Schema Refinement & Normalization Theory

INFS 614

Overview of Normal Forms

- If a relation is placed in a *normal form* (BCNF, 3NF etc.), certain problems are avoided or minimized.
  - redundancy and its associated update/insert/delete anomolies
- Normal forms include: 1NF, 2NF, 3NF, BCNF, 4NF, 5NF, ...
  - we focus on BCNF in this lecture
- Normal forms are achieved by decomposing a schema to isolate certain dependencies
  - we focus only on functional dependencies (FDs)
  - there are other types: e.g., multivalued dependencies (4NF), join dependencies (5NF)

Boyce-Codd* Normal Form (BCNF)

- Reln R with FDs F is in BCNF if, for each $X \rightarrow A$ in F^+ either:
  - 1) $A \subseteq X$ (i.e, A's attributes are all in X, a *trivial* FD), or
  - 2) X is a (super) key for R (i.e., $X \rightarrow R$ is in F^+).
- In other words, R is in BCNF if the only non-trivial FDs involve a key for R.

- If R is in BCNF, there is no redundancy solely due to FD's
- A relation in BCNF corresponds well to an "entity set" or a "relationship set" in a (good) ER design.

* Remember him? (he invented the relational model in 1970)

Testing For BCNF

- The BCNF conditions must hold for ALL FD's!
  - not just those in F, but those in F+

- The hard way:
- 1) generate F+ from F by Armstrong's Axioms
  - a lot of work, potentially!!
- 2) check every FD in F+:
  - is $X \rightarrow Y$ trivial?
  - is X is superkey for R?
- If, for every FD in F+, the answer is "yes" to one of these questions, then R is in BCNF

Testing for BCNF (Easier Way)

- Strategy: For each non-trivial FD $X \rightarrow A$ in F
  - is X a superkey for R?
  - use the attribute closure algorithm: i.e., does X^+ contain all attributes in R?
- Armstrong's Axioms preserve "superkeyness"!
  - (see next slide)
- Thus, for every FD in F+ generated from $X \rightarrow A$, its left hand side is also superkey
  - and thus R is in BCNF
  - and we didn't have to compute F^+ !
- Armstrong's axioms also preserve "trivialness"
  - adding trivial FD's has no impact on BCNF.

Remember These?

- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If $Y \subseteq X$ (i.e., $X \supseteq Y$), 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 $X \rightarrow Z$

Testing for BCNF: Example

- Given ABCD and $F = \{B \rightarrow C, C \rightarrow D, C \rightarrow A\}$
  - (Exercise 19.7 #1)
- Is relation ABCD in BCNF?
  - none of these FDs are trivial (must check all)
  - $B \rightarrow C$: $B^+ = \{ABCD\}$ *OK*
  - $C \rightarrow D$: $C^+ = \{ACD\}$ *Whoops! (Not in BCNF)*
- We will have to decompose ABCD to achieve BCNF!
- *Strong Hint: Know how to test for BCNF!*

Decomposition of a Relation Schema

- When a relation schema is not in BCNF: **decompose**.
- Decompose R into R_1 ... R_n, where
  - each R_i is a projection of R (contains a subset of R's attributes)
  - $\bigcup_{i=1}^{N}$ Atts (R_i) = Atts (R) (the union of the attributes of each R_i = the attributes of R)
  - each R_i is in BCNF
  - intuitively, "rogue dependencies" are broken out into their own tables

Testing for BCNF: Hard Case

- What if the atts of F are not contained entirely within the relation R we are testing?
- This can happen in a decomposition of R:
  - E.g. Consider R_1 (A, B, C, D), with $F = \{ A \rightarrow B, B \rightarrow C\}$
    - Now decompose R_1 into $R_2(A,B)$ and $R_3(A,C,D)$
    - Although neither dependency in F contains only attributes from (A,C,D) R_3 does not satisfy BCNF!
    - Dependency $A \rightarrow C$ in F^+ shows R_3 is not in BCNF.
- To test if a *decomposed relation* R_d is in BCNF:
  - 1) compute the key for R_d,
  - 2) for all (non-trival) FDs $X \rightarrow Y$ in F+ whose attrs are entirely within R_d, is X a superkey for R_d?

Decomposition example

Original relation

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

=

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 at least four potential problems to consider:
  1) The decomposed relation instances are not "join lossless":
     - The original R cannot be recaptured via joins (very bad!)
  2) The decomposed instances are not "dependency preserving"
     - Checking some FDs may require joins
  3) Some queries become more expensive.
     - e.g., How much did Attishoo earn? (earn = W*H, requires a join)
  4) Some decompositions fragment important conceptual entities
     - "overnormalization"; classic example: an address
- Design considerations: Compare these costs vs. tolerating some redundancy (and associated anomolies)
  - BCNF avoids 1), and is conceptually easy.
  - 3NF avoids both 1) and 2), but permits some redundancy
  - sometimes we can use views cleverly (especially materialized) to compensate for 3) and 4)

Example: Join “Lossiness” Problem

I made that word up

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

$\bowtie$

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

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

- It is always true that

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

- The other direction does not always hold!
  - *unless the common (join) attributes of the decomposed tables form a key for at least one of those tables*

Further Example: Lossy

r

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

$\pi_{AB}(r)$

A	B
1	2
4	5
7	2

$\pi_{BC}(r)$

B	C
2	3
5	6
2	8

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

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

Note: B is not a key for AB or BC!

Further Example: Lossless

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

Now, B is a key for BC.

Lossless Join Decomposition

- The decomposition of R into R_1 and R_2 wrt F is join-lossless if:
  - Atts of $(R_1 \cap R_2) \rightarrow R_1$, or
  - Atts of $(R_1 \cap R_2) \rightarrow R_2$
  - I.e.: if the join attributes form a key for R_1 or R_2

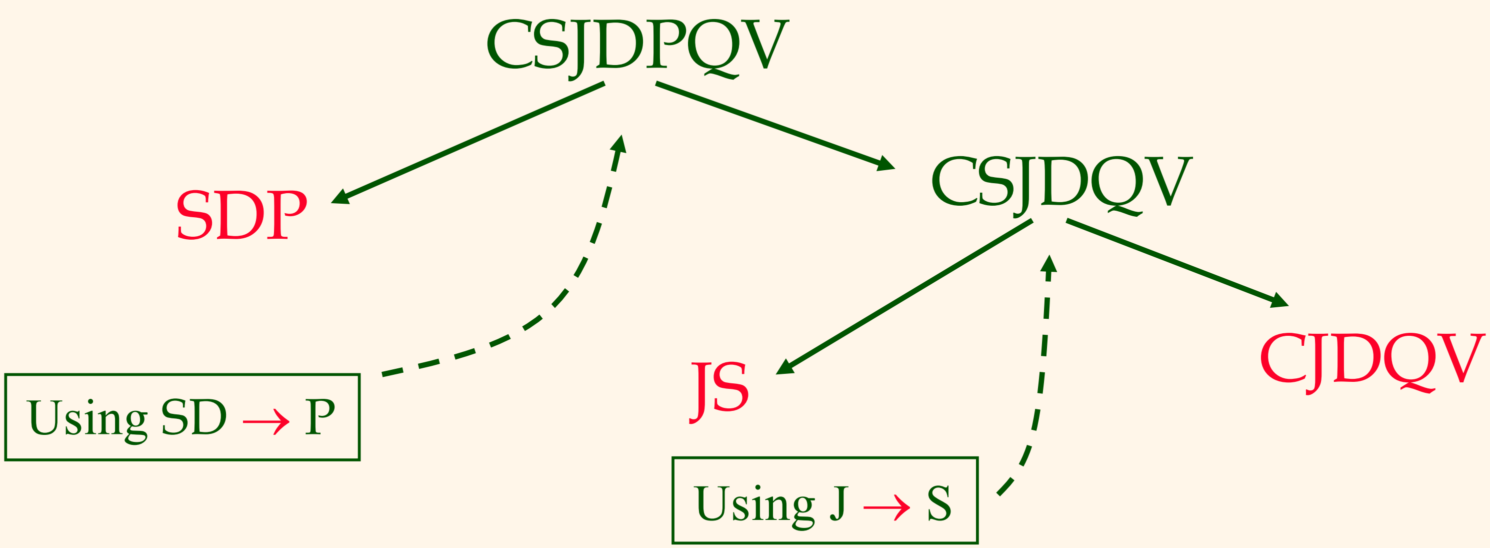
- Important special case: if $X \rightarrow Y$ holds on R, the decomposition of R into (R-Y) and (XY) is join-lossless
  - X and Y do not share attributes.
  - X is common to both tables, and is a key for XY

A Simple "Tree" Algorithm for Decomposition into BCNF

- Consider relation R with FDs F. If $X \rightarrow A$ in F^+ over R, violates BCNF, i.e.,
  - the attributes of XA are all in R, and yet:
  - $X \rightarrow R$ is not in F^+ (X is not a key for R)
  - A is not in X ($X \rightarrow A$ is not trivial)
- Then: decompose R into two new relations
  - XA
  - R - A
- Repeated application of this decomposition:
  - results in a collection of relations that are in BCNF
  - produces a lossless join decomposition
  - is guaranteed to terminate.

BCNF Decomposition Example

- Assume relation schema CSJDPQV
  - key C, JP → C, SD → P, J → S (call the last 2 F_{BAD})
- Remove SD → P from F_{BAD}, and decompose into SDP ("XA"), CSJDQV ("R-A").
  - Test if these are individually in BCNF.
  - If not, recurse! (on either side; see below)
- Remove J → S from F_{BAD}, decompose into JS, CJDQV.
  - We are done now. WHY? (The LHS of all fds in F, and F^+, is a key)
- Note: it helps to draw this tree as you go!



BCNF Decomposition

- In general: several dependencies may cause violation of BCNF.
- The order in which we "deal with" them could lead to different (valid) sets of BCNF relations.

Dependency Preservation

- 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!
  - e.g., does an insertion to SDP violate JP $\rightarrow$ C? How can we tell?
- Dependency preservation problem:
  - If R is decomposed into X, and Y, then all FDs that were given to hold on R should still hold
  - but we can't (easily) enforce them if their attributes are "scattered" across multiple relations!

BCNF and Dependency Preservation

- In general, there may <u>not</u> be a dependency preserving decomposition into BCNF.

The Projection of a set of FDs

- Given F, let R be decomposed into X and Y
- The projection of F onto X (denoted F_X) is the set of FDs $U \rightarrow V$ in F^+ such that the attributes of U, V are entirely in X.

Dependency Preserving Decompositions

- Decomposition of R into X and Y is ***dependency preserving*** if $F^+ = (F_X \text{ union } F_Y)^+$
  - Example: R = ABC, F = {$A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$}, and R is decomposed into AB and BC.
  - Is this dependency preserving? Yes.
  - (Note: Consider F^+, not just F, in definition of F_x and F_y!)
  - By transitivity, F+ also includes $A \rightarrow C$, $B \rightarrow A$, $C \rightarrow B$
  - Thus F_X = {$A \rightarrow B$ and $B \rightarrow A$}
  - Thus F_Y = {$B \rightarrow C$ and $C \rightarrow B$}
  - Thus $(F_X \text{ union } F_Y)^+$ includes $C \rightarrow A$ (by transitivity)
  - So the AB, BC decomposition will enforce/preserve $C \rightarrow A$!!
- Dependency preserving does not imply lossless join:
  - ABC, $A \rightarrow B$, decomposed into AB and BC.
- And vice-versa! (Example?)

Example

- Assume CSJDPQV is decomposed into

  SDP, JS, CJDQV

  this 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.
- Adding table JPC gives us a dependency preserving decomposition.
  - Note: JPC tuples stored only for checking FD!
  - But it is redundant ...
- Is there a general way to ensure dep. pres?

Third Normal Form (3NF)

- R with FDs F is in 3NF if, for all $X \to A$ in F^+
  - A in X (i.e., FD is trivial), or
  - X contains a key for R, or
  - A is part of some (candidate, not super) key for R.
    - a bit weaker than BCNF!
- It is always possible to decompose into 3NF and preserve dependencies
  - and be join-lossless
  - but you may have some FD-induced redundancy
- 3NF is a compromise
  - if you really want dependency preservation
  - or (more likely) if BCNF is just too hard to achieve

Third Normal Form (3NF)

- A relation in 3NF is free of:
  - partial dependencies: e.g. given R(ABC), $AB \rightarrow C$ (AB is a key), $A \rightarrow C$
  - transitive dependencies: e.g. given R(ABC), $A \rightarrow B$, $B \rightarrow C$
- 2NF is free of partial dependencies only

1NF

- First normal form requires every column in a relation to be atomic (i.e. not a repeating group)
- This is the default in the relational model.

- XML DTD's can define non-1NF models.

```
<!DOCTYPE BOOKLIST [
<!ELEMENT LISTNAME (#PCDATA)>
<!ELEMENT LIST (BOOK)*>
    <!ELEMENT BOOK (AUTHOR, TITLE)>
        <!ELEMENT AUTHOR (#PCDATA)>
        <!ELEMENT TITLE (#PCDATA)>
    <!ELEMENT TITLE (#PCDATA)
]>
```

A nested structure (within a book)

An entity set (of books)

Overnormalization

- Address entity:
  - Bob Jones, 12 Main Street, Springfield IL, 12345
- Address (Firstname, Lastname, Number, Street, City, State, Zip)
  - FLNSCTZ
  - NSCT → Z

- Do we *really* want to do this?
  - so what if zip codes are a bit redundant
  - insert, update, delete anomolies are managable!

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.

Three Normalization Lessons

1) Intuitively, the left hand side of every FD wants to be the key of *some* relation.

2) "Perfect" decompositions don't always happen.
   - But you should still know BCNF decomposition for the final :-)

3) You can overnormalize.

Big Picture Takeaways

- What can FDs & normalization do for you?
  - Reduce redundancy, reducing storage costs
  - Improve data consistency (by having only one copy)
  - Improve concurrency (less needs to be locked)
  - Provide insight into any database design (e.g., causal links)
- How could normalization hurt you?
  - Requiring (potentially lossy) joins to put your data back together
  - Require you to understand all FDs before you can design, use your database (big issue in analytic databases).
    - This takes time (doesn't lend itself to rapid prototyping).
  - If you don't know BCNF on the final ☺
- Normalization is a tool (like ER design). Designing great databases is up to you!

Backup

What Does 3NF Achieve?

- If 3NF is violated by $X \rightarrow A$, one of the following holds:
  - X is a proper subset of some key K
    - We store (X, A) pairs redundantly.

      E.g.: SBDC, SBD is the only key, $S \rightarrow C$. Pairs (S,C) redundant.
  - 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.

      E.g.: Hourly_Emps SNLRWH, S is the only key, $R \rightarrow W$.
- But: even if reln is in 3NF, these problems could arise.
  - e.g., Reserves SBDC, $S \rightarrow C$, $C \rightarrow S$ is in 3NF, but for each reservation of sailor S, same (S, C) pair is stored.
- Thus, 3NF is indeed a compromise with respect to BCNF.

Testing for 3NF

- Optimization: Need to check only FDs in *F*, need not check all FDs in F^+.
- Use attribute closure to check, for each dependency $X \rightarrow A$, if X is a superkey.
- If X is not a superkey, we have to verify if A is part of some candidate key of *R*
  - this test is rather expensive, since it involves finding candidate keys

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!
- 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 $EF \rightarrow H$

Dependency-Preserving Decomposition into 3NF

- Given R with a set F of FDs that is a minimal cover. Let R1,…,Rn be a lossless-join decomposition of R in 3NF. Let Fi be the projection of F onto Ri. Do:
  - Identify the set N of dependencies in F that is not preserved, i.e. not included in the closure of the union of F1,…Fn;
  - For each FD $X \rightarrow A$ in N, create a relation schema XA and add it to the decomposition of R.
- The resulting decomposition is a lossless-join and dependency-preserving decomposition of R into 3NF relations.

Problem

- Given a relation schema R(ABCDE) with FD's

  $F = \{A \rightarrow B, C \rightarrow D, B \rightarrow AE\}$

Find all the candidate keys for R.

Problem (contd.)

Proceed as follows:

(a) Identify the attributes that are in the relation schema and not in the set of functional dependencies: None.

(b) Identify the attributes that appear only on the left hand-side in F (*these attributes will belong to any single key of the relation schema*): C.

(c) Identify the attributes that appear only on the right hand-side in F (*these attributes are not part of any key*): DE.

(d) Combine the set of attributes obtained in (a) and (b) with the attributes that are <u>not</u> obtained in (c). Compute their closure to check whether they are a key.

Problem (contd.)

- From (a) and (b), we get C;
- The attributes not obtained in (c) are: A,B,C.
- We compute the closure of C: C+ = CD. Thus: *C is not a key*.
- We compute the closure of CA: CA+ = CABDE. Thus: *CA is a key*.
- We compute the closure of CB: CB+ = CBDAE. Thus: *CB is a key*.
- Note: We know from (c) that CD and CE are not keys;
- Then, we consider combinations of three attributes that don't contain CA or CB (since they are keys), and include C. The only possible combination is CDE, but D and E are not part of any key.
- Therefore, CA and CB are the only keys in R.