CS122A: Introduction to Data Management

Lecture #13: Relational DB Design Theory (II)

Instructor: Chen Li

Third Normal Form (3NF)

- Relation R is in 3NF if it is in 2NF and it has no *transitive* dependencies to non-prime attributes.

Violation

$$\begin{tikzcd}
\text{Candidate Key} & \text{Non-prime attribute} & \text{Non-prime attribute} \\
\boxed{X} \arrow[r] & \boxed{Y} \arrow[r] & \boxed{Z}
\end{tikzcd}$$

Example

- 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 → A in *F*+
  - A I X (*trivial* FD), or else
  - *X is a superkey* (i.e., contains a key) for R.

Violation

$$\begin{tikzcd}
\text{Not a superkey} & \text{Not a subset of X} \\
\boxed{X} \arrow[r] & \boxed{A}
\end{tikzcd}$$

Boyce-Codd *Normal Form* (*BCNF*)

- R is in BCNF if the *only* non-trivial FDs that hold over R are *key constraints*! (i.e., *key* → *attr*)
  - Everything depends on "the key, the whole key, and nothing but the key" (so help me Codd ☺)

Boyce-Codd Normal Form (Cont'd.)

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

  where: sno → sname, sname → 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.)

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

3NF Revisited (Alternative Def'n)

- Rel'n R with FDs *F* is in 3NF if, for all X → A in F+
  - A I 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.
- 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.*

Decomposition of a Relation Scheme

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

Example Decomposition

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

Reminder:

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?

Decompositions: Possible Problems

- There are three potential problems to consider:
  1. Some queries become more expensive.
     - E.g., how much did sailor Joe earn? (S = 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"...)
     - Fortunately, not a problem in the SNLRWH example!
  3. Checking some dependencies may require joining the instances of the decomposed relations.
     - Fortunately, also not in the SNLRWH example.
- *Tradeoff*: Must consider these issues vs. 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:
  - $\pi_X(r) \bowtie \pi_Y(r) = r$
- 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 in a straightforward way.
- *Decompositions* ***must*** *be lossless! (Avoids Problem (2).)*

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, 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 onto **X** (denoted F_X) is the set of FDs U → V in F^+ (***closure*** *of F*) such that U,V both in **X**.

Dependency Preserving Decomp. (Cont'd.)

- 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 **F^+**, not F, in this definition:
  - R(ABC), $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, decomposed into AB and BC.
  - Is this dependency preserving? (Is $C \rightarrow A$ preserved?)
- Dependency preserving does *not* imply lossless join:
  - R(ABC), only $A \rightarrow B$, if decomposed into AB and BC.
- And vice-versa! (So we need to check for both.)
  - Must make sure **some** relation contains a **key** for R**!!!**