Database design:

You are given a big relation schema R and a set of fds F involving attributes of R.

Need to come up with a set of probably smaller schemas that can store the same information.

<u>Consider:</u>

Enrol = (s_id, c_id, grade, gpa)

F = { s_id, c_id → grade
s_id → gpa }.

What is the problem with this schema?

If a student is enrolled in two or more courses, student's gpa is repeated.

"Duplication of information, i.e. redundancy".

Bad for updates - Need to keep them consistent

If a student is not enrolled in any course but has a 'gpa', how to store this information?

"Inability to represent certain info" ?

Normal Forms:

Tell you when a relation schema is "good" with respect to a set of fds.

BCNF : Boyce Codd Normal Form:

Def: A relation schema R is in BCNF with respect to a set of fds F if one of the following conditions satisfied for every fd $\alpha \to \beta$ in F^+:

(a) $\alpha \to \beta$ is a trivial fd (i.e., $\beta \subseteq \alpha$)

(b) α is a super key of R, i.e., $\alpha \to R$ is in F^+

examples: ex 1: Accounts = (acct #, bal, br-name)

F = { acct # → bal, br-name }

ex 2: Accounts = (acct #, bal, br-name, cust-name)

F = { acct # → bal, br, name, cust-name } ex 3: Enrol = (S-id, C-id, grade, gpa)

F = { S-id, C-id → grade,
S-id → gpa }

not in BCNF

ex 4: Account-Deposit = (Cust-name, acct #, bal, br-name)

F = { acct # → bal, br-name }

not in BCNF

What should be done if not in BCNF?

Split the relation schema, i.e. decompose into smaller schemas.

Decomposition: A decomposition of R is a set of relation schemas $(R_1, R_2, \ldots R_k)$ such that

$$R = R_1 \cup R_2 \cup \ldots \cup R_k.$$

Given a relation schema R and a set of fds F, when is a decomposition good?

Ex: Accounts-deposit = (cust-name, Acct #, bal, br-name)

F = { Acct # → bal, br-name }

Consider the decomposition:

R_1 = (cust-name, br-name), R_2 = (Acct #, bal, br-name)

is this a good decomposition?

No.

why? Consider the instance of Accounts-deposit

cust-name	Acct#	bal	br-name
John	100	10K	UIC
Mary	100	10K	UIC
Jeff	200	5K	UIC

instance of R_1

cust-name	br-name
John	UIC
Mary	UIC
Jeff	UIC

instance of R_2

Acct#	bal	br-name
100	10K	UIC
200	5K	UIC

The only way to get the original instance of Accounts-deposit is to take the join of the two instances of

R_1, R_2.

~~when~~ The result of join:

Cust-name	Acct#	bal	br-name
John	100	10K	UIC
John	200	5K	UIC
Mary	100	10K	UIC
Mary	200	5K	UIC
Jeff	100	10K	UIC
Jeff	200	5K	UIC

get additional tuples.

Means loss of information!!

Loss-less Join decomposition:

Given R, F.

Def: A decomposition $(R_1, R_2, \ldots, R_k)$ of R is a <u>loss less join</u> decomposition w.r.t. F if for every <u>legal</u> instance r of R w.r.t. F the following condition holds:

$$\pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \ldots \bowtie \pi_{R_k}(r) = r. \qquad -(1)$$

(Note that the left side in (1) is always a super set of the right side)

Clearly, the above does not hold for the previous

example.
Consider the following decomposition of Accounts-deposit

$R_1 = (\text{Cust-name}, \text{Acct\#})$ $\quad R_2 = (\text{Acct\#}, \text{bal}, \text{br-name})$

Recall $F = \{ \text{Acct\#} \rightarrow \text{bal}, \text{br-name} \}$

In this case, the decomposition is a <u>loss less join</u> decomposition.

why?

Sufficient Condition for loss-less join property:

Thm: Given R, F, ~~the~~ and given the decomposition (R_1, R_2) of R, this decomposition satisfies the Loss-less join property if

$R_1 \cap R_2 \rightarrow R_1$ is in F^+ (i.e, $R_1 \cap R_2$ is a Super key of R_1)

or

$R_1 \cap R_2 \rightarrow R_2$ is in F^+ (i.e., $R_1 \cap R_2$ is a Super key of R_2).

Proof: Suppose r is a legal instance of R and $R_1 \cap R_2 \rightarrow R_1$ is in F^+, i.e., $R_1 \cap R_2$ is a super key of R_1.

Now consider the join of the tabes $\Pi_{R_1}(r)$ and $\Pi_{R_2}(r)$ #2

For every tuple t_2 in $\Pi_{R_2}(r)$ there is exactly one tuple t_1 in $\Pi_{R_1}(r)$ with which t_2 joins.

Hence we get exactly r, i.e., ~~there~~ will not be any extra tuples in the result.

Suppose we perform a sequence of decompositions each of which is loss less then the final decomposition is loss less.

R

R₁ R₂ — are loss less.

R_3 R_4 R_5 R_6

The final decomposition is (R_3, R_4, R_5, R_6)

Ex: R = (cust-name, street, cust-city, acct#, bal, br-name, br-city, assets)

F = { cust-name → street, cust-city
acct # → bal, br-name
br-name → br-city, assets }

R

R_1 → (cust-name, street, cust-city)

R_2 (cust-name, acct#, bal, br-name, br-city, assets)

R_3 (cust-name, acct#, bal, br-name)

R_4 (br-name, br-city, assets)

R_5 (cust-name, acct#)

R_6 (acct#, bal, br-name)

Final decomposition (R_1, R_5, R_6, R_4) is a loss-less join decomp.

"Loss-less join property" is critical and a must.

Another property called "dependency preservation" is desirable.

Given R, F

Consider a decomposition $(R_1, R_2, \ldots, R_k)$ of R.

$F_1 \quad F_2 \quad F_k$

Let F_1 be the set of all fds in F^+ whose attributes are from R_1

F_i be the set of all fds in F^+ whose attributes are from R_i; i.e, $F_i = \{\alpha \to \beta \in F^+ \mid \alpha, \beta \subseteq R_i\}$

Thus we have F_1 on R_1

F_2 on R_2

$\vdots$

F_k on R_k

The decomposition is dependency preserving if

$$F \subseteq (F_1 \cup F_2 \cup \cdots \cup F_k)^+$$

Ex: $R = (\text{City}, \text{Street}, \text{ZipCode})$

$F = \{ \text{City}, \text{Street} \to \text{ZipCode}$

$\text{ZipCode} \to \text{City} \}$

$R_1 = (\text{Street}, \text{ZipCode})$, $R_2 = (\text{City}, \text{ZipCode})$

$F_1 = $ Trivial fds

$F_2 = \{ \text{ZipCode} \to \text{City} \} \cup$ Trivial fds.

$F \nsubseteq (F_1 \cup F_2)^+$. Is not dependency preserving?

Algorithm for obtaining a loss-less join and BCNF decomposition:

Input: R, F

Output: a loss-less join and BCNF decomposition. (i.e., all relation schemas in the output are in BCNF).

final output in 'result'.

1. Compute F^+

2. result := {R}

3. done := False;

While not done

{ **if** $\exists R_i \in$ result such that R_i is not in BCNF

{ let $\alpha \to \beta \in F^+$ be such that

$\alpha, \beta \subseteq R_i$, $\alpha \cap \beta = \phi$ and

α is not a super key of R_i

(i.e., $\alpha \to R_i \notin F^+$;

result := result $- \{R_i\} \cup \{\alpha\beta\} \cup \{R_i - \beta\}$

else

done := true;

}

Features of the alg.:

Starts with result = $\{R\}$

- At each step choses a relation schema R_i in result that is not in BCNF.
- It choses an "offending" fd $\alpha \to \beta \in F^+$ such that $\alpha \cap \beta = \phi$, α is not a super key of R_i;

It replaces R_i by $(R_i - \beta)$ and $(\alpha\beta)$

(i.e., it decomposes R_i into $(R_i - \beta)$, $(\alpha\beta)$)

(the decomposition is loss-less because α is a super key of $(\alpha\beta)$.

)

- It stops when all the relation schemas in result are in BCNF.

Ex:

R = (cust-name, acct #, bal, br-name, br-city, assets,
cust-city, street)

F = { cust-name → cust-city, street
acct # → bal, br-name
br-name → br-city, assets }

result = { R }

R not in BCNF

1. Choose the fd : cust-name → cust-city, street

result = { $\overbrace{\text{(cust-name, acct\#, bal, br-name, br-city, assets)}}^{R_1}$,
$\underbrace{\text{(cust-name, cust-city, street)}}_{R_2}$ }

The first relation R_1 not in BCNF;
choose the fd : ~~acct # → bal, br-name~~ br-name → br-city, assets

2. decompose R_1 into
R_3 = (cust-name, acct #, bal, br-name)
R_4 = (br-name, br-city, assets)

R_3 not in BCNF
choose acct # → bal, br-name

3. decompose R_3 into
R_5 = (cust-name, acct #) R_6 = (acct #, bal, br-name)

Final result : { R_2, R_4, R_5, R_6 }

Does the decomposition Satisfy "dependency preservation" ?

$F_2 = \{ \text{Cust-name} \rightarrow \text{Cust-city, street} \} \cup$ Trivial fds

$F_4 = \{ \text{br-name} \rightarrow \text{br-city, assets} \} \cup$ Trivial fds

$F_5 =$ Trivial fds

$F_6 = \{ \text{acct\#} \rightarrow \text{bal, br-name} \} \cup$ Trivial fds

Clearly $\quad F \subseteq (F_1 \cup F_2 \cup F_3 \cup F_4)^+$.

Hence dependency preserving.

Suppose we choose the fd

acct# → bal, br-name

in the 2nd step, we get a different decomp.

acct# → bal, br-name

acct# → br-city, assets

R

R_2

(cust-name, cust-city, street)

R_1

(cust-name, acct#, bal, br-name, br-city, assets)

cust-name → cust city, street

(acct#, bal, br-name)

R_3

R_4

(cust-name, acct#, br-city, assets)

acct# → br-city, assets

(acct#, br-city, assets)

R_5

(cust-name, acct#)

R_6

R_2, R_3, R_5, R_6

F_6 = ...

F_2

F_3 = acct# → bal, br-name

F_5 = {acct# → br-city, assets}

br-name → br-city, assets

The previous algorithm does not guarantee dependency preservation!

For a given R and F: Can we get a <u>loss-less</u>, <u>BCNF</u> and <u>dependency preserving</u> decomposition

Consider the following example:

$$R = (City, Street, Zipcode)$$

$$F = \{ City, Street \rightarrow ZipCode$$
$$Zipcode \rightarrow City \}$$

Cleary R is not in BCNF.

Consider the following decomposition:

$R_1 = (ZipCode, City)$ $R_2 = (Street, ZipCode)$

$F_1 = \{ Zipcode \rightarrow City \} \cup$ trivial $F_2 = \phi \cup$ trivial

It is a loss-less join decomposition.

Both R_1, R_2 are ~~depend~~ in BCNF.

It is not dependency preserving.

BCNF ~~prop~~ normal form is too strong!!

3rd Normal Form:

Def: Relation schema R is in 3NF with respect to a set F of fds, if for every $\alpha \rightarrow \beta$ in F^+, ~~one~~ at least one of the following conditions is satisfied:

(i) $\alpha \rightarrow \beta$ is a trivial fd (i.e, $\beta \subseteq \alpha$)

(ii) α is a super key of R (i.e, $\alpha \rightarrow R \in F^+$)

(iii) each attribute in $\beta - \alpha$ is contained in a candidate key of R. (called a prime attribute)

Ex1: The relation schema R = (City, street, Zipcode)

F: { City, street → ZipCode, ZipCode → City }

is in 3NF with resp. F.

Ex2: R = (Cust-name, br-name, banker-name)

F = { Cust-name, br-name → banker-name

banker-name → br-name }

is in 3NF (not in BCNF)

3NF allows some duplication of information

Any R that is in BCNF with resp. to F

is in 3NF with resp. to F.

The Converse is not always true!

~~We~~ Given R, F ; we can always get

a decomposition that ~~is~~ satisfies loss-less join

property, that is dependency preserving

and all relations schemas are in 3NF.

~~Ideally~~

Given R, F, ideally we like to get a decomposition

Satisfying ~~(i)~~

(i) BCNF property (i.e., every relation schema is in BCNF)

(ii) loss-less join property

(iii) dependency preserving.

If it is not possible we get a decomposition

satisfying (i) 3NF property (i.e, every schema is in 3NF)

and satisfying properties (ii) and (iii).

R = (cust-name, banker-name, br-name, Cust-city, Cust-street)

F = { Cust-name → Cust-city, Cust-street

banker-name → br-name

Cust-name, br-name → banker-name }

R_1 = (cust-name, Cust-city, Cust-street)

R_2 = (Cust-name, br-name, banker-name).

The above decomposition ~~is~~ is a 3NF decomposition satisfying loss-less join property and dependency preservation.