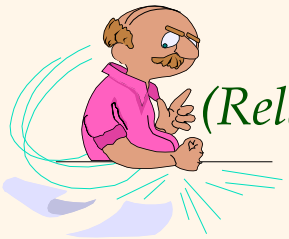
Introduction to Data Management



Lecture #8
(Relational Design Theory, cont.)

Instructor: Mike Carey
mjcarey@ics.uci.edu

Announcements

- Homework stuff
  - HW #2 is due this Friday (before class)
  - HW #3 will be similarly due the following Friday
- Exam stuff (time flies!)
  - Midterm #1 is a week from next Monday (**in class**)!
  - We'll use assigned seating (more info next week), so you'll want to show up a bit early to get settled in
  - An 8.5"x11" 2-sided cheat sheet will be permitted
- Today's plan:
  - Relational DB design theory (*II*)
  - *Disclaimer*: Still not the most exciting CS122A topic… ☺

Reasoning About FDs (Review)

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, c and transitivity)
(g) AD→CD		(d and augmentation)
(h) **AD→E**		(g, c and transitivity)
(i) **AD→C** (j) **AD→D**		(g and decomposition)
(k) AD→BD		(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) ....

Candidate key!

Note: If some attribute *X* is not on the RHS of *any* initial FD, then *X* must be *part of the key*!

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:
  - First compute *attribute closure* of X (denoted **X+**) w.r.t. *F:*
    - Set of all attributes A such that X→A is in F+ (i.e., all F+ attributes)
    - There is a *linear time algorithm* to compute this (look here): start with X and keep adding attributes that can be inferred via the FDs.
  - Then check to see 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+**?

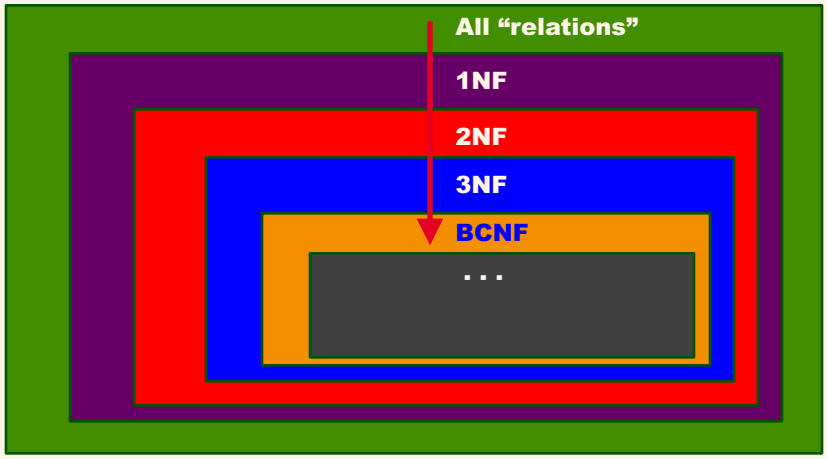
FDs & Redundancy

- Role of FDs in detecting redundancy in a schema:
  - Consider a relation R with 3 attributes, ABC.
    - If **no** (non-trivial) FDs hold: There is no redundancy here then. (Think about this – in fact, think hard...!)
      - ***Ex:*** Prescriptions(doc_name, patient_name, drug_name)
    - 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! (Thus if A is repeated for some reason, it will always have the same B "tagging along for the ride".)
      - **Ex:** Employee(emp_name, dept_no, mgr_name)
        if dept_no → mgr_name

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 forms* (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 – clearer soon!
- Checking for which normal form a relation is in will help us decide whether to decompose the relation
  - E.g., no point in decomposing a BCNF relation!

Normal Forms



Some Terms and Definitions (Review)

- If X is part of a (candidate) key, we will say that X is a *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".
- Some types of dependencies (*on a key*):
  - *Trivial*: **XY → X**
  - *Partial*: XY is a key, **X → Z**
  - *Transitive*: X → Y, Y → Z, Y is non-prime, **X → Z**

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

(1NF is different than the other normal forms.)

Second Normal Form (2NF)

- Rel'n R is in 2NF if it is in 1NF and no non-prime attribute is *partially* dependent on a candidate key of R.
- *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?

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

Supplier(<u>sno</u>, sname, saddr)
Part(<u>pno</u>, pname, pcolor)
Supply(<u>sno, pno</u>)

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

Third Normal Form (3NF)

- Rel'n R is in 3NF if it is in 2NF and it has no *transitive* dependencies to non-prime attributes.
- *Ex*: Workers(eno, ename, esal, dno, dname, dfloor)

  where: eno→ename, eno→esal, eno→dno, dno→dname, dno→dfloor

  Q1: What are the candidate keys for Workers?

  Q2: What are the prime attributes for Workers?

  Q3: Why is Workers not in 3NF?

  Q4: What's the fix?

  A1: eno

  A2: eno

  A3: Two inferable FDs, eno→ dname and eno→dfloor, each violate 3NF.

  Emp(eno, ename, esal, dno)

  Dept(dno, dname, dfloor)

  Don't forget this!

  (Else "lossy join" !!)

  Note: *A lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations is* ***always possible****.*