CS122A: Introduction to Data Management

Lecture #6

Instructor: Chen Li

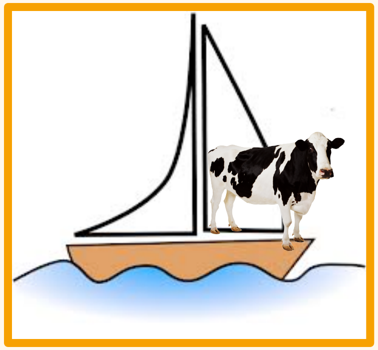
Next: Relational Database Design

- There are two aspects to this problem:
  - Logical schema design: We just saw one approach, namely, doing ER modeling followed by an ER → relational schema translation step
  - Physical schema design: Later, once we learn about indexes, when should we utilize them?
- We will look at both problem aspects this term, starting first with relational schema design
  - Our power tools will be functional dependencies (FDs) and normalization theory
  - Note: FDs also play an important role in other contexts as well, e.g., in SQL query optimization

So, Given a Relational Schema...

- How do I know if my relational schema is a "good" logical database design or not?
  - What might make it "not good"?
  - How can I fix it, if indeed it's "not good"?
  - How "good" is it, after I've fixed it?
- Note that your relational schema might have come from one of several places
  - You started from an ER model (but maybe that model was "wrong" in some way?)
  - You went straight to relational in the first place
  - It's not your schema – you inherited it! ☺

Ex: *Wisconsin Sailing Club*

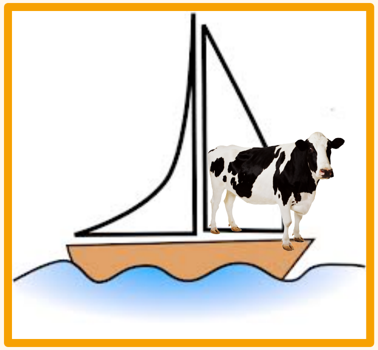


Proposed schema design #1:

sid	sname	rating	age	date	bid	bname	color
22	Dustin	7	45.0	10/10/98	101	Interlake	blue
22	Dustin	7	45.0	10/10/98	102	Interlake	red
22	Dustin	7	45.0	10/8/98	103	Clipper	green
22	Dustin	7	45.0	10/7/98	104	Marine	red
31	Lubber	8	55.5	11/10/98	102	Interlake	red
31	Lubber	8	55.5	11/6/98	103	Clipper	green
31	Lubber	8	55.5	11/12/98	104	Marine	red
…	…	…	…	…	…	…	…

Q: Do you think this is a "good" design? (Why or why not?)

Ex: *Wisconsin Sailing Club*



Proposed schema design #2:

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
…	…	…	…

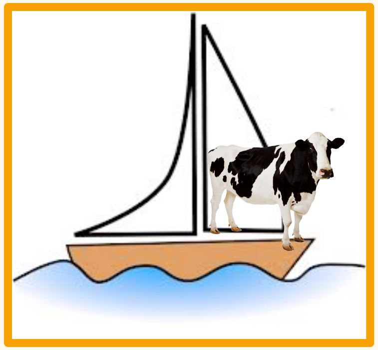
sid	bid	date
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
…	…	…

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Q: What about *this* design?

- Is #2 "better than #1…? Explain!
- Is it a "best" design?
- How can we go from design #1 to this one?

Ex: Wisconsin Sailing Club



Proposed schema design #3:

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
…	…	…	…

sid	bid	date
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
…	…	…

bid	bname
101	Interlake
102	Interlake
103	Clipper
104	Marine

bid	color
101	blue
102	red
103	green
104	red

Q: What about *this* design?

- Is #3 "better" or "worse" than #2…?
- What sort of tradeoffs do you see between the two?

The Evils of Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  - Redundant storage
  - Insert/delete/update anomalies

<u>*Basic rule of thumb:*</u>

"One fact, one place!"

- *Functional dependencies* can help in identifying problem schemas and suggesting refinements.
- Main refinement technique: <u>*decomposition*</u>, e.g., replace R(ABCD) with R1(AB) + R2(BCD).
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - Does the decomposition cause any problems?

** *Functional Dependencies (FDs)*

- A functional dependency X → Y holds over relation R if, for every allowable instance *r* of R:
  - For *t1* and *t2* in *r*, *t1*.X = *t2*.X implies *t1*.Y = *t2*.Y
  - I.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y can be *sets* of attributes.)
- An FD is a statement about *all* allowable relations.
  - Identified based on application semantics (similar to E-R).
  - Given some instance *r1* of R, we can check to see if it violates some FD *f*, but we cannot tell if *f* holds over R!
- Saying K is a candidate key for R means that K→ R
  - Note: K → R does not require K to be *minimal*! If K minimal, then it is a candidate key.

Example: Constraints on an Entity Set

- Suppose you're given a relation called HourlyEmps:
  - HourlyEmps (*<u>ssn</u>, name, lot, rating, hrly_wages, hrs_worked*)
- *<u>Notation</u>*: We will denote this relation schema by simply 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., HourlyEmps for SNLRWH).
- Suppose we also have some FDs on HourlyEmps:
  - *ssn* is the key: S → SNLRWH
  - *rating* determines *hrly_wages*: R → W

Example (Cont'd.)

Wages

R	W
8	10
5	7

HourlyEmps2

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

- 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 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

How about two smaller tables?