CMSC 424 – Database design
Lecture 11
Normalization

Mihai Pop

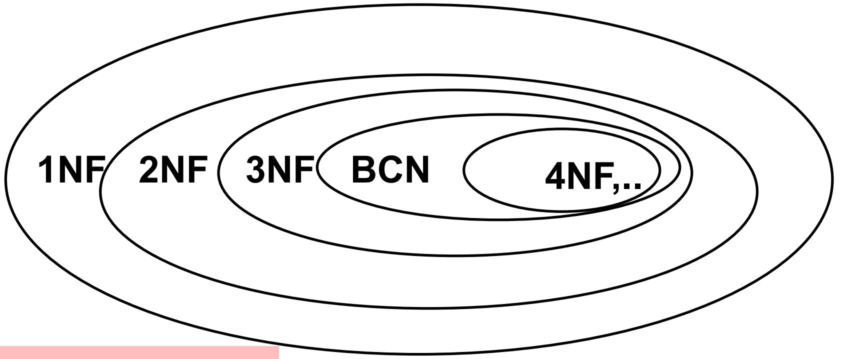
The Normal Forms

- 1NF: every attribute has an atomic value (not a set value)

- 2NF: we will not be concerned in this course

- 3NF: if for each FD X→Y either
  - it is trivial or
  - X is a superkey
  - Y-X is a proper subset of a candidate key

- BCNF: if for each FD X→Y either
  - it is trivial or
  - X is a superkey



- 4NF,…: we are not concerned in this course.

Goals

- Lossless decomposition
- Dependency preservation
- Recap: FD closure, attribute closure

FDs, Normal forms, etc..., why?

- Start with a schema
- Decompose relations until in a normal form
- Functional dependencies (constraints we'd like preserved) drive the decomposition
- The resulting schema is “better”

- Note that functional dependencies can either be:
  - explicit: we want to enforce these constraints irrespective of data in the relations – can be encoded in SQL
  - implicit: the data happen to satisfy them (see netflix example)

  Normalization only concerned with explicit FDs
  Privacy/anonymization – need to worry about implicit FDs

Boyce-Codd Normal Form

A relation schema *R* is in BCNF with respect to a set *F* of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

$$\alpha \rightarrow \beta \text{ is trivial (i.e. } \beta \subseteq \alpha \text{)}$$

$$\alpha \text{ is a superkey for R, i.e. } \alpha + = R$$

Example schema *not* in BCNF:

bor_loan = (*customer_id, loan_number, amount*)

because *loan_number* $\rightarrow$ *amount* holds on *bor_loan* but *loan_number* is not a superkey

Decomposing a Schema into BCNF

- Suppose we have a schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

  We decompose R into:

$$\alpha \cup \beta$$
$$R - (\beta - \alpha)$$

- In our example,
  - $\alpha = loan\_number$
  - $\beta = amount$

  and *bor_loan* is replaced by

  - $\alpha \cup \beta$ = (*loan_number, amount*)
  - $R - (\beta - \alpha)$ = (*customer_id, loan_number*)

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  - compute α^+ (the attribute closure of α), and
  - verify that it includes all attributes of *R*, that is, it is a superkey of *R*.

- Simplified test: To check if a relation schema *R* with a given set of functional dependencies F is in BCNF, it suffices to check only the dependencies in the given set *F* for violation of BCNF, rather than checking all dependencies in F^+.
  - We can show that if none of the dependencies in *F* causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either.

Testing for BCNF...cont

- However, using only F is incorrect when testing a relation in a decomposition of R
  - E.g. Consider R (A, B, C, D), with $F = \{ A \rightarrow B, B \rightarrow C\}$
    - Decompose R into $R_1(A,B)$ and $R_2(A,C,D)$
    - Neither of the dependencies in F contain only attributes from (A,C,D) so we might be mislead into thinking R_2 satisfies BCNF.
    - In fact, dependency $A \rightarrow C$ in F^+ shows R_2 is not in BCNF.

- Simplified test: Avoids computing F+
  - For every subset α of R_i compute α+ under F
  - Then either α+ includes no attributes of R_i-α or includes all attributes of R_i
    - In $R_2(A,C,D)$ above A+=ABC, A+-(A)=(BC) includes an attribute of R_i but not all (violation)
    - Then $\alpha \rightarrow (\alpha+ - \alpha) \cap R_i$ is the violator A→BC ∩ (ACD)=C is an FD (actually in F+) which violates BCNF

BCNF Decomposition Algorithm

result := $\{R\}$;
done := false;
compute F^+;
while (not *done*) do
  if (there is a schema R_i in *result* that is not in BCNF)
    then begin
      let $\alpha \rightarrow \beta$ be a nontrivial functional
        dependency that holds on R_i
        such that $\alpha \rightarrow R_i$ is not in F^+,
        and $\alpha \cap \beta = \emptyset$;
        $result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta);$
      end
    else *done* := true;

Note: each R_i is in BCNF, and decomposition is lossless-join.

Example of BCNF Decomposition

- R = *(branch-name, branch-city, assets, customer-name, loan-number, amount)*

- F = *(branch-name → assets branch-city loan-number → amount branch-name)*

  Key = *{loan-number, customer-name}*

- Decomposition
  - R_1 = *(branch-name, branch-city, assets)*
  - R_2 = *(branch-name, customer-name, loan_number, amount)*
  - R_3 = *(branch-name, loan-number, amount)*
  - R_4 = *(customer-name, loan-number)*

- Final decomposition

  R_1, R_3, R_4

R=(Bn,Bc,As,Cn,Ln,Am)

F={Bn→As Bc,

Ln→Am Bn,

Ln Cn→Bn Bc As Am} ← key

1) Bn→As Bc in R

Bn+={Bn As Bc} ← not SK

Decompose

***R*1 = (Bn,Bc,As)**

***R*2 = (Bn,Cn,Ln,Am)**

2) Ln→Am Bn in *R*2

Ln+={Ln Am Bn As Bc} ← not SK

decompose

R3=(Ln Am Bn)

R4=(Ln Cn)

BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation

- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving*.

- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form*.

Third Normal Form

- A relation schema *R* is in third normal form (3NF) if for all:

$$\alpha \rightarrow \beta \text{ in } F+$$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for *R*
  - Each attribute *A* in $\beta - \alpha$ is contained in a candidate key for *R*.

    (**NOTE**: each attribute may be in a different candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).

3NF (Cont.)

- Example
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$
  - Two candidate keys: JK and JL
  - R is in 3NF

    $JK \rightarrow L$ JK is a superkey
    $L \rightarrow K$ K is contained in a candidate key

Redundancy in 3NF

- Example of problems due to redundancy in 3NF
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

A schema in 3NF but not in BCNF has the following problems:

- redundancy of information
- need to use null values (e.g. to represent relationship l_2k_2, when there is no corresponding j value)

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 $\alpha \rightarrow \beta$, if α is a superkey.

- If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of *R*
  - this test is more expensive, since it involve finding ALL candidate keys
  - testing for 3NF has been shown to be NP-hard

Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B, \;\; B \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS: $\{A \rightarrow B, \;\; B \rightarrow C, \;\; A \rightarrow CD\}$ can be simplified to
      $$\{A \rightarrow B, \;\; B \rightarrow C, \;\; A \rightarrow D\}$$
    - E.g.: on LHS: $\{\mathrm{A} \rightarrow B, \;\; B \rightarrow C, \;\; AC \rightarrow D\}$ can be simplified to
      $$\{\mathrm{A} \rightarrow B, \;\; B \rightarrow C, \;\; A \rightarrow D\}$$
- Intuitively, a canonical cover of F is a "minimal" set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies

Extraneous Attributes

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.
  - Attribute A is extraneous in α if $A \in \alpha$ and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute A is extraneous in β if $A \in \beta$ and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.
- *Note:* implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one
- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping B from $AB \rightarrow C$).
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting C

3NF Decomposition/"construction" Algorithm

Let F_c be a canonical cover for F;
$i := 0$;
for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**
 if none of the schemas R_j, $1 \leq j \leq i$ contains $\alpha\,\beta$
  then begin
   $i := i + 1$;
   $R_i := \alpha\,\beta$
  end
if none of the schemas R_j, $1 \leq j \leq i$ contains a candidate key for R
 then begin
  $i := i + 1$;
  $R_i :=$ any candidate key for R;
 end
return $(R_1, R_2, ..., R_i)$

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - the dependencies are preserved

- It is always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

More Examples

- SUPPLY(<u>sno,pno,jno</u>,scity,jcity,qty) **1NF**
  - sno,pno,jno is the candidate key,
  - sno → scity, jno= → jcity
- ED(<u>eno</u>,ename,byr,sal,dno,dname,floor,mgr) **1NF**
  - eno → dno → mgr
- TEACH(<u>student</u>,teacher,<u>subject</u>) **3NF**
  - student,subject → teacher
  - teacher → subject

Normalization Using FDs

Check whether a particular relation R is in "good" form: BCNF or 3NF

If not, decompose R into a set of relations $\{R_1, R_2, \ldots, R_n\}$ such that

- No redundancy: The relations R_i preferably should be in either Boyce-Codd Normal Form or Third Normal Form.
- Lossless-join decomposition: Otherwise you have information loss.
- Dependency preservation: Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - Preferably the decomposition should be dependency preserving, that is, $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
  - Otherwise, checking during updates for violation of functional dependencies may require expensive joins operations
- The theory is based on functional dependencies

BCNF and Over-normalization

- 3NF relation has redudancy anomalies: TEACH(<u>student</u>,teacher,<u>subject</u>)
  - insertion: cannot insert a teacher until we had a student taking his subject
  - deletion: if I delete the last student of a teacher, then I loose the subject he teaches

- What is really the problem? schema <u>*overload.*</u> We are trying to capture two meanings:
  1. subject X is (or can be) taught by teacher Y
  2. student Z takes subject W from teacher V

- it makes no sense to say we loose the subject he teaches when he does not have a student! Who does he teach to?

- normalizing it to BCNF cannot preserve dependencies. Therefore, it is better to stay with the 3NF TEACH and another relation SUBJECT_TAUGHT:

TEACH(<u>student</u>,teacher,<u>subject</u>) **3NF**

SUBJECT-TAUGHT(<u>teacher</u>,subject) **BCNF**

Summary...practical issues

- Normalization
  - Create a good schema – low redundancy, no loss of information
- Functional dependencies
  - Specify constraints that must be encoded in our schema
  - Note: SQL does not allow us to specify FDs other than key constraints (PRIMARY KEY, UNIQUE)
- Typical design process:
  - Decompose to BCNF
  - Use materialized views to preserve any additional FDs