Normal forms

- Main topic: Boyce-Codd Normal Form (BCNF)
- For this we need to study functional dependencies (FD)
- Basics on FDs
- BCNF
- More on FD inference needed for BCNF

Functional dependencies

Functional dependency (FD): A general type of constraints on relations

Syntax: Let $R(A1, \ldots, An, B1, \ldots, Bm, C1, \ldots, Ck)$ be a relation schema

$A1 \ldots An \rightarrow B1 \ldots Bm$ is an FD

Meaning: Whenever two tuples t_1 and t_2 in an instance of R agree on attributes $A1$, ..., A_n, then they must agree on the attributes B_1, ..., B_m

If we can be certain that *for every instance of a relation* this FD will hold, we say that R satisfies the FD

Example

Consider the following relation (note that is different from the schema we have used until now)

`Movies(title, year, length, genre, studioName, starName)`

What FDs hold?

$$\texttt{title year} \rightarrow \texttt{length genre studioName}$$

But

$$\texttt{title year} \rightarrow \texttt{starName}$$

Does not hold

Some notation:

- We concatenate attributes instead of using set notation
- $AB \to C$ means that the set of attributes $\{A, B\}$ functionally determines the set of attributes $\{C\}$.
- Note that $AB \to CD$ is equivalent to $AB \to C$ and $AB \to D$
- But this is *not* equivalent to $A \to CD$ and $B \to CD$
- We typically use A, B, ... for single attributes and X, Y, ... for sets of attributes

Example

title	year	length	genre	studioName	starName
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

Note the redundant data

Why is this bad?

- Space
- Update costs
- Update anomalies

But: Fewer joins needed in queries

Keys

First, we study keys as FDs

Let

`A1,...,An`

be some of the attributes of `R`.

`A1,...,An` is a *key* if

- `A1,...,An` functionally determine all the attributes of `R`
- No proper subset of `A1,...,An` functionally determines all the attributes of `R`.

Example:

`Movies(title,year,genre,length,studioName)`

(`title`, `year`) is a key since the following FD holds

`title year` $\rightarrow$ `length genre studioName`

But neither of the following hold

`title` $\rightarrow$ `year length genre studioName`

`year` $\rightarrow$ `title length genre studioName`

(`title`, `year`,`length`) is a *superkey*

`title year length` $\rightarrow$ `genre studioName`

What is wrong with this schema?

title	year	length	genre	studioName	starName
Star Wars	1977	124	SciFi	Fox	Carrie Fisher
Star Wars	1977	124	SciFi	Fox	Mark Hamill
Star Wars	1977	124	SciFi	Fox	Harrison Ford
Gone With the Wind	1939	231	drama	MGM	Vivien Leigh
Wayne's World	1992	95	comedy	Paramount	Dana Carvey
Wayne's World	1992	95	comedy	Paramount	Mike Meyers

- Redundancy
- Update cost. To change the length of "Star Wars", we have to modify 3 tuples
- *Update anomalies*. If we only change 2 of these, we get inconsistent data
- *Deletion anomalies*. Suppose we delete "Viven Leigh" from the stars of "Gone with the Wind". We might lose all the other data about the movie

Decomposition

The basic idea is to decompose the relation into two different relations, to avoid these problems

In this case, one relation contains Movie information

`Movies2(title,year,length,genre,studioName)`

And the other contains star information

`Movies3(title,year,starName)`

No problems with anomalies, but

- How do we find the data for the two new relations?
- Can we reconstruct the original relation

To get the data: Project onto the attributes

$$\texttt{Movie2} = \pi_{\texttt{title,year,genre,studioName}}(\texttt{Movies})$$

$$\texttt{Movie3} = \pi_{\texttt{title,year,starName}}(\texttt{Movies})$$

To reconstruct the original data take

$$R = \texttt{Movie2} \bowtie \texttt{Movie3}$$

Is `Movie = R` ?

Abstract example

$R(A, B, C)$

Take $S(A, B) = \pi_{A,B}(R)$ and $T(A, C) = \pi_{A,C}(R)$

Is $R = S \bowtie T$?

Example 1:

$R(A, B, C)$

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

$S(A, B)$

A	B
1	2
1	4

$T(A, C)$

A	C
1	3
1	5

$S \bowtie T$

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

Not equal to R

Question: What FDs hold in R? What about anomalies?

Example 2:

$R(A, B, C)$

A	B	C
1	2	3
1	2	5

$S(A, B)$

A	B
1	2

$T(A, C)$

A	C
1	3
1	5

$S \bowtie T$

A	B	C
1	2	3
1	2	5

Equal to R

Are there any anomalies?

What is the main difference between the two examples?

(We assume that FDs that hold in these instances hold in R)

- Example 1: Only key is ABC. No nontrivial FDs
- Example 2: Only key is ABC, but FD $A \rightarrow B$ holds

The existence of an FD whose left hand side is not a key is

- A reason to decompose a relation
- A reason why the decomposition works

The problem

Given $R(A, B, C)$, $A \rightarrow B$

Is $R = \pi_{A,B}(R) \bowtie \pi_{A,C}(R)$?

If so, we call the decomposition *lossless*, i.e., we do not lose information

Warning: If a decomposition is not lossless, it means we lose information, not that we lose tuples. We lose information if we get

- Too few tuples
- Too many tuples

For the decompositions that we are looking at, the first case never happens

Theorem: For any relation $R(A, B, C)$,

$$R \subseteq \pi_{A,B}(R) \bowtie \pi_{A,C}(R)$$

This means that we never lose tuples. (Note that the theorem does not make any assuptions about FDs)

Proof:

Let $(a, b, c) \in R$.

Then $(a, b) \in \pi_{A,B}(R)$ and $(a, c) \in \pi_{A,C}(R)$.

Joining these two tuples, $(a, b, c) \in \pi_{A,B}(R) \bowtie \pi_{A,C}(R)$.

$\pi_{A,B}(R) \bowtie \pi_{A,C}(R) \subseteq R$ does not hold in general (see our example)

But we can show:

Theorem: If $A \to B$ holds in R, then

$$\pi_{A,B}(R) \bowtie \pi_{A,C}(R) \subseteq R$$

Proof:

Let $(a, b, c) \in \pi_{A,B}(R) \bowtie \pi_{A,C}(R)$

Then $(a, b) \in \pi_{A,B}(R)$ and $(a, c) \in \pi_{A,C}(R)$

Therefore, there exist c' and b' such that $(a, b, c') \in R$ and $(a, b', c) \in R$

But $(a, b, c) \in R$ and $(a, b', c) \in R$ implies $b = b'$.

But then $(a, b, c) = (a, b', c) \in R$.

BCNF - Boyce-Codd Normal form

Definition:

A relation is in BCNF iff whenever there is a nontrivial FD

$$A_1 1 A_2 \ldots A_n \rightarrow B_1 B_2 \ldots B_m$$

for R, then $\{A_1, \ldots, A_n\}$ is a key (or superkey) for R

Example

$R(A, B, C)$ with only $A \rightarrow B$ is not in BCNF, since A is not a superkey (only key is AC).

We have seen that there is a losless decomposition into $T(A, B)$ and $U(A, C)$ (which happen to be in BCNF). We shall see that this works in general

Example

`Movies1(title,year,length,genre,studioName,starName)`

is not in BCNF, since the FD

$$\texttt{title}, \texttt{year} \rightarrow \texttt{length}, \texttt{genre}, \texttt{studioName}$$

holds, but $\{\texttt{title}, \texttt{year}\}$ is not a key

On the other hand,

`Movies2(title,year,length,genre,studioName)`

is in BCNF, since $\{\texttt{title}, \texttt{year}\}$ is a key

Decomposition into BCNF

- Repeatedly apply the technique above
- At the end, every schema is in BCNF
- The decomposition is lossless, i.e., we can always reconstruct the original database
- Technique: Look for nontrivial FD whose right-hand side is not a superkey, and break this schema into two
- Important point: What are the FDs in the new schemas?
- For now, we shall derive them informally. We discuss a formal approach later

Example

$R(A, B, C)$ with $A \rightarrow B$

Not in BCNF, since A is not a key

We can decompose into

- $R1(A, B)$: Right-hand side, plus the attributes from the left-hand side
- $R2(A, C)$: Right-hand side, plus all the remaining attributes

We have seen that the decomposition is lossless, i.e., if we project R onto the schemas of $R1$ and $R2$ and then take the join, we get back the original relation

Example

Relation $R(T, Y, L, G, Su, St)$

(Title, Year, Length, Genre,StudioName,StarName)

FD

$$TY \rightarrow LGSu$$

TY not a key. So we can decompose into:

- $R1(T, Y, L, G, Su)$ (attributes of FD)
- $R2(T, Y, St)$ (right-hand side and remaining attributes)

The same proof technique shows that this decomposition is lossless

Another example

Relation R representing

- Movie title (T)
- Movie year (Y)
- Studio name (S)
- President name (P)
- President address (A)

$R(T, Y, S, P, A)$

What are the problems with this schema?

$R(T, Y, S, P, A)$

FDs:

TY $\rightarrow$ S
S $\rightarrow$ P
P $\rightarrow$ A

TY is the only key

Why?

So the last two FDs violate BCNF

Decomposition

We could decompose based on either of these FDs. Let's pick $S \rightarrow P$.

We get two relations

- $R1(S, P)$
- $R2(T, Y, S, A)$

Are these in BCNF?

Intuitively no: the second relates movies to studios and president addresses.

But we appear to have no FD relating S and A

FD inference

However we can *infer* such an FD

Claim:

If $S \to P$ and $P \to A$ hold in a relation, so does $S \to A$

Proof: Direct application of the definition of FD

Therefore, we have (in $R2$) $S \to A$ and S is not a key.

Decompose $R2$:

- $R3(S, A)$
- $R4(T, Y, S)$

$R1(S, P)$, $R3(S, A)$, and $R4(T, Y, S)$ is a decomposition of R into BCNF

Decomposition Algorithm

BCNF Decomposition Algorithm:

INPUT: A relation R_0 with a set of functional dependencies F_0

OUTPUT: A decomposition of R_0 into a collection of relations, all of which are in BCNF.

Apply the following recursively, starting with $R = R_0$ and $S = S_0$.

1. Check whether R is in BCNF. If so, return $\{R\}$.
2. If there are BCNF violations, let one be $X \to Y$ Choose $R_1 = X \cup Y$ and let R_2 have as attributes X and those attributes of R that are not in Y. Find the sets of FDs that hold on R_1 and R_2 (we'll study that next)
3. Recursively decompose R_1 and R_2. Return the union of the results of these decompositions.

Missing step

- Not sufficient to use explicitly given FDs
- If we have $A \to B$ and $B \to C$, then $A \to C$ must hold
- Sometimes we have to use FDs that are derived from existing ones
- In particular: for recursive decompositions, we must know which FDs hold on the new relations

Inference rules

- Trivial FDs: If

$$\{B1, B2 \ldots, Bm\} \subseteq \{A1, A2, \ldots, An\}$$

  then

$$A1, A2, \ldots, An \rightarrow B1, B2, \ldots, Bm$$

- Augmentation: If

$$A1, A2, \ldots, An \rightarrow B1, B2, \ldots, Bm$$

  then

$$A1, A2, \ldots, An, C1, \ldots, Ck \rightarrow B1, B2, \ldots, Bm$$

  for any $C1$, $C2$, $\ldots$, Ck

Inference rules

- Transitivity: If

$$A1, A2, \ldots, An \rightarrow B1, B2, \ldots, Bm$$

and

$$B1, B2, \ldots, Bm \rightarrow C1, C2, \ldots, Ck$$

then

$$A1, A2, \ldots, An \rightarrow C1, C2, \ldots, Ck$$

This is in principle all we need to test for FD inference

In practice, we need something better

FD inference

- Given a set of FDs F, we want to know whether $X \to Y$ is a consequence of X
- But there are lots of possible X and Y!
- Our approach: Given X find the *closure* X^+ of X, which is the set of all attributes that *could be* on the right side of X
- Formally

$$X^+ = \{A \mid X \to A \text{ follows from } F\}$$

Computing the closure

Algorithm:

Input: F set of FDs and set $\{A_1, \ldots, A_n\}$ of attributes. Assume that each FD has only one attribute on the right

Output: Closure $\{A_1, \ldots, A_n\}^+$

1. Set X to be $\{A_1, \ldots, A_n\}$. When the algorithm terminates, X will be the closure
2. Find an FD in F of the form

   $$B_1 B_2 \cdots B_m \rightarrow C$$

   such that B_1, ..., $B_m \in X$ and $C \notin X$
3. If no such F exists, terminate
4. Otherwise, add C to X and go to step 2

Termination:

We only add attributes to X, and there are only a finite number of attributes

FD implication:

To test whether

$$A_1 A_2 \cdots A_n \rightarrow B$$

follows from F, test if B is in $\{A_1 A_2 \cdots A_n\}^+$

Proof: Immediate from the definition of X^+

Example

$R(A, B, C, D, E, F)$

FDs:

$AB \to C$, $BC \to AD$, $D \to E$ and $CF \to B$

What is $\{A, B\}^+$?

First rewrite the second FD as $BC \to A$ and $BC \to D$

- Set $X = \{A, B\}$
- Using $AB \rightarrow C$, X becomes $\{A, B, C\}$
- Using $BC \rightarrow D$, X becomes $\{A, B, C, D\}$
- Using $D \rightarrow E$, X becomes $\{A, B, C, D, E\}$
- No more FDs can be applied, so $\{A, B\}^+ = \{A, B, C, D, E\}$

FD implication

Does $AB \to D$ follow from these FD?

$\{A, B\}^+$ contains D, so it does

On the other hand $AB \to F$ does not

What about $D \to A$?

The algorithm gives $X = \{D\}$, $\{D, E\}$ and terminates. Since A is not in this set the FD does not follow

Why the closure algorithm works

- Fix set of FDs F, and set S of attributes. X be the set computed by the algorithm, and let Y be the closure. We must show $X = Y$
- First we show $X \subseteq Y$
- Proof by induction on the number of steps of the algorithm
- Base case: All attributes in X are already in S
- Assume that $S \to A$ for all $A \in X$. If we apply the algorithm using an FD whose right-hand side is contained in X, we can use the transitivity of FDs to show that all new attributes are also determined by X
- Transitivity: $A \to B$ and $B \to C$ imply $A \to C$

- Other direction: $Y \subseteq X$
- Suppose C is in the closure but not in X, i.e., the algorithm claims that C does not follow from S
- Let the schema of R be $R(A_1 \cdots A_n B_1 \cdots B_m)$ where $X = \{A_1, \ldots, A_n\}$. Consider the instance I:

	A_1	$\ldots$	A_n	B_1	$\ldots$	B_n
t:	1	$\ldots$	1	0	$\ldots$	0
s:	1	$\ldots$	1	1	$\ldots$	1

- If C is in Y and not in X, then it must have value 0 in s and 1 in t
- By induction, show that at each step of the algorithm the new attributed added to X must have value 1 in both tuples
- A contradiction

Back to BCNF

- We know how to find all FDs implied by the given set
- We can therefore find all BCNF violations and decompose the relation if needed
- But we need to find the FDs that hold on the new relations
- We therefore study *projection of FDs*
- Given R, L a subset of attributes of R, and Fds F. What FDs hold on $R_1 = \pi_L(R)$?
- Formally: What FDs follow from F and involve only attributes of L?1
- We shall find a *basis*, a minimal set of FDs that imply all FDs that hold on R_1

Algorithm

Input: R, L, F as above

Output: Set of FDs that hold in $\pi_L(R)$

- Set T to be the empty set, At the end it will be the output
- For all $X \subseteq L$, compute X^+. This may contain attributes not in L
- Add to T all nontrival FDs of the form $X \to A$ when $A \in X^+ \cap L$
- Minimize this set by repeating the following as often as possible
  - If there is an FD in T that follows from the others, delete it
  - If $YZ \to B$ is in T and $Z \to B$ follows from the F, replace the FD by $Z \to B$

Example

- $R(A, B, C, D)$ with FDs $A \to B$, $B \to C$ and $C \to D$. $L = ACD$
- Some closures
  - $\{A\}^+ = \{A, B, C, D\}$. This gives us $A \to C$ and $A \to D$
  - $\{C\}^+ = \{C, D\}$, giving $C \to D$
  - $\{D\}^+ = \{D\}$
  - Other closures give us nothing new
  - We have $A \to C$, $A \to D$ and $C \to D$
  - But the first and the third imply the second
  - The projection is therefore $A \to C$ and $C \to D$

Other normal forms

- First normal form (attributes have simple structure)
- Second normal form (historical interest only)
- Third normal form (3NF): Discussed below
- BCNF
- Fourth normal form (4NF): Discussed below

Any schema in a higher normal form is always in the lower ones

3NF

- Any schema in BCNF is automatically in 3NF, but not vice versa
- Why might we want to not used BCNF?

Example: Relation `Bookings` for movie theaters. Attributes

- `name` (name of movie)
- `theater` (movie theater - unique name)
- `city`
- we abbreviate N, T, C

FDs

- $T \rightarrow C$ (names are unique)
- $NC \rightarrow T$ (don't book same movie in 2 theaters in the same city)

Keys?

- $T^+ = TC$
- $N^+ = N$
- $C^+ = C$
- $NC^+ = NCT$
- $NT^+ = NTC$
- $CT^+ = CT$

So NC and NT are keys

$T \rightarrow C$ violates BCNF

Decomposition: TC and TN

If we represent the data in this form, there is no way to enforce the FD $NC \rightarrow T$, as the attributes are split between different relations

Third normal form (3NF) is a somewhat technical definition that does not imply BCNF

There is always a 3NF decomposition that preserves all the dependencies

In this example it will be the same as the original schema

Note: Because 3NF is not always in BCNF, it may have problems with update anomalies, and the decision which to use is a tradeoff between anomalies and preserving dependencies

Fourth normal form (4NF)

Consider the relation

`Movies (starName, street, city, title, year)`

No nontrivial dependencies!

- `starName` not a key, as may be in several movies
- `title, year` not a key as may have several stars
- Etc. (try other combinations to convince yourself)

But there is clearly redundancy

starName	street	city	title	year
C. Fisher	123 Maple	Hollywood	Star Wars	1977
C. Fisher	5 Locust	Malibu	Star Wars	1977
C. Fisher	123 Maple	Hollywood	Empire strikes back	1980
C. Fisher	5 Locust	Malibu	Empire strikes back	1980
C. Fisher	123 Maple	Hollywood	Return of Jedi	1983
C. Fisher	5 Locust	Malibu	Return of Jedi	1983

Note that whenever a star has several addresses, every address must be paired with every movie of this star

Formally, if $(sn, st_1, c_1, t_1, y_1) \in R$ and $(sn, st_2, c_2, t_2, y_2) \in R$ then $(sn, st_1, c_1, t_2, y_2)$ and $(sn, st_2, c_2, t_1, y_1)$ must also be in R

This is called a *multivalued dependency* and is written

$$\texttt{starName} \twoheadrightarrow \texttt{street}, \texttt{city}$$

or, equivalently

$$\texttt{starName} \twoheadrightarrow \texttt{title}, \texttt{year}$$

There are techniques to infer MVDs, and to decompose into 4NF where there are no MVDs except where the left-hand side is a superkey (an FD is automatically a MVD)

However

- MVD inference is much harder than FD inference
- It is difficult to determine which MVDs hold (though there are techniques to help find them)
- But it's important to be aware of the problem and try to decompose when possible