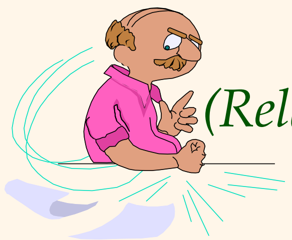
Introduction to Data Management

Lecture #9
(Relational Design Theory, cont.)



Instructor: Mike Carey
mjcarey@ics.uci.edu

It's time again for....

Friday Nights with Databases

Brought to you by…

Announcements

- Homework stuff
  - HW #2 due (albeit late) by tomorrow at 5PM
  - HW #3 is now available on the wiki
- Exam stuff (time flies!)
  - Midterm #1 is a week from Monday (**in class**)
  - We'll use assigned seating (more info next week), so you'll want to arrive early to get settled in
  - An 8.5"x11" 2-sided cheat sheet will be permitted
- Today's plan:
  - Relational DB design theory (*III*)
  - *Good news*: You're almost there… ☺

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(<u>eno</u>, ename, esal, dno)
  Dept(<u>dno</u>, dname, dfloor)

  Don't forget this!
  (Else "lossy join" !!)

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

Boyce-Codd Normal Form (BCNF)

- Rel'n R with FDs *F* is in BCNF if, for all X $\rightarrow$ A in *F+*
  - A $\in$ X (***trivial* FD**), or else
  - *X is a superkey* (i.e., **contains a key**) for R.
- In other words, R is in BCNF if the *only* non-trivial FDs that hold over R are *key constraints*! (i.e., *key* $\rightarrow$ *attr)*
  - Everything depends on "the key, the whole key, and nothing but the key" (so help me Codd ☺)

整个candidate key，是后面每一个attribute的充分必要条件

key之间没有关系，attribute之间没有关系

Boyce-Codd Normal Form (Cont'd.)

- *Ex*: Supply2(sno, sname, pno)

  Given FDs: sno $\rightarrow$ sname, sname $\rightarrow$ sno

  Q1: What are the candidate keys for Supply2?

  Q2: What are the prime attributes for Supply2?

  Q3 Is Supply2 in 3NF?

  Q4: Why is Supply2 not in BCNF?

  Q5: What's the fix?

  Supplier2(sno, sname)

  Supplies2(sno, pno)

Note: Overlapping...!

A1: (sno, pno), (sname, pno)

A2: sno, sname, pno

A3: Yes, it is in 3NF.

A4: Each of its FDs has a left-hand-side that isn't a candidate key. (Just a part of one.)

Note: *A lossless-join, dependency-preserving decomposition of R into a collection of BCNF relations is* ***NOT always possible.***

因为她的non-prime也可以指向key，当prime也可以指向key的时候就不是dependcy-preserving

3NF Revisited (Alternative Def'n)

- Rel'n R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in F+
  - $A \in X$ (*trivial* FD), or else
  - X is a superkey (i.e., contains a key) for R, or else
  - A is part of some key for R (i.e., it's a prime attribute).
- If R is in BCNF, clearly it is also in 3NF.
- If R is in 3NF, some redundancy is possible. 3NF is a compromise to use when BCNF isn't achievable (e.g., no "good" decomp, or performance considerations).
  - ***Important***: *A lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations is* ***always*** *possible.*

Reminder:

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

- Problems due to $R \rightarrow W$:
  - *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 or her rating?
  - *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Reminder:

- 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 ssn	N name	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

lot

rating

hrly_wages

hrs_worked

Reminder:

- 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

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

How about two smaller tables?

Decomposition of a Relation Scheme

- Suppose a relation R contains attributes *A1 ... An.* A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation contains a subset of the attributes of R (and no attributes that did not appear in R ☺), and
  - Every attribute of R appears as an attribute of at least one of the new relations.
- Intuitively, decomposing R means we will store instances of the relations from the decomposition ***instead*** of instances of R.
- E.g., decompose SNLRWH into RW and SNLRH.

Example Decomposition

- Decompositions should be used only when needed.
  - Suppose **SNLRWH** has 2 FDs: S → SNLRWH and R → W
  - Second FD violates 3NF (W values repeatedly associated with R values). Easiest fix creates a relation RW to store the associations, then removes W from the main schema:
    - I.e.: Decompose SNLRWH into SNLRH and RW.
- The information to be stored consists of SNLRWH tuples. So if we just store the projections of these tuples onto SNLRH and RW, are there potential problems that we should be aware of? ...→

Decompositions: Possible Problems

- There are three potential problems to consider:
  1. Some queries become more expensive.
     - E.g., how much did sailor Joe earn? (W*H now requires a join)
  2. Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation! (If "lossy"...)
     - Not a problem in the SNLRWH example! (thanks to ***R***)
  3. Checking some dependencies may require joining the instances of the decomposed relations.
     - Fortunately, also not a problem in the SNLRWH example.
- *Tradeoff*: Consider these issues vs. the redundancy.

Lossless Join Decompositions

- Decomposition of **R** into **X** and **Y** is *lossless-join* w.r.t. a set of FDs F if, for every instance r that satisfies F:
  - $\pi_X(r) \bowtie \pi_Y(r) = r$ (***Note***: *relational agebra)*
- It is always true that $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - In general, the other direction does not hold! If it does, then the decomposition is called lossless-join.
  - Must ensure that X and Y *overlap*, and that the overlap contains a key for one of the two relations.
- Definition extends to decomposition into 3 or more relations as you would expect.
- *Decompositions **must** be lossless! (Avoids Problem (2).)*

The JOIN operation is denoted by the ⋈ symbol and is used to compound similar tuples from two Relations into single longer tuples. Every row of the first table is joined to every row of the second table. The result is tuples taken from both tables

Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP → C and SD → P.
  - BCNF decomposition: Two tables, CSJDQV and SDP.
  - **Problem:** Checking JP → C now requires a join!
- Dependency preserving decomposition (intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y, 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 Decomposition

- Consider CSJDPQV, C is key, JP → C and SD → P.
  - BCNF decomposition: Two tables, CSJDQV and SDP.
  - **Problem:** Checking JP → C now requires a join!
- Dependency preserving decomposition (intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y, 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**.

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

- The decomposition of R into two tables X and Y is <u>*dependency preserving*</u> 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:
  - <u>*Ex:*</u> EmpDeptMix(eid, email, ename, did, dname) with eid$\rightarrow$email, email$\rightarrow$eid, eid$\rightarrow$ename, email$\rightarrow$did, did$\rightarrow$dname
    - Emp(eid, email, ename) - eid$\rightarrow$email, email$\rightarrow$eid, eid$\rightarrow$ename
    - Dept(did, dname) - did$\rightarrow$dname
    - Work(eid, did) - eid$\rightarrow$did (instead of email$\rightarrow$did)

Must check for both!

- Dependency preserving does ***not*** imply lossless join:
  - <u>*Ex:*</u> ABC with A$\rightarrow$B, if decomposed into AB and BC. (<u>***Q:***</u> Key?)

We stopped here….

- … and we'll finish up FD/NF theory Monday!
- *REMINDER*:
  - The first midterm will be given a week from Monday *IN CLASS*! (Everyone needs to be there, arriving a little early in fact, to take it then!)

Decomposing a Design 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...)*
- <u>*Ex:*</u> CSJDPQV with C→CSJDPQV, *JP*→*C*, *SD*→*P*, and *J*→*S*.
  - To deal with *SD*→*P*, decompose into SDP, CSJDQV.
  - To deal with *J*→*S*, decompose CSJDQV into JS and CJDQV.
- Note that in general, several dependencies may cause violations of BCNF. (And the order in which we deal with them can lead to different sets of relations!)

BCNF and Dependency Preservation

- In general, there simply may not be a dependency preserving decomposition into BCNF.
  - E.g., R(CSZ) with *CS* → *Z*, *Z* → *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* → *C*, *SD* → *P*, *J* → *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 tuples would be used only for checking FD! (*Redundancy!*)

Decomposition into 3NF

- Hmm … 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→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 **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→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!)*

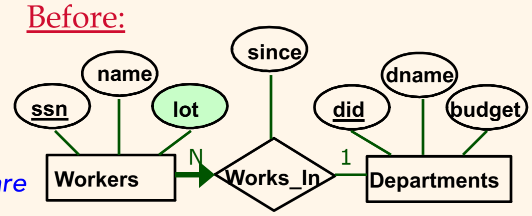
Obtaining that 3NF Decomposition

- Start by computing 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:
  - α→Y1, α→Y2, ... α→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 from R
  - 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?)***

On Refining ER Based Designs

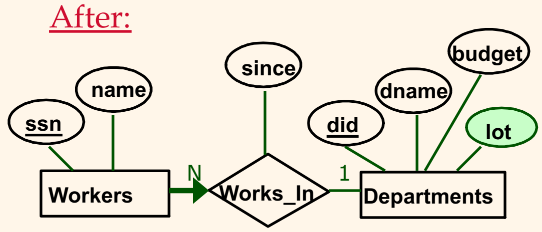
- 1st diagram translated:
  Workers(S,N,L,*D*,S)
  Departments(D,M,B)
  - Lots associated with workers.
- *Suppose all workers in a dept are assigned the same lot: D → L ....*
- Redundancy; fixed by:
  Workers2(S,N,*D*,S)
  WorkersLots(D,L)
  Departments(D,M,B)
- Can further fine-tune this:
  Workers2(S,N,*D*,S)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

Before:



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

After:



Relational 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.