Schema Refinement & Normalization Theory

Module 10

Prof. Alex Brodsky

Database Systems

What's the Problem

- Consider relation obtained (call it SNLRHW)

  Hourly_Emps(*ssn, name, lot, rating, hrly_wages, hrs_worked*)

- What if we ***know the FD: R→W holds***?

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

Redundancy

- When part of data can be derived from other parts, we say *redundancy* exists.
  - Example: the hrly_wage of Smiley can be derived from the hrly_wage of Attishoo because they have the same rating and we know rating determines hrly_wage.
- Redundancy exists because of of the existence of *integrity constraints* ***(e.g., FD: R→ W).***

What's the problem, again

- *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!

What do we do? Decomposition

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

=

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

⋈

R	W
8	10
5	7

Functional Dependencies (FDs)

- A functional dependency (FD) has the form: $X \rightarrow Y$, where X and y are two sets of attributes.
  - Examples: $R \rightarrow W$
- The FD $X \rightarrow Y$ ***is satisfied by a relation instance r if:***
  - for each pair of tuples t1 and t2 in r:

    $t1[X] = t2[X]$ implies $t1[Y] = t2[Y]$
  - i.e., given any two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:
  - $ssn \rightarrow did$, $did \rightarrow lot$ implies $ssn \rightarrow lot$
  - $A \rightarrow BC$ implies $A \rightarrow B$
- An FD f is *logically implied by* a set of FDs F, denoted by $F \models f$, if for every relational instance r that satisfies all fd's in F, f is also satisfied.
  - F^+ = *closure of F* is the set of all FDs that are implied by *F*.

Reasoning about FDs

- How do we get all the FDs that are logically implied by a given set of FDs?
- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If $X \supseteq Y$, then $X \rightarrow Y$
  - *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Reasoning About FDs (Contd.)

- Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs!)
- Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs F. An efficient check:
  - Compute *attribute closure* of X (denoted X^+) wrt F:
    - Set of all attributes A such that $X \rightarrow A$ is in F^+
    - There is a linear time algorithm to compute this.
  - Check if Y is in X^+
- Does $F = \{A \rightarrow B,\ B \rightarrow C,\ C\ D \rightarrow E\}$ imply $A \rightarrow E$?
  - i.e, is $A \rightarrow E$ in the closure F^+? Equivalently, is E in A^+?

Computing X^+

- Input F (a set of FDs), and X (a set of attributes)
- Output: Result=X^+ (under F)
- Method:
  - Step 1: Result := X;
  - Step 2: Take $Y \rightarrow Z$ in F, and Y is in Result, do:
    Result := Result *union* Z
  - Repeat step 2 until Result cannot be changed and then output Result.

Example of computing X^+

- F={A →B, AC →D, AB →C}
- X=A
- Result should be X^+=ABCD

Normal Forms

- The first question: Is any refinement needed!
- Normal forms:
  - 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: There is no redundancy here.
    - Given $A \rightarrow B$: Several tuples could have the same A value, and if so, they'll all have the same B value!

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs *F* is in BCNF if, for each non-trivial fd $X \rightarrow A$ in F^+, X is a (super) key for R (i.e., $X \rightarrow R$ in F^+).

- In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.

- If BCNF:
  - No "data" in R can be predicted using FDs alone. Why:
  - Because X is a key,

    we can't have two different tuples that agree on the X value

X	Y	A
x	y1	a
x	y2	?

Decomposition of a Relation Scheme

- When a relation schema is not in BCNF: **decompose**.
- 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 at least 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.

Decomposition example

Original relation
(not stored in DB!)

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

Decomposition
(in the DB)

=

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

$\bowtie$

R	W
8	10
5	7

Problems with Decompositions

- There are three potential problems to consider:
  - ❶ Some queries become more expensive.
    - e.g., How much did sailor Attishoo earn? (salary = W*H)
  - ❷ 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.
  - ❸ 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.

Example of problem 2

Student_ID	Name	Dcode	Cno	Grade
123-22-3666	Attishoo	INFS	501	A
231-31-5368	Guldu	CS	102	B
131-24-3650	Smethurst	INFS	614	B
434-26-3751	Guldu	INFS	614	A
434-26-3751	Guldu	INFS	612	C

$\neq$

Name	Dcode	Cno	Grade
Attishoo	INFS	501	A
Guldu	CS	102	B
Smethurst	INFS	614	B
Guldu	INFS	614	A
Guldu	INFS	612	C

$\rhd\lhd$

Student_ID	Name
123-22-3666	Attishoo
231-31-5368	Guldu
131-24-3650	Smethurst
434-26-3751	Guldu

Lossless Join Decompositions

- Decomposition of R into R_1 and R_2 is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F, we have:

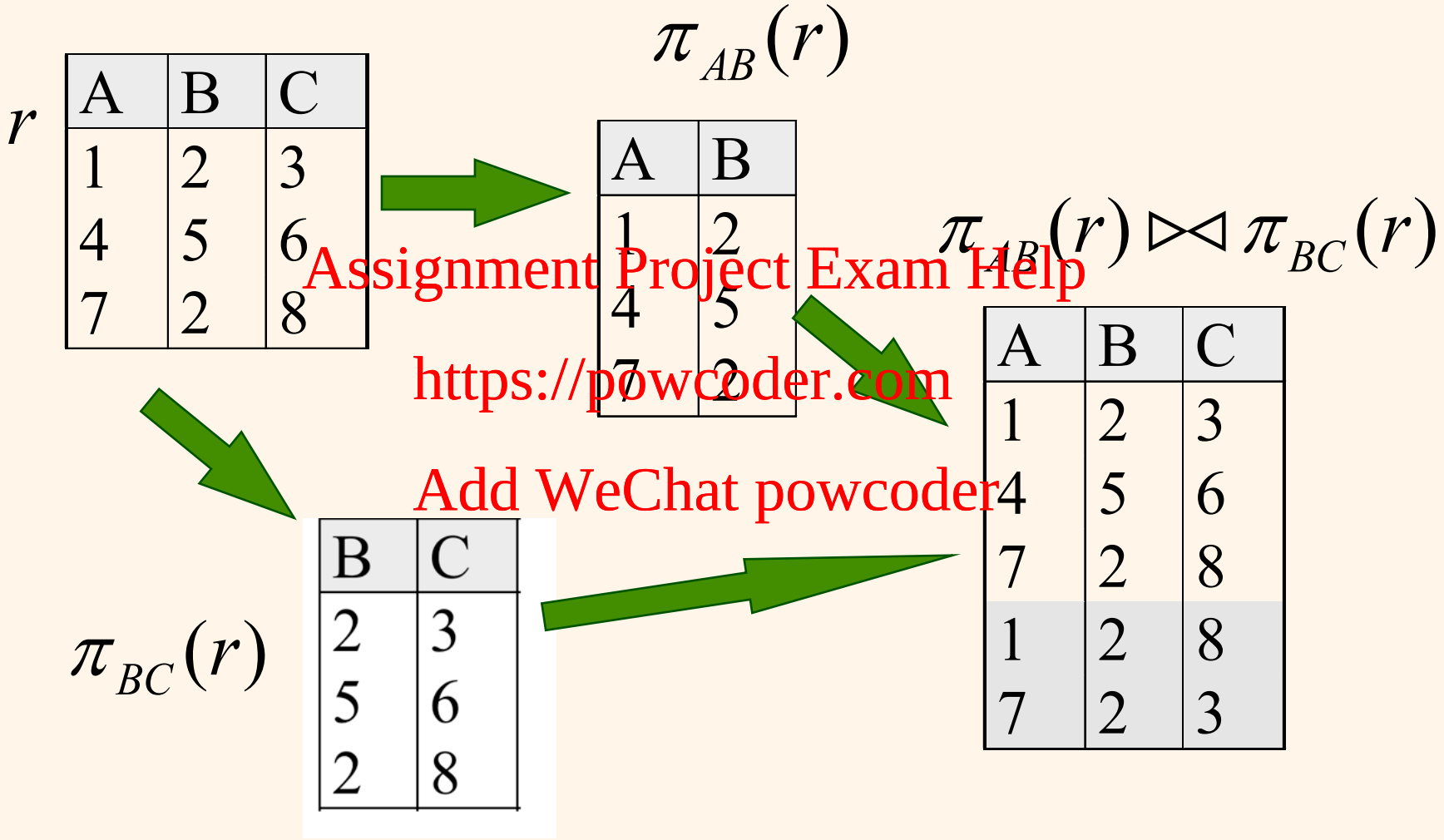
$$\pi_{R_1}(r) \bowtie \pi_{R_2}(r) = r$$

- It is always true that

$$r \subseteq \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$$

- In general, the other direction does not hold! If it does, the decomposition is *lossless-join*.

Example (lossy decomposition)



Example (lossless join decomposition)

r

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

$\pi_{AB}(r)$

A	B
1	2
4	5
7	2

$\pi_{BC}(r)$

B	C
2	3
5	6

$\pi_{AB}(r) \bowtie \pi_{BC}(r)$

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

$We\ have\ (AB \cap BC) \rightarrow BC$

Lossless Join Decomposition

- The decomposition of R into R_1 and R_2 is lossless-join wrt F if and only if F^+ contains:
  - $R_1 \cap R_2 \rightarrow R_1$, or
  - $R_1 \cap R_2 \rightarrow R_2$
- In particular, the decomposition of R into (UV) and (R-V) is lossless join if $U \rightarrow V$ holds on R
  - assume U and V do not share attributes.
  - WHY?

Decomposition

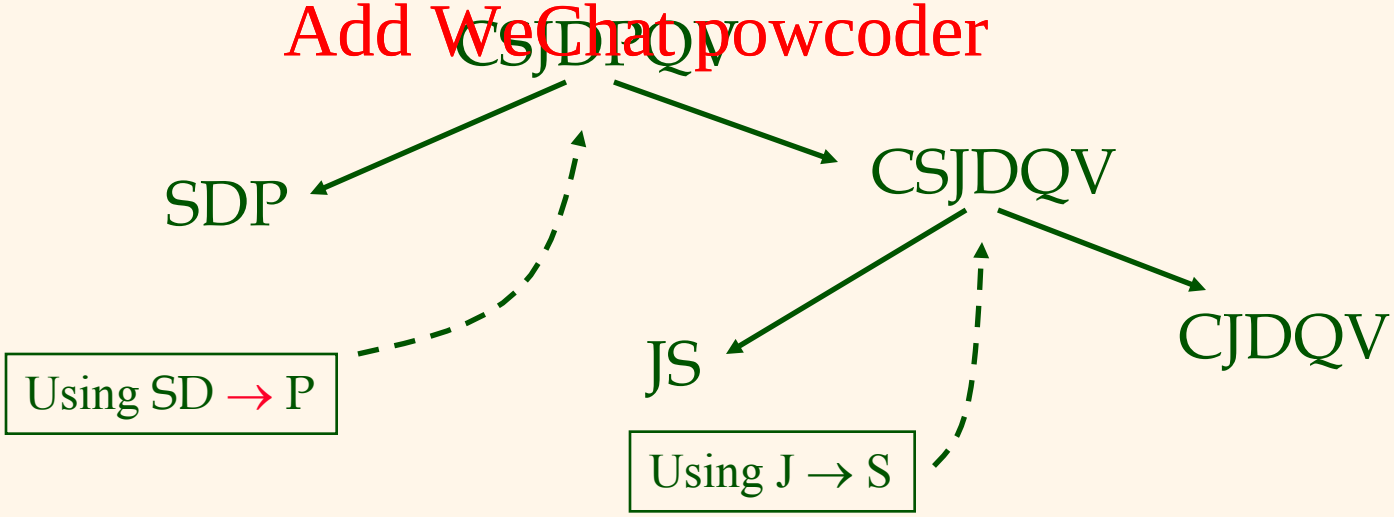
- Definition extended to decomposition into 3 or more relations in a straightforward way.
- *It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem (2).)*

Decomposition into BCNF

- Consider relation R with FDs F. If $X \rightarrow A$ in F^+ over R, violates BCNF, i.e.,
  - XA are all in R
  - A is not in X
  - $X \rightarrow R$ is not in F^+
- Then: decompose R into R - A and XA.
- Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.

BCNF Decomposition Example

- Assume relation schema 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
- A tree representation of the decomposition:



BCNF Decomposition

- In general, several dependencies may cause violations of BCNF. The order in which we ``deal with'' them could lead to very different sets of relations!

How do we know R is in BCNF?

- If R has only two attributes, then it is in BCNF
- If F only uses attributes in R, then:
  - R is in BCNF *if and only if* for each $X \rightarrow Y$ in F (***not F⁺!***), X is a superkey of R, i.e., $X \rightarrow$ R is in F^+ (not F!).
- In general (F may use attributes outside of R! See example earlier for CSJDQV) ,
  - Need to consider all FD $X \rightarrow A$ in F^+ (***not F!).***

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.
- E.g., schema CSZ with FDs: $CS \rightarrow Z$, $Z \rightarrow C$
- Can't decompose while *preserving* $CS \rightarrow Z$, but CSZ is not in BCNF

Dependency Preserving Decomposition

- Consider CSJDPQV, C is key, JP $\rightarrow$ C and SD $\rightarrow$ P.
  - BCNF decomposition: CSJDQV and SDP
  - Problem: Checking JP $\rightarrow$ 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).)*

What FD on a decomposition?

- *Projection of set of FDs F*: If R is decomposed into X, ... the projection of F onto X (denoted F_X) is the set of FDs $U \rightarrow V$ 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?)

Another example

- Assume CSJDQV is decomposed into

  SDP, JS, 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!

Third Normal Form (3NF)

- Reln R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in F^+
  - A in X (i.e., FD is trivial), or
  - X contains a key for R, or
  - A is part of some (candidate) key for R.
- *Minimality* of a (candidate) key is crucial in third condition above!

Third Normal Form (3NF)

- 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'' decomposition, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

What Does 3NF Achieve?

- If 3NF is violated by X→A, one of the following holds:
  - X is a subset of some key K
    - We store (X, A) pairs redundantly.
  - X is not a proper subset of any key.
    - There is a chain of FDs $K \rightarrow X \rightarrow A$, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.
- But: even if reln is in 3NF, these problems could arise.
  - e.g., Reserves SBDC, S →C, C →S is in 3NF, but for each reservation of sailor S, same (S, C) pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.

Decomposition into 3NF

- Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier).
- To ensure dependency preservation, one idea:
  - If $X \rightarrow Y$ is not preserved, add relation XY.
  - Problem is that XY may violate 3NF! e.g., consider the addition of CJP to `preserve' $JP \rightarrow C$. What if we also have $J \rightarrow C$?
- Refinement: Instead of 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.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get 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. Decomp!!! (in book)

Summary of Schema Refinement

- 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 heuristic.
- If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.
  - Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.