CS145 Lecture Notes #5

Relational Database Design: FD's & BCNF

Motivation

- Automatic translation from E/R or ODL may not produce the best relational design possible
- Sometimes database designers like to start directly with a relational design, in which case the design could be really bad

Notation

- R, S, ... denote relations
- $attrs(R)$ denotes the set of all attributes in R
- A, B, ... denote attributes
- X, Y, ... denote sets of attributes

Functional Dependencies

A *functional dependency* (FD) has the form $X \to Y$, where X and Y are sets of attributes in a relation R

- Formally, $X \to Y$ means that whenever two tuples in R agree on all the attributes of X, they must also agree on all the attributes of Y

Example: FD's in `Student(SID, SS#, name, CID, grade)`

Some FD's are more interesting than others:

- *Trivial* FD: $X \to Y$ where Y is a subset of X
  Example:
- *Nontrivial* FD: $X \to Y$ where Y is not a subset of X
  Example:
- *Completely nontrivial* FD: $X \to Y$ where Y and X do not overlap
  Example:

Once we declare that an FD holds for a relation R, this FD becomes a part of the relation schema

$\leadsto$ Every instance of R must satisfy this FD

$\leadsto$ This FD should better make sense in the real world!

A particular instance of R may coincidentally satisfy some FD

$\leadsto$ But this FD may not hold for R in general

Example: `name` $\to$ `SID` in `Student`?

FD's are closely related to:

- Multiplicity of relationships
  Example: Queens, Overlords, Zerglings

- Keys
  Example: { `SID`, `CID` } is a key of `Student`

$\leadsto$ Another definition of key: A set of attributes K is a key for R if

(1) $K \to attrs(R)$; i.e., K is a *superkey*

(2) No proper subset of K satisfies (1)

Closures of Attribute Sets

Given R, a set of FD's $\mathcal{F}$ that holds in R, and a set of attributes Z in R:

- The *closure of Z with respect to $\mathcal{F}$* (denoted Z^+) is the set of all attributes that are functionally determined by Z

$\leadsto$ Yet another definition of key: A set of attributes K is a key for R if

(1) $K^+ = attrs(R)$; i.e., K is a superkey

(2) No proper subset of K satisfies (1)

Question: Given R and $\mathcal{F}$, what is the closure of Z?

- Start with Z
- If $X \to Y$ is a given FD and X is already inside the closure, then also add Y to the closure
- Repeat until the closure cannot be changed

Example: $\{\texttt{SID}, \texttt{CID}\}^+ = attrs(\texttt{Student})$

Question: Given R and $\mathcal{F}$, what are the keys of R?

- Brute-force approach: for every subset of $attrs(R)$, compute its closures and see if it covers $attrs(R)$

$\rightsquigarrow$ Trick: start with small subsets; if $X^+ = attrs(R)$, no need to try any superset of X

$\rightsquigarrow$ Trick: if A does not appear on the right-hand side of any FD, then every key must contain A

Example: what are the keys of `Student`?

Closures of FD Sets

Given R and a set of FD's $\mathcal{F}$ that holds in R:

- The *closure of* $\mathcal{F}$ in R (denoted $\mathcal{F}^+$) is the set of all FD's in R that are logically implied by $\mathcal{F}$

Question: Given R and $\mathcal{F}$, is $X \rightarrow Y$ implied by $\mathcal{F}$?
(Or, given R and $\mathcal{F}$, is $X \rightarrow Y$ in $\mathcal{F}^+$?)

- Method 1: compute X^+ and check if it contains Y
- Method 2: try to prove $X \rightarrow Y$ using *Armstrong's Axioms*:
  - Reflexivity: if $Y \subseteq X$, then $X \rightarrow Y$
  - Augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any set Z
  - Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

  or using other rules that follow from the axioms:
  - Splitting: if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - Combining: if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$

Example: prove that `SS#`, `CID` $\rightarrow$ `name`, `grade`

Basis

When specifying FD's for a relation R:

- Obviously we do not want to list *all* FD's that hold in R
- Instead, it suffices to specify a set of FD's from which all other FD's will follow logically; this set of FD's is a *basis* for the FD's in R
- In fact, we should specify a *minimal* basis
  - Every FD in the minimal basis is necessary; it cannot be proven using other FD's in the minimal basis
  - Sounds tough, but in practice the minimality comes naturally
  - There might be multiple minimal bases

Example: what is a minimal basis for the FD's in `Student`?

BCNF (Boyce-Codd Normal Form)

A relation R is in *BCNF* if:

- For every nontrivial FD $X \rightarrow Y$ in R, X is a superkey

In other words:

- All FD's follow from the fact "key $\rightarrow$ everything"

Intuition:

- When an FD is *not* of the form "superkey $\rightarrow$ other attributes", then there is typically an attempt to cram too much into one relation; this relation needs to be decomposed

Example: `SID` $\rightarrow$ `SS#` is a BCNF violation

$\rightsquigarrow$ the `SID`/`SS#` association is repeated multiple times

BCNF Decomposition Algorithm

- Start with the relation in question
- Repeat until no BCNF violation can be found in any of your relations:
  - Find a BCNF violation $X \rightarrow Y$ in R
  - Decompose R into two relations:
    - ◦ One with $X \cup Y$ as its attributes (i.e., everything in the FD)
    - ◦ One with $X \cup (attrs(R) - X - Y)$ as its attributes (i.e., left side of the FD plus everything not in the FD)

Example:
`Students(SID,SS#,name,CID,grade)`
SID $\rightarrow$ SS#
SS# $\rightarrow$ name
SS# $\rightarrow$ SID
SID, CID $\rightarrow$ grade

- In general, you may need to decompose several times
- To check for BCNF violations in R, we need to know:
  - *All* keys of R
  - A basis for the FD's that hold in R
  - Do we need to check any FD that is not in the basis but follows from the basis?
    $\rightsquigarrow$ No. If there is no BCNF violation in a basis, then there is no BCNF violation at all *(why?)*
- After the first iteration, the algorithm requires FD's to be "projected" onto smaller relations
  $\rightsquigarrow$ Be careful when deriving an FD basis for a smaller relation: don't miss any FD that follows from the FD's in the original relation (see textbook for an exhaustive algorithm; can usually do it with common sense though)
  Example: SID $\rightarrow$ name
- An optimization: instead of decomposing on any BCNF violation $X \rightarrow Y$, decompose on $X \rightarrow X^+$
  $\rightsquigarrow$ This strategy avoids excessive fragmentation
  Example: decompose on SID $\rightarrow$ SS#, name instead of SID $\rightarrow$ SS#

BCNF = Good Design?

- BCNF removes all redundancies caused by FD's
- BCNF can decompose relations "too much" and complicate queries and constraint enforcement
  Example: if we decompose Student on SID $\rightarrow$ SS#, it will be difficult to enforce SS# $\rightarrow$ name*
- BCNF does *not* remove all redundancies in general
  Example: Student(SID, club, CID) has no FD's, but still redundancy

---

*Actually this example is not good: it turns out that we can enforce SS# $\rightarrow$ name by enforcing SS# $\rightarrow$ SID and SID $\rightarrow$ name independently in two different relations. For an example that makes more sense, stay tuned for the next lecture on the theory of decomposition.