Relational Algebra

$f(x)$ $\quad$ $A + B$ $\quad$ A'

- is a procedural query language

- fundamental operations $\qquad D_1 \times D_2 \times D_3$

  - Select
  - Project
  - union
  - set difference
  - Cartesian Product
  - rename

unary operations

binary operations

- set intersection
- natural join
- division
- assignment

Select (σ) — selects tuples that satisfy a given predicate (appears as a subscript to σ)

query in RA: $\sigma_{br\_nm = \text{"mall\_Rd"}}(account)$

~~relation~~ Anonymous

o/p

mall_Rd	A-101	500
mall_Rd	–	–

=, ≠, <, >, ≤, ≥ — allowed

$\wedge$ — ~~and~~ AND

$\vee$ — OR

query in RA: $\sigma_{br\_nm = \text{"mall\_Rd"} \wedge amount > 1200000}(loan)$

Project operation (π)

- Returns its argument relation with certain att's left out. duplicate rows are eliminated. Att's that we wish to appear in the resultant relation is given as subscript to π

$$\pi_{ln\text{-}num,\ amount}(loan)$$

o/p

ln-num	amount

write a RA exp to find name of customers who live in Delhi

$$\pi_{cust\text{-}nm}\left(\sigma_{cust\_cty="Delhi"}(customer)\right)$$

o/p

$\pi_{cust\text{-}nm}$ (

cust-nm	cust-st	cust-cty
x	a	Delhi
y	b	Delhi
z	c	Delhi

)

o/p relation

cust-nm
x
y
z

Union operation — binary operator

R_1

x	y	z
a	c	e
b	d	f

$\cup$

R_2

x	y	z
g	i	k
h	j	l

x	y	z

$\cup$

u	v	w

wrong

R

$r \rightarrow$ ~~$R = R_1 \cup R_2$~~

$r = r_1 \cup r_2$

x	y	z
a	c	e
b	d	f
g	i	k
h	j	l

rows/tuples

- Union are taken between compatible relations only
- for $r \cup s$ to be valid following conditions must hold

(i) relations r & s must be of same arity i.e. they must have same no. of att's

(ii) domain's of i^{th} att of r and i^{th} att of s must be same $\forall i$

r − s
s − r

Set difference − (−) ex r − s

− used to find tuples that are in one relation but are not there in second relation

union compatibel

RA → find customers name who have an account in the bank but have not taken any loan from the bank.

$$\pi_{cust\text{-}nm}(\underline{depositor}) - \underbrace{\pi_{cust\text{-}nm}(\underline{borrower})}$$

Cartesian Product (×)

- allows us to combine inf^n from any two relation

- relations that are arguments of cartesian product operation must have distinct names.

~~if~~ ~~e~~ $r = borrower \times loan$

5 n_1 no of tuples n_2 no. of tuples 10

r — $n_1 \times n_2$ — no. of tuples

$r = r_1 \times r_2$ — $5 \times 10 = 50$ tuples

- Cartesian product takes all possible pairing of one tuple from r_1 with one tuple of r_2

borrower

custnm	ln-no
a	L-1
b	L-2
c	L-3

loan

br-nm	ln-num	Amount
x	L-1	5
x	L-2	2
y	L-3	4

$\rightarrow \sigma_{b.lno = ln.ln\text{-}no} (b \times l)$

custnm	ln-no	br-nm	ln-num	Amt	
a	L-1	x	L-1	5	
a	L-1	x	L-2	2	xx
a	L-1	y	L-3	4	xx
b	L-2	x	L-1	5	xx
b	L-2	x	L-2	2	

→ spurious tuples

b	L-2	y	L-3	4	xx
c	L-3	x	L-1	5	xx
c	L-3	x	L-2	2	xx
c	L-3	y	L-3	4	

Find names of cust's who have taken loan at ~~all the~~ mall-Rd branch.

RA query

$$\pi_{borrower.cust\_nm}\left(\sigma_{loan.br\_nm = \text{"mall Rd"}}\left(\sigma_{borrower.ln\text{-}num = loan.ln\text{-}num}(borrower \times loan)\right)\right)$$

$\sigma_{\cancel{bl}.ln} \quad \rho_{bl}(borrower \times loan)$

(bl)

$\sigma_{borrower.ln\text{-}num = loan.ln\text{-}num}$

Rename operation (ρ)

$\rho_x(E)$ → relation or RA expression

↳ newname

$\underbrace{\text{RA expression}}$ ↳ o/p relation/table

$\rho_{x(A_1, A_2, ---, A_n)}(E)$

custn , ln-no.

$\rho_b(\text{borrower})$

$\rho_{b(C,L)}(\text{borrower})$

b

$\rho_{bl}(\underbrace{\text{borrower} \times \text{loan}})$

Find the largest acc balance in the bank.

$$\underbrace{\Pi_{bal}(account) - \Pi_{account.bal}\left(\sigma_{account.bal < d.bal}\left(account \times \rho_d(account)\right)\right)}_{\text{balance which not less i.e. they are highest}}$$

Relation

account

br_nm	acc_no	bal
a	A-1	5
b	A-2	6
c	A-4	9

d

br_nm	acc_no	bal
a	A-1	5
b	A-2	6
c	A-4	9

a	A-1	5	a	A-1	5
a	A-1	5	b	A-2	6
a	A-1	5	c	A-4	9

5

5 → 9   5   5 → 9 ← Ans

Additional operations

1) Set Intersection ($\cap$)

$$\underbrace{r \cap s}_{\text{tuples which are present in both } r \text{ \& } s} = \boxed{r - (r - s)}$$

tuples which are present in both r & s

find cust's have taken ~~both~~ loan from the bank and are also having account.

depositor ∩ borrower

names of cust

Natural join — ⋈

- binary operation — it allows us to combines certain selections and a cartesian product into one operation
- natural join forms a cartesian product of its two arguments, performs selection forcing equality on those atts that appear in both relations and finally removes duplicates

r , s

$$\sigma_{=} (r \times s)$$

Find names of all cust's who have a loan at the bank & find the amount of the loan.

$$\pi_{custnm,\, amount,\, loan\text{-}no} \left(\sigma_{borrower.loan\text{-}no = loan.loan\text{-}no} (loan \times borrower) \right)$$

$$\downarrow$$

$$\pi_{cust\text{-}nm,\, amount,\, loan\text{-}no} (loan \bowtie borrower)$$

Division Operation $\div$

Suited for queries that include phrase "for all"
Ex. find all customers who have an account at all branches located in Kanpur.

$$r_1 = \pi_{brnm}(\sigma_{brcty = \text{"Kanpur"}}(branch))$$

$$r_2 = \pi_{custnm, brnm}(depositor \bowtie account)$$

We need to find customers who appear in r_2 With every branch name in r_1

$$r_2 \div r_1$$

ex say $r(R)$

A	B
a_1	b_1
a_2	b_3
a_1	b_2
a_4	b_1

$s(S)$

B
b_1
b_2

$R-S$

$r \div S =$

A
a_1

Views

- It is not desirable for all users to see the entire logical schema
- There may be certain security reasons that require that only certain part of the logical schema to be visible to the users
- Any relation that is not part of logical model but is made visible to a user as virtual relation is referred to as a view

Create view V as <query expression>

ex – branch(brnm, brcty, assets)

→ create view br as $\sigma_{brnm,\ assets}$ (branch)

→ create view allcusts as

$\pi_{brnm,\ custnm}$ (depositor ⋈ account)

∪

$\pi_{brnm,\ custnm}$ (borrower ⋈ loan)

Tuple relational calculus

- Is a non procedural query language
- Query in TRC is expressed as

$$\{t \mid P(t)\}$$

- Set of all tuples t s.t. predicate P is true for t
- Several tuple variables may appear in a formula
- Tuple variable is said to be a free variable unless it is quantified by a $\exists$ or $\forall$

ex $t \in loan \wedge \exists s \in customer(t[brnm] = s[brnm])$

↳ free variable (t)

bounded variable (s)

- A tuple relational calculus formula is built out of atoms.
- An atom has one of the following forms

(i) $s \in r$, s is a tuble variable, r is a relation

(ii) $s[x] \ominus u[y]$,

s, u – tuble variables

x – is att of s

- y is att of u
- $\ominus$ is comparision operation $<, >, \leq, \geq, =, \neq$
- x, y - att's domain is s.t. they can be compared

(iii) $s[x] \ominus c$, where c is a constant

- Formulae are build from atoms using following rules
- An atom is a formula
- If P_1 is a formula then are $\neg P_1$ and (P_1)
- If p_1 and p_2 are formula then so are

$$P_1 \vee P_2, \; P_1 \wedge P_2, \; P_1 \Rightarrow P_2$$

If $p_1(s)$ is a formula containing free tuple variable s and r is a relation then

$$\exists s \in r \, (P_1(s))$$
$$\forall s \in r \, (P_1(s))$$

are also formulae

Example: find brnm, lnnum,amount for loans of 1200000 rupees

$$\{t \mid t \in loan \land t[amount] > 1200000\}$$

Find lnnum for each loan of amount grater than 1200000 rupees

$$\{t \mid \exists s \in loan\,(t[lnnum] = s[lnnum] \land s[amount] > 1200000\}$$

Find names of all customers who have loan from mall-road branch

$$\{t \;\; \exists s \in borrower\,(t[custnm] = s[custnm] \land \exists u \in loan\,(u[lnnum] = s[lnnum] \land u[brnm] = \text{"mallRd"}))\}$$

p ➡ q, means if p is true then q must be true, used for division operation

Find all customers who have an account at all branches located in kanpur

$$\{t \mid \forall u \in branch\,(u[brcty] = \text{"kanPur"} \Rightarrow \exists s \in depositor\,(t[custnm] = s[custnm] \land \exists w \in account\,(w[accnum] = s[accnum] \land w[brnm] = u[brnm])))\}$$

Safety of an expression

$\{t \mid \neg(t \in loan)\}$ – may produce Infinite many tuples

- Such expressions are not desirable in database
- For safe expressions we define the domain of a formula dom(p)
- Dom(p) is the set of all values referenced by p i.e. values mentioned in p itself also the values that appear in tuples of relation mentioned in p.
- { t | p(t) } is safe if all values appearing in the result are from dom(p)

Domain relational Calculus

- Uses domain variables
- Domain variables takes values from domain of attributes
- A DRC expression is of the form

$$\{<\underbrace{x_1}_{\text{domain variable}}, x_2, ---, x_n> \mid \underbrace{P(x_1, x_2, ---, x_n)}_{\text{formula composed of atoms}}\}$$

Example:

Find brnm, lnnum, amount for loans over 1200000 rupees

$$\{<b, l, a> \mid <b, l, a> \in loan \wedge a > 1200000\}$$

Find names of customers who have an account in NOIDA, lives in Delhi and balance more than 500000 rupees

$$\{ cn> \mid \exists cs(<cn, \text{"DELHI"}, cs> \in customer \wedge$$
$$\exists an(<cn, an> \in depositor \wedge$$
$$\exists bn, bal(<an, bn, bal> \in account \wedge$$
$$bal > 500000 \wedge \exists as(<bn, \text{"NOIDA"}, as) \in branch))))\}$$

Find the names of all customers who have an account at all branches located in kanpur

$$\{\langle c\rangle \mid \forall\, x, y, z\ (\langle x, y, z\rangle \in branch) \wedge y = \text{"kanpur"}$$
$$\Rightarrow \exists\, a, b\ (\langle x, a, b\rangle \in account \wedge (c, a) \in depositor)\}$$

Assertions

An assertion is a predicate expressing a condition that we wish the database always to satisfy.
Domain constraints and referential integrity constraints are special forms of assertions.
There are many constraints that we cannot express using domain constraints & referential integrity constraints.

Ex. – sum of all loan amounts for each branch must be less than the sum of all account balance at the branch

Ex. every loan has at least one customer who maintains an account with a min bal of $1000.00

Creat assertion <assertion_name> check <predicate>

```
        Create assertion bal_constraint check
                (Not exists (select * from laon
Where not exists (select * from barrower, depositor, account
Where loan.loan_num = borrower.loan_num
and
borrower.cust_nm = depositor.cust_nm
and
depositor.acc_num = acc.acc_num
and
account.bal>=1000)))
```


When an assertion is created system tests it for validity. If the assertion is valid then any further modification to the database is allowed only it, it does not cause the assertion to be violated.

Triggers

- A trigger is a statement that is executed automatically by the system as a side effect of a modification to the db
- for trigger two requirements, 1. Specify the conditions under which the trigger is to be executed, 2. Specify the actions to be taken when the trigger executes.

Ex. instead of allowing –ve are balances, the bank deals with overdrafts by setting the acc bal to zero, & creating an loan in the amount of the overdraft. This loan is given a ln number indentical to the acc num of the overdrawn account.

Defin trigger overdraft on update of account t

(if new T.bal<0

then (insert into loan values

(T.br_nm,T.acc_num, -new T.bal)

insert into borrower

(select cust_nm, acc_num from depositor

Where T.acc_num = depositor.acc_num)

Update accounts

Set s.bal = 0

Where s.acc_num = T.acc_num))

New – used so that value of T.bal after apdate should be used if it is omitted the value before the update is used.

Functional dependencies

- Functional dependency is a generalization of the notion of key
- FD are constraints on the set of legal relations.

FD $\alpha \rightarrow \beta$ holds on R if, in any legal relation r(R) $\forall$ pairs of tubles t_1 & t_2 in r s.t. $t_1[\alpha] = t_2[\alpha]$ then it is also the case that $t_1[\beta] = t_2[\beta]$.

Using the above defined FD notion, we say that K is a Superkey of R if K $\rightarrow$ R. That is , K is a Superkey if, Whenever t1[K] = t2[K] then it is also the case that t1[R] = t2[R]. In other wards t1 = t2.

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4

We use assertions in SQL to enforce FD's

A $\rightarrow$ C satisfied
C $\rightarrow$ A not satisfied
AB $\rightarrow$ D satisfied

Trivial FD's
A $\rightarrow$ A
AB $\rightarrow$ A
In general FD of the form $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

FD's are used in two ways

- To specify constraints on the set of legal relations
- To test relations to see whether the latter are legal under a given set of FD's

extraneous att's on the left Side of a FD

db-college

FD $\alpha \to \beta$

L R

extraneous att in this FD

ex (Rollno, Name) $\to$ address, branch, section

Rollno $\to$ address, branch, section

suppose $\alpha \to \beta$ and $A \subseteq \alpha$ s.t. $(\alpha - A) \to \beta$

maybe more than one att.

then A is a extraneous att in FD $\alpha \to \beta$

Left irreducible FD

CK relation can have many CK's, one of them consider to be PK.

Suppose $\alpha \to \beta$ holds R and x is the Set of extraneous attributes on left side of $\alpha \to \beta$ then $(\alpha - x) \to \beta$ is called left irreducible

Prime att / Key att (

An attribute $A \in R$ is called a Prime (or Key) att if it forms Part of any of the CK's of R.

$R(\alpha, \beta, \gamma, \delta)$ R CK1 (α, β) CK2 (α, γ) $\underbrace{\alpha\, \beta\, \gamma}_{\text{Prime att's}}$

non Prime / non key att –

Att's those does not form Part of any of the CK's

Logically implied FD's or Logically inferred FD's

A FD $\alpha \rightarrow \beta \notin F$ is said to be logically implied by F iff it is satisfied by each legal relation r(R) that satisfies F

Say R(A, B, C) - be a relation

and

$$F = \{\underbrace{A \rightarrow B, B \rightarrow C}\}$$ $\rightarrow$ infer / imply / deduce $\rightarrow$ $A \rightarrow C$

closure of a FD set

suppose F is the set of FD's holding on R and F_1 is the complete set of FD's logically implied by F then closure of F (denoted by F^+) is defined as union of F & F_1

$$F^+ = F \cup F_1$$

Rules for logical inferrence of FD's

Amstrong rules (Axioms)

Rule1 – Reflexivity rule

if $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ holds

this will generate Trivial FD's

$AB \rightarrow B$, $AB \rightarrow A$, $A \rightarrow A$, $B \rightarrow B$

grandson $\alpha \rightarrow \beta$

A B C D

father son

Rule 2 – Augmentation rule. if $\alpha \rightarrow \beta$ then

$\alpha\gamma \rightarrow \beta\gamma$

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

$AD \rightarrow BD$

Rule3 – Transitivity rule

if $\alpha \rightarrow \beta$, $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$

Additional rules

Rule 4 – Union rule – if $\alpha \to \beta$ & $\alpha \to \gamma$
then $\alpha \to \beta\gamma$ $\alpha \to \gamma\beta$
(" , ")

Rule 5 – Decomposition rul if $\underbrace{\alpha \to \beta\gamma}$ then
$\alpha \to \beta$, $\alpha \to \gamma$

L cannot be decomposed

$AB \to C$
$AB \to D$
$AB \to CD$ ✓
$AB \to CD$

$AB \to C$ ✓
$AB \to D$ ✓

✓ $\boxed{AB} \to C$
$A \to C$
$B \to C$
wrong

$A \to C$
$B \to D$
$AB \to CD$ – wrong
cannot union on L side of a FD.

Rule 6 – pseudotransitivity rule

if $\alpha \to \beta$ & $\beta\delta \to \gamma$

then. $\alpha\delta \to \gamma$

ex $R(A, B, C)$, $F = \{A \to B, B \to C\}$

$F^+ = \{ A \to B, B \to C, A \to C, A \to A, B \to B, C \to C,$
$AB \to B, AB \to A, BC \to B, BC \to C, CA \to C,$
$CA \to A, ABC \to AB, ABC \to BC, ABC \to CA$ ...
$- - - - - - - \}$

Algorithm to determine closure of FD set F

(i) $F^+ = F$

(ii) Repeat

Save the value of F^+

- Use Armstrongs reflexivity rule to logically infer FD's from F^+ and those FD's to F^+
- Use Armstrong Augmentation rule to logically infer FD's from F^+ & add those to F^+
- To each FD pair of the form ~~add FD α~~ $\{\alpha \to \beta, \beta \to \gamma\}$ in F^+ add $\alpha \to \gamma$ to F^+

Until $F^+ = save\_F^+$ = no more new FD's appear

Cover of an FD set F

closure, cover

A FD set say G is said to be cover of FD set F

if $\boxed{F \subseteq G^+}$

$R(A, B, C)$

ex $F = \{A \to B,\ AB \to BC\}$

$G = \{A \to B,\ A \to C\}$ — ideally G^+

$AB \to BC$

$\alpha \to \beta$

$\alpha\gamma \to \beta\gamma$

$A \to C$ — Augment by B

$\Rightarrow \boxed{AB \to BC}$

now $A \to B$ & $A \to C$ of G are in F

hence G is cover of F

equivalent FD sets

Two set of FD's F & G are said to be equivalent if both F & G are cover of each other

i.e. $F \subseteq G^+$ and $G \subseteq F^+$

therefore $F^+ = G^+$

hence $F \equiv G$

F^+, F_c

Canonical cover (Minimal Cover) of a FD set

A FD set $\boxed{F_c}$ is said to be canonical cover (minimal cover) of a FD set F iff $F_c \equiv F$ and satisfies the following 3 conditions

(i) each FD in F_C must be in Canonical form

$AB \rightarrow C$

$A \rightarrow C$
$B \rightarrow C$ ✗

$\underbrace{AB}_{\alpha} \rightarrow CDEF$

$\underbrace{AB}_{\alpha} \rightarrow C$, $\underbrace{AB}_{\alpha} \rightarrow D$, $AB \rightarrow E$, $AB \rightarrow F$ ← Canonical form

$\alpha \rightarrow \beta$ $\quad \alpha \rightarrow \gamma$ $\quad \alpha \rightarrow$

$\left.\begin{matrix} AB \rightarrow C \\ AB \rightarrow D \end{matrix}\right\}\Big\}$ $AB \rightarrow CD$

must have only one att on its right side

(ii) All FD's in F_C must be in left irreducible form i.e. no FD in F_C should have any extraneous att.

$A \rightarrow B$ ✓

$AC \rightarrow BC$

(iii) NO FD in F_C should be extraneous.

Algo to determine Canonical cover / minimal cover of a FD set

(i) $F_c = F$

(ii) For each FD of the form $A \to B_1, B_2, \ldots, B_n$ in F_c replace by n FD's
$\{A \to B_1, A \to B_2, \ldots\ldots A \to B_n\}$

(iii) for each FD $\alpha \to \beta \in F_c$ and for each att $A \in \alpha$

ABC

if $\{\{F_c - \{\alpha \to \beta\}\} \cup \{(\alpha - A) \to \beta\}\}^+ = F_c^+$

if that happens then

$F_c = \{F_c - \{\alpha \to \beta\}\} \cup \{(\alpha - A) \to \beta\}$

(iv) for each FD $\alpha \rightarrow \beta \in \underline{\underline{F_c}}$

$$\text{if } \{ F_c - \{\alpha \rightarrow \beta\} \}^+ = F_c^+$$

$$\text{then } \underline{\underline{F_c}} = F_c - \{\alpha - \beta\} ;$$

* A set of FD may have more than one Cannonical cover.

~~$F_c\{\ \}$ $\quad f_c = \{\ \}$~~

All the cannonical covers will be equivalent to the given <u>FD set</u>.

closure of an att set α under FD set F.

$R(A, B, C, D, E, F, G, H, I)$

$FD = \{AB \rightarrow C,\ CD \rightarrow EF,\ F \rightarrow GHI\}$

$\{ABD\}^+$

$\{ABD\}^+ = ABD$

$AB \rightarrow C$

$\{ABD\}^+ \rightarrow ABCD$

$\cancel{AB \rightarrow C}$

$CD \rightarrow EF$

$\{ABD\}^+ \rightarrow ABCDEF$

$ABD \rightarrow F,\quad F \rightarrow GHI$

$ABD \rightarrow GHI$

$\boxed{\{ABD\}^+ \rightarrow \{A, B, C, D, E, F, G, H, I\}}$

$\{ABD\}^+ \rightarrow R$

$\{ABD\} \Rightarrow CK$

$AB \rightarrow ?$

$\{AB\}^+ = AB$

$\{AB\}^+ \rightarrow ABC$

$\{CD\}^+$

$\{CD\}^+ \rightarrow CD$

$CD \rightarrow EF$

$CD \rightarrow CDEF$

$\{CD\}^+ \rightarrow CDEFGHI$

Algo to determine closure of ($\underline{\alpha} \subseteq R$) under FD set F on R.

$\alpha^+ := \alpha$ by reflexivity rule

Repeat

save_$\alpha^+ := \alpha^+$

For each FD $\underline{\underline{\gamma \to \beta}}$ in F^+

~~if each~~ FD

if $\gamma \subseteq \alpha^+$ ($\therefore \alpha^+ \to \beta$) then

$\alpha^+ := \alpha^+ \cup \beta$

until (save_$\alpha^+ = \alpha^+$) no change is made in the last iteration

save_α^+ is closure of $\underline{\alpha}$.

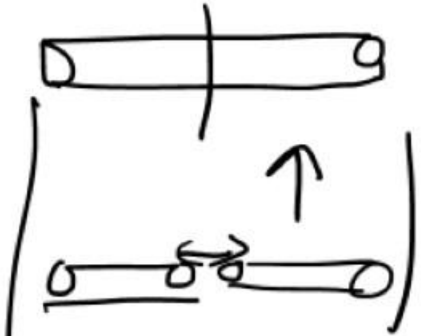
Lossless join Decomposition

ex

R

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2
a_3	b_2	c_1

$FD = \{A \to B, A \to C, \cancel{A \to BC}\}$



$R_1 \cap R_2 = A$

$R_1 \cap R_2 \to R_1$ or $R_1 \cap R_2 \to R_2$

$A \to R_1$ or $A \to R_2$

decomposition I

$A \to R_1$

$A \to A, B$

R_1 CK $= A$

R_2 CK A

R_1

A	B
a_1	b_1
a_2	b_1
a_3	b_2

$A \to B$

$\bowtie$

R_2

A	C
a_1	c_1
a_2	c_2
a_3	c_1

$A \to C$

$A \to R_2$

$A \to AC$

$R_1 \bowtie R_2$

R'

A	B	C
a_1	b_1	c_1
a_2	b_1	c_2
a_3	b_2	c_1

Lossless join decomposition

decomposition II + . R

$BC \rightarrow B$

$BC \rightarrow C$

R_1

A	B
a_1	b_1
a_2	b_1
a_3	b_2

CK = A

$A \rightarrow B$

$\bowtie$

R_2 CK = BC

Prime att

B	C
b_1	c_1
b_1	c_2
b_2	c_1

$\cancel{B \rightarrow C}$ $C \not\rightarrow B$

$R_1 \bowtie R_2 \rightarrow$

R'

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_2
a_3	b_2	c_1

Spurious tuples

Lossy join decomposition

$R_1 \cap R_2 = B \rightarrow \times CK$

$R_1 \cap R_2 \rightarrow R_1$

or $R_1 \cap R_2 \rightarrow R_2$

The decomposition $\{R_1, R_2\}$ of schema R

where $\underbrace{R_1}_{A,B} \cup \underbrace{R_2}_{A,\cancel{C}} = R$ $= \cancel{A},\cancel{B}, C$

is called Lossless if

$\underbrace{R_1 \cap R_2}$ $\rightarrow R_1$

or

$R_1 \cap R_2 \rightarrow R_2$

common att between R_1, R_2

R(A,B,C)

A B C D R

A B C   B C   D ?

R_1 R_2

$R_1 \cup R_2 \neq R$

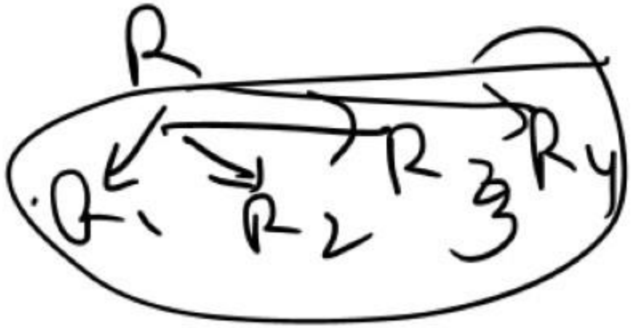
When a schema is decomposed into R_1 & R_2 for the purpose of Normalization. It must satisfy the following conditions — mandatory

(i) Preservation of att's

if decompose R into R_1, R_2
then $R_1 \cup R_2 = R$

(ii) The decomposition must be lossless

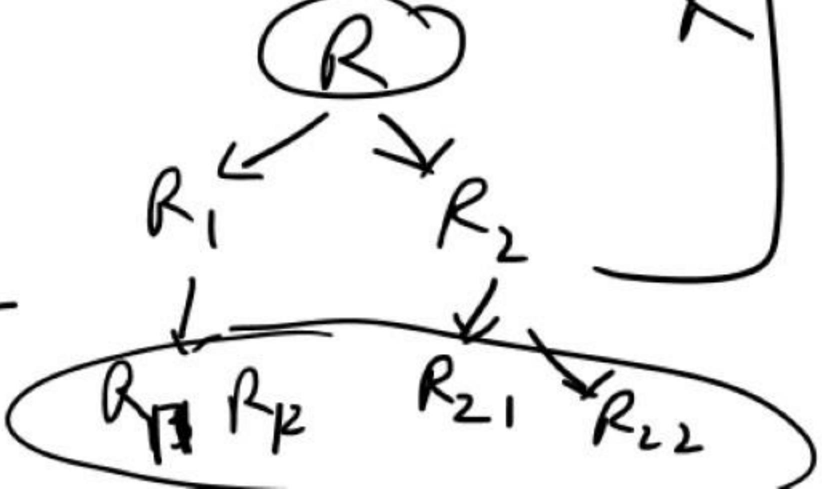
R



$R_1 \cap R_2 \rightarrow R_1$

or

$R_1 \cap R_2 \rightarrow R_2$



Desirable conditions

(i) All FD's holding on R should preferably be preserved in the decomposition

(ii) The subschemas obtained after decomposition must be in the desired Normal form.

Dependency preserving decomposition

Suppose $*(R_1, R_2, R_3, ---, R_n)$ is a decomposition of R s.t. $(R_1 \cup R_2 \cup R_3 \cup --- \cup R_n) = R$. Suppose F is the set of FD's holding on R.

Let $F' = \bigcup_{i=1}^{i=n} F_i$ if $\{F'\}^+ = F^+$ then the decomposition $*(R_1, R_2, R_3, ---, R_n)$ is called dependency preserving else it is not. We want it as desirable but not mandatory.

ex $R(A, B, C)$ lossless

$F = \{A \to B, B \to C\}$

$R_1(A, B)$ | $R_2(B, C)$

$A \to B$ | $B \to C$

So dependency is Preserved

not dependency Preserving

ex $R(A, B, C)$ $CK = A$

$F = \{A \to B, B \to C\}$ ← lost

$R_1(A, B)$ | $R_2(A, C)$

$A \to B$ | $CK = A$

$CK = A$ | $A \to C$

Lossless

$\to$ A

$A \to R_1$

$A \to B$

$B \to C$

$A \to C$

$B \to C$ Lost

$F' = \{A \to B, A \to C\}$

$[F']^+ = A \to B, A \to C$

$A \to A$

$B \to B$

$C \to C$

$[F]^+ =$

$A \to B$ $A \to A$

$B \to C$ $B \to B$

$A \to C$ $C \to C$

~~$AB \to C$~~

$$[F']^{+} \subset F^{+}$$
$$\neq$$

Normalization

1NF – first normal form.

A schema R is said to be in 1NF iff domain of each attribute of R is atomic

i.e. each att must be simple, Single valued att

Full FD

If a nonkey att $A \in R$ can only be determined by the Full CK of R (and not Part of CK) then that FD is called Full FD

$R(A, B, C, D, E)$ $CK_1 = A, B$ Key att = A, B, C

$CK_2 = C$ nonkey = D, E

FD $B \to D$ not full FD

Part of CK

D is being determined by part of CK.

$AB \to D$ ✓

$C \to D$ ✓ } – Full FD

Partial FD

if a nonkey att $A \in R$ can be determined by Part of CK of R then it is called Partial FD

2NF

- A schema R is said ~~to~~ be in 2NF if
  - it is in 1NF
  - no nonkey att of R has Partial FD on its CK's.

Heath's theorem

If schema $R(\alpha, \beta, \gamma)$ has FD $\underbrace{\alpha \to \beta}$ holding on it then it can be losslessly decomposed into $\underbrace{R_1(\alpha, \beta)}_{\alpha \to \beta}$, $\underbrace{R_2(\alpha, \gamma)}$.

CK − α

$R_1 \cap R_2 \to R_1$ ✓

or $R_1 \cap R_2 \to R_2$

comm

$R_1 \cap R_2 = \alpha$

$\alpha \to R_1$

1NF ✓ , 2NF ✗ because of Partial FD

SPO (S#, Sname, Scity, Status, P#, Pname, Price, Qty)

F = { S# → Sname, Scity, status

Scity → Status

P# → Pname, Price

(S#, P#) → Qty }

CK = (S#, P#)

S# → Sname, Scity, Status

P# → Pname, Price

(S#, P#) → Qty

(S#, P#) → Sname, Scity, Status, Qty

(S#, P#) → Sname, Scity, status, Qty, Pname, Price

(S#, P#) → S#, P#

$\alpha \to \beta$

$\alpha\gamma \to \beta\gamma$

S# → Sname, Scity, Status

(S# P#) → Sname, Sci---, — —, Qty

S# → Sname, Scity, Status - violating 2NF

Anomalies

Insertion Anomaly

Deletion Anomaly

Updation Anomaly

---

decomposition

because SPO is not in 2NF, to have schema in 2NF we will have to decompose SPO

we will use Heath's theorem.

$SPO(\underbrace{S\#}_{\alpha}, \underbrace{Sname, Scity, Status}_{\beta}, \underbrace{P\#, Pname, Price, Qty}_{\gamma})$

FD - $\underbrace{S\#}_{\alpha} \rightarrow \underbrace{Sname, Scity, Status}_{\beta} \leftarrow$

$P\# \rightarrow Pname, Price.$

$R(\alpha, \beta, \gamma)$ FD $\alpha \rightarrow \beta$

$R_1(\alpha, \beta)$ $R_2(\alpha, \gamma) \leftarrow$ lossless

$\alpha \rightarrow \beta$

$R_1(\underbrace{S\#}_{\alpha}, \underbrace{Sname, Scity, Status}_{\beta})$

CK = S#

FD (S# $\rightarrow$ Sname, Scity, Status)

is R_1 2NF $\rightarrow$ key - S#, nonkey - Sname, Scity, Status

Yes

~~True~~

$R_2(\underbrace{S\#}_{\alpha}, \underbrace{P\#, Pname, Price, Qty}_{\gamma})$

FD { $P\# \rightarrow Pname, Price$
$(S\#, P\#) \rightarrow Qty$ }

CK = ? $(S\#, P\#) \rightarrow Pname, Price, Qty$

CK $(S\#, P\#) \rightarrow R_2 \rightarrow S\#, P\#, Pname, Price, Qty$

$R_2(\underbrace{S\#}_{\gamma}, \underbrace{P\#}_{\alpha}, \underbrace{Pname, Price}_{\beta}, \underbrace{Qty}_{\gamma})$

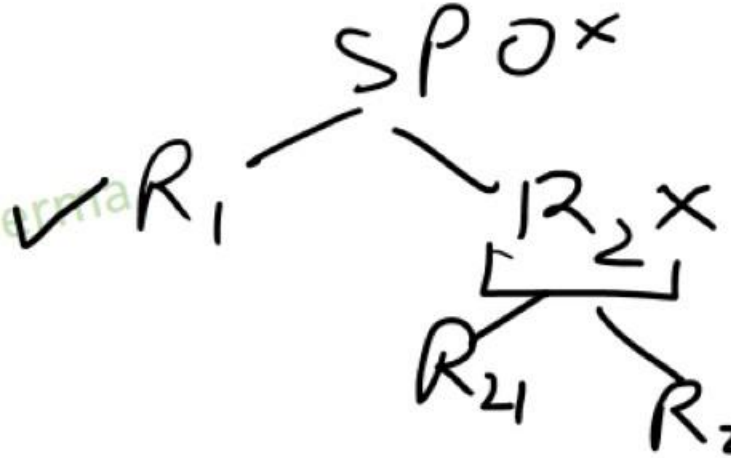
$FD = \{ P\# \rightarrow Pname, Price$ ✓ $\quad CK = (S\#, P\#)$

$\rightarrow (S\#, P\#) \rightarrow Qty \}$

is R_2 is 1NF yes

is R_2 is 2NF — NO

R_2 is NOT in 2NF



DB to be in 2NF

Why

key att — $S\#, P\#$

nonatt — Pname, Price, Qty

is Pname fully FD on CK — NO

" Price " " " " — NO

" Qty " " " " — yes

violating 2NF.

$\rightarrow \underbrace{P\#}_{\alpha} \rightarrow \underbrace{Pname, Price}_{\beta}$

R_{21} (P#, Pname, Price)

$R_{21}(\alpha, \beta)$

$R_{22}(\alpha, \gamma)$

FD = {P# → Pname, Price}

2NF — yes

CK — P#

key att — P#

nonkey — Pname, Price

R_{22} (P#, S#, Qty)

FD = {(S#, P#) → Qty}

R_{22} CK — (S#, P#) → Qty — fully FD on nonkey

key att — S#, P#

nonkey — Qty

$(S\#, P\#) \rightarrow \underbrace{S\#, P\#, Qty}_{R_{22}}$ CK

2NF yes

CK (S#, P#) → R_{22}

DB will have.

$R_1 \longrightarrow$ (S#, Sname, Scity, Status)

$R_{21} \longrightarrow$ (P#, Pname, Price)

$R_{22} \longrightarrow$ (S#, P#, Qty)

— 2NF
