Functional Dependencies, Schema Refinement, and Normalization for Relational Databases

CSC 375, Fall 2019

Chapter 19

> Science is the knowledge of consequences, and dependence of one fact upon another.
>
> *Thomas Hobbes (1588-1679)*

Review: Database Design

- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - high level descr (often done w/ER model)
- Logical Design
  - translate ER into DBMS data model
- Schema Refinement
  - **consistency, normalization**
- Physical Design - indexes, disk layout
- Security Design - who accesses what

Related Readings…

- **Check the following two papers on the course webpage**
  - *Decomposition of A Relation Scheme into Boyce-Codd Normal Form*, D-M. Tsou
  - *A Simple Guide to Five Normal Forms in Relational Database Theory*, W. Kent

Informal Design Guidelines for Relation Schemas

- **Measures of quality**
  - Making sure attribute semantics are clear
  - Reducing redundant information in tuples
  - Reducing NULL values in tuples
  - Disallowing possibility of generating spurious tuples

What is the Problem?

- **Consider relation obtained (call it SNLRHW)**

  `Hourly_Emps(ssn, name, lot, rating, hrly_wage, hrs_worked)`

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- **What if we *know* that `rating` determines `hrly_wage`?**

What is the Problem?

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- **Update anomaly**
  - Can we change **W** in just the 1st tuple of SNLRWH?

What is the Problem?

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- **Insertion anomaly:**
  - What if we want to insert an *employee* and don't know the *hourly wage* for his rating?

What is the Problem?

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

- **Deletion anomaly**
  - If we delete all employees with *rating* 5, we lose the information about the *wage* for rating 5!

What do we do?

- **When part of data can be derived from other parts, we say *redundancy* exists**
  - Example: the `hrly_wage` of Smiley can be derived from the `hrly_wage` of *Attishoo* because they have the same rating and we know rating determines `hrly_wage`.
- **Redundancy exists because of of the existence of integrity constraints (e.g., FD: R→W).**

What do we do?

- ***Redundancy* is at the root of several problems associated with relational schemas:**
  - redundant storage, insert/delete/update anomalies
- **Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.**
- **Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD).**
- **Decomposition should be used judiciously:**
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

Decomposing a Relation

- **Redundancy can be removed by "chopping" the relation into pieces.**
- **FD's (more about this one later) are used to drive this process.**

  R → W is causing the problems, so decompose SNLRWH into what relations?

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2

R	W
8	10
5	7

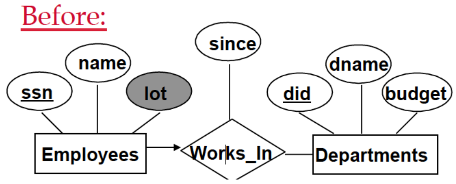
Wages

Refining an ER Diagram

- **1st diagram translated:**

  Employees(S,N,L,D,S2)
  Departments(D,M,B)

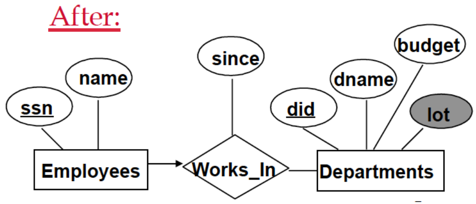
  Before:



  - Lots associated with employees
- **Suppose all employees in a dept are assigned the same lot: D → L**
- **Can fine-tune this way:**

  Employees2(S,N,D,S2)
  Departments(D,M,B,L)

  After:



Normalization

- **Normalization is the process of organizing the data into tables in such a way as to remove anomalies.**
  - Based on the observation that relations with certain properties are more effective in inserting, updating and deleting data than other sets of relations containing the same data
  - A multi-step process beginning with an "unnormalized" relation

Normal Forms

- **First Normal Form (1NF)**
- **Second Normal Form (2NF)**
- **Third Normal Form (3NF)**
- **Boyce-Codd Normal Form (BCNF)**
- **Fourth Normal Form (4NF)**
- **Fifth Normal Form (5NF)**

Recall

- **A key is a set of attributes that uniquely identifies each tuple in a relation.**
- **A candidate key is a key that is minimal.**
  - If AB is a candidate key, then neither A nor B is a key on its own.
- **A superkey is a key that is not necessarily minimal (although it could be)**
  - If AB is a candidate key then ABC, ABD, and even AB are superkeys.

Functional Dependencies (FDs)

- **Formal tool for analysis of relational schemas**
- **Enables us to detect and describe some of the above-mentioned problems in precise terms**

Functional Dependencies (FDs)

- **A functional dependency (FD) has the form: X→Y, where X and Y are two sets of attributes**
  - Examples: rating → hrly_wage, AB →C
- **The FD X→Y is satisfied by a relation instance r if:**
  - for each pair of tuples t1 and t2 in r:
    - t1.X = t2.X implies t1.Y = t2.Y
  - i.e., given any two tuples in r, if the X values agree, then the Y values must also agree. (X and Y are sets of attributes)
- **Convention: X, Y, Z etc denote sets of attributes, and A, B, C, etc denote attributes.**

In other Words…

- A functional dependency X → Y holds over relation schema R if, for every allowable instance *r* of R**:**

$$t1 \in r,\ t2 \in r,\ \prod_X(t1) = \prod_X(t2) \text{ implies } \prod_Y(t1) = \prod_Y(t2)$$

Given two tuples in *r*

if the *X* values agree

then the *Y* values must also agree

Y and *Y* are sets of attributes

- **Example: SSN → StudentNum**

FD's Continued

- **The FD *holds* over relation name R if, for every *allowable* instance r of R, r satisfies the FD.**
- **An FD, as an integrity constraint, is a statement about *all* allowable relation instances**
  - Given some instance r1 of R, we can check if it violates some FD *f* or not
  - But we cannot tell if *f* holds over R by looking at an instance!
    - Cannot prove non-existence (of violation) out of ignorance
  - This is the same for all integrity constraints!

FD's Continued

- **Functional dependencies are semantic properties of the underlying domain and data model**

- **FDs are NOT a property of a particular instance of the relation schema *R*!**
  - The *designer* is responsible for identifying FDs
  - FDs are *manually defined* integrity constraints on *R*
  - All extensions respecting *R*'s functional dependencies are called legal extensions of *R*

Example: Constraints on Entity Set

- **Consider relation obtained from Hourly_Emps:**
  - Hourly_Emps (*ssn*, *name*, *lot*, *rating*, *hrly_wages*, *hrs_worked*)
    S N L R W H
- ***Notation***: **We will denote this relation schema by listing the attributes: SNLRWH**
  - This is really the *set* of attributes {S,N,L,R,W,H}.
  - Sometimes, we will refer to all attributes of a relation by using the relation name (e.g., Hourly_Emps for SNLRWH)
- **Some FDs on Hourly_Emps:**
  - *ssn* is the key: S $\rightarrow$ SNLRWH
  - *rating* determines *hrly_wages*: R $\rightarrow$ W
  - *lot* determines *lot*: L $\rightarrow$ L ("trivial" dependency)

Detecting Reduncancy

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

Q: Why is R $\rightarrow$ W problematic, but S$\rightarrow$W not?

One More Example

A	B	C
1	1	2
1	1	3
2	1	3
2	1	2

FDs with A as the left side	Satisfied by the relation instance?
$A \rightarrow A$	Yes
$A \rightarrow B$	Yes
$A \rightarrow C$	No
$A \rightarrow AB$	Yes
$A \rightarrow AC$	No
$A \rightarrow BC$	No
$A \rightarrow ABC$	No

How many possible FDs on this relation instance?

Violation of FD by a relation

- **The FD $X \rightarrow Y$ is NOT satisfied by a relation instance *r* if:**
  - There exists a pair of tuples t1 and t2 in r such that:
    `t1.X = t2.X but t1.Y ≠ t2.Y`
  - i.e., we can find two tuples in *r*, such that X agree, but Y values don't.

Some Other FDs

A	B	C
1	1	2
1	1	3
2	1	3
2	1	2

FDs with A as the left side	Satisfied by the relation instance?
$C \rightarrow B$	Yes
$C \rightarrow AB$	No
$B \rightarrow C$	No
$B \rightarrow B$	Yes
$AC \rightarrow B$	Yes
…	…

Relationship between FDs and Keys

- **How are FD's related to keys?**
  - if "$K \rightarrow$ all attributes of R" then K is a superkey for R
    - Does not require K to be minimal.
- **Given R(A, B, C)**
  - $A \rightarrow ABC$ means that A is a key

What do we need to proceed?

- **A compact representation for sets of FD constraints**
- **No redundant FDs**
- **An algorithm to compute the set of all implied FDs**
- **Given some FDs, we can usually infer additional FDs:**
  - ssn $\rightarrow$ did, did $\rightarrow$ lot $\Rightarrow$ ssn $\rightarrow$ lot
  - A$\rightarrow$BC $\Rightarrow$ A$\rightarrow$B

Reasoning About FDs

- An FD *f* is *implied by* a set of FDs *F* if
  - *f* holds whenever all FDs in *F* hold.

- How can we find all implied FDs?
  - Closure of F, F+
- How can we find a minimal set of FDs that implies others?
  - Minimal Cover

- F^+ = *closure of F* is the set of all FDs that are implied by *F*. (includes "trivial dependencies")

- Fortunately, the closure of *F* can easily be computed using a small **set of inference rules**

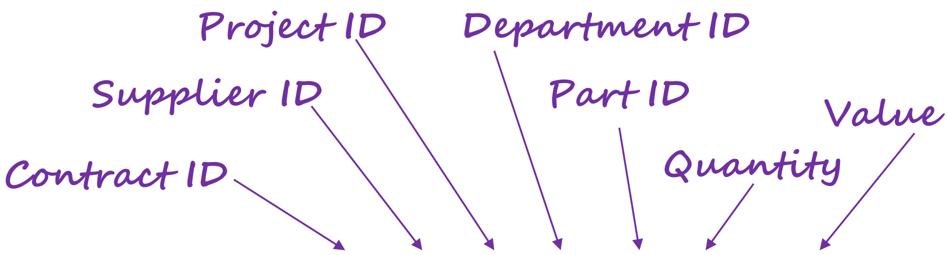
Rules of Inference

- Armstrong's Axioms (X, Y, Z are <u>sets</u> of attributes):
  - *Reflexivity*: If X $\supseteq$ Y, then X $\rightarrow$ Y
  - *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

- **These are *sound* and *complete* inference rules for FDs!**
  - i.e., using AA you can compute all the FDs in F+ and only these FDs.
    - *Completeness*: Every implied FD can be derived
    - *Soundness:* No non-implied FD can be derived

- **Some additional rules (that follow from AA):**
  - *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

Reasoning About FDs - Example



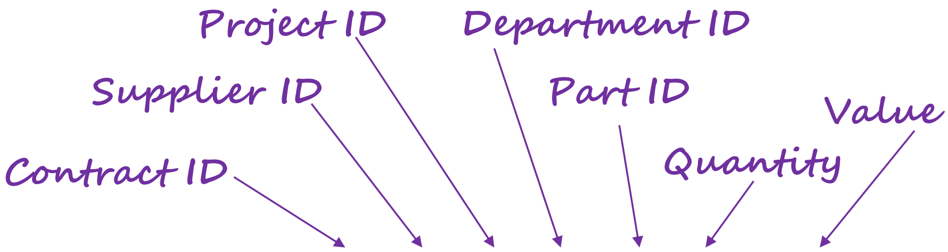
Example: Contracts(*cid,sid,jid,did,pid,qty,value*), and:

- C is the key: C $\longrightarrow$ CSJDPQV (C is a candidate key)
- Project purchases each part using single contract: JP $\longrightarrow$ C
- Dept purchases at most one part from a supplier: SD $\longrightarrow$ P

- **JP $\longrightarrow$ C, C $\longrightarrow$ CSJDPQV imply JP $\longrightarrow$ CSJDPQV**
- **SD $\longrightarrow$ P implies SDJ $\longrightarrow$ JP**
- **SDJ $\longrightarrow$ JP, JP $\longrightarrow$ CSJDPQV imply SDJ $\longrightarrow$ CSJDPQV**

These are also candidate keys

Reasoning About FDs - Example



Example: Contracts(*cid,sid,jid,did,pid,qty,value*), and:

- C is the key: C → CSJDPQV (C is a candidate key)
- Project purchases each part using single contract: JP → C
- Dept purchases at most one part from a supplier: SD → P

- **Since SDJ → CSJDPQV can we now infer that SD → CSDPQV (i.e., drop J on both sides)?**

No! FD inference is not like arithmetic multiplication.

Computing F+

- Recall that F^+ = *closure of F* is the set of all FDs that are implied by *F*. (includes "trivial dependencies")

- **In principle, we can compute the closure *F+* of a given set *F* of FDs by means of the following algorithm:**
  - Repeatedly apply the six inference rules until they stop producing new FDs.

- **In practice, this algorithm is hardly very efficient**
  - However, there usually is little need to compute the closure per se
  - Instead, it often suffices to compute a certain subset of the closure: namely, that subset consisting of all FDs with a certain left side

Example on Computing F+

- **F = {A → B, B → C, C D → E }**
- **Step 1: For each f in F, apply reflexivity rule**
  - We get: CD → C; CD → D
  - Add them to F:
    - F = {A → B, B → C, C D → E; CD → C; CD → D }
- **Step 2: For each f in F, apply augmentation rule**
  - From A → B we get: A → AB; AB → B; AC → BC; AD → BD; ABC →BC; ABD → BD; ACD →BCD
  - From B → C we get: AB → AC; BC → C; BD → CD; ABC → AC; ABD → ACD, etc etc.
  - Step 3: Apply transitivity on pairs of f's
  - Keep repeating... You get the idea

Attribute Closure

- Size of F^+ is exponential in # attributes in R; can be expensive.

- If we just want to check if a given FD $X \rightarrow Y$ is in F^+, then:
- Compute the *attribute closure* of X (denoted X^+) wrt F
  - X^+ = Set of all attributes A such that X → A is in F^+
    - initialize $X^+ := X$
    - Repeat until no change:
      if U → V in F such that U is in X^+, then add V to X^+
  - Check if Y is in X^+

- Can also be used to find the keys of a relation.
    - If all attributes of R are in the closure of X then X is a superkey for R.
    - Q: How to check if X is a "candidate key"?

Attribute Closure

- The following algorithm computes (*X*, *F*)+:

```
Input F (a set of FDs), and X (a set of attributes)
• Output: Result = X+(under F)
• Method:
   • Step 1: Result := X;
   • Step 2: Take Y → Z in F, and Y is in Result, do:
      • Result := Result ∪ Z
• Repeat step 2 until Result cannot be changed and then
  output Result
```


Attribute Closure (example)

- **R = {A, B, C, D, E}**
- **F = { B →CD, D → E, B → A, E → C, AD →B }**
- **Is B → E in F$^+$?**

  $B^+ = B$

  $B^+ = BCD$

  $B^+ = BCDA$

  $B^+ = BCDAE$ ... Yes!

  and B is a key for R too!

- **Is D a key for R?**

  $D^+ = D$

  $D^+ = DE$

  $D^+ = DEC$

  ... Nope!

Reflexivity: If $Y \supseteq X$, then $X \rightarrow Y$
Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Attribute Closure (example)

- **Does F = {A → B, B → C, C D → E } imply A → E?**
  - i.e, is A→E in the closure F+? Equivalently, is E in A+?
- **Does F = {A → B, B → C, C D → E } imply A → E?**
  - i.e, is A→E in the closure F+? Equivalently, is E in A+?
- **Step 1: Result = A**
- **Step 2: Consider A → B, Result = AB**
  - Consider B → C, Result = ABC
  - Consider CD → E, CD is not in ABC, so stop
- **Step 3: A+= {ABC}**
  - E is NOT in A+, so A → E is NOT in F+

Attribute Closure (example)

- **F = {A →B, AC →D, AB →C}?**
- **What is X+ for X = A? (i.e. what is the attribute closure for A?)**
- **Answer: A+= ABCD**

Attribute Closure (example)

- **R = (A, B, C, G, H, I)**
- **F = {A → B; A → C; CG → H; CG → I; B → H}**
- **(AG)+= ?**
  - Answer: ABCGHI
- **Is AG a candidate key?**
  - This question involves two parts:
    - 1. Is AG a super key?
      - Does AG → R?
    - 2. Is any subset of AG a superkey?
      - Does A → R?
      - Does G → R?

Uses of Attribute Closure

- **There are several uses of the attribute closure algorithm:**
  - Testing for *superkey*:
    - To test if X is a superkey, we compute X+, and check if X+ contains all attributes of R.
- **Testing functional dependencies**
  - To check if a functional dependency X → Y holds (or, in other words, is in F+), just check if Y ⊆ X+.
  - That is, we compute X+ by using attribute closure, and then check if it contains Y.
  - Is a simple and cheap test, and very useful
- **Computing closure of F**

Thanks for that…

- So we know a lot about FDs
- We could care less, right?

Normal Forms

- **Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!**

  Definition. Denormalization is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.

- **Normalization results with high quality designs that meet the desirable properties stated previously**
  - Pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF
  - If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized.
- **Do not need to normalize to the highest possible normal form**
- **Used to help us decide whether decomposing the relation will help.**

Normal Forms

- **Role of FDs in detecting redundancy:**
  - Consider a relation *R* with 3 attributes, *ABC*.
    - Given A ⟶ B: Several tuples could have the same A value, and if so, they'll all have the same B value - redundancy!
    - No FDs hold: There is no redundancy here
    - **Note**: A ⟶ B potentially causes problems. However, if we know that no two tuples share the same value for A, then such problems cannot occur (a normal form)

Normal Forms

- **First normal form (1NF)**
  - Every field must contain atomic values, i.e. no sets or lists.
  - Essentially all relations are in this normal form
- **Second normal form (2NF)**
  - Any relation in 2NF is also in 1NF
  - All the non-key attributes must depend upon the WHOLE of the candidate key rather than just a part of it.
- **Boyce-Codd Normal Form (BCNF)**
  - Any relation in BCNF is also in 2NF
- **Third normal form (3NF)**
  - Any relation in BCNF is also in 3NF

First Normal Form

DEPARTMENT

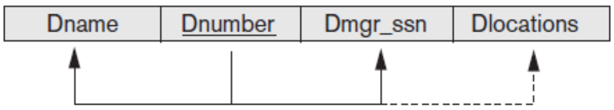
Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

To move to First Normal Form a relation must contain only atomic values at each row and column.

First Normal Form

(a)

DEPARTMENT



(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 15.9
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

First Normal Form

- **Does not allow nested relations**
  - Each tuple can have a relation within it
- **To change to 1NF, multi-valued attributes must be normalized, e.g., by**
  - A) Introducing a new relation for the multi-valued attribute
  - B) Replicating the tuple for each multi-value
  - C) introducing an own attribute for each multi-value (if there is a small maximum number of values)
- **Solution A is usually considered the best**

Second Normal Form

- **Second normal form (2NF)**
  - Any relation in 2NF is also in 1NF
  - All the non-key attributes must depend upon the WHOLE of the candidate key rather than just a part of it.
    - It is only relevant when the key is composite, i.e., consists of several fields.
  - e.g. Consider a relation:
    `Inventory(part, warehouse, quantity, warehouse_address)`
    - Suppose {part, warehouse} is a candidate key.
    - `warehouse_address` depends upon warehouse alone - 2NF violation
    - Solution: decompose

Unnormalized Relation

Patient #	Surgeon #	Surg. date	Patient Name	Patient Addr	Surgeon	Surgery	Postop drug	Drug side effects
1111	145 311	Jan 1, 1995; June 12, 1995	John White	15 New St. New York, NY	Beth Little Michael Diamond	Gallstones removal; Kidney stones removal	Penicillin, none-	rash none
1234	243 467	Apr 5, 1994 May 10, 1995	Mary Jones	10 Main St. Rye, NY	Charles Field Patricia Gold	Eye Cataract removal Thrombosis removal	Tetracycline none	Fever none
2345	189	Jan 8, 1996	Charles Brown	Dogwood Lane Harrison, NY	David Rosen	Open Heart Surgery	Cephalosporin	none
4876	145	Nov 5, 1995	Hal Kane	55 Boston Post Road, Chester, CN	Beth Little	Cholecystectomy	Demicillin	none
5123	145	May 10, 1995	Paul Kosher	Blind Brook Mamaroneck, NY	Beth Little	Gallstones Removal	none	none
6845	243	Apr 5, 1994 Dec 15, 1984	Ann Hood	Hilton Road Larchmont, NY	Charles Field	Eye Cornea Replacement Eye cataract removal	Tetracycline	Fever

First Normal Form

Patient #	Surgeon #	Surgery Date	Patient Name	Patient Addr	Surgeon Name	Surgery	Drug admin	Side Effects
1111	145	01-Jan-95	John White	15 New St. New York, NY	Beth Little	Gallstones removal	Penicillin	rash
1111	311	12-Jun-95	John White	15 New St. New York, NY	Michael Diamond	Kidney stones removal	none	none
1234	243	05-Apr-94	Mary Jones	10 Main St. Rye, NY	Charles Field	Eye Cataract removal	Tetracycline	Fever
1234	467	10-May-95	Mary Jones	10 Main St. Rye, NY	Patricia Gold	Thrombosis removal	none	none
2345	189	08-Jan-96	Charles Brown	Dogwood Lane Harrison, NY	David Rosen	Open Heart Surgery	Cephalosporin	none
4876	145	05-Nov-95	Hal Kane	55 Boston Post Road, Chester, CN	Beth Little	Cholecystectomy	Demicillin	none
5123	145	10-May-95	Paul Kosher	Blind Brook Mamaroneck, NY	Beth Little	Gallstones Removal	none	none
6845	243	05-Apr-94	Ann Hood	Hilton Road Larchmont, NY	Charles Field	Eye Cornea Replacement	Tetracycline	Fever
6845	243	15-Dec-84	Ann Hood	Hilton Road Larchmont, NY	Charles Field	Eye cataract removal	none	none

Second Normal Form

Patient #	Surgeon #	Surgery Date	Surgery	Drug Admin	Side Effects
1111	145	01-Jan-95	Gallstones removal	Penicillin	rash
1111	311	12-Jun-95	stones removal	none	none
1234	243	05-Apr-94	Eye Cataract removal	Tetracycline	Fever
1234	467	10-May-95	Thrombosis removal	none	none
2345	189	08-Jan-96	Open Heart Surgery	Cephalospori n	none
4876	145	05-Nov-95	Cholecystect omy	Demicillin	none
5123	145	10-May-95	Gallstones Removal	none	none
6845	243	15-Dec-84	Eye cataract removal	none	none
6845	243	05-Apr-94	Eye Cornea Replacement	Tetracycline	Fever

Third Normal Form

- **A relation is said to be in *Third Normal Form* if there is no transitive functional dependency between *nonkey* attributes**
  - When one nonkey attribute can be determined with one or more nonkey attributes there is said to be a transitive functional dependency.
- **The side effect column in the Surgery table is determined by the drug administered**
  - Side effect is transitively functionally dependent on drug so Surgery is not 3NF

Third Normal Form

Patient #	Surgeon #	Surgery Date	Surgery	Drug Admin
1111	145	01-Jan-95	Gallstones removal	Penicillin
1111	311	12-Jun-95	Kidney stones removal	none
1234	243	05-Apr-94	Eye Cataract removal	Tetracycline
1234	467	10-May-95	Thrombosis removal	none
2345	189	08-Jan-96	Open Heart Surgery	Cephalosporin
4876	145	05-Nov-95	Cholecystectomy	Demicillin
5123	145	10-May-95	Gallstones Removal	none
6845	243	15-Dec-84	Eye cataract removal	none
6845	243	05-Apr-94	Eye Cornea Replacement	Tetracycline

Normal Forms

- Question: is any refinement needed??!

- If a relation is in a *normal form* (BCNF, 3NF etc.):
  - we know that certain problems are avoided/minimized.
  - helps decide whether decomposing a relation is useful.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No (non-trivial) FDs hold: There is no redundancy here.
    - Given $A \rightarrow B$: If A is not a key, then several tuples could have the same A value, and if so, they'll all have the same B value!

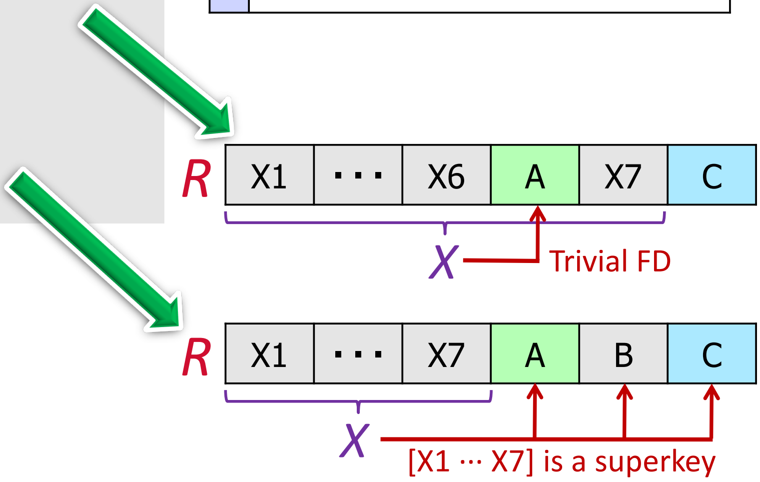
- 1st Normal Form – all attributes are atomic (i.e., "flat tables")
- 1st $\supset$ 2nd (of historical interest) $\supset$ 3rd $\supset$ Boyce-Codd $\supset$ …

Boyce-Codd Normal Form (BCNF)

Relation *R* is in BCNF if, for all *X* → *A* in *F*,

- $A \in X$ (called a *trivial* FD), or
- *X* is a superkey (i.e., contains a key of *R*)

R	A relation
F	The set of FD hold over *R*
X	A subset of the attributes of *R*
A	An attribute of *R*



BCNF is Desirable

Consider the relation:

X	Y	A
x	y1	y2
x	y2	?

→A

Can you guess?

Should be y2

Not in BCNF

"X →A" ⇒ The 2nd tuple also has y2 in the third column

⇒ an example of redundancy

Such a situation cannot arise in a BCNF relation:

BCNF ⇒ X must be a key

⇒ we must have X→Y

⇒ we must have "y1 = y2" (1)

X→A ⇒ The two tuples have the same value for A (2)

(1) & (2) ⇒ The two tuples are identical

⇒ This situation cannot happen in a relation

Boyce-Codd Normal Form (BCNF)

- In other words, if you can guess the value of the missing attribute then the relation is not in BCNF

X	Y	A
x	y1	a
x	y2	?

BCNF: Desirable Property

A relation is in BCNF

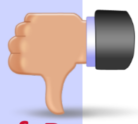
⇒ every entry records a piece of information that cannot be inferred (using only FDs) from the other entries in the relation instance

⇒ No redundant information !

Key constraint is the only form of FDs allowed in BCNF

A relation *R*(*ABC*)

- ***B*→*C***: The value of *B* determines *C*, and the value of *C* can be inferred from another tuple with the same *B* value
  ⇒ redundancy ! (not BCNF)



- ***A*→*BC***: Although the value of *A* determines the values of *B* and *C*, we cannot infer their values from other tuples because no two tuples in *R* have the same value for *A*
  ⇒ no redundancy ! (BCNF)

Boyce-Codd Normal Form

- **Most 3NF relations are also BCNF relations.**
- **A 3NF relation is NOT in BCNF if:**
  - Candidate keys in the relation are composite keys (they are not single attributes)
  - There is more than one candidate key in the relation, and
  - The keys are not disjoint, that is, some attributes in the keys are common

Boyce-Codd Normal Form - Alternative Formulation

"The key, the whole key, and nothing but the key, so help me Codd."

Most 3NF Relations are also BCNF – Is this one?

Patient #	Patient Name	Patient Address
1111	John White	15 New St. New York, NY
1234	Mary Jones	10 Main St. Rye, NY
2345	Charles Brown	Dogwood Lane Harrison, NY
4876	Hal Kane	55 Boston Post Road, Chester,
5123	Paul Kosher	Blind Brook Mamaroneck, NY
6845	Ann Hood	Hilton Road Larchmont, NY

BCNF Relations

Patient #	Patient Name
1111	John White
1234	Mary Jones
2345	Charles Brown
4876	Hal Kane
5123	Paul Kosher
6845	Ann Hood

Patient #	Patient Address
1111	15 New St. New York, NY
1234	10 Main St. Rye, NY
2345	Dogwood Lane Harrison, NY
4876	55 Boston Post Road, Chester,
5123	Blind Brook Mamaroneck, NY
6845	Hilton Road Larchmont, NY

Decomposition of a Relation Scheme

- If a relation is not in a desired normal form, it can be *decomposed* into multiple relations that each are in that normal form.

- Suppose that relation R contains attributes $A_1 \ldots A_n$. A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R, and
  - Every attribute of R appears as an attribute of at least one of the new relations.

Example

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

- SNLRWH has FDs S → SNLRWH and R → W
- Q: Is this relation in BCNF?

No, The second FD causes a violation;
W values repeatedly associated with R values.

Decomposing a Relation

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2

R	W
8	10
5	7

Wages

•Q: Are both of these relations now in BCNF?

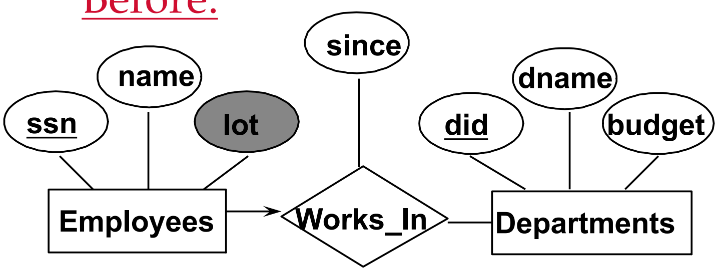
•**Decompositions should be used only when needed.**

–Q: potential problems of decomposition?

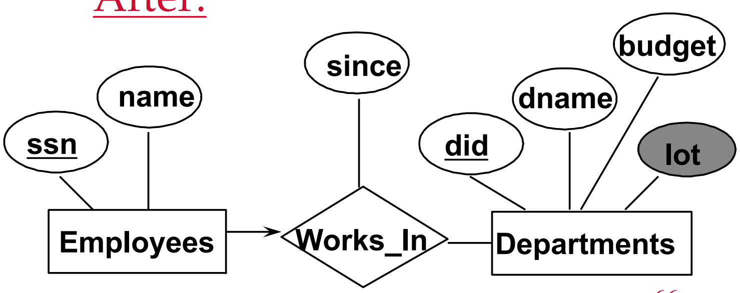
Refining an ER Diagram

- **1st diagram becomes:**
  Workers(S,N,L,D,Si)
  Departments(D,M,B)
  - Lots associated with workers.
- **Suppose all workers in a dept are assigned the same lot: D → L**
- **Redundancy; fixed by:**
  Workers2(S,N,D,Si)
  Dept_Lots(D,L)
  Departments(D,M,B)
- **Can fine-tune this:**
  Workers2(S,N,D,Si)
  Departments(D,M,B,L)

Before:



After:



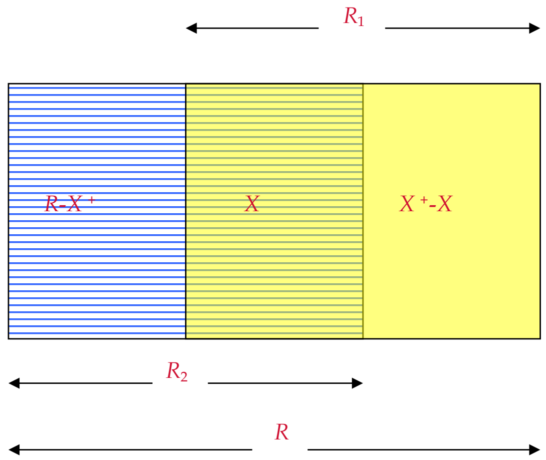
Example: Decomposition into BCNF

- **Given: relation R with FD's F.**
- **Look among the given FD's for a BCNF violation X ->B.**
  - 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 -> B

- **Replace R by relations with schemas:**
  - R1 = X +.
  - R2 = (R – X +) U X.
- **Project given FD's F onto the two new relations.**
  - Compute the closure of F = all nontrivial FD's that follow from F.
  - Use only those FD's whose attributes are all in R1 or all in R2.

Decomposition Picture



Problems with Decompositions

- There are three potential problems to consider:
  1) May be impossible to reconstruct the original relation! (Lossiness)
     - Fortunately, not in the SNLRWH example.
  2) Dependency checking may require joins.
     - Fortunately, not in the SNLRWH example.
  3) Some queries become more expensive.
     - e.g., How much does Guldu earn?

  Lossiness (#1) cannot be allowed

  #2 and #3 are design tradeoffs: Must consider these issues vs. redundancy.

Lossless Decomposition (example)

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

$\bowtie$

R	W
8	10
5	7

=

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Lossy Decomposition (example)

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

$\Rightarrow$

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

A → B; C → B

A	B
1	2
4	5
7	2

$\bowtie$

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 Decomposition

- **Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F:**

$$\pi_X(r) \bowtie \pi_Y(r) = r$$

- The decomposition of R into X and Y is **lossless with respect to F** *if and only if* F^+ contains:

$X \cap Y \rightarrow X$, **or**

$X \cap Y \rightarrow Y$

In other words, the common attributes must contain a key for either

in previous example: decomposing ABC into AB and BC is lossy, because intersection (i.e., "B") is not a key of either resulting relation.

- Useful result: If $W \rightarrow Z$ holds over R and $W \cap Z$ is empty, then decomposition of R into R-Z and WZ is loss-less.

Lossless Decomposition (example)

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

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B$; $C \rightarrow B$

A	C
1	3
4	6
7	8

⋈

B	C
2	3
5	6
2	8

=

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

But, now we can't check $A \rightarrow B$ without doing a join!

Dependency Preserving Decomposition

- **If we decompose a relation R into relations R1 and R2, All dependencies of R either must be a part of R1 or R2 or must be derivable from combination of FD's of R1 and R2.**

Dependency Preserving Decomposition

- **Dependency preserving decomposition (Intuitive):**
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold individually on X, on Y and on Z, then all FDs that were given to hold on R must also hold. *(Avoids Problem #2 on our list.)*

- The projection of F on attribute set X (denoted F_X) is the set of FDs $U \rightarrow V$ in F^+ (*closure of F , not just F*) such that all of the attributes on both sides of the f.d. are in X.
  - *That is: U and V are subsets of X*

Dependency Preserving Decompositions (Contd.)

- **Decomposition of R into X and Y is *dependency preserving* if $(F_X \cup F_Y)^+ = F^+$**
  - i.e., if we consider only dependencies in the closure F^+ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F^+.
- **Important to consider F^+ in this definition:**
  - ABC, $A \to B$, $B \to C$, $C \to A$, decomposed into AB and BC.
  - Is this dependency preserving? Is $C \to A$ preserved?????
    - note: F^+ contains $F \cup \{A \to C, B \to A, C \to B\}$, so…

- F_{AB} contains $A \to B$ and $B \to A$; F_{BC} contains $B \to C$ and $C \to B$
- So, $(F_{AB} \cup F_{BC})^+$ contains $C \to A$

Decomposition into BCNF

- Consider relation R with FDs F. If $X \to Y$ violates BCNF, decompose R into R - Y and XY (guaranteed to be loss-less).
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDPQV, key C, $JP \to C$, $SD \to P$, $J \to S$
  - *{contractid, supplierid, projectid,deptid,partid, qty, value}*
  - To deal with $SD \to P$, decompose into SDP, CSJDQV.
  - To deal with $J \to S$, decompose CSJDQV into JS and CJDQV
  - So we end up with: SDP, JS, and CJDQV

- Note: several dependencies may cause violation of BCNF. The order in which we fix them could lead to very different sets of relations!

BCNF and Dependency Preservation

- **In general, there may not be a dependency preserving decomposition into BCNF.**
  - e.g., CSZ, CS → Z, Z → C
  - Can't decompose while preserving 1st FD; not in BCNF.
- **Similarly, decomposition of CSJDPQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP → C, SD → P and J → S).**
- ***{contractid, supplierid, projectid,deptid,partid, qty, value}***
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - but JPC tuples are stored only for checking the f.d. (*Redundancy!*)

Quiz

- **Consider a schema R(A,B,C,D) and functional dependencies A->B and C->D. Then the decomposition of R into R1(AB) and R2(CD) is:**

  A. dependency preserving and lossless join
  B. lossless join but not dependency preserving
  C. dependency preserving but not lossless join
  D. not dependency preserving and not lossless join

Quiz

- **Consider a schema R(A,B,C,D) and functional dependencies A->B and C->D. Then the decomposition of R into R1(AB) and R2(CD) is:**
  - A. dependency preserving and lossless join
  - B. lossless join but not dependency preserving
  - C. dependency preserving but not lossless join
  - D. not dependency preserving and not lossless join

Summary of Schema Refinement

- **BCNF: each field contains information that cannot be inferred using only FDs.**
  - ensuring BCNF is a good heuristic.
- **Not in BCNF? Try decomposing into BCNF relations.**
  - Must consider whether all FDs are preserved!
- **Lossless-join, dependency preserving decomposition into BCNF impossible? Consider lower NF e.g., 3NF.**
  - Same if BCNF decomp is unsuitable for typical queries
  - Decompositions should be carried out and/or re-examined while keeping performance requirements in mind.

Higher Normal Forms

- **BCNF is the "ultimate" normal form when using only functional dependencies as constraints**
  - "Every attribute depends on a key, a whole key, and nothing but a key, so help me Codd."

- **However, there are higher normal forms (4NF to 6NF) that rely on generalizations of FDs**
  - 4NF: Multivalued dependencies
  - 5NF/6NF: Join dependencies

Fourth Normal Form

- **Any relation is in Fourth Normal Form if it is BCNF *and any multivalued dependencies are trivial***
- **Eliminate non-trivial multivalued dependencies by projecting into simpler tables**

Fifth Normal Form

- **A relation is in 5NF if every join dependency in the relation is implied by the keys of the relation**
- ***Implies that relations that have been decomposed in previous NF can be recombined via natural joins to recreate the original 1NF relation***

Normalization

- **Normalization is performed to reduce or eliminate Insertion, Deletion or Update anomalies.**
- **However, a completely normalized database may not be the most efficient or effective implementation.**
- **"Denormalization" is sometimes used to improve efficiency.**

Denormalizatin

- **Normalization in real world databases:**
  - Guided by normal form theory
  - But: Normalization is not everything!
  - Trade-off: Redundancy/anomalies vs. speed
  - General design: Avoid redundancy wherever possible, because redundancies often lead to inconsistent states
  - An exception: Materialized views (≈ precomputed joins) – expensive to maintain, but can boost read efficiency

Denormalization

- **Usually, a schema in a higher normal form is better than one in a lower normal form**
  - However, sometimes it is a good idea to artificially create lower-form schemas to, e.g., increase read performance
  - This is called denormalization

- **Denormalization usually increases query speed and decreases update efficiency due to the introduction of redundancy**

Denormalization

- **Rules of thumb:**
  - A good data model almost always directly leads to relational schemas in high normal forms
    - Carefully design your models!
    - Think of dependencies and other constraints!
    - Have normal forms in mind during modeling!
- **– Denormalize only when faced with a performance problem that cannot be resolved by:**
  - Money
  - hardware scalability
  - current SQL technology
  - network optimization
  - Parallelization
  - other performance techniques

ADDITIONAL SLIDES (FYI)

BCNF vs 3NF

- BCNF: **For every functional dependency X->Y in a set *F* of functional dependencies over relation *R*, either:**
  - Y is a subset of X or,
  - X is a *superkey* of R
- 3NF: **For every functional dependency X->Y in a set *F* of functional dependencies over relation *R*, either:**
  - Y is a subset of X or,
  - X is a *superkey* of R, or
  - Y is a subset of K for some key K of R
    - *N.b.,* no subset of a key is a key

3NF Schema

For every functional dependency X->Y in a set *F* of functional dependencies over relation *R*, either:

- Y is a subset of X or,
- X is a *superkey* of R, or
- Y is a subset of K for some key K of R

Client, Office -> Client, Office, Account
Account -> Office

Account	Client	Office
A	Joe	1
B	Mary	1
A	John	1
C	Joe	2

3NF Schema

For every functional dependency X->Y in a set *F* of functional dependencies over relation *R*, either:
- Y is a subset of X or,
- X is a *superkey* of R, or
- Y is a subset of K for some key K of R

Client, Office -> Client, Office, Account
Account -> Office

Account	Client	Office
A	Joe	1
B	Mary	1
A	John	1
C	Joe	2

BCNF vs 3NF

For every functional dependency X->Y in a set *F* of functional dependencies over relation *R*, either:
- Y is a subset of X or,
- X is a *superkey* of R
- Y is a subset of K for some key K of R

3NF has some redundancy
BCNF does not

Unfortunately, BCNF is not *dependency preserving*, but 3NF is

Account	Client	Office
A	Joe	1
B	Mary	1
A	John	1
C	Joe	2

Client, Office -> Client, Office, Account
Account -> Office

Account	Office
A	1
B	1
C	2

Account -> Office

Account	Client
A	Joe
B	Mary
A	John
C	Joe

No non-trivial FDs

Lossless decomposition

Closure

- **Want to find all attributes A such that X -> A is true, given a set of functional dependencies F**

define closure of X as X*

<u>Closure(X)</u>:

c = X
Repeat
    old = c
    if there is an FD Z->V such that
    Z $\subset$ c and
    V $\not\subset$ c then
        c = c U V
until old = c
return c

<u>Closure(**X**)</u>:
c = **X**
Repeat
    old = **c**
    if there is an FD **Z**->**V** such that
    Z $\subset$ **c** and
    V $\not\subset$ **c** then
        c = **c** U **V**
until **old** = **c**
return **c**

BCNFify

For every functional dependency X->Y in a set *F* of functional dependencies over relation *R*, either:
- Y is a subset of X or,
- X is a *superkey* of R

<u>BCNFify(schema **R**, functional dependency set **F**):</u>
D = {{**R**,**F**}}
while there is a schema **S** with dependencies **F'** in **D** that is not in BCNF, do:
    given **X**->**Y** as a BCNF-violating FD in **F**
    such that **XY** is in **S**
    replace **S** in **D** with
        S1={**XY**,**F1**} and
        S2={(**S**-**Y**) U **X**, **F2**}
    where **F1** and **F2** are the FDs in **F** over **S1** or **S2**
    (may need to split some FDs using decomposition)
End
return **D**

Third Normal Form (3NF)

- **Reln R with FDs *F* is in 3NF if, for all X → A in F^+**
  - A ∈ X (called a *trivial* FD), or
  - X is a superkey of R, or
  - A is part of some candidate key (not superkey!) for R. (sometimes stated as "A is *prime*")
- ***Minimality* of a key is crucial in third condition above!**
- **If R is in BCNF, obviously in 3NF.**
- **If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomp, or performance considerations).**
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

What Does 3NF Achieve?

- **If 3NF violated by X → A, one of the following holds:**
  - X is a subset of some key K ("partial dependency")
    - We store (X, A) pairs redundantly.
    - e.g. Reserves SBDC (C is for credit card) with key SBD and S → C
  - X is not a proper subset of any key. ("transitive dep.")
    - There is a chain of FDs K → X → A, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value (different K's, same X implies same A!) – problem with initial SNLRWH example.
- **But: even if R is in 3NF, these problems could arise.**
  - e.g., Reserves SBDC (note: "C" is for credit card here), S → C, C → S is in 3NF (why?), but for each reservation of sailor S, same (S, C) pair is stored.
- **Thus, 3NF is indeed a compromise relative to BCNF.**

Decomposition into 3NF

- **Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier) but does not ensure dependency preservation.**
- **To ensure dependency preservation, one idea:**
  - If X → Y is not preserved, add relation XY.

  Problem is that XY may violate 3NF! e.g., consider the addition of CJP to `preserve' JP → C. What if we also have J → C ?
- **Refinement: Instead of the given set of FDs F, use a *minimal cover for F*.**

Minimal Cover for a Set of FDs

- ***Minimal cover* G for a set of FDs F:**
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- **Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.**
- **e.g., A → B, ABCD → E, EF → GH, ACDF → EG has the following minimal cover:**
  - A → B, ACD → E, EF → G and EF → H
- **M.C. implies 3NF, Lossless-Join, Dep. Pres. Decomp!!!**
  - (in book)