

## ↙ Note that

1. The best publication system will be selected to be used in Innopolis to manage publications here. Thus, you need to think about the usability and situations that fits here not only importing dblp records.
2. Due to the good and ontime feedbacks from the representative of BS#1-3, the final scores of all students in the group will plus 2 as a reward.
3. The representatives of other groups can visit me in 403 on this Thursday 1pm in the case that you would have collected feedbacks.
4. We will teach schema+sql/psm next week.

# Normal Forms



# Relational Schema Design

- ↙ Goal of relational schema design is to avoid anomalies and redundancy.

*Update anomaly*

*Deletion anomaly*

*Insert anomaly*

# Example of Bad Design

Drinkers(name, addr, beersLiked, manf, favBeer)

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	???	WickedAle	Pete's	???
Spock	Enterprise	Bud	???	Bud

Data is redundant, because each of the ???'s can be figured out by using the FD's **name -> addr favBeer** and **beersLiked -> manf**.



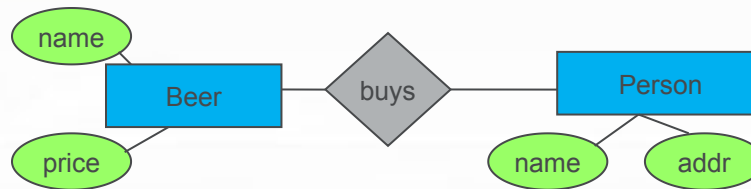
# This Bad Design also Exhibits Anomalies

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

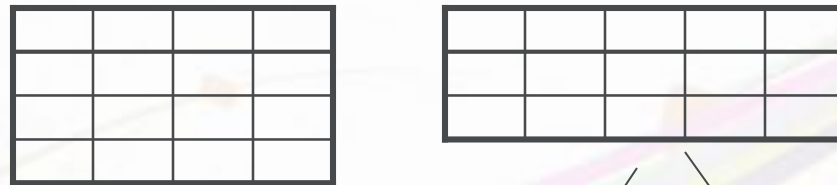
- **Update anomaly:** if Janeway is transferred to *another addr*, will we remember to change each of her tuples?
- **Deletion anomaly:** If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.
- **Insert anomaly:** if Artur is moved to Innopolis, nulls exist.

# Relational Schema Design

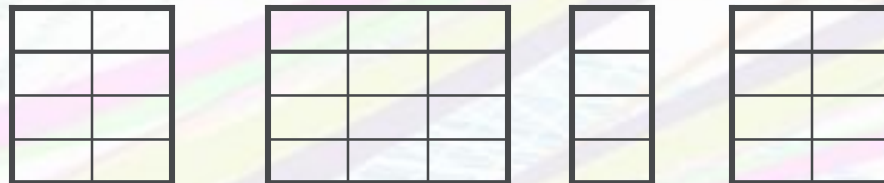
Conceptual Model:



Relational Model:  
(plus FD's)



Normalization:



# Decomposition of Relations

✦ Bad relations combine several concepts

decompose them so that each concept in one relation

$$R \rightarrow R_1, \dots, R_n$$

## 1. Lossless Decomposition

$$R = R_1 \bowtie R_2 \bowtie \dots \bowtie R_n$$

## 2. Preservation of Dependencies

- $FD(R)^+ = (FD(R_1) \cup \dots \cup FD(R_n))^+$

# First Normal Form

⚡ Only atomic domains (as in SQL 92)

<i>Parents</i>		
<i>Father</i>	<i>Mother</i>	<i>Children</i>
Johann	Martha	{Else, Lucie}
Johann	Maria	{Theo, Josef}
Heinz	Martha	{Cleo}

<i>Parents</i>		
<i>Father</i>	<i>Mother</i>	<i>Child</i>
Johann	Martha	Else
Johann	Martha	Lucie
Johann	Maria	Theo
Johann	Maria	Josef
Heinz	Martha	Cleo

vs.



# Second Normal Form

↙  $R$  is in 2NF iff every non-key attribute is minimally dependent on every key.

StudentAttends			
<u>StuID</u>	<u>CourseID</u>	Name	Semester
26120	5001	Fichte	10
27550	5001	Schopenhauer	6
27550	4052	Schopenhauer	6
28106	5041	Carnap	3
28106	5052	Carnap	3
28106	5216	Carnap	3
28106	5259	Carnap	3
...	...	...	...

↙ **StudentAttends is not in 2NF!!!**

$\{StuID\} \rightarrow \{Name, Semester\}$  (violation:  $X \rightarrow A$  in  $F^+$ ,  $A$  is non-key attribute,  $X$  is a subset of a key)

# Second Normal Form

## ✦ Solution: Decompose into two relations

`attends: {[studID, courseID]}`

`Student: {[stuID, Name, Semester]}`

## ✦ Student, attends are in 2NF. The decomposition is lossless and preserves dependencies.

## Generalization:

### ✦ Violation of 2NF

mixing an entity with an N:M (or 1:N) relationship

E.g., mixing Student (entity) with attends (N:M)

### ✦ Solution

Separate: entity and relationship

i.e., implement entity and relationship in separate relations

# Boyce-Codd Normal Form

- ✧ We say a relation  $R$  is in **BCNF** if whenever  $X \rightarrow Y$  is a nontrivial FD that holds in  $R$ ,  $X$  is a superkey.
- Remember: *nontrivial* means  $Y$  is not contained in  $X$ .
- Remember, a *superkey* is any superset of a key.

# Example

Drinkers(name, addr, beersLiked, manf, favBeer)

FD's: name->addr favBeer, beersLiked->manf

- ✚ Only key is {name, beersLiked}.
- ✚ In each FD, the left side is *not* a superkey.
- ✚ Any one of these FD's shows *Drinkers* is not in BCNF

# Another Example

Beers(name, manf, manfAddr)

FD's: name->manf, manf->manfAddr

⚡ Only key is {name} .

⚡ name->manf does not violate BCNF, but manf->manfAddr does.



# Decomposition into BCNF

- ↙ Given: relation  $R$  with FD's  $F$ .
- ↙ Look among the given FD's for a BCNF violation  $X \rightarrow Y$ .  
If any FD following from  $F$  violates BCNF, then there will surely be an FD in  $F$  itself that violates BCNF.
- ↙ Compute  $X^+$ .  
Not all attributes, or else  $X$  is a superkey.

# Decompose $R$ Using $X \rightarrow Y$

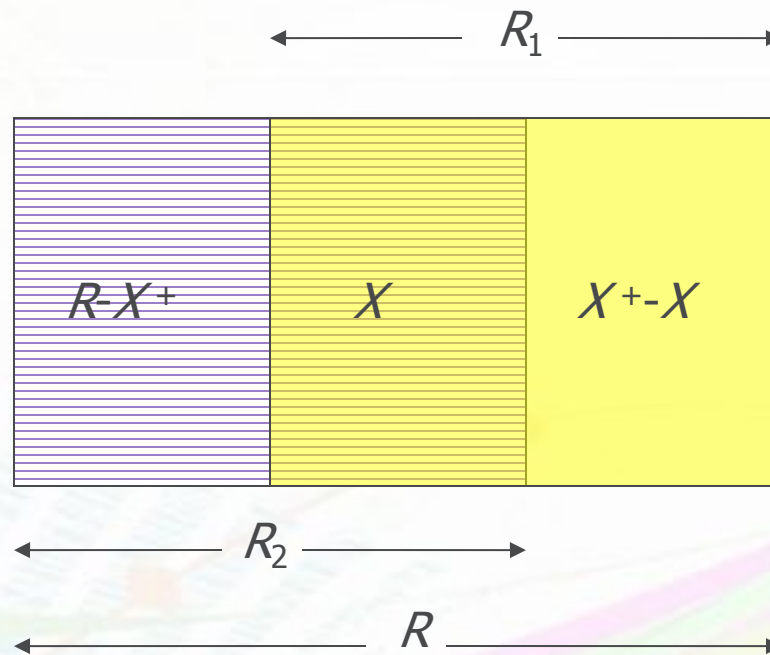
↙ Replace  $R$  by relations with schemas:

1.  $R_1 = X^+.$

2.  $R_2 = R - (X^+ - X).$

↙ *Project* given FD's  $F$  onto the two new relations.

# Decomposition Picture



# Example: BCNF Decomposition

Drinkers(name, addr, beersLiked, manf, favBeer)

$F = \text{name} \rightarrow \text{addr}, \text{name} \rightarrow \text{favBeer}, \text{beersLiked} \rightarrow \text{manf}$

- ✚ Pick BCNF violation  $\text{name} \rightarrow \text{addr}$ .
- ✚ Close the left side:  $\{\text{name}\}^+ = \{\text{name}, \text{addr}, \text{favBeer}\}$ .
- ✚ Decomposed relations:
  1. Drinkers1(name, addr, favBeer)
  2. Drinkers2(name, beersLiked, manf)

## Example -- Continued

- ✚ We are not done; we need to check Drinkers1 and Drinkers2 for BCNF.
- ✚ Projecting FD's is easy here.
- ✚ For **Drinkers1**(name, addr, favBeer), relevant FD's are **name->addr** and **name->favBeer**.  
Thus, {**name**} is the only key and Drinkers1 is in BCNF.



## Example -- Continued

- ↙ For `Drinkers2(name, beersLiked, manf)`, the only FD is `beersLiked->manf`, and the only key is `{name, beersLiked}`.

Violation of BCNF.

- ↙ `beersLiked+ = {beersLiked, manf}`, so we decompose *Drinkers2* into:
  1. `Drinkers3(beersLiked, manf)`
  2. `Drinkers4(name, beersLiked)`

## Example -- Concluded

✚ The resulting decomposition of *Drinkers* :

1. *Drinkers1*(name, addr, favBeer)
2. *Drinkers3*(beersLiked, manf)
3. *Drinkers4*(name, beersLiked)

✚ Notice: *Drinkers1* tells us about drinkers, *Drinkers3* tells us about beers, and *Drinkers4* tells us the relationship between drinkers and the beers they like.

# Third Normal Form -- Motivation

⚡ There is one structure of FD 's that causes trouble when we decompose.

⚡  $AB \rightarrow C$  and  $C \rightarrow B$ .

Example:  $A$  = street address,  $B$  = city,  
 $C$  = zip code.

⚡ There are two keys,  $\{A, B\}$  and  $\{A, C\}$ .

⚡  $C \rightarrow B$  is a BCNF violation, so we must decompose into  $AC$ ,  $BC$ .

# We Cannot Enforce FD's

- ↙ The problem is that if we use  $AC$  and  $BC$  as our database schema, we cannot enforce the FD  $AB \rightarrow C$  by checking FD 's in these decomposed relations.
- ↙ Example with  $A = \text{street}$ ,  $B = \text{city}$ , and  $C = \text{zip}$  on the next slide.

# An Unenforceable FD

street	zip
545 Tech Sq.	02138
545 Tech Sq.	02139

city	zip
Cambridge	02138
Cambridge	02139

Join tuples with equal zip codes.

street	city	zip
545 Tech Sq.	Cambridge	02138
545 Tech Sq.	Cambridge	02139

Although no FD's were violated in the decomposed relations,  
FD **street city -> zip** is violated by the database as a whole.



# 3NF Let's Us Avoid This Problem

- ↙ 3<sup>rd</sup> Normal Form (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation.
- ↙ An attribute is *prime* if it is a member of any key (key attribute).
- ↙  $X \rightarrow A$  violates 3NF if and only if  $X$  is not a superkey, and also  $A$  is not prime.

## Example: 3NF

- ↙ In our problem situation with FD's  $AB \rightarrow C$  and  $C \rightarrow B$ , we have keys  $AB$  and  $AC$ .
- ↙ Thus  $A$ ,  $B$ , and  $C$  are each prime.
- ↙ Although  $C \rightarrow B$  violates BCNF, it does not violate 3NF.

# What 3NF and BCNF Give You

- ⚡ There are two important properties of a decomposition:
  1. *Lossless Join* : it should be possible to project the original relations onto the decomposed schema, and then reconstruct the original.
  2. *Dependency Preservation* : it should be possible to check in the projected relations whether all the given FD' s are satisfied.

## 3NF and BCNF -- Continued

- ✚ We can get (1) with a BCNF decomposition.
- ✚ We can get both (1) and (2) with a 3NF decomposition.
- ✚ But we **can't always** get (1) and (2) with a BCNF decomposition.  
street-city-zip is an example.

# Testing for a Lossless Join

- ↙ If we project  $R$  onto  $R_1, R_2, \dots, R_k$ , can we recover  $R$  by rejoining?
- ↙ Any tuple in  $R$  can be recovered from its projected fragments.
- ↙ So the only question is: when we rejoin, do we ever get back something we didn't have originally?



# The Chase Test

- ↙ Suppose tuple  $t$  comes back in the join.
- ↙ Then  $t$  is the join of projections of some tuples of  $R$ , one for each  $R_i$  of the decomposition.
- ↙ Can we use the given FD's to show that one of these tuples must be  $t$ ?

## The Chase – (2)

- ↙ Start by assuming  $t = abc\dots$ .
- ↙ For each  $i$ , there is a tuple  $s_i$  of  $R$  that has  $a$ ,  $b$ ,  $c, \dots$  in the attributes of  $R_i$ .
- ↙  $s_i$  can have any values in other attributes.
- ↙ We'll use the same letter as in  $t$ , but with a subscript, for these components.

## Example: The Chase

- ✚ Let  $R = ABCD$ , and the decomposition be  $AB$ ,  $BC$ , and  $CD$ .
- ✚ Let the given FD's be  $C \rightarrow D$  and  $B \rightarrow A$ .
- ✚ Suppose the tuple  $t = abcd$  is the join of tuples projected onto  $AB$ ,  $BC$ ,  $CD$ .

The tuples  
of  $R$  projected onto  
 $AB, BC, CD$ .

# The *Tableau*

$A$	$B$	$C$	$D$
$a$	$b$	$c_1$	$d_1$
$a_2$	$b$	$c$	$d_2$
$a_3$	$b_3$	$c$	$d$

Use  $B \rightarrow A$

Use  $C \rightarrow D$

We've proved the  
second tuple must be  $t$ .

# Summary of the Chase

1. If two rows agree in the left side of a FD, make their right sides agree too.
2. Always replace a subscripted symbol by the corresponding unsubscripted one, if possible.
3. If we ever get an unsubscripted row, we know any tuple in the project-join is in the original (the join is lossless).
4. Otherwise, the final tableau is a counterexample.

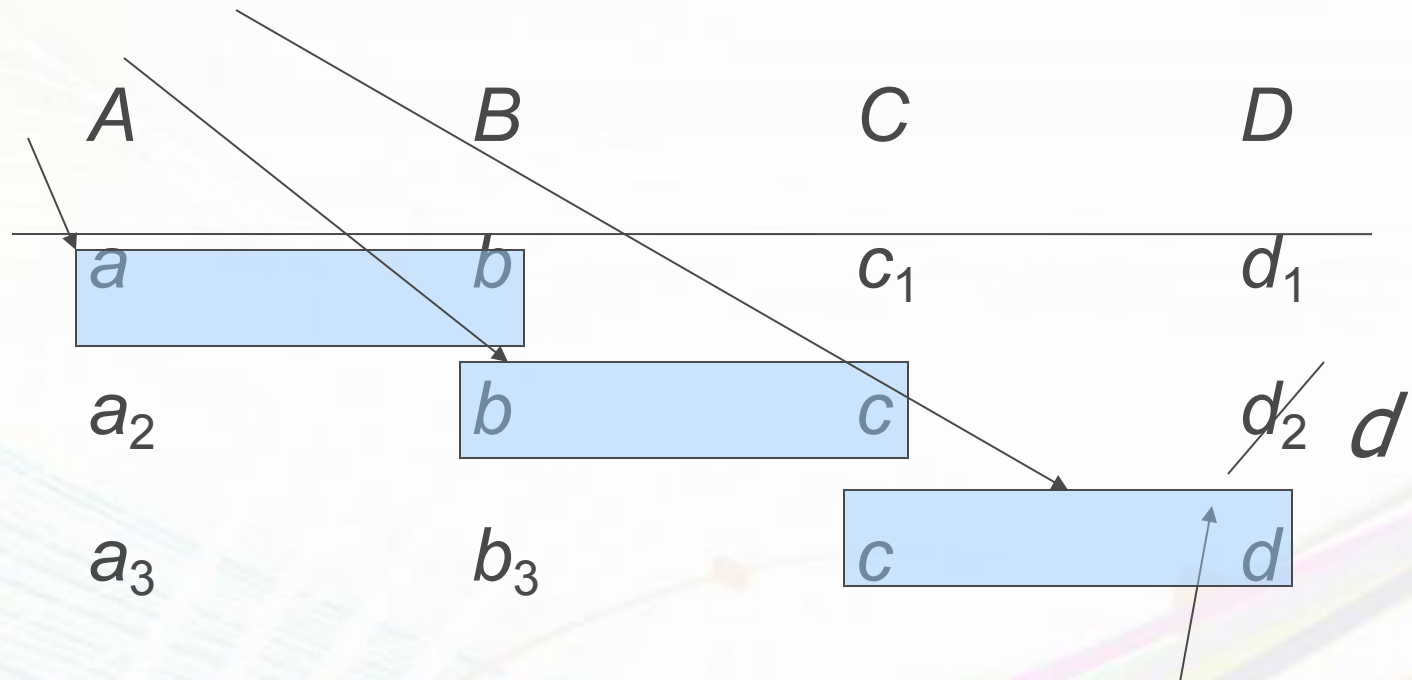
## Example: Lossy Join

- ↙ Same relation  $R = ABCD$  and same decomposition.
- ↙ But with only the FD  $C \rightarrow D$ .



These projections  
rejoin to form  
*abcd*.

## The *Tableau*



These three tuples are an example  
*R* that shows the join lossy. *abcd*  
is not in *R*, but we can project and  
rejoin to get *abcd*.

Use *C* > *D*

# 3NF Synthesis Algorithm

- ✚ We can always construct a decomposition into 3NF relations with a lossless join and dependency preservation.
- ✚ Need *minimal basis* for the FD's:
  1. Right sides are single attributes.
  2. No FD can be removed.
  3. No attribute can be removed from a left side.

# Constructing a Minimal Basis

1. Split right sides.
2. Repeatedly try to remove an FD and see if the remaining FD 's are equivalent to the original.
3. Repeatedly try to remove an attribute from a left side and see if the resulting FD 's are equivalent to the original.

# Computing the Minimum Basis

1. Reduction of left sides of FDs. Let  $\alpha \rightarrow \beta \in F$ ,  $A \in \alpha$ :  
if  $\beta \subseteq \text{Closure}(F, \alpha - A)$   
then replace  $\alpha \rightarrow \beta$  with  $(\alpha - A) \rightarrow \beta$  in  $F$
2. Reduction of right sides of FDs. Let  $\alpha \rightarrow \beta \in F$ ,  $B \in \beta$ :  
if  $B \in \text{Closure}(F - (\alpha \rightarrow \beta) \cup (\alpha \rightarrow (\beta - B)), \alpha)$   
then replace  $\alpha \rightarrow \beta$  with  $\alpha \rightarrow (\beta - B)$  in  $F$
1. Remove FDs:  $\alpha \rightarrow \emptyset$  (clean-up of Step 2)
2. Apply Union rule to FDs with the same left side.

## 3NF Synthesis – (2)

- ↙ One relation for each FD in the minimal basis.

Schema is the union of the left and right sides.

- ↙ If no key is contained in an FD, then add one relation whose schema is some key.

# Example: 3NF Synthesis

- ↙ Relation  $R = ABCD$ .
- ↙ FD's  $A \rightarrow B$  and  $A \rightarrow C$ .
- ↙ Decomposition:  $AB$  and  $AC$  from the FD's, plus  $AD$  for a key.



# Why It Works

- ↙ **Preserves dependencies:** each FD from a minimal basis is contained in a relation, thus preserved.
- ↙ **Lossless Join:** use the chase to show that the row for the relation that contains a key can be made all-unsubscripted variables.

# Summary

- ↙ If all attributes are key attributes, then the design at least meets 3NF (in 3NF, A must be a non-key attribute).
- ↙ If all keys are singleton, then the design at least meets 2NF (in 2NF, X must be a proper subset of a key).
- ↙ 3NF implies 2NF
- ↙ BCNF implies 3NF
- ↙ Preserving dependencies up to 3NF