Introduction to Data Management

Lecture #10
(Relational Design Theory, cont.)

Instructor: Mike Carey
mjcarey@ics.uci.edu

Announcements

- Homework stuff
  - HW #3 is due on Wednesday, not Friday
  - HW #4 will come out on Friday (after the exam)
- Exam stuff (time flies!)
  - Midterm #1 is on Friday (in class)
  - We’ll use assigned seating – come early!
  - You may bring an 8.5”x11” (2-sided) cheat sheet
- Today’s plan:
  - Relational DB design theory (IV & Final!)
  - Good news: This is really the end!…😊
Reminder: Normal Forms

Dependency Preserving Decomposition

- The decomposition of $R$ into two tables $X$ and $Y$ is dependency preserving if $(F_X \cup F_Y)^+ = F^+$
  - I.e., if the resulting set of checkable FDs still implies what the initial set of FDs implied

- Important to consider $F^+$, not $F$, in this definition:
  - $\text{Ex}$: $\text{EmpDeptMix}(\text{eid, email, ename, did, dname})$ with $\text{eid} \rightarrow \text{email, email} \rightarrow \text{eid, eid} \rightarrow \text{ename, email} \rightarrow \text{did, did} \rightarrow \text{dname}$
    - $\text{Emp}(\text{eid, email, ename}) - \text{eid} \rightarrow \text{email, email} \rightarrow \text{eid, eid} \rightarrow \text{ename}$
    - $\text{Dept}(\text{did, dname}) - \text{did} \rightarrow \text{dname}$
    - $\text{Work}(\text{eid, did}) - \text{eid} \rightarrow \text{did}$ (instead of $\text{email} \rightarrow \text{did}$)
      - I.e., we can enforce $\text{email} \rightarrow \text{did}$ by enforcing $\text{email} \rightarrow \text{eid}$ on Emp and $\text{eid} \rightarrow \text{did}$ on Work (using PK and UNIQUE constraints appropriately)

- (Remember: Dependency preserving does not imply lossless join, so you need to check for both)
Decomposing a Design into BCNF

- Consider a relation R with FDs F. If $X \rightarrow Y$ 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...)

- Ex: CSJDPQV with $C \rightarrow CSJDPQV$, $JP \rightarrow C$, $SD \rightarrow P$, and $J \rightarrow S$.
  - To deal with $SD \rightarrow P$, decompose into $SDP$, $CSJDQV$.
  - To deal with $J \rightarrow S$, decompose $CSJDQV$ into $JS$ and $CJDQV$.

- Note that in general, several of the dependencies may cause violations of BCNF. (And the order in which we process them can lead to different decompositions … only some of which may be dependency preserving!)

BCNF and Dependency Preservation

- In general, there simply may not be a dependency preserving decomposition into BCNF.
  - E.g., R(CSZ) with $CS \rightarrow Z$, $Z \rightarrow C$.
  - Can’t decompose preserving the first FD; not in BCNF...

- Consider again decomposing the relation CSJDPQV into relations SDP, JS and CJDQV:
  - Not dependency preserving (w.r.t. $JP \rightarrow C$, $SD \rightarrow P$, $J \rightarrow S$).
  - However, it is indeed a lossless join decomposition.
  - In this case, adding JPC to the collection of relations would give us a dependency preserving decomposition.
    - But: JPC data would be used only for FD checking! (Redundancy!)
Decomposition into 3NF

- The lossless join decomposition algorithm for BCNF can also be used to obtain a lossless join decomposition into 3NF (and might stop earlier).
- One idea to ensure dependency preservation:
  - If \( X \rightarrow Y \) is not preserved in the BCNF decomposition, add relation \( XY \).  
  - Problem is that \( XY \) may violate 3NF (or even 2NF), so this approach won’t work in general.
- The real fix: Instead of using the given set of FDs \( F \) to guide the decomposition, use a minimal cover for \( F \).

Minimal Cover for a Set of FDs

- **Minimal cover** \( G \) for a set of FDs \( F \) such that:
  - Closure of \( G \) = closure of \( F \), i.e., \( G^+ = F^+ \).
  - Right hand side (RHS) of each FD in \( G \) is a single attribute.
  - If we change \( G \) by deleting any FD or deleting attributes from the LHS of any FD in \( G \), the closure would change.
- Intuitively: Every FD in \( G \) is needed, with \( G \) as “as small as possible” to have the same closure as \( F \).
- **E.g.**, \( A \rightarrow B \), \( ABCD \rightarrow E \), \( EF \rightarrow GH \), \( ACDF \rightarrow EG \) has the following minimal cover:
  - \( A \rightarrow B \), \( ACD \rightarrow E \), \( EF \rightarrow G \) and \( EF \rightarrow H \)
- **M.C. \( \rightarrow \) lossless-join, dep. pres. 3NF decomposition!**
Computing the Minimal Cover

1. Put the set of given FDs in a Standard Form.
   - This turns F into a set G of equivalent FDs with a single attribute on the right-hand side.
2. Minimize the left-hand side of each FD in G.
   - For each FD in G, check each LHS attribute to see if it can be deleted without breaking the equivalence G+ = F+.
3. Delete redundant FDs.
   - For any FDs that remain, check to see if it can be deleted without breaking the equivalence G+ = F+.

And voila – you now have a minimal cover for F…!

Obtaining that 3NF Decomposition

I. Compute the minimal cover G (which is also sometimes denoted as F-).
II. Search for dependencies in F- that have the same attribute set on their left hand side, α:
   a. α→Y1, α→Y2, ..., α→Yk
   b. Construct one relation as (α,Y1, Y2, ...Yk )
   c. Repeat this process for all of the FDs’ α’s
   d. If none of the relations from above contains a candidate key for the original relation R, add one more relation with (just) the attributes of a candidate key for R.
   (Q: Why...?)
Testing Your Understanding…

- Now that you now how to compute BCNF and 3NF decompositions, try it on our earlier examples!

  - $\neq 2NF$: Supplies($sno$, $sname$, $saddr$, $pno$, $pname$, $pcolor$)
    with: $sno \rightarrow sname$, $sno \rightarrow saddr$, $pno \rightarrow pname$, $pno \rightarrow pcolor$

  - $\neq 3NF$: Workers($eno$, $ename$, $esal$, $dno$, $dname$, $dfloor$)
    with: $eno \rightarrow ename$, $eno, ename \rightarrow esal$, $eno \rightarrow dno$, $dno \rightarrow dname$,

  - $\neq BCNF$: Supply2($sno$, $sname$, $pno$)
    with: $sno \rightarrow sname$, $sname \rightarrow sno$

Note: I changed the $\neq 3NF$ example’s FDs to be equivalent to our earlier FDs but messier to better illustrate the nature of the minimal cover algorithm’s operation.

Testing Your Understanding (cont.)…

- $\neq 3NF$:
  Workers($eno$, $ename$, $esal$, $dno$, $dname$, $dfloor$)
  with: $eno \rightarrow ename$, $eno, ename \rightarrow esal$, $eno \rightarrow dno$, $dno \rightarrow dname$,

$3NF$ M.C. step 1: $\rightarrow$ $3NF$ M.C. step 2:
  * $eno \rightarrow ename$
  * $eno, ename \rightarrow esal$ $\rightarrow$ $eno \rightarrow esal$
  * $eno \rightarrow dno$
  * $dno \rightarrow dname$
  * $dno \rightarrow dfloor$

$3NF$ step II:
  * Emp($eno$, $ename$, $esal$, $dno$)
  * Dept($dno$, $dname$, $dfloor$)

Q1: What is the attribute closure of $eno$ – and what does that mean...?
We got lucky! (No lossy join!)

Q2: What if the Emp-Dept relationship had been M:N?
Testing Your Understanding (cont.)...

- ≠ 3NF:
  Workers(eno, ename, esal, dno, dname, dfloor)
  
  with: eno \rightarrow ename, eno, ename \rightarrow esal, eno \rightarrow dno, dno \rightarrow dname, dfloor

  - eno \rightarrow ename
  - eno, ename \rightarrow esal \rightarrow eno \rightarrow esal
  - eno \rightarrow dno
  - dno \rightarrow dname
  - dno \rightarrow dfloor

  Q1: What is the attribute closure of eno – and what does that mean...?

  \{eno\}

  \{eno, ename\}

  \{eno, ename, esal\}

  \{eno, ename, esal, dno\}

  \{eno, ename, esal, dno, dname\}

  \{eno, ename, esal, dno, dname, dfloor\}

  \rightarrow That’s everything in Workers!  (Therefore…?)

Testing Your Understanding (cont.)...

- ≠ 3NF:
  Workers(eno, ename, esal, dno, dname, dfloor)
  
  with: eno \rightarrow ename, eno, ename \rightarrow esal, eno \rightarrow dno, dno \rightarrow dname, dfloor

  - eno \rightarrow ename
  - eno, ename \rightarrow esal \rightarrow eno \rightarrow esal
  - eno \rightarrow dno
  - dno \rightarrow dname
  - dno \rightarrow dfloor

  Q2: What if the Emp-Dept relationship had been M:N?

  Emp(eno, ename, esal, dno)

  Dept(dno, dname, dfloor)

  Works(eno, dno)

  Else we’d have a lossy join…!
Relational Design Theory Summary

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

On Refining ER Based Designs

- 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: \( D \rightarrow L \) ....
- 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\((D,M,B,L)\)

Notice: Lot wasn’t really a “Worker attribute”!

![Diagram](image.png)
PS: On Refining ER Based Designs

Before:

- 1st diagram translated:
  - Workers(S, N, L, D, S)
  - Departments(D, M, B)
  - Lots assoc.

- Suppose all assigned the same lot: D \rightarrow 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)

Note:

- In many cases the relational translation of an ER design will take you right to 3NF (and BCNF)…!
- Entity key \rightarrow attributes for entity sets.
- Relationship key \rightarrow attributes for relationship sets.
  (But problems could arise with FDs within attributes.)

Questions…?