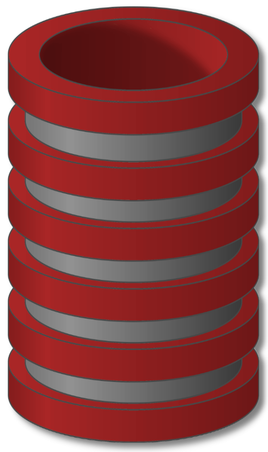


Relational Design Theory

Functional Dependencies

Relational design by decomposition

- "Mega" relations + properties of the data
- System decomposes based on properties
- Final set of relations satisfies normal form
  - No anomalies, no lost information
- Functional dependencies ⇒ Boyce-Codd Normal Form
- Multivalued dependences ⇒ Fourth Normal Form

Functional dependencies are generally useful concept

- Data storage – compression
- Reasoning about queries – optimization

Example: College application info.

```
Student(SSN, sName, address,
        HScode, HSname, HScity, GPA, priority)
Apply(SSN, cName, state, date, major)
```

$GPA \geq 3.8 \wedge prio = 1$

$3.3 \leq GPA < 3.8 \wedge prio = 2$

$GPA < 3.3 \wedge prio = 3$

FD: $GPA \rightarrow Priority$

f.a. $r, s \in Student$: $r[GPA] = s[GPA] \wedge r[Prio] = s[prio]$


```
Student(SSN, sName, address,
        HScode, HSname, HScity, GPA, priority)
```

Suppose `priority` is determined by `GPA`

FD: $GPA \rightarrow priority$

if f.a. $t, s \in R$

$t[A_1, \ldots, A_n] = s[A_1, \ldots, A_n]$

then

$\Rightarrow t[B_1, \ldots, B_k] = s[B_1, \ldots, B_k]$

then the FD $\underbrace{A_1, \ldots, A_n}_{\bar{A}} \rightarrow \underbrace{B_1, \ldots, B_k}_{\bar{B}}$ holds

Functional Dependencies

`Student(SSN, sName, address,`
`        HScode, HSname, HScity, GPA, priority)`

Two tuples with same `GPA` have same `priority`

To test whether SSN → SNAME holds, we can use query:

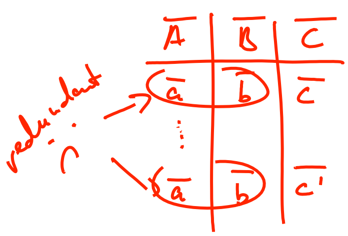
FD_viol(SSN, SName1, SName2) ← Student(SSN, SName1, ...), Student(SSN, SName2, ...), SName1 ≠ SName2.

Functional Dependencies

Functional Dependency

- Based on knowledge of real world
- All instances of relation must adhere

FD: $\bar{A} \to \bar{B}$



Functional Dependencies

Example

```
Student(SSN, sName, address,
        HScode, HSname, HScity, GPA, priority)
```

FDs:

SSN → SName
SSN → addr
HSCode → HSName, HSCity
HSName, HSCity → HSCode
SSN → GPA
GPA → priority
SSN → priority (transitivity)

Functional Dependencies

```
Apply(SSN, cName, state, date, major)
```

SSN, cName → major (?)
cName, state → date

Functional Dependencies and Keys

- Relation with no duplicates
- Suppose $\bar{A} \rightarrow$ all attributes

keys are special FDs!

FD : $\bar{A} \rightarrow \bar{B}$

$\bar{A}$	$\bar{B}$
$\bar{a}$	$\bar{b}$
⋮	
$\bar{a}$	$\bar{b}$

if $\bar{A}$ is a key

Functional Dependencies

normally not considered

Trivial Functional Dependency

$\bar{A} \rightarrow \bar{B}$ $\qquad \bar{A} \supseteq \bar{B}$

SSN, SName → SName

Nontrivial FD

$\bar{A} \rightarrow \bar{B}$ $\qquad \bar{A} \not\supseteq \bar{B}$

$X, Y \rightarrow X, Z$

Completely nontrivial FD

$\bar{A} \rightarrow \bar{B}$ $\qquad \bar{A} \cap \bar{B} = \phi$

$X, Y \rightarrow Z$

Rules for Functional Dependencies

Splitting rule

$$\frac{\bar{A} \to B_1, \ldots, B_k}{\begin{array}{c}\bar{A} \to B_1 \\ \vdots \\ \bar{A} \to B_k\end{array}}$$

Can we also split left-hand-side?

$$\frac{A_1, A_2 \to \bar{B}}{\begin{array}{c}A_1 \to \bar{B} \\ A_2 \to \bar{B}\end{array}} \; ?$$

NO

HsName, HsCity → HsCode

Functional Dependencies

Rules for Functional Dependencies

Combining rule

$$\frac{\begin{array}{c}\bar{A} \to B_1 \\ \bar{A} \to B_2\end{array}}{\bar{A} \to B_1, B_2}$$

Functional Dependencies

Rules for Functional Dependencies

Trivial-dependency rules

1. $\bar{A} \rightarrow \bar{A} \cup \bar{B}$

2 $\bar{A} \rightarrow \bar{A} \cap \bar{B}$

Functional Dependencies

Rules for Functional Dependencies

Transitive rule

$$\frac{\begin{array}{c}\bar{A} \rightarrow \bar{B} \\ \bar{B} \rightarrow \bar{C}\end{array}}{\bar{A} \rightarrow \bar{C}}$$

Closure of Attributes

- Given relation, FDs, set of attributes Ā
- Find all B such that Ā → B

$S := \{\overbrace{A_1, \ldots, A_n}^{\bar{A}}\}$

Repeat

if $\bar{X} \to \bar{Y} \in$ FDs

and $\bar{X} \subseteq S$

then

$S := S \cup \bar{Y}$

Until S doesn't change

$\bar{A}^+ :=$ closure$(\bar{A}) := S$

Closure Example

```
Student(SSN, sName, address,
        HScode, HSname, HScity, GPA, priority)
```

```
SSN → sName, address, GPA
GPA → priority
HScode → HSname, HScity
```

Functional Dependencies

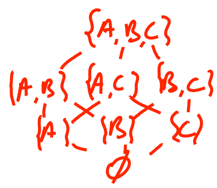
Closure and Keys

Is $\bar{A}$ a key for R ?

1. compute $\bar{A}^+$
2. if $\bar{A}^+ = attr(R)$ then $\bar{A}$ is a key!

How can we find all keys given a set of FDs?

R(A,B,C)



Functional Dependencies

Specifying FDs for a relation

- S_1 and S_2 sets of FDs
- S_2 "follows from" S_1 if every relation instance satisfying S_1 also satisfies S_2

How to test?
Does A → B follow from S ?

(i) Armstrong Relations
(ii) A^+, check if B is in set

Specifying FDs for a relation

Want: *Minimal set of completely nontrivial FDs such that all FDs that hold on the relation follow from the dependencies in this set*

Functional dependencies are generally useful concept

- Relational design by decomposition
  Functional dependencies ⇒ Boyce-Codd Normal Form
- Data storage – compression
- Reasoning about queries – optimization

Summary on Functional Dependencies (FDs)

- FD $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$
  - ... *functionally determines* …
  - Check this at runtime via a "denial query"
  - But here it's about *reasoning* for all db instances
- Generalization of keys:
  - Relation R(**A**, **B**)
  - **A** → **B** (and thus **A** → **A**,B**)**
  - … thus **A** is a (super)key of R

Summary on Functional Dependencies (FDs)

- Trivial vs (completely) nontrivial FDs
- Inference rules for FDs
  - Splitting rule
  - Combining rule
  - Transitive rule
- Armstrong rules (cf. Textbook)
  - Reflexivity: **X** → **X**
  - Augmentation: **X** → **Y** implies **XZ** → **YZ**
  - Transitivity: **X** → **Y** and **Y** → **Z** implies **X** → **Z**
- Armstrong rules are *sound* & *complete*:
  - Sound: inferences are correct
  - Complete: all implied FDs can be inferred

Summary on Functional Dependencies (FDs)

- Closure Algorithm $\mathbf{A}^+$
  - Given set **F** of FDs over R, set **A** of attributes
  - $\mathbf{A}^+ := \mathbf{A}$
  - Repeat
    - for all $\mathbf{X} \rightarrow \mathbf{Y}$ in **F**:
    - if **X** subset $\mathbf{A}^+$
    - … then $\mathbf{A}^+ := \mathbf{A}^+$ union **Y**
  - Until no change

Functional Dependencies

Closure and Keys

Check: Is $\bar{A}$ a key for R ?

1. Compute $\bar{A}^+$
2. if $\bar{A}^+ = \text{attr}(R)$ then $\bar{A}$ is a (super-) key!

How can we find all keys given a set of FDs?

Consider all subsets $S \subseteq \bar{A}$ $\rightsquigarrow |\bar{A}| = n$

$\rightsquigarrow 2^n$ subsets

http://class2go.stanford.edu/db/Winter2013

Functional Dependencies

Specifying FDs for a relation

- S_1 and S_2 sets of FDs
- S_2 "follows from" S_1 if every relation instance satisfying S_1 also satisfies S_2

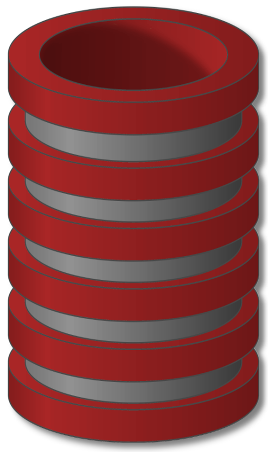
How to test?
Does $\bar{A} \rightarrow \bar{B}$ follow from S ?

(i) $\bar{A}^+$ based on S, see if $\bar{B} \subseteq \bar{A}^+$
(ii) Armstrong's Rules

Understanding FDs with Datalog …

⇒ web site

```
% R(A,B,C)
r(a1,b1,c1).
r(a1,b1,c2).
%r(a1,b2,c1).

% FD: R.A --> R.B
%:- r(A,B1,_), r(A,B2,_), B1 != B2.

% Report violation of FD: R.A --> R.B
ic(fdRAB,A,B1,B2) :- r(A,B1,_), r(A,B2,_), B1 != B2.

% Check for BCNF:
% Given FD R.A --> R.B, then A should be a key!

notBCNF1(A,B1,B2) :- r(A,B1,C1), r(A,B2,C2), B1 != B2.
notBCNF2(A,C1,C2) :- r(A,B1,C1), r(A,B2,C2), C1 != C2.

% DECOMPOSE!
r1(A,B) :- r(A,B,_).
r2(A,C) :- r(A,_,C).

% See whether we can get r back via a joining the decomposed relations:

rj(A,B,C) :- r1(A,B), r2(A,C).

diff(r_too_big, A,B,C) :- r(A,B,C), not rj(A,B,C).
diff(rj_too_big, A,B,C) :- rj(A,B,C), not r(A,B,C).
```



Relational Design Theory

Boyce-Codd Normal Form

BCNF

Relational design by decomposition

- "Mega" relations + properties of the data
- System decomposes based on properties
- Final set of relations satisfies normal form
  - No anomalies, no lost information
- FDs Functional dependencies ⇒ Boyce-Codd Normal Form
- MVDs Multivalued dependences ⇒ Fourth Normal Form

BCNF

Decomposition of a relational schema

$R(A_1, \ldots, A_n)$ $\bar{A}$

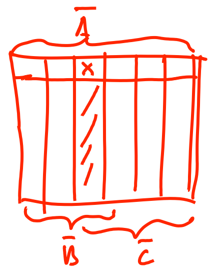
$\hookrightarrow R_1(B_1 .. B_m)$ $\bar{B}$

$\hookrightarrow R_2(C_1 .. C_l)$ $\bar{C}$

$\bar{A} = \bar{B} \cup \bar{C}$

$R = R_1 \bowtie R_2$

lossless-join property (textbook, Ch 16: non-additive)



$R_1 = \pi_{\bar{B}}(R)$

$R_2 = \pi_{\bar{C}}(R)$

$\underbrace{R_1 \bowtie R_2}_{\text{this can be larger than } R} \supseteq R$

$\Rightarrow$ lossless-join violated

BCNF

Decomposition Example #1

```
Student(SSN, sName, address,
        HScode, HSname, HScity, GPA, priority)
```

FD: HScode → HSName, HScity

$\hookrightarrow S_1$ (<u>HScode</u>, HSNAME, HScity)

$\hookrightarrow S_2$ (HScode, <u>SSN</u>, sName, addr, GPA, priority) key

attr(Student) = attr(S_1) ∪ attr(S_2) ✓

Student $\stackrel{?}{=}$ $S_1 \bowtie S_2$??

BCNF

Decomposition Example #2

```
Student(SSN, sName, address,
        HScode, HSname, HScity, GPA, priority)
```

S_1(SSN, SName, HSCode, HSName, HSCity)

S_2(SName, HSName, GPA, priority)

$\overline{\text{Student}} = \overline{S_1} \cup \overline{S_2}$ ✓

$S_1 \bowtie S_2 \overset{?}{=}$ Student ↝ no! (in general $S_1 \bowtie S_2 \supseteq$ Student, $\neq$)

BCNF

Relational design by decomposition

- "Mega" relations + properties of the data
- System decomposes based on properties

❖ "Good" decompositions only ↝ lossless (nonadditive) join

❖ Into "good" relations ↝ BCNF

BCNF

Def.

Boyce-Codd Normal Form

Relation R with FDs is in BCNF if:

For each Ā → B, Ā is a key (superkey) BCNF condition

nontrivial FD

What happens if BCNF condition violated?

R:

Ā	B	Rest
$\bar{a}$	b	..
$\bar{a}$	b	...

redundant info in R! :(

BCNF

Test

BCNF? Example #1

Student(SSN, sName, address,
HScode, HSname, HScity, GPA, priority)

FDs

SSN → sName, address, GPA

GPA → priority

HScode → HSname, HScity

key = {SSN, HScode}

Violates BCNF!

Student/8 is not in BCNF (neither of

- SSN
- GPA
- HScode

is a key)

BCNF

BCNF? Example #2

`Apply(SSN, cName, state, date, major)`

FD: $\underbrace{\texttt{SSN, cName, state}}_{\text{Key}} \rightarrow \texttt{date, major}$

→ BCNF ✓

BCNF

Relational design by decomposition

- "Mega" relations + properties of the data
- System decomposes based on properties

❖ "Good" decompositions only

❖ Into "good" relations

BCNF

BCNF decomposition algorithm

Input: relation R + FDs for R

Output: decomposition of R into BCNF relations with “lossless join”

1. Compute keys for R $A \rightarrow A^+$

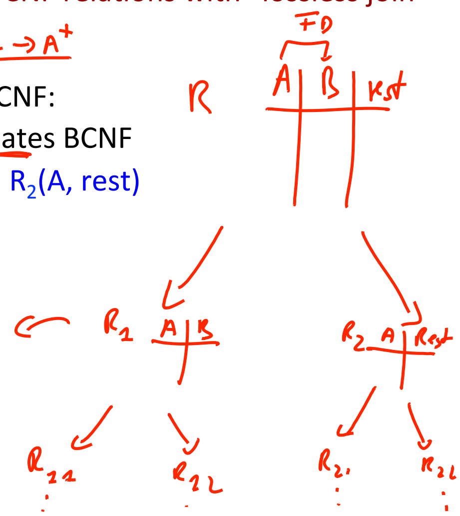
2 Repeat until all relations are in BCNF:

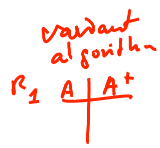
  Pick any R' with $A \rightarrow B$ that violates BCNF

non-det. Decompose R' into R_1(A, B) and R_2(A, rest)

  Compute FDs for R_1 and R_2

  Compute keys for R_1 and R_2





BCNF

BCNF Decomposition Example

```
Student(SSN, sName, address, HScode, HSname, HScity,
        GPA, priority)
```

SSN → sName, address, GPA   GPA → priority

HScode → HSname, HScity

BCNF

Does BCNF guarantee a good decomposition?

- Removes anomalies?
- Can logically reconstruct original relation?

Too few or too many tuples?

FD FD are not given here ↝ invalid decomposition

R

A	B	C
1	2	3
4	2	5

$\xrightarrow{\text{decompose}}$

R_1

A	B
1	2
4	2

R_2

B	C
2	3
2	5

$$R' = R_1 \bowtie R_2 \stackrel{?}{=} R$$

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