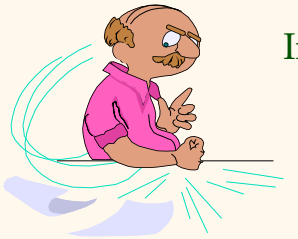
Introduction to Data Management

Lecture #10
(DB Design Wrap-up)



Instructor: Mike Carey
mjcarey@ics.uci.edu

Announcements

- HW and exams:
  - HW #3 in progress (shorter than HW #1, #2)
  - Midterm Exam 1 in one week (at this time)
- Today's plan:
  - Relational DB design theory, Episode 3
  - Relational query languages, Episode 1
  - This week's evening discussion session plan is
    - Relational DB design quiz/discussion
    - Q&A about our given HW #2 solution
    - (All at your scheduled time and place ☺)

Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP → C and SD → P.
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking JP → C requires a join!
- Dependency preserving decomposition (intuitive):
  - If R is decomposed into X and Z, and we enforce the FDs that hold on X and on Z, then all FDs that were given to hold on R must also hold. *(Avoids Problem (3).)*
- *Projection of set of FDs F*: If R is decomposed into **X**, ... projection of F into **X** (denoted F_X) is the set of FDs U → V in F^+ (***closure*** *of F*) where U,V are both in **X**.

Part ID

Department ID

ContractID

Project ID

SupplierID

Dependency Preserving Decomp. *(Cont'd.)*

- The decomposition of R into X and Y is *dependency preserving* if $(F_X \text{ union } F_Y)^+ = F^+$
  - 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, they ***imply*** all dependencies in F^+.
- Important to consider $\mathbf{F}^+$, not F, in this definition:
  - R(ABC), A→B, B→C, C→A, decomposed into AB and BC.
  - Is this dependency preserving? (Is C→A preserved?)
- Dependency preserving does *not* imply lossless join:
  - R(ABC), A→B, if decomposed into AB and BC. *(Q: Key?)*
- And vice-versa! (So we need to check for both.)
  - Must make sure **some** relation contains a **key** for R **(!!)**

Decomposition into BCNF

- Consider relation R with FDs F. If X → Y violates BCNF, decompose R into R–Y and XY. *(R-Y has X still!)*
  - Repeated application of this idea will yield a collection of relations that are BCNF, a lossless join decomposition, and guaranteed to terminate. *(Didn't say dependency preserving...)*
  - E.g., CSJDPQV, C→CSJDPQV, JP→C, *SD→P*, *J→S*.
  - To deal with *SD→P*, decompose into SDP, CSJDQV.
  - To deal with *J→S*, decompose CSJDQV into JS and CJDQV.
- In general, several dependencies may cause violations of BCNF. The order in which we "deal with them" can lead to different sets of relations!

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.
  - E.g., R(CSZ) with CS → Z, Z → C.
  - Can't decompose preserving 1st FD, not in BCNF.
- Decomposition of CSJDPQV into SDP, JS and CJDQV is *not* dependency preserving (*w.r.t.* the FDs JP → C, SD → P and J → S).
  - However, it ***is*** a lossless join decomposition.
  - In this case, ***adding*** JPC to the collection of relations would give us a dependency preserving decomposition.
    - But: JPC tuples stored only for checking FD! (*Redundancy!*)

Decomposition into 3NF

- Obviously, the algorithm for lossless join decomp into BCNF can also be used to obtain a lossless join decomp into 3NF (and might possibly stop earlier).
- To ensure dependency preservation, one idea:
  - If X→Y is not preserved, add relation XY.
  - Problem is that XY may violate 3NF! E.g., consider the addition of CJP to "preserve" JP→ C. What if we also have J→C ? (Which of course implies JP→ C.)
- The fix: Instead of using 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, i.e., $F^+ = G^+$.
  - Right hand side (RHS) of each FD in G is a *single* attribute.
  - If we modify G by deleting an FD or by deleting attributes from a LHS of a FD in G, the closure changes. (So we can't!)
- Intuitively, every FD in G is needed, and G is "*as small as possible*" to have the same closure as F.
- *E.g.*, A→B, ABCD→E, EF→GH, ACDF→EG has the following minimal cover:
  - A→**B**, ACD→**E**, EF→**G** and EF→**H**
- *M.C.*→ lossless-join, dep. pres. 3NF decomp! *(See book!)*

Obtaining that 3NF Decomposition

- Compute the minimal cover G, which is sometimes denoted F- *(see book for how)*
- Search for dependencies in F- with the same attribute set on the left hand side, α:
  - $\alpha \rightarrow Y1$, $\alpha \rightarrow Y2$, ... $\alpha \rightarrow Yk$
  - Construct one relation as (α ,Y1, Y2, ...Yk)
  - Repeat this process for all the FDs' α's
  - Construct a relation with any leftover attributes
  - If none of the relations contains a candidate key for the original relation R, add one *more* relation containing the attributes of a candidate key for R. *(Q: Why?)*

An Example (…that should help!)

- Analyze the given table/comments for FDs and keys and normalize it if need

(ssn, did) → ssn, name, lot, since, did, dname, budget

Workers (ssn, name, lot, since, did, dname, budget)
PRIMARY KEY (ssn, did)

ssn → name, lot

(ssn, did) → since

did → lot

a. Every worker has their own unique ssn.
b. Workers also have a name and an assigned parking lot.
c. Department ids are unique for departments.
d. A department has a name and a budget.
e. An employee works in a department (as of some starting date).
f. If two employees are in the same department their lots are the same.
g. Different departments with the same name always have different budgets.

did→ dname, budget

(dname, budget) → did

An Example (continued) …

- To summarize: FDs
  - (ssn, did) → ssn, name, lot, since, did, dname, budget
    - Primary Key from schema definition
  - ssn → name, lot [from a & b]
  - did → dname, budget [from c & d]
  - (ssn, did) → since [from e]
  - did → lot [from f]
  - (dname, budget) → did [from g]

- What are the candidate keys here?

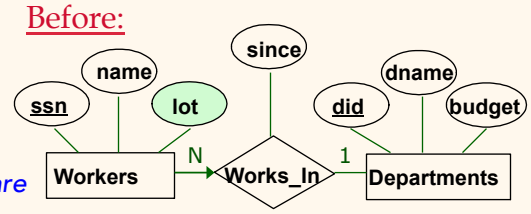
  (ssn, did) and **(ssn, dname, budget)**

- What NF is this in? 1NF? 2NF?

On Refining an ER Based Design

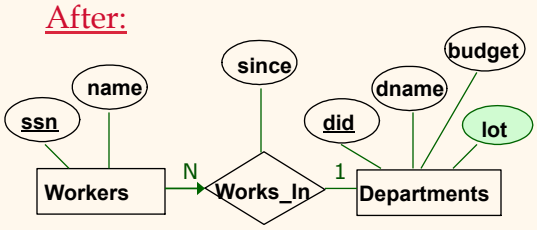
- 1st diagram translated:
  Workers(S,N,L,*D*,S)
  Departments(D,M,B)
  - Lots associated with workers.
- *Suppose all workers in a dept are assigned the same lot: D → L ….*
- Redundancy; fixed by:
  Workers2(S,N,*D*,S)
  WorkersLots(D,L)
  Departments(D,M,B)
- Can further fine-tune this:
  Workers2(S,N,*D*,S)
  Departments(D,M,B,L)

Before:



Notice: Lot **wasn't** really a "Worker attribute"!

After:



DB Design Theory Summary

- If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good goal.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Are all FDs preserved? If a lossless-join, dependency-preserving decomposition into BCNF is not possible (or unsuitable for typical queries), consider 3NF instead.
  - Decompositions should be carried out while also keeping *performance requirements* in mind.