Course Name:
Database Management Systems

Lecture 16
Topics to be covered

- Normalization

NORMAL FORMS

- The normal forms based on FDs are *rst normal form (1NF)*, *second normal form (2NF)*, *third normal form (3NF)*, and *Boyce-Codd normal form (BCNF)*.
- These forms have increasingly restrictive requirements: Every relation in BCNF is also in 3NF,
- every relation in 3NF is also in 2NF, and every relation in 2NF is in 1NF.
- A relation
- is in **first normal form** if every field contains only atomic values, that is, not lists or sets.
- This requirement is implicit in our defition of the relational model.
- Although some of the newer database systems are relaxing this requirement
- 2NF is mainly of historical interest.
- 3NF and BCNF are important from a database design standpoint.

Normal Forms

- Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!
- If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No FDs hold: $\rightarrow$ There is no redundancy here.
    - Given A   B: Several tuples could have the same A value, and if so, they'll all have the same B value!

First Normal Form

- 1NF (First Normal Form)
  - a relation R is in 1NF if and only if it has only single-valued attributes (atomic values)
  - EMP_PROJ (<u>SSN</u>, <u>PNO</u>, HOURS, ENAME, PNAME, PLOCATION)

    PLOCATION is not in 1NF (multi-valued attrib.)
  - solution: decompose the relation

    EMP_PROJ2 (<u>SSN</u>, <u>PNO</u>, HOURS, ENAME, PNAME)

    LOC (<u>PNO</u>, <u>PLOCATION</u>)

Second Normal Form

- 2NF (Second Normal Form)
  - a relation R in 2NF if and only if it is in 1NF and every nonkey column depends on a key not a subset of a key
  - all nonprime attributes of R must be fully functionally dependent on a whole key(s) of the relation, not a part of the key
  - no violation: single-attribute key or no nonprime attribute

Second Normal Form (Contd)

- 2NF (Second Normal Form)
  - violation: part of a key → nonkey

    EMP_PROJ2 (<u>SSN</u>, <u>PNO</u>, HOURS, ENAME, PNAME)

    SSN → ENAME

    PNO → PNAME
  - solution: decompose the relation

    EMP_PROJ3 (<u>SSN</u>, <u>PNO</u>, HOURS)

    EMP (<u>SSN</u>, ENAME)

    PROJ (<u>PNO</u>, PNAME)

Third Normal Form

- 3NF (Third Normal Form)
  - a relation R in 3NF if and only if it is in 2NF and every nonkey column does not depend on another nonkey column
  - all nonprime attributes of R must be non-transitively functionally dependent on a key of the relation
  - violation: nonkey $\rightarrow$ nonkey
  - fig14.10: 2NF & 3NF normalization

Third Normal Form (Contd)

- 3NF (Third Normal Form)
  - SUPPLIER (SNAME, STREET, CITY, STATE, TAX)

    SNAME $\rightarrow$ STREET, CITY, STATE

    STATE $\rightarrow$ TAX (nonkey $\rightarrow$ nonkey)

    SNAME $\rightarrow$ STATE $\rightarrow$ TAX (transitive FD)
  - solution: decompose the relation

    SUPPLIER2 (SNAME, STREET, CITY, STATE)

    TAXINFO (STATE, TAX)

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs *F* is in BCNF if, for all $X \rightarrow A$ in F^+
  - $A \in X$ (called a *trivial* FD), or
  - X 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.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the A value in the other.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

X	Y	A
x	y1	a
x	y2	?

Third Normal Form (3NF)

- Reln R with FDs *F* is in 3NF if, for all X $\longrightarrow$ A in $F+$
  - A $\in$ X (called a *trivial* FD), or
  - X contains a key for R, or
  - A is part of some key for R.
- *Minimality* of a key is crucial in third condition above!
- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomp, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations 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 relation schemes produced by the decomposition, instead of instances of R.
- E.g., Can decompose SNLRWH into SNLRH and RW.

Example Decomposition

- Decompositions should be used only when needed.
  - SNLRWH has FDs $S \longrightarrow SNLRWH$ and $R \longrightarrow W$
  - Second FD causes violation of 3NF; W values repeatedly associated with R values. Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
    - i.e., we 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?

Problems with Decompositions

- There are three potential problems to consider:
  - Some queries become more expensive.
    - e.g., How much did sailor Joe earn? (salary = W*H)
  - Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
    - Fortunately, not in the SNLRWH example.
  - Checking some dependencies may require joining the instances of the decomposed relations.
    - Fortunately, 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, the decomposition is lossless-join.
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem (2).)*

More on Lossless Join

- The decomposition of R into X and Y is lossless-join wrt F if and only if the closure of F contains:
  - $X \cap Y \rightarrow X$, or
  - $X \cap Y \rightarrow Y$
- In particular, the decomposition of R into UV and R - V is lossless-join if $U \rightarrow V$ holds over R.

A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP $\longrightarrow$ C and SD $\longrightarrow$ P.
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking JP $\longrightarrow$ 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 $U \longrightarrow V$ in F^+ (*closure of F*) such that U, V are in X.

Dependency Preserving Decompositions (Contd.)

- 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, these imply all dependencies in F^+.
- Important to consider F^+, not F, in this definition:
  - 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:
  - ABC, $A \rightarrow B$, decomposed into AB and BC.
- And vice-versa! (Example?)

Decomposition into BCNF

- Consider relation R with FDs F. If X $\rightarrow$ Y violates BCNF, decompose R into R - Y and XY.
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDPQV, key C, JP $\rightarrow$ C, SD $\rightarrow$ P, 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
- In general, several dependencies may cause violation of BCNF. The order in which we ``deal with'' them could lead to very different sets of relations!

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.
  - e.g., CSZ, CS $\rightarrow$ Z, Z $\rightarrow$ C
  - Can't decompose while preserving 1st FD; not in BCNF.
- Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP $\rightarrow$ C, SD $\rightarrow$ P and J $\rightarrow$ S).
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - JPC tuples stored only for checking FD! (*Redundancy!*)