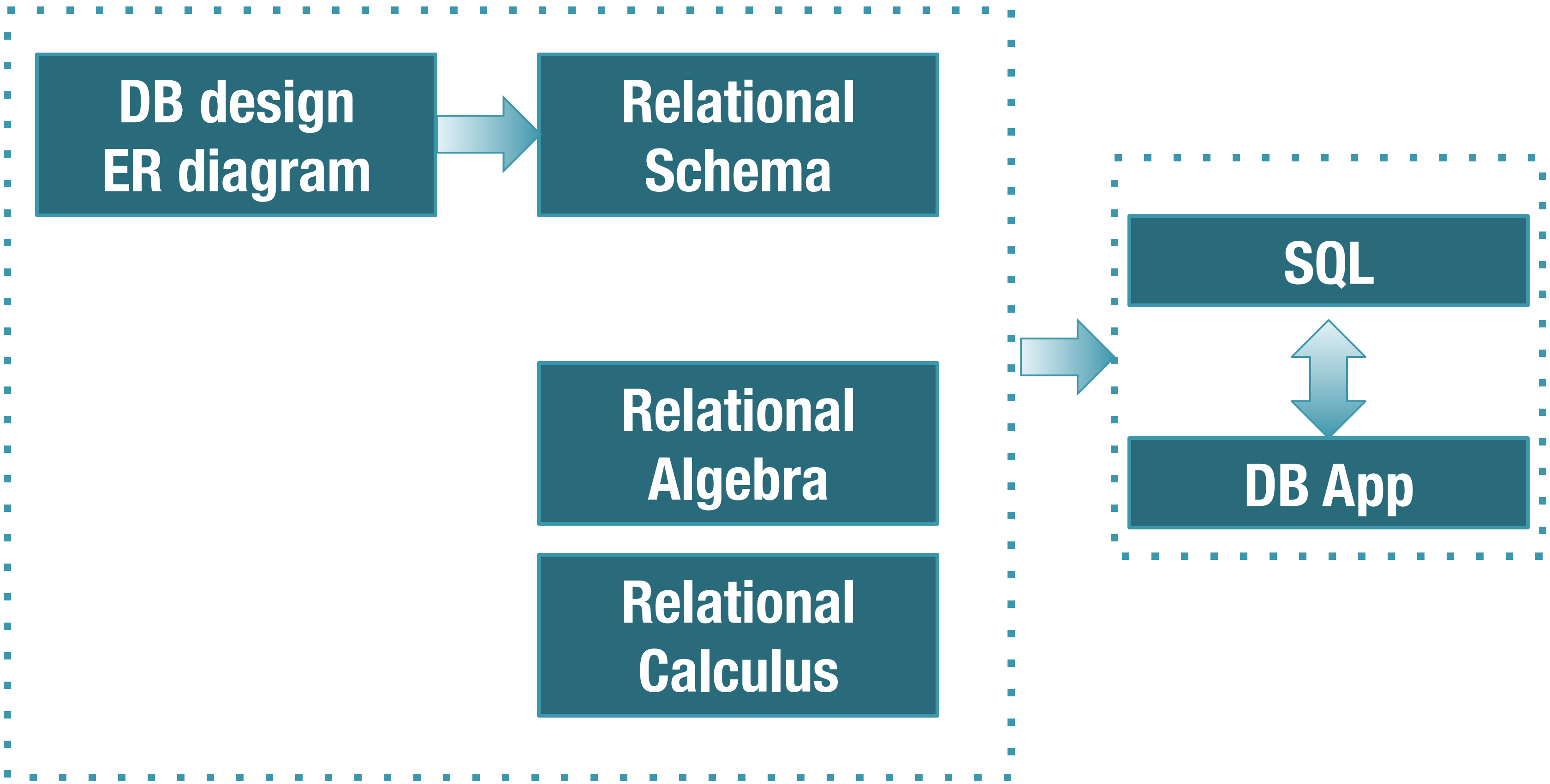
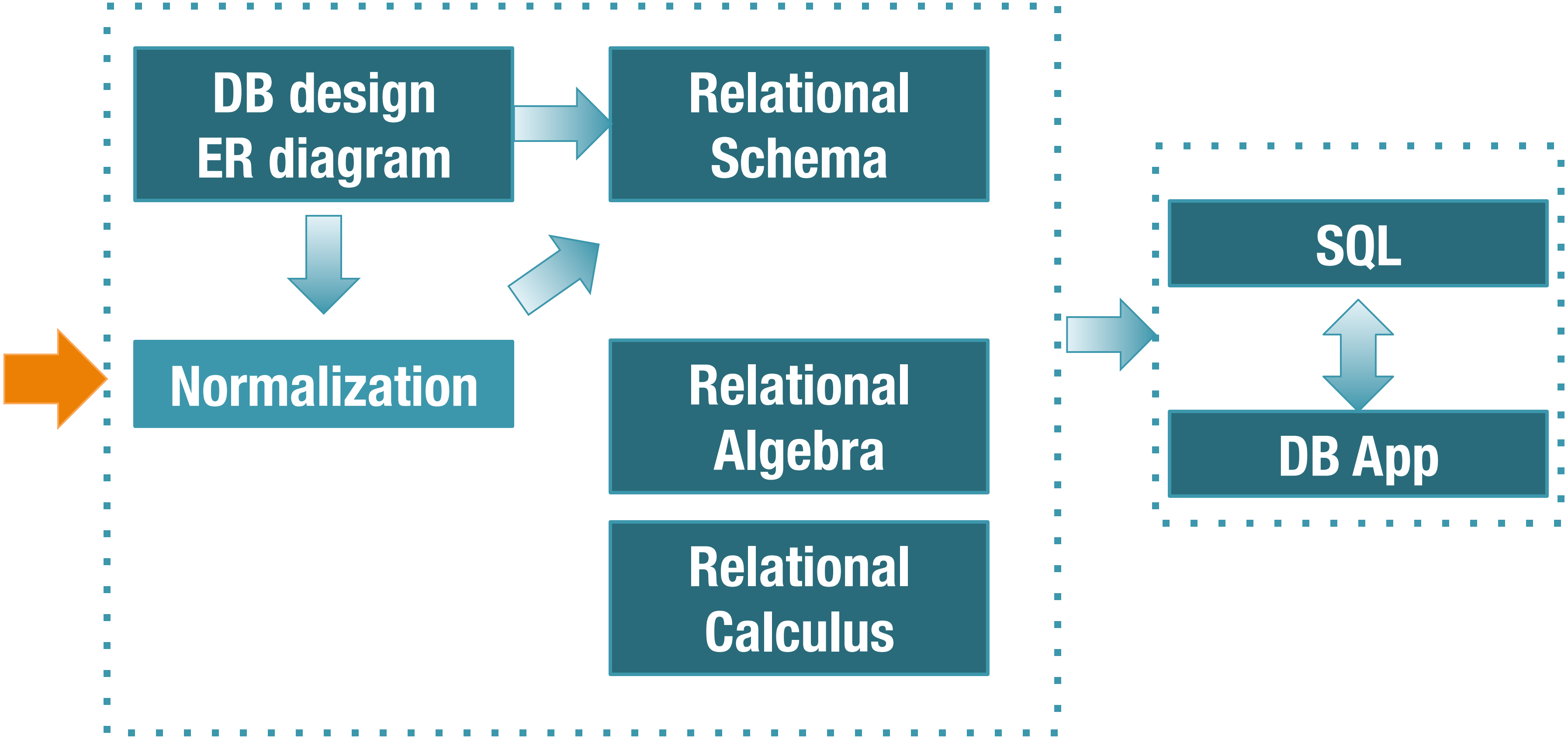
# Normalization

Chapter 19

# Review


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

# Today


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

# Database Design: The Story So Far

- Requirements Analysis
  - Data stored, operations, apps, …
- Conceptual Database Design
  - Model high-level description of the data, constraints, ER model
- Logical Database Design
  - Choose a DBMS and design a database schema
- Schema Refinement
  - Normalize relations, avoid redundancy, anomalies…
- Physical Database Design
  - Examine physical database structures like indices, restructure…
- Security Design

# 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 |

Good or bad 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 |

Problems

- (Supplier ID, item) appears to be the key, but Supplier ID is NULL in many places.
- Addresses can be multi-valued

# Normalization

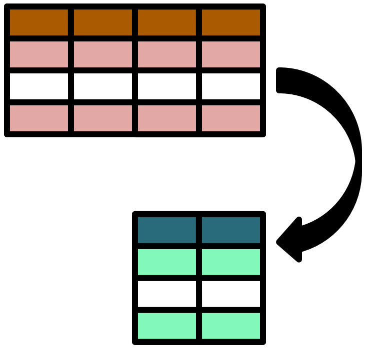
| Supplier ID | Supplier Name | Supplier Address | Item | Desc | Price |
|---|---|---|---|---|---|
| 1 | Acme | A1 | Dynamite | boom | $7 |
| | | | Paint | blue | $10 |
| | | | Flowers | pink | $3 |
| 2 | Beanery | A2, A3 | Dynamite | boom | $8 |

The above is not a good table!

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

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

# 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:

  Create an ER model and then map to tables

- Approach 2 [Today]:
  - State dependencies between attributes of tables
  - Map dependencies to tables. Can be done automatically!

# 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 | A2 | 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.

# Redundancy and Errors I

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

Address mismatch

# Redundancy and Errors II

(Remember, we are trying to do this without using an ER)

| 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 |

**Redundant storage**: For each different item, we also store the address of the supplier.

# Redundancy Problems

- Redundant storage (space)
  - A supplier supplies multiple items
- Update anomalies
  - Change address of a supplier
  - Need to change all instances!
- Insertion anomalies
  - Insert a supplier
  - Has to insert other (unrelated) info too
- Deletion anomalies
  - What if we want to delete the last item tuple?
  - Has to delete other (unrelated) info too

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

# Dealing with Redundancy

- Redundancy arises when schema forces an unnatural association among attributes
- The new trick we will learn today is to use the notion of functional dependencies

# Solution to Redundancy: Decomposition

- Split large relations to smaller ones

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

?

?

- Decomposition should be used judiciously:
  - Normal forms: guarantees against (some) redundancy
  - But, there can be a performance hit in going to smaller tables

# Functional Dependencies (FD)

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 | A2 | Flowers | pink | $3 |
| 2 | Beanery | A2 | Dynamite | boom | $8 |
| 2 | Beanery | A3 | Dynamite | boom | $8 |

- ‘Supplier Name’ depends on the ‘Supplier ID’
- What does ‘depends on’ mean?

# FD: Definition

- Notation: a → b
- Read as: ‘a’ functionally determines ‘b’
- **Informally**: If you know ‘a’, there is only one ‘b’ to match.

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

# FD: Definition

- Notation: a → b
- Read as: ‘a’ functionally determines ‘b’
- **Informally**: If you know ‘a’, there is only one ‘b’ to match.
- **Formally**: A form of Integrity Constraint

D: $X \rightarrow Y$ X and Y subsets of relation R’s attributes

$t1 \in r, \; t2 \in r, \quad \prod_X (t1) = \prod_X (t2) \Rightarrow \prod_y (t1) = \prod_y (t2)$

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

# 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 | A2 | Flowers | pink | $3 |
| 2 | Beanery | A2 | Dynamite | boom | $8 |
| 2 | Beanery | A3 | Dynamite | boom | $8 |

# 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 | A2 | 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

Other FDs?

# 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

- Role of FDs in detecting redundancy:

  Relation R with 4 attributes, XYZK

  $X \rightarrow Y$ is violated in this table

  (X,Y)=(1,1) or (1,2)

| X | Y | Z | K |
|---|---|---|---|
| 1 | 1 | 11 | A |
| 1 | 2 | 12 | A |
| 2 | 2 | 22 | A |
| 2 | 2 | 22 | B |

# Basic Normalization

- Write keys -> attribute mappings
- One table for each

| 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 |

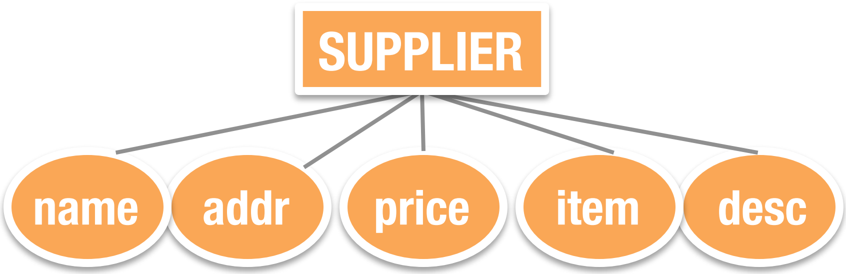
- Supplier ID → Supplier Name
- Item → Desc
- Supplier ID, Item → Price

| 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 |

# Example: Constraints on Entity Set





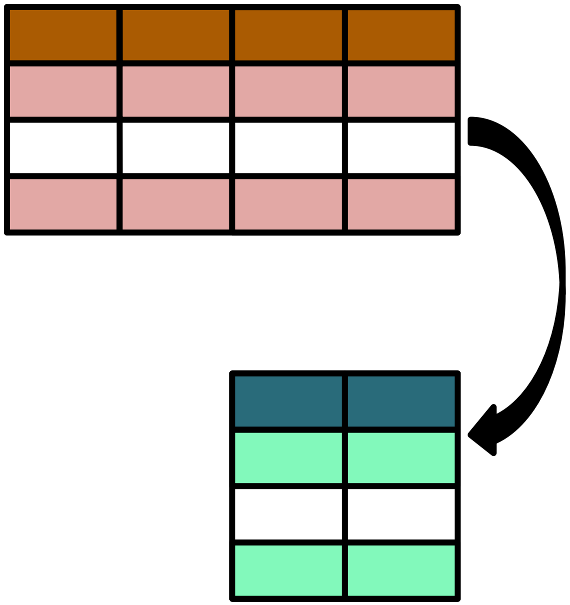
- S(name, item, desc, addr, price)
  - FD: {n,i} → {n,i,d,a,p}
  - FD: {n} → {a}
  - FD: {i} → {d}
- Decompose to: NA, ID, INP

- Spl(name, item, price)
  - FD: {n,i} → {n, i, p}
- Sup(name, addr)
  - FD: {n} → {n, a}
- Item (item, desc)
  - FD: {i} → {i, d}

ER design is subjective and can have many E + Rs
FDs: deeper understanding of schema

# Master goal

- Given relations
  - Supplier(sid, sname, …)
  - Item( iid, desc, …)
- And FD (sid → …, iid → …)
- WRITE CODE
- To automatically generate ‘good’ schemas

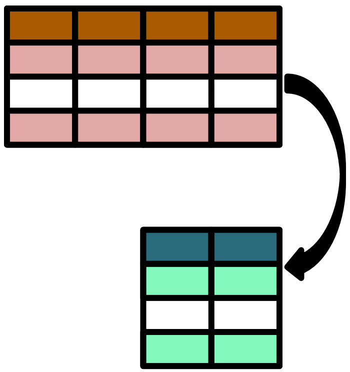


# How to find all the implied FDs?

- F+: Closure of F = Set of all FDs (including the derived ones)

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

- How to obtain?
  - F+ obtained by repeatedly applying Armstrong's Axioms



# Armstrong's Inference Axioms

- Axiom#1: Reflexive Property:
  - (Supplier ID, Item#) -> Item#
  - Obviously, if we know (Supplier ID and item#) pair, we know the item#
- In general, given attribute sets X and Y
  - Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$
- In the above example, Y is [Item#]. X is [Supplier ID, Item#].
- This is called a trivial dependency.

# All the Inference Axioms

- Armstrong's Axioms (X, Y, Z are sets of attributes):

  'sound' and 'complete'

  - 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 $\rightarrow$ ejob, ejob $\rightarrow$ esal; $\Rightarrow$ ename $\rightarrow$ 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

# Solution to Redundancy: Decomposition

- Split large relations to smaller ones

| 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 |

?

?

- Advantages:
  - Eliminate redundancy and anomalies
  - Lossless join when done right

# Solution to Redundancy: Decomposition

- Split large relations to smaller ones

| 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 |

?

?

- Problems with decomposition
  - Some queries become more expensive (more joins)
  - Lossless Join: Can we reconstruct the original relation from instances of the decomposed relations?

**Must have**

  - Dependency Preservation: Checking dependencies may require joining the instances of the decomposed relations.

**Good to have**

# Lossless Join Decompositions

- Relation R, FDs F; Decomposed to X, Y
- Lossless-Join decomposition if:

  $\pi_X(r) \bowtie \pi_Y(r) = r$ for **every** instance r of R
- Note, $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$ is always true, not vice versa, unless the join is lossless
- Can generalize to three or more relations

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

| A | B |
|---|---|
| 1 | 2 |
| 4 | 5 |
| 7 | 2 |

| B | C |
|---|---|
| 2 | 3 |
| 5 | 6 |
| 2 | 8 |

| 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

(Note: Different test needed for decomposition into more than two relations)

**Lossless join decomposition is always required!**

# Lossless Decomposition: Example

**R1**

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

ssn, c-id -> grade

**R2**

| 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

# 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}.
- The above is not dependency preserving

# Normal Forms

- Certain kind of decomposition
- 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

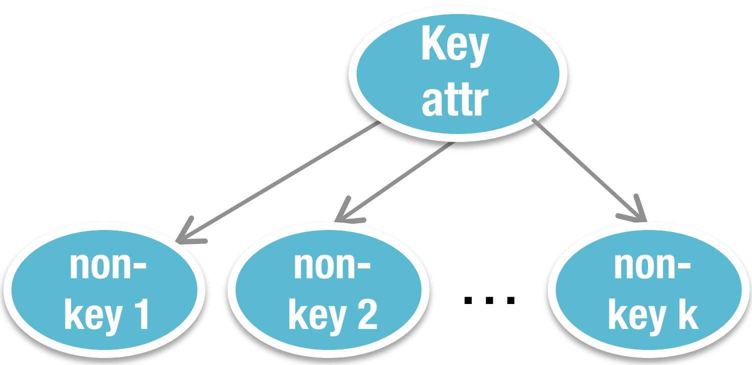
# Boyce-Codd Normal Form (BCNF)

- 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

  i.e. all non-trivial FDs over R are due to keys.
- No redundancy in R (at least none that FDs detect)
- Most desirable normal form

X:subset of attributes

A: single attribute



# 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

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

X:subset of attributes

A: single attribute

- No redundancy in R (at least none that FDs detect)
- Most desirable normal form
- Consider a relation in BCNF and FD: $X \rightarrow A$, two tuples have the same X value
  - Can the y values be different?
  - NO! non-trivial dependency
    - => X is a (super) key ⇒ the '?' must be y1

| X | Y | A |
|---|---|---|
| x | y1 | a |
| x | ? | a |

# 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. not a super 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

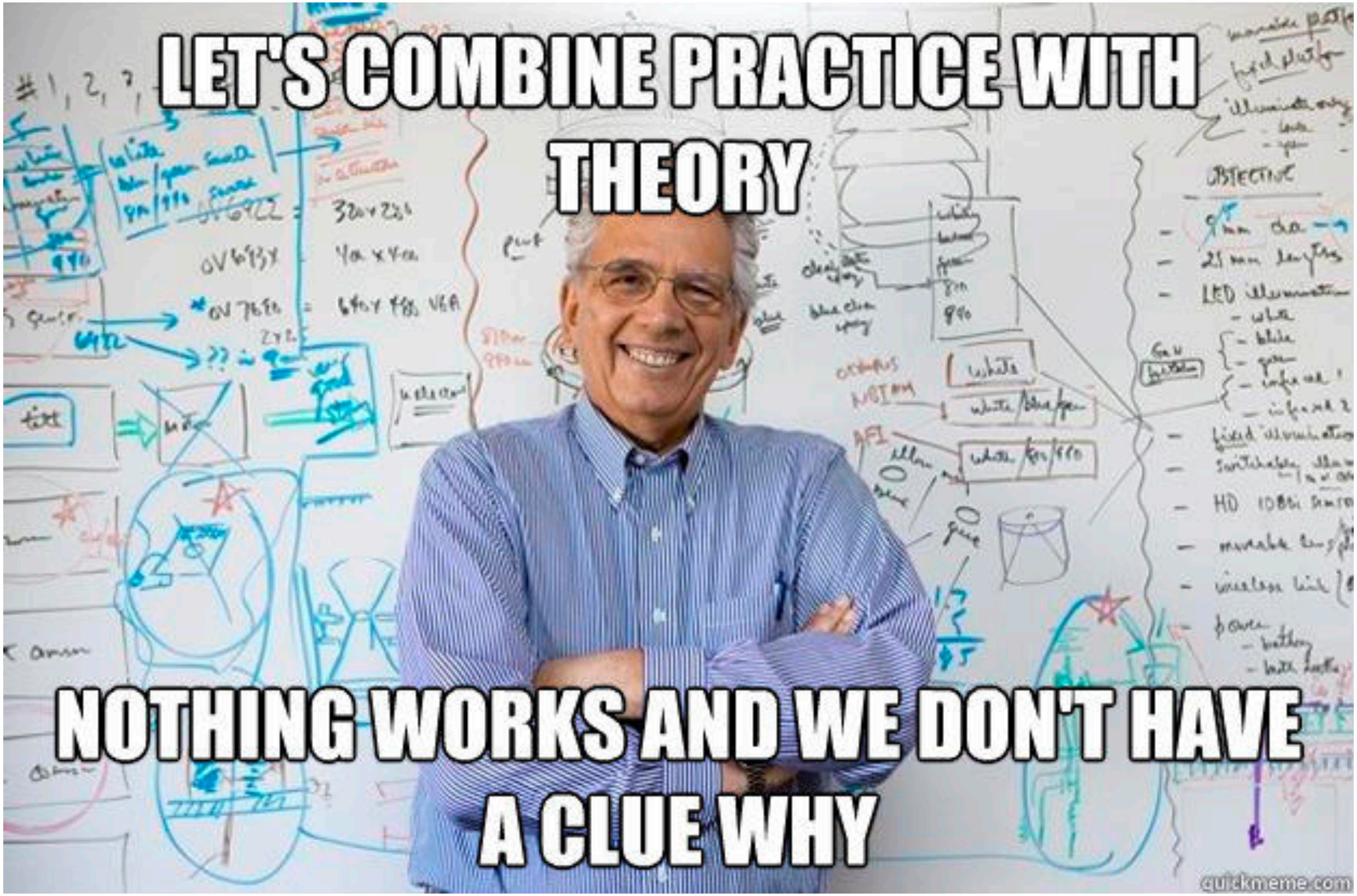
Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations is always possible.

Relation R with FDs F is in 3NF if, for all X → 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:

  $A \rightarrow BC$

  $CD \rightarrow E$

  $B \rightarrow D$

  $E \rightarrow A$

- 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 $X \rightarrow Z$

# 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. not a super 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!

# 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}.
- The above is not dependency preserving

# Exercise 2: FDs & Normal Forms

Suppose you are given the following relation R: ABCDEF

BC -> D

CD -> B

D -> E

ACD -> F

IMPORTANT

Optional after-class quiz
Submission: https://goo.gl/CeTFbm
Deadline: Thursday Feb 9 (noon)
*Counts only positively* (*"class" participation*)

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 $X \to Y$ violates BCNF, decompose R into **R - Y and XY**
2. Repeat for every $X \to Y$ 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 $\alpha \cap \beta = \emptyset$ ;

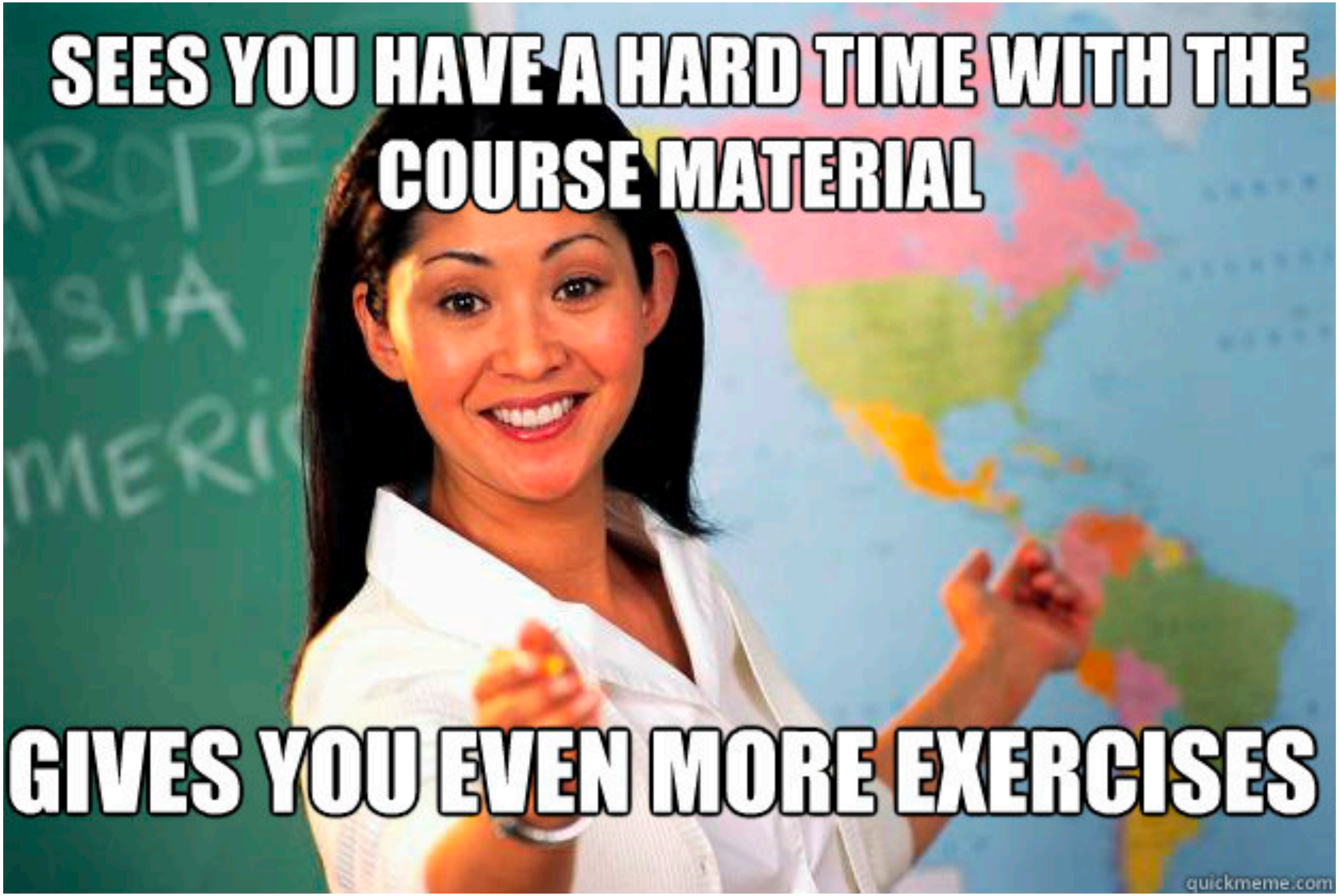
    result=(result-$R_i$) $\cup$ ($R_i$-$\beta$) $\cup$ ($\alpha$,$\beta$) ;

  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!  $R1 \cap R2 = B$ and $B \rightarrow R2$
- Is it dependency preserving?
  - F1: $A \rightarrow BC$, $E \rightarrow A$
  - F2: $B \rightarrow D$
  - No! $CD \rightarrow E$ is not in $(F1 \cup F2)^+$

In this case,
leave it in 3NF


SEES YOU HAVE A HARD TIME WITH THE COURSE MATERIAL
GIVES YOU EVEN MORE EXERCISES
quickmeme.com

# 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,
does each of them satisfy lossless join property?

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

# Exercise 4: Solution

- R1: ACDF, R2: ABCDE

  It is lossless

  Attributes common: ACD - it is a key

- R1: BCD, R2: ABEF

  It is not lossless

  Attributes common: B - not a key

# Refining an ER Diagram


- Suppose you are given the following schema. You are asked to normalize it:
- IS (item, name, desc, loc, price)
  S (name, addr)

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

# Normalization Example

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

**FDs = {i → ndlp, n → la}**


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

# Refining an ER Diagram


- 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)
```

New ER Diagram

# 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
  - 4NF: Use lossless decompositions for multi-valued dependencies