CS-GY 6083 A: Principles of Database Systems

Schema Refinement and Normal Forms

supplementary material:
"Database Management Systems" Sec. 19.1-19.6
class notes

Prof. Julia Stoyanovich
Computer Science and Engineering at Tandon
Center for Data Science
New York University

Motivating Example

Hourly_Employees (<u>ssn</u>, name, rating, wages, hours)

<u>ssn</u>	name	rating	wages	hours
123-22-3666	Ann	5	10	40
231-31-5368	Bob	5	10	30
131-24-3650	Cate	2	7	30
434-26-3751	Dan	2	7	32
612-67-4134	Eve	5	10	40

Observe: the value in the **wages** column is determined by the values in the **rating** column

Redundancy!

Why is this a problem?

The Evils of Redundancy

- Redundancy is at the root of several problems associated with relational schemas
  - redundant storage
  - update anomalies: if one copy with redundant info is updated and other copies are not
  - insertion anomalies: cannot insert a tuple for an employee unless we know the hourly wage corresponding to his rating
  - deletion anomalies: if we delete all tuples with rating 5, we won't know the hourly wage for that rating
- Integrity constraints (ICs) can be used to identify schemas with such problems and to suggest refinements
- Main refinement technique: decomposition

Decomposition

Hourly_Employees (<u>ssn</u>, name, rating, wages, hours)

<u>ssn</u>	name	rating	wages	hours
123-22-3666	Ann	5	10	40
231-31-5368	Bob	5	10	30
131-24-3650	Cate	2	7	30
434-26-3751	Dan	2	7	32
612-67-4134	Eve	5	10	40

Hourly_Employees2 (<u>ssn</u>, name, rating,hours) Rating_Wages(rating, wages)

<u>ssn</u>	name	rating	hours
123-22-3666	Ann	5	40
231-31-5368	Bob	5	30
131-24-3650	Cate	2	30
434-26-3751	Dan	2	32
612-67-4134	Eve	5	40

<u>rating</u>	wages
1	6
2	7
3	8
4	9
5	10

Functional Dependencies

A functional dependency is an integrity constraint (IC) that generalizes the concept of a key

Consider two non-empty sets of attributes of relation *R*:

$$A = \{A_1, A_2, \ldots, A_n\} \quad \text{and} \quad B = \{B_1, B_2, \ldots, B_m\}$$

A functional dependency (FD) $A \rightarrow B$ holds over relation *R* if, for every legal instance of *R*, and for any two tuples t_1 and t_2, if t_1 and t_2 agree on the values of all attributes in ***A***, then they agree on the values of all attributes in ***B***.

(denoting sets of attributes with bold letters, e.g., ***A***)

$$A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m$$

Functional Dependencies

- An FD is a statement about all legal instances of *R*
  - An FD is given, not determined based on the data
  - Given a legal instance of *R*, we can check if it violates some FD, but we cannot tell if an FD holds over all legal instances

In particular, if *K* is a candidate key for *R*, then *K* functionally determines all other attributes in *R.*

Functional Dependencies: Example

Notation: refer to relation *R* by the first letters of its attribute names.

R: <u>K</u>ABCD

K is a key

<u>**K**</u>	**A**	**B**	**C**	**D**
k1	a1	b1	c1	d1
k2	a1	b1	c1	d2
k3	a1	b2	c2	d1
k4	a1	b3	c3	d1

Is this FD satisfied: $K \rightarrow ABCD$ *yes, we were told K is a key*

Is this FD satisfied: $AB \rightarrow C$ *yes*

Is this FD satisfied: $A \rightarrow B$ *no, violated for tuples k3 and k4*

Is this FD satisfied: $AB \rightarrow B$ *yes, known as a trivial FD*

What is *Functional* about FDs?

$$A_1 A_2 \ldots A_n \rightarrow B$$

is called a functional dependency because there is a function that takes a list of values, one for each attribute on the left, and produces a value for the attributes on the right.

Importantly, this function is computed in a specific way: by looking up the value of *B* in a relation.

Example:

Rating_Wages

rating	wages
1	6
2	7
3	8
4	9
5	10

FDs and Redundancy

- Given a schema, we need to decide whether it is well-designed (has no redundancy)
- If a schema does have redundancy, we may need to decompose it
- Decomposition is replacing relation *R* with several relations such that their sets of attributes together include all attributes *R*
- FDs help us reason about redundancy
  - Consider relation *R* with attributes ABC, denoted *R*(A, B, C)
  - If *A* is a candidate key and $A \rightarrow B$, there is no redundancy
  - If *B* is not a key and $B \rightarrow C$, then several tuples could have the same *B* value, so they'll have the same *C* value - redundancy!

Reasoning about FDs

Given a set of FDs that relation *R* satisfies, infer additional FDs that hold in *R.*

Two sets of FDs ***S*** and ***T*** are equivalent if the set of instances satisfying ***S*** is the same as the set of instances satisfying ***T****.*

A set of FDs ***S*** follows from a set of FDs ***T*** if every relation instance that satisfies all the FDs in ***T*** also satisfies all the FDs in ***S***.

Basic Rules

The splitting rule

$A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m$ can be replaced by a set of *m* rules

$A_1 A_2 \ldots A_n \rightarrow B_i$ with $i = 1, 2, \cdots m$

The combining rule

$A_1 A_2 \ldots A_n \rightarrow B_i$ with $i = 1, 2, \cdots m$ can be replaced by

$A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m$

The transitive rule

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \wedge$$
$$B_1, B_2, \ldots, B_m \rightarrow C_1, C_2, \ldots, C_k \Rightarrow$$
$$A_1, A_2, \ldots, A_n \rightarrow C_1, C_2, \ldots, C_k$$

Proof for

$A \rightarrow B \wedge B \rightarrow C \Rightarrow A \rightarrow C$

Proof of the transitive rule

$R(A, B, C)$ $\quad A \to B \wedge B \to C \Rightarrow A \to C$

- Consider tuples $t_1(a, b_1, c_1), t_2(a, b_2, c_2)$
- Since $A \to B$ and $t_1.A = t_2.A = a$, then it must be the case that $t_1.B = t_2.B$, i.e., that $b_1 = b_2$
- Similarly, since $B \to C$, and since $t_1.B = t_2.B$ then $t_1.C = t_2.C$, i.e., $c_1 = c_2$

Closure of a set of attributes

Suppose $\boldsymbol{A} = \{A_1, \ldots, A_n\}$ is a set of attributes and $\boldsymbol{S}$ is a set of FDs.

The *closure* of $\boldsymbol{A}$ under the FDs in $\boldsymbol{S}$ is the set of attributes $\boldsymbol{B}$ s.t. every relation that satisfies all the FDs in $\boldsymbol{S}$ also satisfies

$A \rightarrow B$

We denote the closure of $\{A_1, A_2, \ldots, A_n\}$ by $\{A_1, A_2, \ldots, A_n\}^+$

Note that $\{A_1, A_2, \ldots, A_n\} \subseteq \{A_1, A_2, \ldots, A_n\}^+$ Why?

Computing the closure of a set of attributes

Algorithm *AttributeClosure*

Input: a set of attributes $\{A_1, A_2, \ldots, A_n\}$ and a set of FDs ***S***

Output: the closure $\{A_1, A_2, \ldots, A_n\}^+$

1. Split the FDs of S using the splitting rule, so that each FD has one attribute on the right

2. Initialize $\{A_1, A_2, \ldots, A_n\}^+ \leftarrow \{A_1, A_2, \ldots, A_n\}$

3. Repeatedly search for some FD $B_1, B_2, \ldots, B_m \rightarrow C$ such that

$$\{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\}^+ \wedge C \notin \{A_1, A_2, \ldots, A_n\}^+$$

4. Stop when no more attributes can be added to $\{A_1, A_2, \ldots, A_n\}^+$

Example

$$R(A,B,C,D) \qquad S = \{AB \to C, C \to D, D \to A\}$$

$$\{A,B,D\}^+ = \{A,B,D\} \bigcup_{AB \to C} \{C\} = \{A,B,C,D\}$$

$$\{C,D\}^+ = \{C,D\} \bigcup_{D \to A} \{A\} = \{A,C,D\}$$

Note that B is not in the closure of {CD}. Generally, since B does not appear on the right-hand side of any rule, it will not be in the closure of any set that does not already contain B.

Example

$R(A,B,C,D) \qquad S = \{A \rightarrow B, BC \rightarrow A\}$

$\{A\}^+ =$

$\{B\}^+ =$

$\{B,C\}^+ =$

$\{A,B,C\}^+ =$

$\{B,C,D\}^+ =$

Example (solution)

$R(A, B, C, D) \qquad S = \{A \rightarrow B, BC \rightarrow A\}$

$\{A\}^+ =$

$\{B\}^+ =$

$\{B, C\}^+ =$

$\{A, B, C\}^+ =$

$\{B, C, D\}^+ =$

What can we do with the closure?

Given ***S*** and an FD not in ***S***, we can compute whether that FD follows from ***S***.

Consider $B_1 B_2 \ldots B_m \to C$

1. Compute the closure of the attributes on the left, $\{B_1, B_2, \ldots, B_m\}^+$

2. Check whether the attribute on the right, *C*, is in the closure.
   If so - FD follows from ***S***, otherwise it does not.

More generally, $A_1 A_2 \ldots A_n \to B_1 B_2 \ldots B_m$

follows from ***S*** if and only if $\{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\}^+$

Example

$$R(A,B,C,D) \qquad S = \{AB \to C, C \to D, D \to A\}$$

Check whether the following FDs follow from ***S***:

Compute the closure of the left-hand side, check whether the attribute on the right-hand side is in the closure. If so - yes, if not - no.

$C \to A \qquad \{C\}^+ = \{A,C,D\}$ yes

$CD \to B \qquad \{C,D\}^+ = \{A,C,D\}$ no

$AB \to D \qquad \{A,B\}^+ = \{A,B,C,D\}$ yes

Correctness of the closure algorithm

The algorithm is sound: it computes only the true FDs

The algorithm is complete: it computes all the true FDs

In a relation R, for which set or sets of attributes does the closure correspond to all attributes of R?

Closure of a Set of FDs

- For a given FD, we can decide whether it follows from a given set of FDs ***S*** using the closure of the set of attributes algorithm
- Alternatively, we can compute the closure of the set of FDs ***S*** and check whether the FD in question is in that set. We can do this using Armstrong's axioms.

Reflexivity (trivial FDs)

$$If\ \{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\}$$
$$then\ A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$$

Augmentation

$$If\ A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$$
$$then\ A_1A_2 \ldots A_nC_1C_2 \ldots C_k \rightarrow B_1B_2 \ldots B_mC_1C_2 \ldots C_k$$

Transitivity

$$If\ A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m\ and\ B_1B_2 \ldots B_m \rightarrow C_1C_2 \ldots C_k$$
$$then\ A_1A_2 \ldots A_n \rightarrow C_1C_2 \ldots C_k$$

Minimal basis of a set of FDs

- For a given relation *R*, there may exist several sets of FDs that are equivalent:
  - they give rise to the same closures of all subsets of *R*'s attributes
  - the same sets of FDs follow from them
  - all such equivalent sets of FDs are called *bases for **S** in R*
- A *minimal basis **B*** is a set of FDs that satisfies 3 conditions

1. All FDs in ***B*** have 1 attribute on the right (are in a standard form)
2. If any FD is removed from ***B***, the result is no longer a basis
3. If for any FD in ***B*** we remove 1 attribute on the left, the result is no longer a basis

Example

$$R(A,B,C) \qquad S = \{A \to BC, B \to AC, C \to AB\}$$

What is the full set of FDs with 1 attribute on the right that follow from ***S***?

To work with S more easily, use the splitting rule to transform each FD into 2 FDs

Is ***S*** a minimal basis? $S = \{A \to B, A \to C, B \to A, B \to C, C \to A, C \to B\}$

No, e.g., removing A->B does not change the closure of any attribute.

Give 2 different minimal bases for ***S*** in R.

$$S_1 = \{A \to B, B \to C, C \to A\}$$

$$S_2 = \{A \to C, C \to B, B \to A\}$$

Computing the minimal basis

Called "Minimal Cover" in Ramakrishnan & Gehrke, Section 19.6.2

Algorithm *MinimalBasis*

Input: Relation *R, a* set of FDs ***S*** that hold in *R*.

Output: The set of FDs ***T*** that forms a minimal basis for ***S*** in *R*.

1. Transform the FDs in ***S*** into a standard form: if an FD has one attribute on the right, add it to ***T***. Otherwise, if it has *m* attributes on the right, break it up into *m* FDs using the splitting rule (see slide 11) and add each to ***T***.

2. Compute the closure of each set of attributes in the powerset of R (except the empty set) under ***T***.

3. Minimize the left side of each FD in ***T***: For each FD in T with 2 or more attributes on the left, check if an attribute can be deleted while preserving the closures computed in step 2.

4. Delete redundant FDs: check each FD in ***T*** to see if it can be deleted while preserving the closures computed in step 2.

Example

$$R(A,B,C,D,E) \qquad S = \{ABCD \to E, E \to D, A \to B, AC \to D\}$$

$$\text{Step 1}: T = \{ABCD \to E, E \to D, A \to B, AC \to D\}$$

$$\text{Step 2}: \{A\}^+ = \{A,B\} = \{A,B\}^+ \qquad A \to B$$

$$\{E\}^+ = \{E,D\} \qquad E \to D$$

$$\{A,C\}^+ = \{A,B,C,D,E\} \qquad AC \to B, AC \to D, AC \to E$$

$$\{A,E\}^+ = \{A,B,D,E\} \qquad AE \to B, AE \to D$$

$$\text{Step 3}: T = \{AC \to E, E \to D, A \to B, AC \to D\}$$

$$\text{Step 4}: T = \{AC \to E, E \to D, A \to B\}$$

could we have dropped the first two FDs instead?

Closures and keys

Q: How can we tell if a set of attributes $A_1 A_2 \ldots A_n$ is a key of a relation *R*?

A: If $\{A_1, A_2 \cdots A_n\}^+$ = all the attributes in *R*

Q: How can we compute the keys for *R*?

A: Find all sets of attributes that functionally determine all other attributes. Is this enough?

A: Make sure these sets are minimal!

Example

$$Movies(title, year, studio, president, address)$$

$$FDs = \{TY \to S, S \to P, S \to A\}$$

Compute the keys of *Movies*

$$\{S\}^+ = \{S, P, A\}$$

$$\{T\}^+ = \{T\}$$

$$\{Y\}^+ = \{Y\}$$

$$\{P\}^+ = \{P\}$$

$$\{A\}^+ = \{A\}$$

$$\boxed{\{TY\}^+ = \{T, Y, S, P, A\}}$$

Remember that there can be multiple candidate keys, we just happen to have 1 candidate key here!

Example

Compute the keys of R

$$R(A,B,C,D) \quad S=\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$$

$$R(A,B,C,D,E) \quad S=\{AB \rightarrow C, C \rightarrow B, A \rightarrow D\}$$

Example (solution)

Compute the keys of R

$$R(A,B,C,D) \quad S = \{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$$

$$R(A,B,C,D,E) \quad S = \{AB \rightarrow C, C \rightarrow B, A \rightarrow D\}$$

Computing a projection of a set of FDs

Suppose relation *R* is given, with its corresponding set of FDs ***S***. If we take a projection of *R* onto a set of attributes ***L***, what can we say about the FDs of $\pi_L(R)$?

Algorithm *ProjectFDs*

Input: Relations *R* and *R1*= $\pi_L(R)$. A set of FDs ***S*** that hold in *R*.

Output: The set of FDs ***T*** that hold in *R1*.

1. Compute the closure of each subset of attributes of *R1* in ***S***, store the result in ***T***. Add to ***T*** all non-trivial FDs $X \rightarrow A$ s.t. *A* is both in X^+ and an attribute of *R1*.

2. Remove from ***T*** all FDs that involve attributes not in *L* (on either side).

3. Optionally compute the minimal basis of ***T***, remove FDs from ***T*** that do not belong to the minimal basis.

Example

Compute a projection of the set of FDs when R (ABCD) is projected onto ACD.

$$R(ABCD)\ \ A \rightarrow B\ ;\ B \rightarrow C\ ;\ C \rightarrow D$$

$$\pi_{ACD}(R)$$

Compute closures of all subsets of attributes in the projected relation.

$$\{A\}^+ = \{A,B,C,D\}$$

we stop here, since any set that includes A will have the same closure as A alone

$$\{C\}^+ = \{C,D\} = \{C,D\}^+$$

$$\{D\}^+ = \{D\}$$

Compute FDs from these closures that involve only A,C,D on either side.

$$T = \{A \rightarrow C, A \rightarrow D, C \rightarrow D\}$$

Optionally remove redundant FDs, keeping only the minimal basis.

$$T = \{A \rightarrow C, C \rightarrow D\}$$

Why not simply take S and project each FD?

Recap so far

- What are functional dependencies (FDs)?
- We care about FDs because they allow us to reason about redundancy in a relation
- We can use FDs to compute the closure of a set of attributes
- We can use FDs to check which FDs are inferred from them
- We know how to project a set of FDs when the relation is projected onto a subset of its attributes

Next: using FDs to improve schema design (aka normalization)

Normal Forms

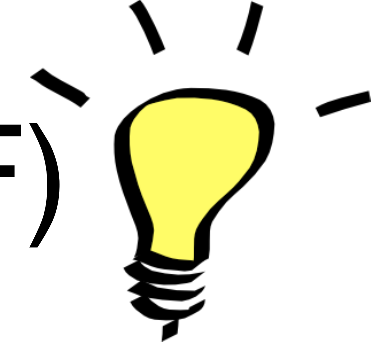
We consider the FDs that hold in a schema, and, based on that information, classify the schema as being in a certain normal form

- First normal form (1NF): all attributes have atomic values; the relational model guarantees 1NF
- Second normal form (2NF), of historical interest only, a less restrictive version of 3NF
- Third normal form (3NF)*
- Boyce-Codd normal form (BCNF)** - the Holy Grail

If R is in BCNF it is also in 3NF; if R is in 3NF it is also in 2NF.

All relations are in 1NF, by definition of the relational model. And so "all relations that are in 2NF are also in 1NF" is a truism. All relations are in 1NF, period.

Boyce-Codd Normal Form (BCNF)



Let *R* be a relation schema, ***S*** be the set of FDs given to hold over *R.*

R is in BCNF if, for every FD $A_1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m$

one of the following statements is true:

1. The FD is trivial: $\{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\}$
2. $A_1 A_2 \ldots A_n$ is a candidate key of *R*
3. $A_1 A_2 \ldots A_n$ is a superkey of *R*

In a BCNF relation, the only set of attributes that determines values for other attributes is a key!

Example

Hourly_Employees (ssn, name, rating, wages, hours)

ssn	name	rating	wages	hours
123-22-3666	Ann	5	10	40
231-31-5368	Bob	5	10	30
131-24-3650	Cate	2	7	30
434-26-3751	Dan	2	7	32
612-67-4134	Eve	5	10	40

Is Hourly_Employees in BCNF?

FDs:

$$S \rightarrow NRWH$$

$$R \rightarrow W$$

no

FDs:

$$S \rightarrow NRWH$$

$$SR \rightarrow W$$

yes

FDs:

$$S \rightarrow NRWH$$

$$NR \rightarrow N$$

yes

A relation with 2 attributes

R(A,B) e.g., SSN and Id are both candidate keys

ssn	id
123-22-3666	1
231-31-5368	2
131-24-3650	3
434-26-3751	4
612-67-4134	5

$$A \rightarrow B$$

$$B \rightarrow A$$

Is this relation in BCNF?

more generally, all 2-attribute relations are in BCNF

To see this, consider all cases of FDs that can hold, see that each case gives rise to a BCNF relation.

Decomposition of Relation Schema

A decomposition of relation *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 new attributes), and
- each attribute of *R* appears in at least one of the new relations.

Decomposition

Hourly_Employees : SNRWH

ssn	name	rating	wages	hours
123-22-3666	Ann	5	10	40
231-31-5368	Bob	5	10	30
131-24-3650	Cate	2	7	30
434-26-3751	Dan	2	7	32
612-67-4134	Eve	5	10	40

FDs:

$$S \rightarrow NRWH$$

$$R \rightarrow W$$

Hourly_Employees2 : SNRH

ssn	name	rating	hours
123-22-3666	Ann	5	40
231-31-5368	Bob	5	30
131-24-3650	Cate	2	30
434-26-3751	Dan	2	32
612-67-4134	Eve	5	40

Rating_Wages : RW

rating	wages
1	6
2	7
3	8
4	9
5	10

Decomposition into BCNF

Let *R* be a relation schema, ***S*** be the set of FDs given to hold over *R*. We decompose *R* by considering FDs that violate BCNF.

Suppose that the following non-trivial FD violates BCNF

$$\boxed{A_1 A_2 \ldots A_n \rightarrow B}$$

This means that

1. $A_1 A_2 \ldots A_n$ functionally determine B
2. $A_1 A_2 \ldots A_n$ is not a key

Intuition

1. Use the offending FD to make a BCNF relation (one where the left side is a candidate key

What relation is that? $A_1 A_2 \ldots A_n \rightarrow ?$

2. Create another relation from which redundancy due to the offending FD has been removed

Decomposition into BCNF

Let *R* be a relation schema, ***S*** be the set of FDs given to hold over *R.* We decompose *R* by considering FDs that violate BCNF.

Algorithm *BCNFDecomposition*

Input: Relation *R,* a set of FDs ***S*** that hold in *R*.

Output: A decomposition of *R* into a set of relations, all of which are in BCNF.

1. Check whether *R* is in BCNF. If so, return {*R*}.
2. Otherwise, let $A_1 A_2 \ldots A_n \rightarrow B$ be an FD that violates BCNF.
   - 2.1. Use *AttributeClosure to compute* $\{A_1, A_2, \ldots, A_n\}^+$
   - 2.2. Decompose *R* into $R_1 = R - B$ and $R_2 = \{A_1 A_2 \ldots A_n\}^+$
   - 2.3. Use *ProjectFDs* to compute FDs of R_1 and R_2
   - 2.4. Recursively decompose R_1 and R_2 using *BCNFDecomposition*

Decomposition into BCNF

Example Hourly_Employees : SNRWH

$$\{S \to N, S \to R, S \to W, S \to H, R \to W\}$$

$R \to W$ *violates BCNF*

decompose into SNRH and RW

$$\begin{tikzcd}
& SNRWH \arrow[dl] \arrow[dr] & \\
SNRH & & RW
\end{tikzcd}$$

$\{S \to N, S \to R, S \to H\}$ $\quad R \to W \quad$ FDs on the decomposed relations

done!

Decomposition into BCNF

If multiple FDs violate BCNF, different orders of decomposition are possible, leading to different results

Example: *R* is ABCDEFG, key A

violate BCNF

FDs: $A \rightarrow BCDEFG \quad C \rightarrow B \quad BD \rightarrow E \quad CD \rightarrow E$

$$ABCDEFG$$

$$BD \rightarrow E$$

$$ABCDFG \quad BDE$$

$$C \rightarrow B$$

$$ACDFG \quad CB$$

also took care of $CD \rightarrow E$

Decomposition into BCNF

If multiple FDs violate BCNF, different orders of decomposition are possible, leading to different results

Example: *R* is ABCDEFG, key A

violate BCNF

FDs: $A \rightarrow BCDEFG$ $\quad C \rightarrow B \quad BD \rightarrow E \quad CD \rightarrow E$

$ABCDEFG$

$BD \rightarrow E$

$ABCDFG \quad BDE$

$C \rightarrow B$

$ACDFG \quad CB$

also took care of $CD \rightarrow E$

$ABCDEFG$

$C \rightarrow B$

$ACDEFG \quad CB$

$CD \rightarrow E$

$ACDFG \quad CDE$

also took care of $BD \rightarrow E$

Properties of a decomposition

- If a relation was not in BCNF, and we decomposed it into a set of BCNF relations, we eliminate the anomalies due to redundancy
- A good decomposition should additionally have two properties:

1. Recoverability of information: Can we recover the original relation from the tuples of its decomposition (by joining)? If so, we have a lossless join decomposition.

2. Preservation of dependencies: When we reconstruct the original relation, does it still satisfy all the original FDs? If so, we have a dependency-preserving decomposition.

Lossless join decomposition

Any decomposition using *BCNFDecomposition* will result in lossless join

- This is because we decompose using an FD, and can reconstruct the original relation (schema and tuples), by taking a natural join. This will compute all the original tuples and no additional ones.
- If we decomposed without an FD - we may generate more tuples by the natural join than were there originally

Recall: Decomposition into BCNF

R: $ABCDEFG$ FDs: $A \rightarrow BCDEFG$ *violate BCNF*

$C \rightarrow B \quad BD \rightarrow E \quad CD \rightarrow E$

$ABCDEFG$

$BD \rightarrow E$

$ABCDFG \quad BDE$

$C \rightarrow B$

$ACDFG \quad CB$

also took care of $CD \rightarrow E$

Are all of the original FDs still enforced?

No, this decomposition is not dependency-preserving!

Dependency-preserving decomposition

In a dependency-preserving decomposition, each original dependency can be checked by looking at just 1 table in the result of the decomposition

R: $ABCDEFG$ FDs: $C \to B \quad BD \to E \quad CD \to E$

$R1(ACDFG) \quad R2(CB) \quad R3(BDE)$ not dependency-preserving

$CD \to E$ not enforced

$R1(ACDFG) \quad R2(CB) \quad R3(CDE)$ not dependency-preserving

$BD \to E$ not enforced

A dependency-preserving decomposition into BCNF is not guaranteed to exist.

Example

$R(ABCD)$ $\quad A \rightarrow B \quad B \rightarrow D \quad AD \rightarrow C \quad BC \rightarrow A$

Decompose R into BCNF. Show keys, projected FDs.

Example (solution)

$R(ABCD)$ $\quad A \rightarrow B \quad B \rightarrow D \quad AD \rightarrow C \quad BC \rightarrow A$

Decompose R into BCNF. Show keys, projected FDs.

Third Normal Form (3NF)

Let *R* be a relation schema, ***S*** be the set of FDs given to hold over *R.*

R is in 3NF if, for every FD $A_1A_2\ldots A_n \rightarrow B_1B_2\ldots B_m$

one of the following statements is true:

same as for BCNF

1. The FD is trivial: $\{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\}$
2. $A_1A_2\ldots A_n$ is a candidate key of *R*
3. $A_1A_2\ldots A_n$ is a superkey of *R*
4. Each B_i is part of some candidate key of *R*

In contrast to BCNF, some redundancy is possible with 3NF. This normal form is a compromise, needed when no dependency-preserving decomposition into BCNF exists.

Decomposition into 3NF

Let *R* be a relation schema, ***S*** be the set of FDs given to hold over *R.* We decompose *R* by considering FDs that violate 3NF.

Algorithm *3NFSynthesisDecomposition*

Input: Relation *R,* a set of FDs ***S*** that hold in *R*.

Output: A decomposition of *R* into a set of relations, all of which are in 3NF.

1. Check whether *R* is in 3NF. If so, return {*R*}.
2. Find a minimal basis for ***S****, say* ***T****.*
3. For each FD in ***T*** of the form $A_1A_2\ldots A_n \rightarrow B_1B_2\ldots B_m$ create a relation

   $A_1A_2\ldots A_nB_1B_2\ldots B_m$ and add it to the decomposition
4. If none of the relations from Step 3 is a key for *R*, another relation to the decomposition, whose schema is a key for *R*

Example

$$R(A,B,C,D,E) \qquad S=\{AB\to C, C\to B, A\to D\}$$

Let's start by computing T, the minimum basis of S.

To do this, we compute attribute closures that appear in S, and the FDs to which they give rise.

$\{A\}^+ = \{A,D\}^+ = \{A,D\}$ $\qquad A\to D$

$\{C\}^+ = \{B,C\}^+ = \{B,C\}$ $\qquad C\to B$

$\{B\}^+ = \{B\} \quad \{D\}^+ = \{D\} \quad \{E\}^+ = \{E\}$

$\{A,B\}^+ = \{A,C\}^+ = \{A,B,C,D\}$ $\qquad AB\to C \quad AB\to D \quad AC\to B \quad AC\to D$

And now pick the minimal set of FDs. $T=\{AB\to C, A\to D, C\to B\}$

For example, we don't add $AB\to D$ because $A\to D$ is in *T*.

Example (continued 1)

$$R(A,B,C,D,E) \qquad S=\{AB \to C, C \to B, A \to D\}$$

On the previous slide, we computed the minimum basis of S and called it T.

$$T=\{AB \to C, A \to D, C \to B\}$$

Next, what are the candidate keys of R?

$$\{A,B,E\}^+ = \{A,B,C,E,D\}$$

$$\{A,C,E\}^+ = \{A,B,C,E,D\}$$

Is R in 3NF? No, the following FD violates 3NF: it's non-trivial, A is not a candidate key or a superkey, and D is not part of any candidate key.

$$A \to D$$

Decompose R into 3NF by synthesis. First add a relation for each FD in T , shown here with the corresponding FDs.

$$R1(ABC) \quad AB \to C$$

$$R2(AD) \quad A \to D$$

$$R3(BC) \quad C \to B$$

Observe that R3 is a proper subset of R1. We never need to create a relation that is a proper subset of another, keep only R1 and R2.

$$R1(ABC) \quad AB \to C, \quad C \to B$$

$$R2(AD) \quad A \to D$$

Example (continued 2)

$$R(A,B,C,D,E) \qquad S = \{AB \to C, C \to B, A \to D\}$$

Two slides ago we computed the minimum basis of S, and called it T.

$$T = \{AB \to C, A \to D, C \to B\}$$

We then computed the candidate keys of R.

$$\{A,B,E\}^+ = \{A,B,C,E,D\}$$

$$\{A,C,E\}^+ = \{A,B,C,E,D\}$$

On the previous slide we computed the decomposition into 2 relations.

$$R1(ABC) \quad AB \to C, \quad C \to B$$

$$R2(AD) \quad A \to D$$

Finally, we need to add a relation that represents some (one) candidate key or R, since neither R1 nor R2 includes a candidate key of R. Include either R3.1 or R3.2 into the final decomposition.

$$R3.1(ABE)$$

$$R3.2(ACE)$$

Example

Are these relations in BCNF? In 3NF?

$$R(ABCD)\ A \rightarrow B\ ;\ B \rightarrow A\ ;\ A \rightarrow D\ ;\ D \rightarrow B$$

$$R(ABCD)\ AB \rightarrow C\ ;\ BCD \rightarrow A\ ;\ D \rightarrow A\ ;\ B \rightarrow C$$

$$R(ABCD)\ FD's:\ AC \rightarrow D\ ;\ D \rightarrow A\ ;\ D \rightarrow C\ ;\ D \rightarrow B$$

Example (solutions)

Are these relations in BCNF? In 3NF?

$$R(ABCD)\ A \rightarrow B\ ;\ B \rightarrow A\ ;\ A \rightarrow D\ ;\ D \rightarrow B$$

$$R(ABCD)\ AB \rightarrow C\ ;\ BCD \rightarrow A\ ;\ D \rightarrow A\ ;\ B \rightarrow C$$

$$R(ABCD)\ FD's: AC \rightarrow D\ ;\ D \rightarrow A\ ;\ D \rightarrow C\ ;\ D \rightarrow B$$

Schema refinement summary

- We discussed schema redundancy
- We presented functional dependencies (FDs), talked about how we can reason about them and use them to determine if a schema exhibits redundant
- Normal forms: BCNF and 3NF
- Properties of a decomposition: (1) eliminate redundancy; (2) lossless join; (3) dependency-preserving
- Algorithms for: closure of a set of attributes, projection of a set of FDs, BCNF decomposition, 3NF synthesis