Functional Dependencies, Schema Refinement, and Normalization for Relational Databases

CSC 375, Fall 2016

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…

- http://vlsi.byblos.lau.edu.lb/classes/csc375/Resources.html
  - *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

Introduction

- **Levels at which we can discuss goodness of relation schemas**
  - Logical (or conceptual) level
  - Implementation (or physical storage) level
- **Approaches to database design:**
  - Bottom-up or top-down

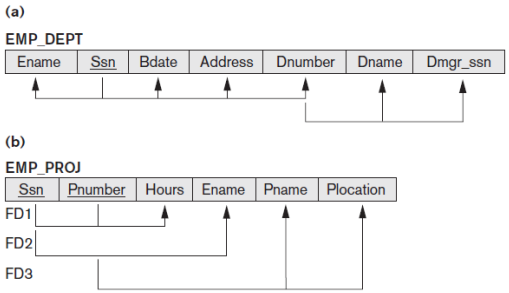
Informal Design Guidelines for Relation Schemas

- **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

Guideline 1

- **Design relation schema so that it is easy to explain its meaning**
- **Do not combine attributes from multiple entity types and relationship types into a single relation**
- **Example of violating Guideline 1:**

Figure 15.3
Two relation schemas suffering from update anomalies. (a) EMP_DEPT and (b) EMP_PROJ.

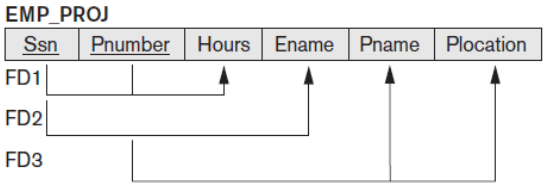


Redundant Information in Tuples and Update Anomalies

- **Storing natural joins of base relations leads to update anomalies**
- **Types of update anomalies:**
  - Insertion
  - Deletion
  - Modification

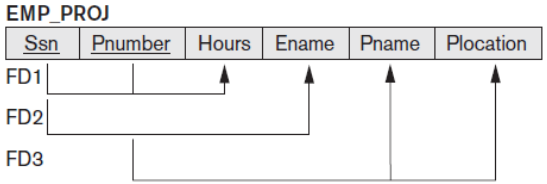
Example of An Update Anomaly

- **Update Anomaly:**
  - Changing the name of project number P1 from "Billing" to "Customer-Accounting" may cause this update to be made for all 100 employees working on project P1.



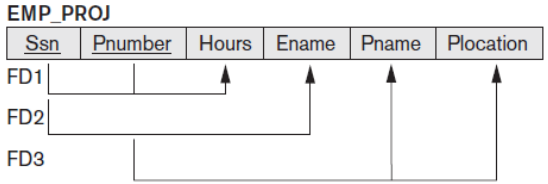
Example of An Insert Anomaly

- **Insert Anomaly:**
  - Cannot insert a project unless an employee is assigned to it.
- **Conversely**
  - Cannot insert an employee unless an he/she is assigned to a project.



Example of A Delete Anomaly

- **Delete Anomaly:**
  - When a project is deleted, it will result in deleting all the employees who work on that project.
  - Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.



Guideline 2

- **Design base relation schemas so that no update anomalies are present in the relations**
- **If any anomalies are present:**
  - Note them clearly
  - Make sure that the programs that update the database will operate correctly

NULL Values in Tuples

- **May group many attributes together into a "fat" relation**
  - Can end up with many NULLs
- **Problems with NULLs**
  - Wasted storage space
  - Problems understanding meaning

Guideline 3

- **Avoid placing attributes in a base relation whose values may frequently be NULL**
- **If NULLs are unavoidable:**
  - Make sure that they apply in exceptional cases only, not to a majority of tuples

Generation of Spurious Tuples

- **NATURAL JOIN**
  - Result produces many more tuples than the original set of tuples
  - Called spurious tuples
  - Represent spurious information that is not valid

Guideline 4

- **Design relation schemas to be joined with equality conditions on attributes that are appropriately related**
  - Guarantees that no spurious tuples are generated
- **Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations**

Recap: Which Design is Better: Green or Blue?

heroID	teamID	heroName	teamName	joinYear
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963
4	1	Hulk	The Avengers	1963
5	1	Captain America	The Avengers	1964
6	2	Invisible Girl	Fantastic Four	1961

heroID	heroName
1	Thor
2	Mister Fantastic
3	Iron Man
4	Hulk
5	Captain America
6	Invisible Girl

heroID	teamID	joinYear
1	1	1963
2	2	1961
3	1	1963
4	1	1963
5	1	1964
6	2	1961

teamID	teamName
1	The Avengers
2	Fantastic Four

Recap: What is Wrong with Blue Design?

heroID	teamID	heroName	teamName	joinYear
1	1	Thor	The Avengers	1963
2	2	Mister Fantastic	Fantastic Four	1961
3	1	Iron Man	The Avengers	1963
4	1	Hulk	The Avengers	1963
5	1	Captain America	The Avengers	1964
6	2	Invisible Girl	Fantastic Four	1961

- **Redundancy:**
  - The fact that certain teams have certain names is represented several times.
- **Inferior expressiveness:**
  - We cannot represent heroes that currently have no team.
- **Modification anomalies:**
  - Insertion anomalies
    - How do you add heroes that currently have no team?
    - How do you (consistently!) add new tuples?
  - Deletion anomalies
    - Deleting Mister Fantastic and Invisible Girl also deletes all information about the Fantastic Four
  - Update anomalies
    - Renaming a team requires updating several tuples (due to redundancy

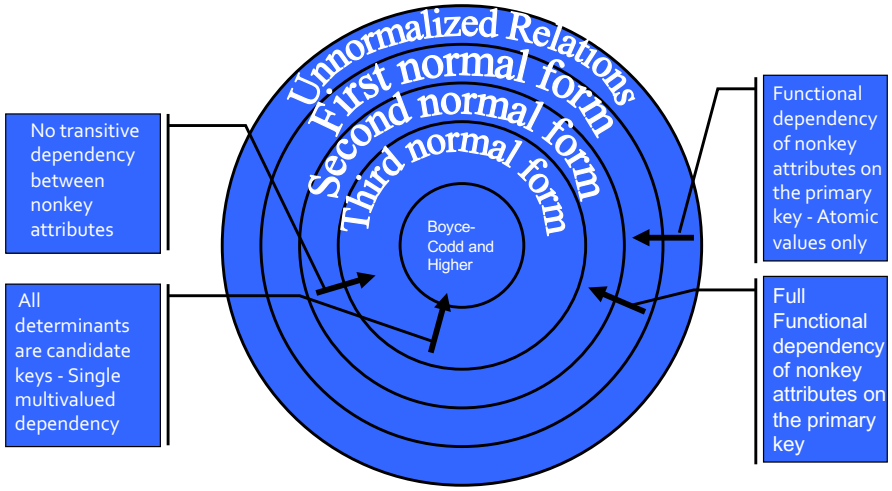
Normalization

- **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)**

Normalization



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.

Recap: The Evils of Redundancy

- ***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?

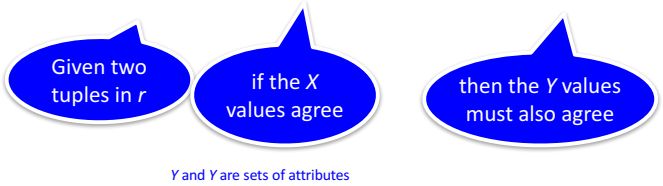
Functional Dependencies (FDs)

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

Functional Dependencies (FDs)

- Formally, a functional dependency $X \rightarrow Y$ holds over relation schema R if, for every allowable instance *r* of R:

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



- **Example: SSN → StudentNum**

FD's Continued

- A FD is a statement about *all* allowable relations.
  - Identified based on application semantics
  - Given some instance *r1* of R, we can check if *r1* violates some FD *f*, but we cannot determine if *f* holds over R.

- **How related to keys?**
  - if "K → all attributes of R" then K is a *superkey* for R
    - Does not require K to be *minimal*.

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 → SNLRWH
  - *rating* determines *hrly_wages*: R → W
  - *lot* determines *lot*: L → L ("trivial" dependency)

Example (Contd.)

Problems due to R → W :

- *Update anomaly*: Can we change W in just the 1st tuple of SNLRWH?
- *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his/her rating?
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

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_Emps2

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

Using two smaller tables is better

Wages

R	W
8	10
5	7

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 → W problematic, but S→W not?

Taming Schema Redundancy

- **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 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

What do we need to proceed?

- **What do we need?**
  - A compact representation for sets of FD constraints
  - No redundant FDs
  - An algorithm to compute the set of all implied FDs

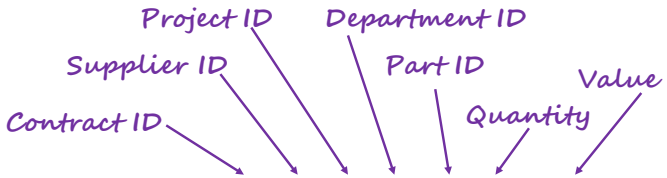
Reasoning About FDs

- 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 sets of attributes):**
  - *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 $X \rightarrow Z$
- **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



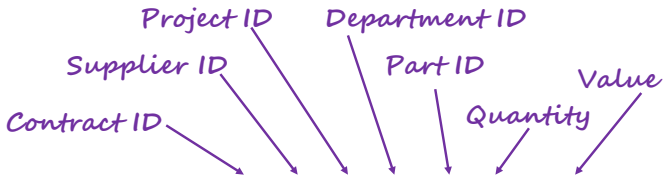
Example: Contracts(*cid,sid,jid,did,pid,qty,value*), and:

- C is the key: C ⟶ 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

- **JP ⟶ C, C ⟶ CSJDPQV imply JP ⟶ CSJDPQV**
- **SD ⟶ P implies SDJ ⟶ JP**
- **SDJ ⟶ JP, JP ⟶ CSJDPQV imply SDJ ⟶ CSJDPQV**

These are also candidate keys

Reasoning About FDs - Example



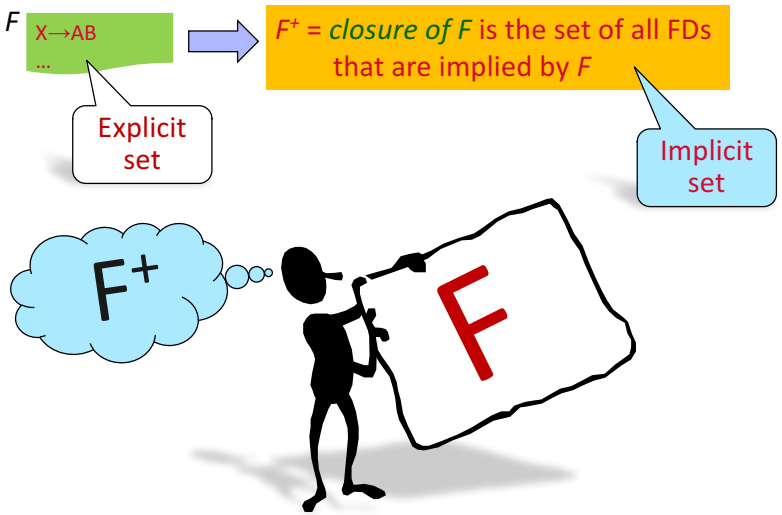
Example: Contracts(*cid,sid,jid,did,pid,qty,value*), and:

- 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.

Closure of a FD set



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

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 $X \rightarrow A$ is in F^+
    - initialize $X^+ := X$
    - Repeat until no change:
      if $U \rightarrow V$ 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*)+:**

```
unused = F
closure = X
repeat
        for each Y → Z ∈ unused do
                if Y ⊆ closure then
                unused = unused \ {Y → Z}
                closure = closure ∪ Z
until unused and closure did not change on this iteration
return closure
```

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

Normal Forms

- **Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!**
- **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)
- **If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.**

Practical Use of Normal Forms

- **Normalization carried out in practice**
  - Resulting designs are of high quality and meet the desirable properties stated previously
  - Pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF
- **Do not need to normalize to the highest possible normal form**

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.

Normal Forms

Normal form	Defined by	Brief definition
First normal form (1NF)	Two versions: E.F. Codd (1970), C.J. Date (2003)[9]	Table faithfully represents a relation and has no *repeating groups*
Second normal form (2NF)	E.F. Codd (1971)[2]	No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key
Third normal form (3NF)	E.F. Codd (1971);[2] see also Carlo Zaniolo's equivalent but differently expressed definition (1982)[10]	Every non-prime attribute is non-transitively dependent on every candidate key in the table. The attributes that do not contribute to the description of the primary key are removed from the table. In other words, no transitivity dependency is allowed.
Elementary Key Normal Form (EKNF)	C.Zaniolo (1982)[10]	Every non-trivial functional dependency in the table is either the dependency of an elementary key attribute or a dependency on a superkey
Boyce–Codd normal form (BCNF)	Raymond F. Boyce and E.F. Codd (1974)[11]	Every non-trivial functional dependency in the table is a dependency on a superkey
Fourth normal form (4NF)	Ronald Fagin (1977)[12]	Every non-trivial multivalued dependency in the table is a dependency on a superkey
Fifth normal form (5NF)	Ronald Fagin (1979)[13]	Every non-trivial join dependency in the table is implied by the superkeys of the table
Domain/key normal form (DKNF)	Ronald Fagin (1981)[14]	Every constraint on the table is a logical consequence of the table's domain constraints and key constraints
Sixth normal form (6NF)	C.J. Date, Hugh Darwen, and Nikos Lorentzos (2002)[15]	Table features no non-trivial join dependencies at all (with reference to generalized join operator)

First Normal Form

- **To move to First Normal Form a relation must contain only *atomic values* at each row and column.**
  - No repeating groups
  - A column or set of columns is called a *Candidate Key* when its values can uniquely identify the row in the relation.
- **Techniques to achieve first normal form**
  - Remove attribute and place in separate relation
  - Expand the key
  - Use several atomic attributes

First Normal Form (cont'd.)

- **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**

A) Introducing a new relation

- **Use old key and multi-attribute as composite key**

heroID	heroName	Powers
1	Storm	weather control, flight
2	Wolverine	extreme cellular regeneration
3	Phoenix	omnipotence, indestructibility, limitless energy manipulation

heroID	Powers
1	weather control
1	flight
2	extreme cellular regeneration
3	omnipotence
3	indestructibility
3	limitless energy manipulation

heroID	heroName
1	Storm
2	Wolverine
3	Phoenix

B) Replicating the tuple for each multi-value

heroID	heroName	Powers
1	Storm	weather control, flight
2	Wolverine	extreme cellular regeneration
3	Phoenix	omnipotence, indestructibility, limitless energy manipulation

heroID	heroName	Powers
1	Storm	weather control
1	Storm	flight
2	Wolverine	extreme cellular regeneration
3	Phoenix	omnipotence
3	Phoenix	indestructibility
3	Phoenix	limitless energy manipulation

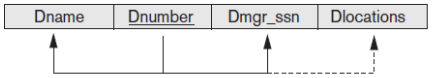
C) Introducing an own attribute for each multi-value

heroID	heroName	Powers
1	Storm	weather control, flight
2	Wolverine	extreme cellular regeneration
3	Phoenix	omnipotence, indestructibility, limitless energy manipulation

heroID	heroName	Power1	Power2	Power3
1	storm	weather control	flight	NULL
2	Wolverine	extreme cellular regeneration	NULL	NULL
3	Phoenix	omnipotence	indestructibility	limitless energy manipulation

Another Example!

(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.

Second Normal Form

- **A relation is said to be in Second Normal Form when every nonkey attribute is fully functionally dependent on the primary key.**
  - That is, every nonkey attribute needs the full primary key for unique identification
  - In the case of a composite primary key, this means that a non-key column cannot depend on only part of the composite key.
- ***This is typically accomplished by projecting (think splitting) the relations into simpler relations with simpler keys***

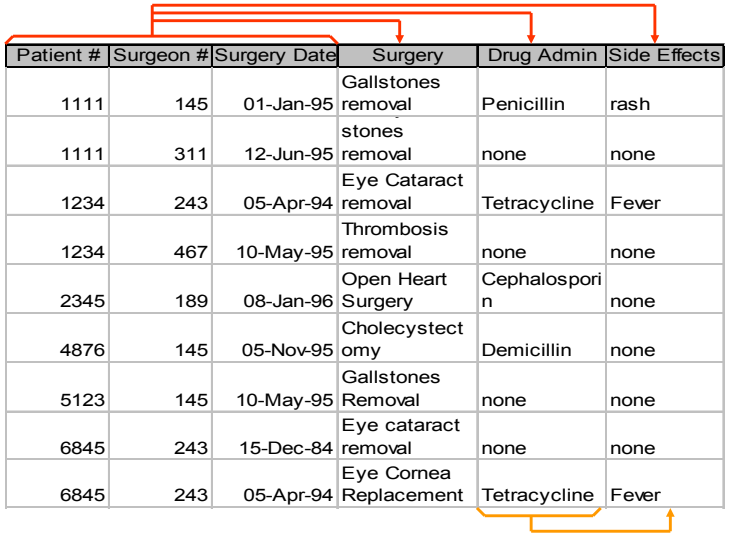
Second Normal Form

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

Second Normal Form

Surgeon #	Surgeon Name
145	Beth Little
189	David Rosen
243	Charles Field
311	Michael Diamond
467	Patricia Gold

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	Cephalosporin	none
4876	145	05-Nov-95	Cholecystectomy	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

- **Third Normal Form (3NF) requires that all columns depend directly on the primary key.**
- **Tables violate the Third Normal Form when one column depends on another column, which in turn depends on the primary key (a transitive dependency).**
- **Definition: A relation schema is in 3NF if and only if:**
  - It is 2NF and
  - No key transitively determines a non-key attribute

Recap: General Definitions of *First*, *Second* and *Third* Normal Forms

Table 15.1 Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

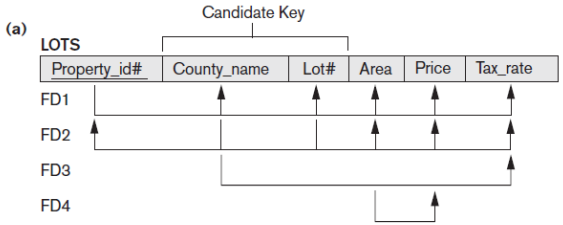
General Definitions of Second and Third Normal Forms (cont'd.)

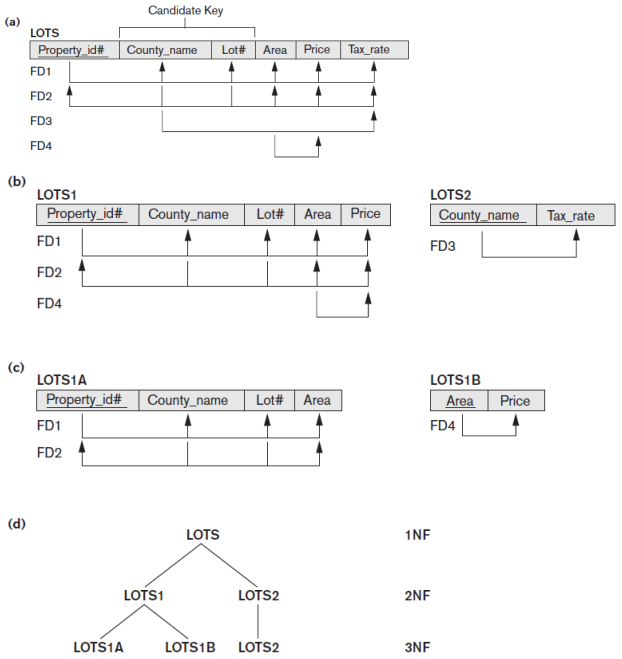
- **Prime attribute**
  - Part of any candidate key will be considered as prime
- **Consider partial, full functional, and transitive dependencies with respect to all candidate keys of a relation**

General Definition of Second Normal Form

Definition. A relation schema R is in **second normal form** (2NF) if every non-prime attribute A in R is not partially dependent on *any* key of R.[11]

Figure 15.12
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.





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 → 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!

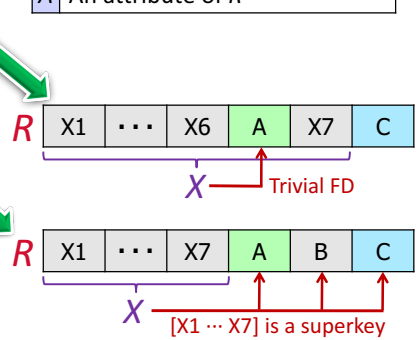
- 1st Normal Form – all attributes are atomic (i.e., "flat tables")
- 1st ⊃ 2nd (of historical interest) ⊃ 3rd ⊃ Boyce-Codd ⊃ ...

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*



In other words, *R* is in BCNF if the only non-trivial FDs that hold over *R* are key constraints (i.e., *X* must be a superkey!)

Boyce-Codd Normal Form (BCNF)

- **Reln R with FDs *F* is in BCNF if, for all X → A in F^+**
  - $A \in X$ (called a *trivial* FD), or
  - X is a superkey for R.
- **In other words: "R is in BCNF if the only non-trivial FDs over R are *key constraints*."**
- **If R in BCNF, then every field of every tuple records information that cannot be inferred using FDs alone.**
  - Say we know FD X → A holds for this example relation:

X	Y	A
x	y1	a
x	y2	?

- Can you guess the value of the missing attribute?
- Yes, so relation is not in BCNF

BCNF is Desirable

Consider the relation:

X	Y	A →A
x	y1	y2
x	y2	?

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

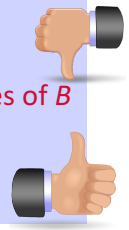
BCNF: Desirable Property

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

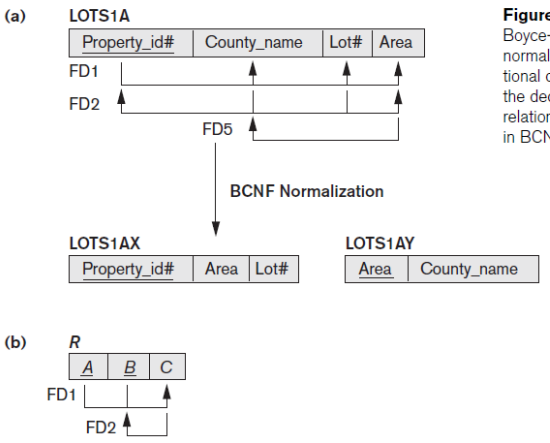


Figure 15.13
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

Boyce-Codd Normal Form - Alternative Formulation

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

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

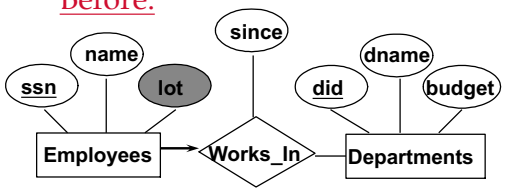
Wages

- Q: Are both of these relations now in BCNF?
- **Decompositions should be used only when needed.**
  - Q: potential problems of decomposition?

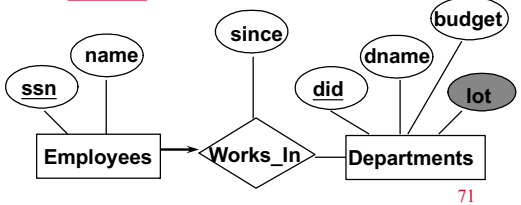
Refining an ER Diagram

- **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:



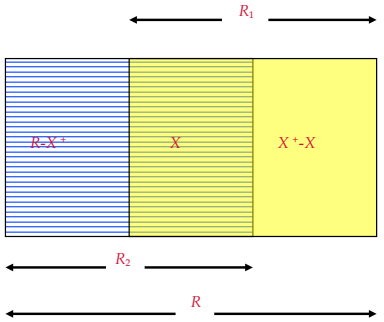
Example: Decomposition into BCNF

- **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



Example

- **Drinkers(name, addr, beersLiked, manf, favBeer)**
- **F = name->addr, name -> favBeer, beersLiked->manf**
- **Pick BCNF violation name->addr.**
- **Close the left side: {name}+ = {name, addr, favBeer}.**
- **Decomposed relations:**
  - Drinkers1(name, addr, favBeer)
  - Drinkers2(name, beersLiked, manf)

Example, Continued

- **We are not done; we need to check Drinkers1 and Drinkers2 for BCNF.**
- **Projecting FD's is complex in general, 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 :**
  - Drinkers1(name, addr, favBeer)
  - Drinkers3(beersLiked, manf)
  - 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.**

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

⋈

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

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

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 → B; C → 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 → B without doing a join!

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, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, decomposed into AB and BC.
  - Is this dependency preserving? Is $C \rightarrow A$ preserved?????
    - note: F^+ contains $F \cup \{A \rightarrow C, B \rightarrow A, C \rightarrow B\}$, so…

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

Decomposition into BCNF

- Consider relation R with FDs F. If $X \rightarrow Y$ 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, $JP \rightarrow C$, $SD \rightarrow P$, $J \rightarrow S$
  - *{contractid, supplierid, projectid,deptid,partid, qty, value}*
  - To deal with $SD \rightarrow P$, decompose into SDP, CSJDQV.
  - To deal with $J \rightarrow S$, 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, $CS \rightarrow Z$, $Z \rightarrow C$
  - 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 $JP \rightarrow C$, $SD \rightarrow P$ and $J \rightarrow S$).**
- ***{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!*)

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)

Third Normal Form (3NF)

- **Reln R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in F^+**
  A $\in$ 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 $X \rightarrow A$, 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 $S \rightarrow C$
  - X is not a proper subset of any key. ("transitive dep.")
    - There is a chain of FDs $K \rightarrow X \rightarrow A$, 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), $S \rightarrow C$, $C \rightarrow S$ 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 $X \rightarrow Y$ is not preserved, add relation XY.

  Problem is that XY may violate 3NF! e.g., consider the addition of CJP to `preserve' $JP \rightarrow C$. What if we also have $J \rightarrow C$?
- **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., $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow GH$, $ACDF \rightarrow EG$ has the following minimal cover:**
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$ and $EF \rightarrow H$
- **M.C. implies 3NF, Lossless-Join, Dep. Pres. Decomp!!!**
  - (in book)