# CS122A: Introduction to Data Management

## Lecture #12: Relational DB Design Theory (1)

Instructor: Chen Li

# *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 E-R 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! ☺

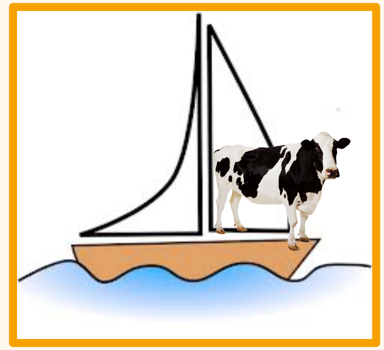
# *Ex:* *Wisconsin Sailing Club*

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
- *Functional dependencies* can help in identifying problem schemas and suggesting refinements.
- Main refinement technique: *decomposition*, 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?

*Good rule to follow:*

***"One fact, one place!"***

# *Functional Dependencies (FDs)*

- A <u>functional dependency</u> 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 (*ssn*, *name*, *lot*, *rating*, *hrly_wages*, *hrs_worked*)
- *Notation*: 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?

# *Reasoning About FDs*

- Given some FDs, we can usually infer additional FDs:
  - $ssn \rightarrow did,\ did \rightarrow lot$ implies $ssn \rightarrow lot$
- An FD $f$ is *implied by* a set of FDs $F$ if $f$ holds whenever all FDs in $F$ hold.
  - $F^+$ = *closure of F* is the set of all FDs that are implied by $F$.
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If $X \subseteq Y$, then $Y \rightarrow X$
  - *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!

# *Reasoning About FDs (Cont'd.)*

*(Recall: "two matching X's always have the same Y")*

- A few additional rules (which follow from AA):
  - *Union*: If $\boxed{X \rightarrow Y}$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- Example: Contracts(*cid,sid,jid,did,pid,qty,value*), and:
  - C is the key: $C \rightarrow CSJDPQV$
  - Project purchases each part using single contract: $JP \rightarrow C$
  - Dept purchases at most one part from a supplier: $SD \rightarrow P$
- $JP \rightarrow C$, $C \rightarrow CSJDPQV$ imply $JP \rightarrow CSJDPQV$
- $SD \rightarrow P$ implies $SDJ \rightarrow JP$
- $SDJ \rightarrow JP$, $JP \rightarrow CSJDPQV$ imply $SDJ \rightarrow CSJDPQV$

# *Reasoning About FDs (Examples)*

Let's consider R(ABCDE), F = {A → B, B → C, C D → E}

- Let's work our way towards inferring F+ ...

| | | |
|---|---|---|
| (a) A → B (b) B → C (c) CD → E | | (given) |
| (d) A → C | | (a, b, and transitivity) |
| (e) BD → CD | | (b and augmentation) |
| (f) BD → E | | (e and transitivity) |
| (g) AD → CD | | (d and augmentation) |
| (h) **AD** → E | *Candidate key!* | (g and transitivity) |
| (i) **AD** → C (j) **AD** → D | | (g and decomposition) |
| (j) AD → BD | If an attribute *X* is not on the RHS of *any* initial FD, *X* must be part of the key! | (a and augmentation) |
| (l) **AD** → B | | (k and decomposition) |
| (m) **AD** → A | | (a and reflexivity) |
| (n) AD → ABCDE | | (h, i, j, l, m, and union) .... |

# *Reasoning About FDs (Cont'd.)*

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in #attrs!)
- Typically, we just want to check if a given FD *X* → *Y* is in the closure of a set of FDs *F*. An efficient check:
  - Compute *attribute closure* of X (denoted X+) wrt *F:*
    - Set of all attributes A such that X→A is in F+
    - There is a linear time algorithm to compute this.
  - Then check if Y is in X+
- 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+?

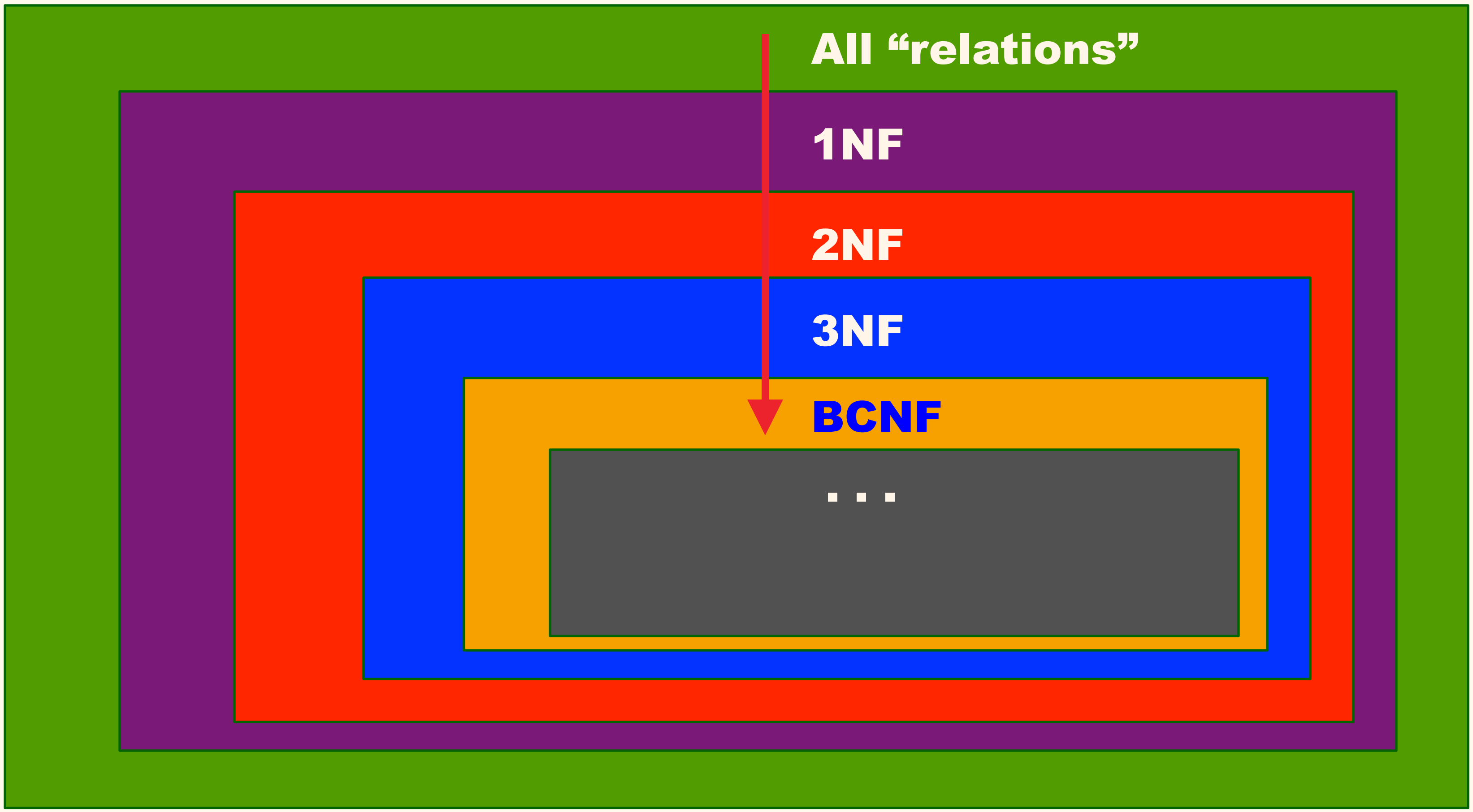
# *FDs & Redundancy*

- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - If no non-trivial FDs hold: There is no redundancy here.
    - Given A → B: Several tuples could have the same A value – and if so, then they'll all have the same B value as well!

# *Normal Forms*

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- We will define various *normal form* (BCNF, 3NF etc.) based on the nature of FDs that hold
- Depending upon the normal form a relation is in, it has different level of redundancy
  - E.g., a BCNF relation has NO redundancy – will be clear soon!
- Checking for which normal form a relation is in will help us decide whether to decompose the relation
  - E.g., no point decomposing a BCNF relation!

# → *Normal Forms* ←



# *Some Terms and Definitions*

- If X is part of a candidate key, we will say that X is a *prime attribute*.
- If X is not part of any candidate key, we will say that X is a *non-prime attribute*.
- If X (an attribute set) contains a candidate key, we will say that X is a *superkey*.
- X → Y can be pronounced as "X *determines* Y", or "Y is *functionally dependent* on X".
- X → Y is *trivial* if $Y \subseteq X$.

# *First Normal Form (1NF)*

- Rel'n R is in 1NF if all of its attributes are atomic.
  - No set-valued attributes! (1NF = "flat" ☺)
  - Usually goes *w/o* saying for relational model (but not for NoSQL systems, as we'll see at the end of the quarter ☺).
  - *Ex:*

| bname | color |
|---|---|
| Interlake | blue, red |
| Clipper | green |
| Marine | red |

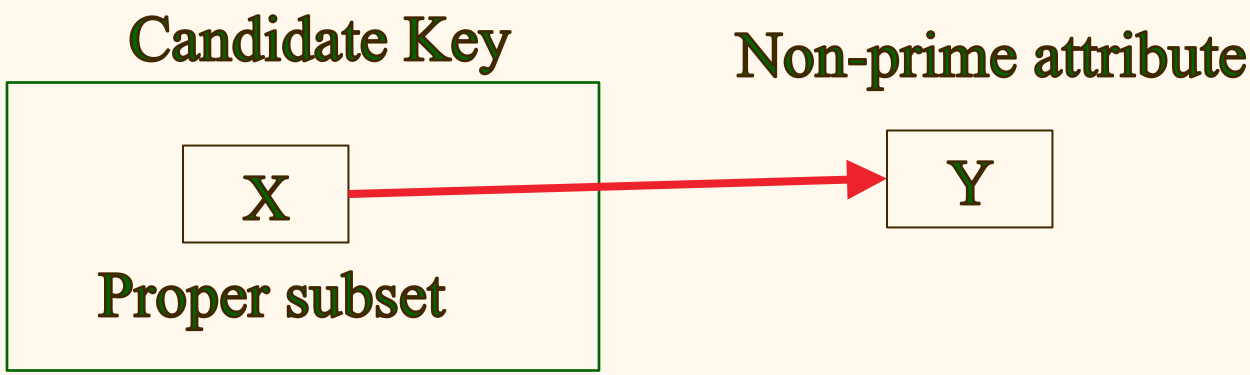
| bname | color |
|---|---|
| Interlake | blue |
| Interlake | red |
| Clipper | green |
| Marine | red |

# *Second Normal Form (2NF)*

Relation R is in 2NF if

- It is in 1NF; and
- Each non-prime attribute is dependent on the *whole* of every candidate key

Violation:



# *Second Normal Form (2NF)*

- *Ex*: Supplies(sno, sname, saddr, pno, pname, pcolor)

  where: sno → sname, sno → saddr, pno → pname, pno → pcolor

  Q1: What are the candidate keys for Supplies?

  Q2: What are the prime attributes for Supplies?

  Q3: Why is Supplies not in 2NF?

  Q4: What's the fix?

Supplier(<u>sno</u>, sname, saddr)

Part(<u>pno</u>, pname, pcolor)

Supply(<u>sno, pno</u>)

A1: (sno, pno)
A2: sno, pno
A3: Each of its four FDs violates 2NF!

**Must not forget this!!**
(Else "lossy join" !!)