5 Normalization: Quality of relational designs

Lit: Kemper/Eickler: chap 6; Garcia-Molina/Ullman/Widom: chap 3.4 ff.; Elmasr/Navathe: chap 14
Lausen: Datenbanken - Grundlagen und XML-Technologien

Roadmap

- Functional dependencies may cause "update anomalies" ☑
- Update anomalies cause troubles
  $\Rightarrow$ find relational schema without "anomalies" in case of update ☑
- Define "Normal forms" for relations which do not show (all) anomalies
- Given a set of functional dependencies, find algorithm which generates a relational schema in some normal form.

5.2 Normal Forms

5.2.1 Informal introduction

First normal form:
all attributes are single valued and atomic
"Movie"-table is in first, but not in second normal form.

Second normal form (2NF):
No non-prime attribute functionally dependent on only part of the primary key
("No partial dependency")

Remove "format" from "Movie"-attributes,
mId is a single attribute key -> no partial dependencies on key -> table in 2 NF

But: there is still a dependeny, which is not a key dependency:
{director} -> {birthdate}

Design quality FDs and Normal Forms

Third normal form (3NF):

– No dependencies of non-prime attributes except those on the whole key or on candidate keys

Example:
Movie (<u>mId</u>, title, director, birthdate, livesInCity,…)
2NF but not in 3NF since
director $\rightarrow$ birthday is a FA

To achieve 3NF, data on directors, i.e. birthdate, livesInCity, have to be put into a different table:

Movie(<u>mId</u>, title, director, year,…) and
Dir (<u>director</u>, birthdate, livesInCity)
are in 3NF

Note: the original table "Movie" can be reconstructed by a join.

5.2.2 Normal forms – definitions

Given a set of Functional Dependencies

- Wanted:
  - Find "normalized" relations from "unnormalized" R
    - Define normalization properly
    - Design algorithm which decomposes R from FDs to normalized relations
  - Or: synthesizes normalized relations from FDs which result in R when joined
- First Normal form: ☑ (no structured attributes)

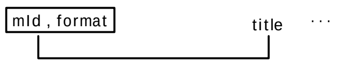
Normal Forms Second normal form

- Second normal form

R is in second normal form (2NF), iff " X $\subseteq \Sigma$ (R),
" a Î Σ (R) : a $\notin$ X , a not prime, X -> a
$\Rightarrow$ X is a key or a superset of a key but not a proper subset of any key of R

This means basically: there is no functional dependency in which a non- prime attribute depends on some part of a key („no partial dependencies on keys")

Example from above:



Normal Forms Second normal form

Removed: partial dependency on key

Movie (<u>mId</u>, title, <u>format</u>, director, birthday, livesInCity)

Movie2(mId, format)

.... but a functional dependency remains

..... since there is a transitive dependency on a (the) key:
mId -> director -> birthday

More general: a non-prime attribute y is transitive dependent on a key K, if K -> X and X -> y and *not* X -> K
Notation: K -> X -> y

Normal Forms Third normal form

Third normal form:

R is in Third Normal Form (3NF) if no non-prime attribute is transitively dependent on a key
or:

If an attribute a of R is transitively dependent on a key k:
k -> X -> a then either

- X contains a key
- a is prime
- a is an element of X

... or more formally:
R is in third normal form (3NF), iff
" X ⊆ S(R)," a Î S(R): a ∉ X, X -> a
⇒ X contains a key or a is prime

Normal Forms Third normal form

– Example
Suppose for each tape the video shop wants to record the company which sold the tape, furthermore its phone number

Tape(id, format, mId, since, back, seller, phone)
'seller' is not a key, 'phone' is not prime
but {id} -> {seller} -> {phone}
'Tape' is in 2NF (why?) , not in 3NF

3NF ⇒ no partial dependencies on a key ⇒ 2NF

DESIGN QUALITY: what do we have?

- Functional Dependencies
- Normal forms
  - 2NF: no functional dependencies of non-prime attributes on part of a key
    "no partial dependencies"
  - 3 NF: no transitive dependency of a non-prime attribute b on a Key K : K -> X -> a
    and ¬ X-> K , a ∉ K
    " no transitive dependencies"

  3NF ⇨ 2NF
  More dependencies??

YES: dependencies between prime attributes!

Decomposition: eliminate FDs

- Given $\Sigma(R) = U$ and DEP the set of FDs
  - Find the set of keys K:
    K -> U ∈ DEP or K -> U ∈ DEP$^+$ (set of all implied dependencies)
  - Eliminate all transitive dependencies by splitting recursively
  - if K -> Y -> a is a transitive FD in R_k, split R_k into R_i, R_j
    $\Sigma(R_i) = \Sigma(R_k) \setminus \{a\}$, $\Sigma(R_j) = Y \cup \{a\}$
    until there is no more relation with a transitive dependency
- Example
  $\Sigma(R) = \{a,b,c\}$, F = {a -> c, a->b, b-> c}
  Key: {a}
  Transitive dependency a -> b -> c
  Normal form: $\Sigma(R1) = \{a,b\}$, $\Sigma(R2) = \{b,c\}$

Normal Forms More normal forms?

– What kind of dependencies remain?
– Remember: "3NF : No other dependencies of non-prime attributes than from a key"

Example:
R(p, o, s, n) with {o,s,n} -> p, p -> o and the keys {o,s,n} and {p,s,n}
R is in 3NF, but there is a transitive dependency in R:
{p,s,n} -> p -> o

e.g. (p,o,s,n) = (PLZ,City,Street,Number)

– Given Relation R with two candidate keys and more than one attribute each:
K = {a,b}, K' = {c,d}
R may be in 3NF but there may exist a FD among key attributes in R

Boyce Codd Normal Form

- Boyce-Codd Normal Form(BCNF) :

  A relation R is in BCNF, if there are no non trivial dependencies X -> a except when X contains (or is) a key.

  Equivalent to: There are no transitive dependencies in R other than trivial ones

  Consequence: BCNF $\Rightarrow$ 3NF

  Equivalent to: X -> a then (i) trivial (ii) X is superkey of R

- Always decompose relations to BCNF?
- Does only work, if the decomposition has particular properties

5.2.4 Lossless property and preserving dependencies

- Normalization (by decomposition) means:

  Split the relation R into relations $R_1, R_2, ..., R_n$ in a way, such that R_i are in normal form (3NF or BCNF) which
  - „preserves information“
  - preserves dependencies

- Criterion for „preserved information“:

  $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n = R$

  „lossless property"

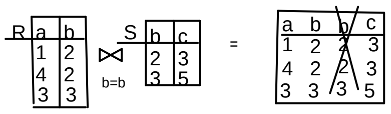
Joining relations

- When relation R has been split into relations $R_1, R_2, ..., R_n$, reconstruction of R from $R_2, ..., R_n$ should be possible
- Join operation (natural join):

  concatenate those typels of R and S which have same name and same value. Eliminate the redundant attribute.

  $R \bowtie S = \{(a1, ...an, b1,...b_{m-k}) \mid (a1,...an) \in R, (b1... b_m) \in S\}$

  R.n = S.n k = numer of attributes R and S with the same name



a	b
1	2
4	2
3	3

$\bowtie$

b	c
2	3
3	5

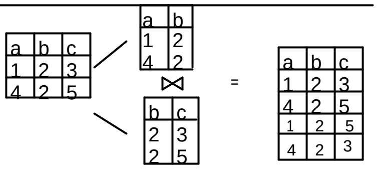
=

a	b	~~b~~	c
1	2	~~2~~	3
4	2	~~2~~	3
3	3	~~3~~	5

"Natural join"

Lossless joins

- Example:



a	b	c
1	2	3
4	2	5

a	b
1	2
4	2

$\bowtie$

b	c
2	3
2	5

=

a	b	c
1	2	3
4	2	5
1	2	5
4	2	3

- Lossless property depends on functional dependencies e.g. {a -> b, c -> b} could hold in the above situation
- If {a -> c, c -> b} the decomposition of R(a,b,c) into R1(a,c), R2(c, b) is lossless (check!)
- In general: Decomposition of R into R1 and R2 is lossless, if

  $\Sigma(R1) \cap \Sigma(R2) \rightarrow \Sigma(R2)$ or $\Sigma(R1) \cap \Sigma(R2) \rightarrow \Sigma(R1)$

Lossless joins

- Lossless decomposition and keys

$\Sigma(R1) \cap \Sigma(R2) \rightarrow \Sigma(R2)$ or $\Sigma(R1) \cap \Sigma(R2) \rightarrow \Sigma(R1)$

means:

The common attribute(s) of R1 and R2 are a key (or a superset of a key) of R1 or R2 (example from above: c is a key of R2)

Important side effect of normalization: Functional dependencies are transformed into key dependent FDs

Advantage: Invariance property expressed by FDs may now be checked by checking the primary key property.

This can efficently be done by any DBS

Preserving Dependencies

If DEP is the set of FDs defined for relation R, decomposition should guarantee:

for each X->Y from DEP there is a relation R_i in the decomposition with $X \cup Y \subseteq \Sigma(R_i)$.

This should be a key dependency, i.e. X should be a (super) key

Means: the set of FDs after decomposition should be the same as before.

Example:
Movie1(mID, title, director), M2(director, birthday)
Dependencies are preserved

- BCNF does not always guarantee both the lossless property and dependency preservation

3NF versus BCNF

Example:
Let (p,s,n) be the key of R(p, o, s, n)
there is a transitive dependency of the (prime) attribute o on (p,s,n).
Normalisation to BCNF:
R1 (p,s,n) and R2(p,o)
Dependency (o,s,n) -> p is lost

- Consequence:
  Normalization to 3NF is the best we can achieve
  - Note the following property.
    If there is at most one key with more than one attribute,
    3NF $\Leftrightarrow$ BCNF

5.2.5 Multivalued dependencies and 4NF

Multiple values: example

Person (<u>name</u>, <u>affiliation</u>, <u>hobbies</u>)

Meier	FUB	skiing
Müller	TUB	trekking
Meier	FUB	trekking
Schulze	HU	skating

Redundancy introduced by multiple values

'hobbies' is multivalued dependent on name

Definition for single attribute multi valued (MV) dependencies:

see below, chap. 6

Let R = (a, y, b),
b is multivalued dependent on a (a ->>b) if for each value v of a $\{v\} \text{ X } (\pi_y(\sigma_{a=v} R)) \text{ X } (\pi_b(\sigma_{a=v} R))$ í R

Example: {'Meier'} X {'FU'} X {'skiing', 'trekking'} Í Person
{'Müller'} X {'TU'} X {'trekking'} Í Person
{'Schulze'} X {'HU'} X {'skating'} Í Person

Fourth Normal Form

A relation R is in Fourth Normal Form
if for every MVD A ->> B
- B í A or
- B = Σ(R) \ A or
- A contains a key

see below, chap. 6

May be easily calculated by splitting up a relation R with a MVD A->> B into R1 and R2 such that
$\Sigma(R1) = A \cup B$, $\Sigma(R2) = \Sigma(R) \setminus \Sigma(R1) \cup A$

Müller	TUB
Meier	FUB
Schulze	HU

Müller	trekking
Meier	trekking
Meier	skiing
Schulze	skating

Better to have multi valued attributes?

5.3 Algorithms for finding Normal Forms

5.3.1 Informal introduction
- Invariants hold in the application domain
  They are made explicit during requirements analysis
  e.g. “A tape may only be borrowed by one client”
  “ A video tape has one and only one format”
  “ A person has exactly one date of birth”

- Wanted: algorithm producing relational schema from the set DEP of all FDs

FDs and Normal Forms

Given a set of dependencies DEP there are two approaches:
- Set up relations in such a way, that
  - All attributes are consumed
  - The relations are in normal form

  Called synthesis of relations

- For a given set of relations find those which are not normalized with respect to DEP and decompose them into normalized relations
  Called decomposition
- Question: how do we find all FDs?

5.3.2 Minimal sets of Functional Dependencies

Task:
Given a set of FDs *F* and a relational schema
-> Find all FDs *F'* implied by *F* (?)
-> Find a canonic set *F''*
-> Find a relational schema in 3NF

How to find all FDs?
- The first step for synthesis or decomposition:
  given a set of dependencies DEP, determine all dependencies of E which must ‘logically” hold:
  $DEP^+ = \{f \mid f$ is a FD in the attribute set, f is implied by DEP$\}$

Implied means: " DEP $\Rightarrow$ f " can be proven

Finding a canonical set

- Dep$^+$ - the set of all implied dependencies of DEP is called the closure of DEP
- Example:
  Movie (<u>mId</u>, title, <u>format</u>, director, birthdate, livesInCity)
  a, b, c, d, , e , g

  DEP = {a-> b, a-> d, ac -> c, d-> e, d -> g} *

  transitivity: a-> e, a-> g
  augmentation: ab -> b, ac-> bc, ad -> bd, ……. , ab-> gb,
  ad -> bd, ad -> gd,
  ….,
  Inclusion: abcd -> abc, abcd -> bcd, …..

  Exponentially many not very interesting dependencies

  * Notation: ab -> c means {a,b} -> {c}

Finding a canonical set

- Different approach
  - Given a set DEP of dependencies, find a minimal one MIN such that: DEP ⊆ MIN$^+$
  - MIN is called a minimal cover of DEP
  - Minimal: MIN \{f} is not a cover for all f ∈ MIN
- Finding a minimal cover
  - First determine the closure X$^+$ of a set of attributes X
  - Closure of attribute set X with respect to the set DEP of FDs is the largest set Y of attributes such that X -> Y ∈ DEP$^+$

Functional Dependencies Closure of X

```
I = 0; X[0] = X;     /* integer I, attr. set X[0] */
REPEAT               /* loop to find larger X[I] */
 I = I + 1;          /* new I */
 X[I] = X[I-1];      /* initialize new X[I] */
 FOR ALL Z ->W in DEP /* loop on all FDs Z ->W in DEP*/
  IF Z Í X[I]        /* if Z contained in X[I] */
  THEN X[I] = X[I]ÈW; /* add attributes in W to X[I]*/
 END FOR             /* end loop on FDs */
UNTIL X[I] = X[I-1]; /* loop till no new attributes*/
RETURN X = X[I] ;    /* return closure of X */
```

Used rule: X -> YZ and Z -> W then X -> YZW
Proof?

Example:
X= X[0] ={a,b} (attributes a and b), DEP = {a -> b, b -> da, e-> d}
X[1] = {a,b,d}
X[2] = X[1]

Finding a canonical set

- Algorithm for determining a minimal cover in polynomial time
- Steps
  1. Replace each FD X -> Y of DEP in which Y contains more than one attribute, by FDs with one attribute on the right hand side
     Example: DEP = {ab -> cd, a -> e} → {ab -> c, ab -> d, a -> e}
  2. Remove redundant FDs
     f is redundant, if (DEP \ {f})$^+$ = DEP$^+$
     Example: {b -> d, d -> e, ef -> a, c -> f, bc -> a}
     b -> d, d -> e ⇒ b -> e, c -> f ⇒ bc -> ef , ef -> a ⇒ bc -> a
     FD bc -> a is redundant
  3. Minimize left hand side of each FD, i.e. if f = X->Y ∈ DEP, a ∈ X and {DEP}+ = (DEP')$^+$, DEP'= {DEP \f} ∪ { X\{a} -> Y}
     then replace DEP by DEP',
     If a FD has been minimized repeat step 2.
     Example: {bcd -> a, c -> e, e -> b} $^+$ = {cd -> a, c -> e, e -> b} $^+$
  4. Make lefthand side of FDs unique by applying the union rule
     Example: {cd -> a, cd -> e, d -> f} becomes {cd -> ae, d -> f}

5.3.3 Synthesis and Decomposition

- Given Σ(R) = U and DEP the set of FDs
  - Find the set of keys K: see above..
    K -> U ∈ DEP or K -> U ∈ DEP$^+$
  - Eliminate all transitive dependencies by splitting recursively
    if K -> Y -> a is a transitive FD in R_k, split R_k into R_i, R_j
    $\Sigma(R_i) = \Sigma(R_k) \setminus \{a\}$, $\Sigma(R_j) = Y \cup \{a\}$
    until there is no more relation with a transitive dependency
- Example
  Σ(R) = {a,b,c}, F = {a->b, b-> c}
  Key: {a}
  Transitive dependency a -> b -> c
  Normal form: Σ(R1) = {a,b}, Σ(R2) = {b,c}
- Disadvantage;
  - May produce more relations than necessary
  - Time complexity, since keys have to be determined in each step

…and Synthesis

- Normalization problem:
  - Given a relation R in 1NF and a set of DEP of FD
    Find a lossless, dependency preserving decomposition $R_1,\dots,R_k$, all in 3NF
- Synthesis Algorithm

```
1. Find minimal cover MIN of DEP;
2. For all X -> Y in MIN define a relation
   RX  with schema S(RX) = X È Y
3. Assign all FDs X' -> Y' with X' È Y' Í  S(RX) to RX
4. If at least one of the synthesized relations RX
   contains a candidate key of R
   skip
   else introduce a relation Rkey which contains a
   candidate key of R
5. Remove relations RY where:  S(RY) Í S(RX)
```

Final result: lossless, dependency preserving decomposition of R

Normal Forms Synthesis

Example

Movie (mId, title, format, director, birthday, livesInCity)

MIN = { mID -> {title, director },
director -> {birthday, livesInCity} ,
format -> format }

R1 = (mId, title, director)

R2=(director, birthday, livesInCity)

R3= (format)

No relation which includes key.

Therefore: R4 = (mId, format)

R3 í R4 : remove R3

5.4 Normal Forms: Critical review

Should relations be always normalized ?

– Yes : makes invariant checking easy, no „update anomalies“

– No : Why should we normalize if there are no updates ?

Example:
Customer (cuId, name, fname , zipCode , city, street , no)
No reason to normalize into e.g.: Cu1(cuId, name, fname) and CuAdr(cuId, zipCode, city, street, no)
if only one address per customer and updates are infrequent

– Yes: consider cost of joins / updates
  - How expensive are selects which need joins because of normalization?

  "Select name from Cu1, Cu2 where Cu1.cuId = Cu2.cuId and..."
  - Updates which cause anomalies?

ER modeling and Normal Forms

- ER and Normal Forms two different mechanisms to set up or enhance a database scheme
- ER more intuitive, NF uses algorithms
- BUT
  - ER-models often already in NF
  - Starting with a universal set of attributes and FDs and synthesizing relations not a "natural way" of modeling
- Use normalization as a complementary design tool
  1. Set up ER model
  2. Transform to relations
  3. Normalize each non normalized relation if the tradeoff of join processing (Select) and updating redundant data suggests to do so