Normalization Theory
Book chapter: Chapter 7

Main question: How do we design "good" tables for a relational database?
- Typically we start with ER and convert it into tables
- Still, different people come up with different ER, and thus
  with different tables. Which one is better? What design should we choose?

****
Warning: The most difficult and theoretical part of the course. Pay attention!
****

****
* MOTIVATION & INTUITION
****

Example:
Students take classes
StudentClass(sid, name, addr, dept, cnum, title, unit)

Q: Is it a good table design?
<Example instance slides>

REDUNDANCY: The same information mentioned multiple times
Redundancy leads to potential anomaly
1) UPDATE ANOMALY: Only some information may be updated
Q: What if a student changes the address?

2) INSERTION ANOMALY: Some informatikon cannot be represented
Q: What if a student does not take any class?

2) DELETION ANOMALY: Deletion of some information may delete others
Q: What if the only class that a student takes is cancelled?

Q: Is there a better design? What tables would you use?

Let us study this question more carefully

Q: Where is the redundancy from?
(Slide on "guessing" missing info)

Some attributes are "determined" by other attrs: FUNCTIONAL DEPENDENCY
ex) **sid -> (name, addr),**
(dept, cnum) -> (title, unit)

Some notations for formal definition

u[X] - values for the attributes X of tuple u
<e.g, u = (sid: 100, name: James, addr: Wilshire)
u[sid, name] = (100, James)>

FUNCTIONAL DEPENDENCY:
X -> Y (where X = A1, ..., An, Y = B1, ..., Bm):
- For any u1, u2 in R, if u1[X] = u2[X}, then u1[Y] = u2[Y]
- No two tuples in R can have the same X values but different Y values

<e.g., StudentClass(sid, name, addr, dept, cnum, title, unit)>

Q: sid -> name?

Q: dept, cnum -> title, unit?

Q: dept, cnum -> sid?

NOTE:
* Whether a FD is true or not depends on real-world semantics

<examples>
A B C   AB -> C
----
a1 b1 c1
a1 b2 c2
a2 b1 c3   Q: Is this okay?

Replace c3 to c1.
A B C   AB -> C
----
a1 b1 c1
a1 b2 c2
a2 b1 c1   Q: Is this okay?

NOTE: AB -> C does not mean no duplicate C values.

Replace b2 to b1.
A B C   AB -> C
----
a1 b1 c1
a1 b1 c2
a2 b1 c3   Q: Is this okay?

<back to StudentClass example>
  sid -> name, addr
  dept,cnum -> title,unit

  (301, James, 11 West) is stored redundantly.
  So is (cs, 143, database, 04).

NOTE: When there is a FD, we might have redundancy.

DECOMPOSITION: When there is a FD, no need to store multiple instances of this relationship. Store it once in a separate table

<StudentClass example again>
<Intuitively explain normalization of StudentClass table>
Two FDs: sid->(name,addr), (dept,cnum)->(title,unit)

(1) sid -> (name, addr) no need to store it multiple time. separate it out

(2) (dept, cnum) -> (title, unit). separate it out

BASIC IDEA OF NORMAL FORM and DECOMPOSITION
- Whenever there is a FD, the table may be "bad" (not in normal form)
- we use FDs to "split" or "decompose" table and remove redundancy

FUNCTIONAL DEPENDENCY AND KEY
- a key determines a tuple
- functional dependency determines other attributes

Q: In previous example, is (A, B) a key of R?

X is a KEY of R iff

1) X -> all attributes of R (ie, X+ = R)
2) No subset of X satisfies 1) (i.e., X is minimal)

PROJECTING FD

R(A, B, C, D): A -> B, B -> A, A -> C

Q: What FDs hold for R'(B, C, D) which is a projection of R?

COMMENTS: In order to find FD's after projection, we first need
to compute F+ and pick the ones with only the attributes after
the projection.

****
* LOSSLESS-JOIN DECOMPOSITION
****

<Remind splitting StudentClass table to make it better>

DECOMPOSITION
- Split table R(A1, ..., An) into R1(A1, ..., Ai) and R2(Aj, ..., An)
- {A1, ... An} = {A1, .., Ai} UNION {Aj, ..., An}

<Conceptual diagram for R(X, Y, Z) -> R1(X, Y) and R2(Y, Z)>

Q: When we decompose, what should we watch out for?

LOSSLESS-JOIN DECOMPOSITION
- R = R1 NJ R2
- Intuitively, we should not lose any information by decomposing R
- Can reconstruct the original table from the decomposed tables

Q: When is decomposition lossless?

<example>

cnum	sid	name
143	1	James
143	2	Elaine
325	3	Susan

<eg1, Decompose into S1(cnum, sid), S2(cnum, name). Lossless?>
<Explain using the example>

- Main problem:
  - A tuple in S1 may join with two tuples in S2
  - A tuple in S2 may join with two tuples in S1
    - Tuples multiply because of these

<eg2, Decompoes into S1(cnum, sid), S2(sid, name). Lossless?>
<Explain using the example>

** DECOMPOSITION R(X, Y, Z) => R1(X, Y), R2(X, Z) IS LOSSLESS IF X->Y OR X->Z **
- i.e., The shared attributes are the key of one of the decomposed tables
- We can use FDs to check whether a decomposition is lossless

<e.g., StudentClass(sid, name, addr, dept, cnum, title, unit)>
sid->(name,addr) (dept,cnum)->(title,unit)

Q: R1(sid, name, addr), R2(sid, dept, cnum, title, unit). Lossless?

****
* FD, KEY & REDUNDANCY
****

StudentClass(sid, name, addr, dept, cnum, title, unit)
Q: sid->(name,addr). Does it cause redundancy?

After decomposition, Student(sid, name, addr)
Q: sid->(name,addr). Does it still cause redundancy?

Q: Why does the same FD cause redundancy in one case, but not in the other?

A: Main difference
- sid is a key for the second example, but not for the first.
- Because sid is a key, (sid, name, addr) is stored only once in the second
  -> no redundancy
- Because sid is not a key, (sid, name, addr) may be stored multiple times in the first.

IN GENERAL, FD X->Y IS BAD IF X DOES NOT CONTAIN A KEY.

****
* BOYCE-CODD NORMAL FORM (BCNF)
****
DEFINITION
- R is in BCNF with regard to F, iff for every non-trivial X -> Y,
  X contains a key

- a "good" table design (no redudancy due to FD)

Q: S(dept, cnum, title, unit). BCNF?
  dept,cnum->title,unit

Q: S(instructor, office, fax). BCNF?
    instructor->office
    office->fax

BCNF NORMALIZATION: decompose tables until all tables are in BCNF
- For each FD that violates the condition, use it to decompose the table and remove the redundancy from it.
- In addition, we want to make sure that the decomposition is lossless.
  -> The algorithm is designed to guarantee this

Naïve DECOMPOSITION ALGORITHM for BCNF:

Use FDs t o decompose relations which are not BCNF ... until they are
All BCNF

<eg, ClassInstructor(dept, cnum, title, unit, instructor, office, fax)>
    F: instructor -> office
       office -> fax
       (dept, cnum) -> title, unit
       (dept, cnum) -> instructor

NOTE: Explain that the algorithm guarantees lossless join decomposition, because after the decomposition based on X->Y, X becomes the key of one of the decomposed table

<example>
R(A, B, C, G, H, I)
A -> B
A -> C
G -> I
B -> H

Q: Does the algorithm lead to a unique set of relations?

<eg., R(A, B, C) A->C, B->C>
If we start with A->C: R1(A, C), R2(A, B)
If we start with B->C: R1(B, C), R2(A, B)

Q: R1(A, B), R2(B, C, D)
A->B
B->A
A->C
Are R1 and R2 in BCNF?

COMMENTS: We have to check all implied FD's for BCNF, not just the given ones.