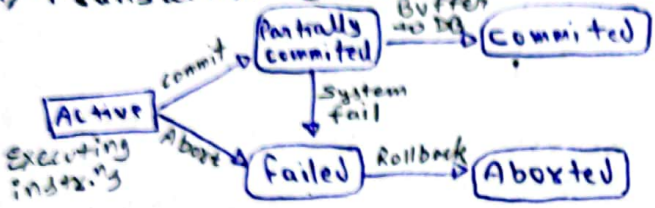
DBMS

#) Transactions & concurrency control

*) Transaction: Bundle of atomic instr^{n}s which maintains consistency in DB

*) Desirable properties of tx.^{n}:
- Atomicity → RCM, TCM
- Consistency → Programmer
- Isolation → CCM
- Durability → RCM

*) Transaction states:



After aborted or commited state, we guarentee DB to be in a consistent state.

*) Problems due to concurrent exec.^{n} of tx.^{n}s:

1) Lost update (w-w problem)

T_1	T_2
R(A)	
w(A)	
	w(A)
	com.
com.	

2) Dirty Read (R-w problem)

T_1	T_2
R(A)	
w(A)	
	R(A)
	com.
com.	

3) Unrepeatable Read

T_1	T_2
R(A)	
	R(A)
	w(A)
R(A)	

4) Phantom read

T_1	T_2
R(A)	
	Delete(A)
R(A)	

Note: If DR there ⇒ irrecoverable

Note: Sol.^{n} of concurrency problems is Schedule.

*) Schedule: Ordering of operations of transactions.

*) Types of schedules:

1) Serial Schedule: • No interleaving
- It's always consistent But less throughput

T_0	T_1
=	
=	
	=
	=

- # Serial Schedules possible $= n!$; n = no of tx.^{n}s

2) Non-Serial schedule: • Interleaving ✓

- # Non-serial Schedules $= \frac{(N_1+N_2+\cdots N_n)!}{N_1!\,N_2!\cdots N_n!}$

N_i → no. of instructions in i^{th} tx.^{n}

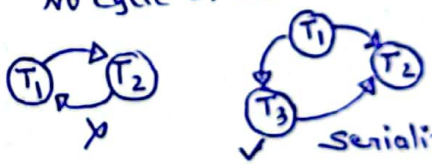
T_0	T_1
=	
	=
=	

1

Note: • conflicting instr.^{n} → 2 instr.^{n}s belonging to different tx.^{n}s, operating on same data item, & one of them is a write operation.

• conflict equivalent → when one schedule can be converted to another schedule by swapping non-conflicting instructions.

3) Conflict serializable:
- Schedule which is conflict equivalent to serial schedule (any one of the serial schedule).
- How to determine CS :-
  → Use precedence graph
  → cycle ⇒ CS ✗
  NO cycle ⇒ CS ✓



Serializ.^{n}: $T_1 \to T_3 \to T_2$

Note: • View equivalent → S & S' are view equivalent if 3 conditions hold :-

i) If T_i reads initial value of A in S, then T_i must read initial value of A in S' as well.

ii) Tx.^{n} T_x which performs final write on A in S must perform final write on A in S' as well.

iii) If T_i reads value of A in updated by T_j in S, then in S' the same should happen.

4) View serializable: • Schedule which is view equivalent to any of the serial schedule.

Ex:

S:

T_3	T_4	T_6
R(Q)		
	w(Q)	
w(Q)		
		w(Q)

i) ✓ ii) ✓ iii) ✓

S' :

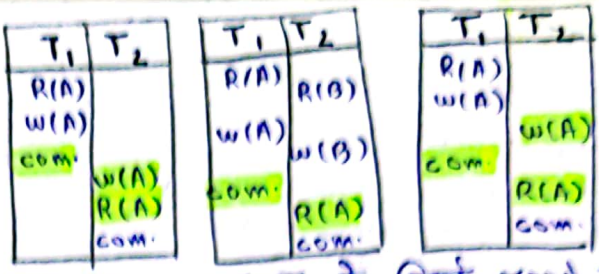
T_3	T_4	T_6
R(Q)		
w(Q)		
	w(Q)	
		w(Q)

Serial schedules possible $= n! = 3! = 6$

Let's take this: 3 + 6, 36 + ...

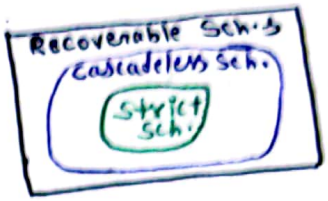
S is view eq. to S'
⇒ S is view serializable

- How to determine VS:
- If CS ✓ then VS ✓
- If CS ✗ then check Blind write
  - If BW ✗ ⇒ VS ✗
  - If BW ✓ ⇒ Tabulate all Serial Schedules & check if my Schedule is viewEq. to any one of it. (like on prev. page)



T_1 का commit T_2 के पहले read या read-write combination के पहले होगा

Note:



Note: Recoverability of Schedules:

1) Non recoverable Sch. → DR & commit before
2) Recoverable Sch. → DR but commit after

Note: Cascading concepts:

*) Cascading rollback → If uncommited reads are present, then cascading rollbacks are there.

Ex:

T_1	T_2	T_3
R(A) W(A)		
	R(A) W(A)	
		R(A)
com.		
	com.	
		com.

failure

*) Cascadeless schedule → No uncommited reads, all reads are commited reads.

T_1	T_2	T_3
R(A) W(A) com.		
	R(A) W(A) com.	
		R(A) com.

*) Shortcut

DR ✗ → • Cascadeless ✓
• Recoverable ✓

DR ✓ → • cascadeless ✗
• If order of commit is same in which DR was done, then rec. ✓
o/w Rec. ✗

5) **Strict schedule:** No uncommited read & write come together (both 2 PL)

- If T_j reads A previously written by T_i then commit/abort of T_i must appear before read & write of T_j
- Examples:-

*) Concurrency control: Design protocols to ensure properties like CS, VS, Rec etc.

*) Concurrency control methods:-

1) Time stamp ordering:
If T_i requests W(Q) or R(Q) then

i) $TS(T_i) < RTS(Q)$ ✗ Rollback T_i
ii) $TS(T_i) \geq RTS(Q)$ ✓ Allow
iii) $TS(T_i) < WTS(Q)$ ✗ Rollback T_i
iv) $TS(T_i) \geq WTS(Q)$ ✓ Allow

Note: Time Stamp
1) For txn → TS(Ti) → unique, determines serializability order
2) For data → RTS(Q) → Largest TS of txn performed R(Q)
WTS(Q) → Largest TS of Txn performed W(Q)

2) Thomas write rule: same as time stamp ordering, just in iii) instead of rollback you ignore.

3) Basic 2 PL: GP SP

4) Conservative 2PL: SP

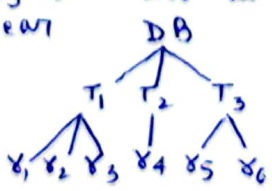
5) Rigorous 2 PL: GP

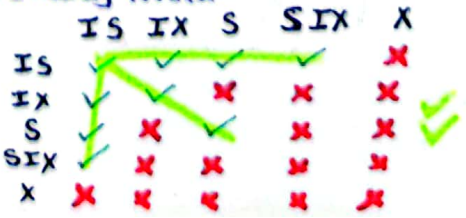
6) Strict 2 PL:

Exclusive lock hold till commit

Note: Lock
Lock-S(Q)
Lock-X(Q)

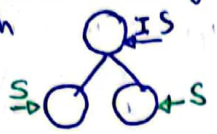
Note: Multiple granularity protocol:
- Data items of multiple sizes
- Represented in a form of tree
- when a node is locked, all the children are automatically locked.





• Intention locks:-

IS: Explicit locking at lower level of tree but only with shared lock.



IX: ..... only with exclusive or shared lock.

SIX: Subtree rooted by that node is locked explicitly in shared mode And explicit locking is done at a lower level with exclusive mode.

Note: validation based protocol:
1) Read phase (Read → Local variables)
2) Validation phase (valid.ⁿ for serializability)
3) write phase (Local var. → DB)

Note:	CS	VS	Rec.	Cascadeless	DL freedom
TS ordering	✓	✓	✗	✗	✓
Thomas WR	✗	✓	✗	✗	✓
B2PL	✓	✓	✗	✗	✗
C2PL	✓	✓	✗	✗	✓
R2PL	✓	✓	✓	✓	✗
S2PL	✓	✓	✓	✓	✗

ER model:

*) Basics:
• Arity/Degree = cols/Att.s
• Cardinality = rows/tuples

*) Attributes:
• Single valued
• multi valued — stored as independent table in RDBMS
• simple
• composite
• Stored
• Derived
• Descriptive

*) Degree in relationship:
• Unary: 1 entity set
• Binary: 2 entity set
• n-ary: n entity set

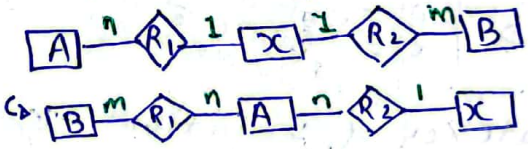
*) Cardinality ratios:
• 1 to 1: No seperate table, Pk of one side as fk on other side
• 1 to m: No seperate table, pk of 1 side as fk on m side
• m to 1: same ....
• m to n: Seperate table needed. Take pk of both relations as pk of new relation.

3

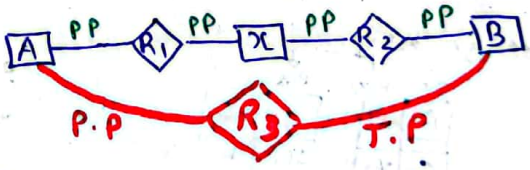
*) Participation:
• Total = min.ᵐ cardinality 1
• Partial = " " 0
• Strong entity set = Own PK
• weak entity set = Have discriminator attr. (partial key)

Partial key + PK of Strong entity set
PK of weak entity set

*) Traps in ER diagram:
1) Fan trap:



2) Chasm trap:



RDBMS:

*) Basics:
• Arity/Degree = col.s/Att.s
• Cardinality = rows/tuples
• Trivial FD = If $y \subseteq x$ then $x \to y$ always holds

*) Attributes closure:
• Ex: R(ABCDEFG)

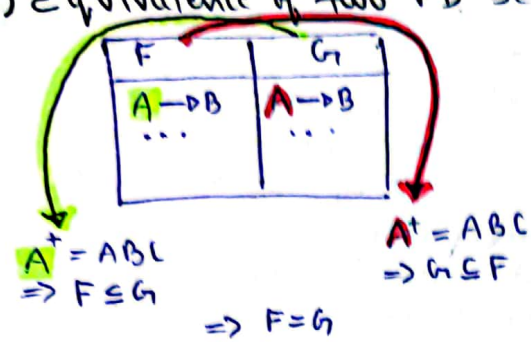
$A \to B$
$BC \to DE$
$AEG \to G$

EA: ACFG
$(AC)^+$ = ACBDE

*) Armstrong Axioms:
• Rat rules:-
Reflexivity: $y \subseteq x \Rightarrow x \to y$
Augmentation: $x \to y \Rightarrow xz \to yz$
Transitivity: $x \to y, y \to z \Rightarrow x \to z$
• Secondary RAT rules:-(UDPC)
Union: If $x \to y$ & $x \to z \Rightarrow x \to yz$
Decompo.ⁿ: If $x \to yz \Rightarrow x \to y$ & $x \to z$
Pseudo Trans: If $x \to y$ & $wy \to z \Rightarrow wx \to z$
composition: If $x \to y$ & $z \to w \Rightarrow xz \to yw$

*) Applications of att. closure:-

1) Equivalence of two FD sets:



2) Minimal cover:

R(ABCD)

A → BC

...

AC → D

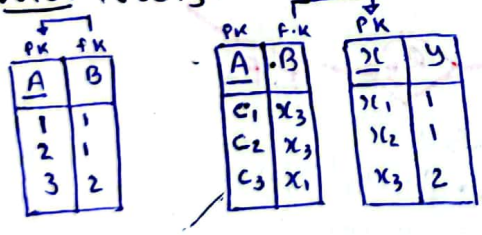
A → B	A^+ =	$\|A^+$ = पूरा attr. आए तो पार्ट नहीं पड़ता। => मिटा दो
A → C	A^+ =	$\|A^+$ =
...		
AC → D		
...	AC^+ =	subset of AC के attr. निकाल दिए तो वही आएगा।
AC → D	A^+ =	
	C^+ =	

o/w No change

3) Candidate Key of relation:

Note: FK ⊆ PK ⊆ CK ⊆ SK

Note: Foreign key:



Note:

- Entity integrity = No duplicate rows in table i.e. No key att. can be null
- Domain integrity = enforce valid entries for a column
- Referential integrity = Rows can't be deleted which are used by other records
- User defined integrity = enforce some specific business logic.

*) Normalization: Decomposition of relation.

- Partial dep.: P → N.P (subset of PK → NPA)
- Transitive dep: NP → NP

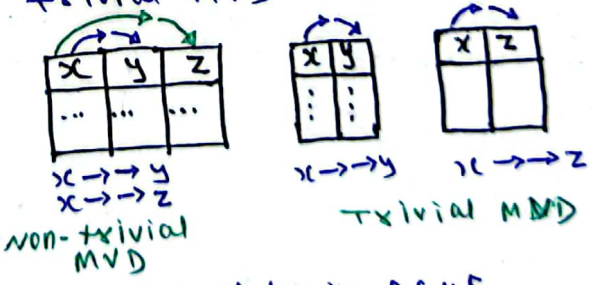
Note:) • 2 attr.s only => Always in BCNF
• Lossy join decomposition not allowed in 2NF, 3NF, BCNF

- BCNF: α (S.K) → β
- 3NF: (No transitive dep.) α (SK) → β (or PA)
- 2NF: (No partial dep.) α (P.A) ↛ β (N.P.A)
- 1NF: -> Each cell have single value
  -> Each row is unique
  -> There must be a P.K
  -> Every column have unique name
  -> Order of row & column is irrelevant.

- Note that :-
  1) In R if all are trivial dep.'s => BCNF
  2) Only 2 attr.'s => BCNF
  3) 3NF && R have only simple CK.'s => BCNF
  4) Only simple CK's [No composite CK] => 2NF
  5) All are PA's => 3NF

*) MVD: • A →→ B ; for every value of A more than one value of B may exist in DB.

- Trivial MVD $X \to\to Y$ iff $Y \subseteq X$ or $X \cup Y = R$
- Non-trivial MVD leads to redundancy
- Each MVD is decomposed into seperate table where it becomes trivial MVD



*) 4NF: • should be in BCNF
• अगर non-trivial MVD ना हो

*) 5NF: • Should be in 4NF
• can't be further non-less decomposed.

*) Lossless join decomposition :-

i) $R_1 \cup R_2 = R$

ii) $R_1 \cap R_2 \neq \phi$

iii) $R_1 \cap R_2$ = a key (any one of R_1, R_2)

- BCNF -> Lossless decomp.n always possible
  Dependency preserving may not possible.
- 3NF -> Lossless & dependency preserving decomposition always possible.

*) Dependency preserving decomposition:

R(ABC) R_1(AB) R_2(BC)

F: A → B, B → C, C → A

(F_1) $A^+ = ABC$, $B^+ = ABC$

F_1: A → B, B → A

(F_2) $B^+ = ABC$, $C^+ = ABC$

F_2: B → C, C → B

$F_1 \cup F_2$: A → B, B → A, B → C, C → B

F: A → B, B → C, C → A

$A^+ =$, $B^+ =$, $C^+ =$

$\Rightarrow F \subseteq F_1 \cup F_2$

$A^+ =$, $B^+ =$, $C^+ =$

$\Rightarrow F_1 \cup F_2 \subseteq F$

$\Rightarrow F_1 \cup F_2 = F$

$\Rightarrow$ Yes, Dep. pr.

Note: If indirect preserve हो रही FD, तब भी preserved मानते हैं।

#) File organization, Indexing, B tree, B+ tree:

*) File organization: Organization of records in a file

- Ordered → B-Search
- Unordered → L-Search

*) Indexing:

- Index file record contains only two col.s, Key (A# on which searching is done), Block ptr. (BA of block of main file which contains the record holding the key)
- Index file is always ordered
- main file maybe ordered/unordered.
- Types of indexing:-

Single level:
- Primary indexing
- clustering
- Secondary indexing

(index file for main file)

Multi level:
- Simple multilevel
- B tree
- B+ tree

(Further index the index file)

- Terminology:-

Blocking factor = No. of records per block = $\left\lfloor \frac{\text{block size}}{\text{record size}} \right\rfloor = \left\lfloor \frac{B}{R} \right\rfloor$

No. of blocks required by file = $\left\lceil \frac{\text{No. of records}}{\text{Blocking factor}} \right\rceil = \left\lceil \frac{r}{Bf} \right\rceil = n$

No. of Block access required = $\lceil \log_2(n) \rceil + 1$ ← if accessing block in main file

- Primary indexing: (PK + Ordered)
  - → Main file always sorted acc. to P.K
  - → Sparse indexing
  - → No. of entries in index file = No. of Blocks occupied by main file
  - → Example:

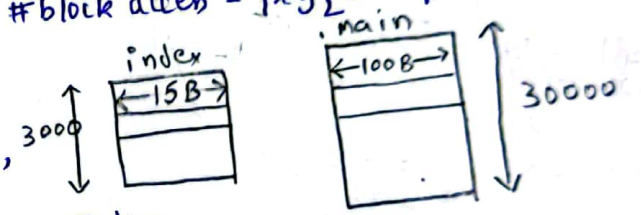
$r = 30000$, $B = 1024B$, $R = 100B$

ordering key = 9B, Block ptr = 6B] 15B

main file: $Bf = \left\lfloor \frac{1024}{100} \right\rfloor = 10$

Blocks req. by main file = $\left\lceil \frac{30000}{10} \right\rceil = 3000 = n$

#block access = $\lceil \log_2 3000 \rceil = 12$



Index file: $Bf = \left\lfloor \frac{1024}{15} \right\rfloor = 68$

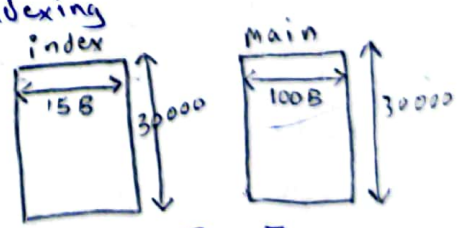
Blocks req. = $\left\lceil \frac{3000}{68} \right\rceil = 45$

#block access = $\lceil \log_2 45 \rceil + 1 = 7 + 1 = 8$

- Clustured indexing: (key att. + ordered)
  - → main file ordered on a non-key att.
  - → No. of entries in index file = No. of unique values of att. on which indexing is done (key or non-key)
  - → Sparse and dense.
- Secondary indexing: (key or non-key + ordered)
  - → #entries in index file = #entries in main file
  - → Dense indexing
  - → Example:

$r = 30000$, $B = 1024B$, $R = 100B$

ordering key = 9B, Block ptr = 6B



main file: $Bf = \left\lfloor \frac{1024}{100} \right\rfloor = 10$ Blocks req. = $\left\lceil \frac{30000}{10} \right\rceil = 3000$

#Block access = $\lceil \log_2 3000 \rceil = 12$

Index file: $Bf = \left\lfloor \frac{1024}{15} \right\rfloor = 68$

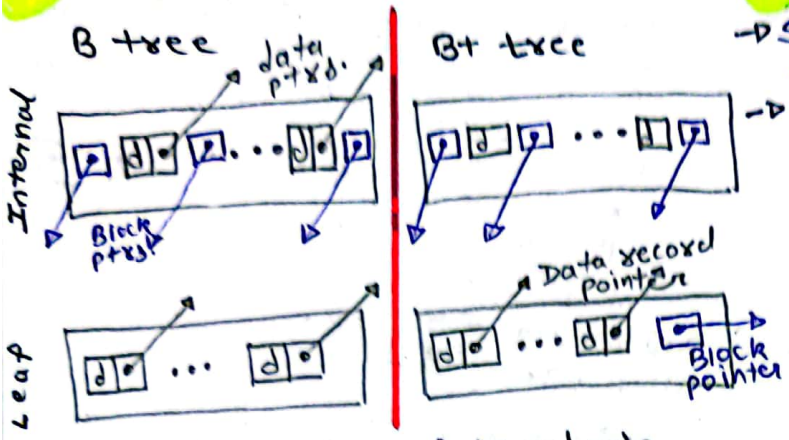
Blocks req. = $\left\lceil \frac{30000}{68} \right\rceil = 442$

Block access = $\lceil \log_2 442 \rceil + 1$

$= 9 + 1$

$= 10$

B & B+ tree :

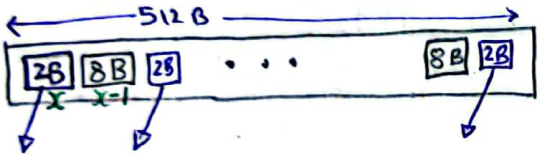


B & B+ tree contd...

- Specially designed DS for sorted stored index files in DB.
- If order = m

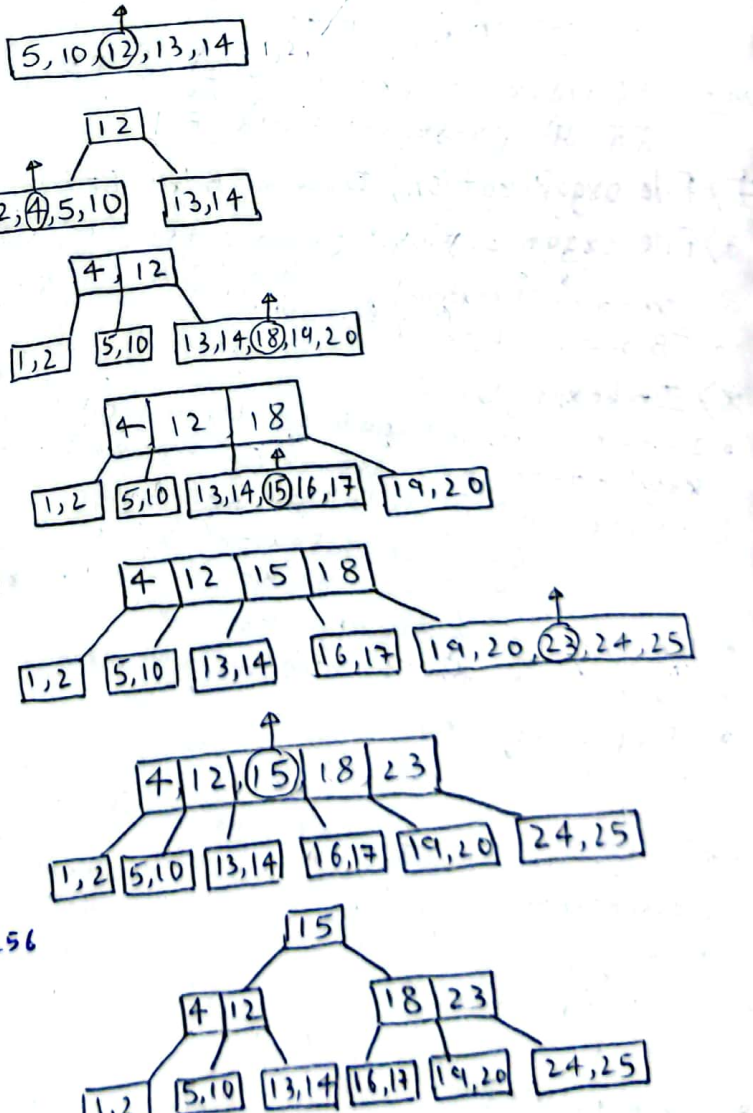
		max	min
Root :-	child	m	0
	Data	m-1	1
Internal :- (except root)	child	m	$\lceil m/2 \rceil$
	Data	m-1	$\lceil m/2 \rceil - 1$
Leaf :-	child	0	0
	Data	m-1	$\lceil m/2 \rceil - 1$

Example: B+ tree - internal node
Search key = 8 B
Block size = 512 B
Block ptr = 2 B = Tree/Disk ptr.



$2(x) + 8(x-1) \le 512$
$x \le 52$
$x = 52$

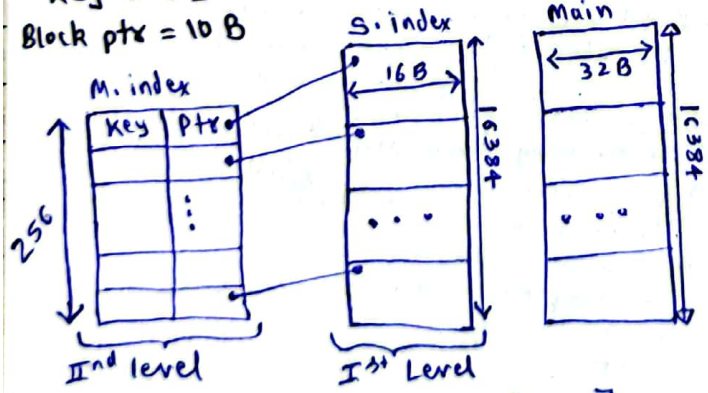
→ Insertion in B tree
5, 10, 12, 13, 14, 7, 4, 3, 20, 18, 19, 17, 16, 15, 25, 23, 24
Order = 5
child = n = 5 } max.
Data = n-1 = 4



Simple multi-level indexing :

Example:
r = 16384
B = 1024 B
R = 32 B
Key = 6 B
Block ptr = 10 B

i) Sec. indexing on key field of main file
ii) Multilevel indexing on Secondary index file.



main file: $Bf = \lfloor \frac{1024}{32} \rfloor = 32$; Blocks $= \lceil \frac{16384}{32} \rceil = 512$
Block access $= \lceil \log_2 512 \rceil = 9$

Sec. index file: $Bf = \lfloor \frac{1024}{16} \rfloor = 64$; Blocks $= \lceil \frac{16384}{64} \rceil = 256$
Block access $= \lceil \log_2 256 \rceil + 1 = 9$

multilevel index file: $Bf = 64$; Blocks $= \lceil \frac{256}{64} \rceil = 4$
Block access $= \lceil \log_2 4 \rceil + 1 + 1 = 4$

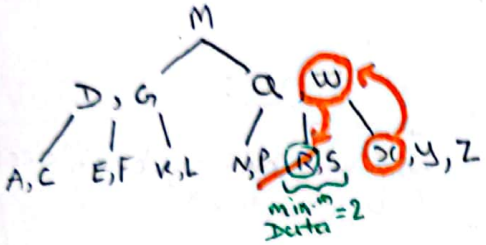
→ In B & B+ tree : $L \le R$

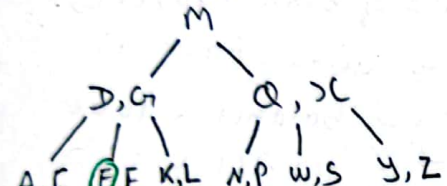
→ Deletion in B tree :

order = 6

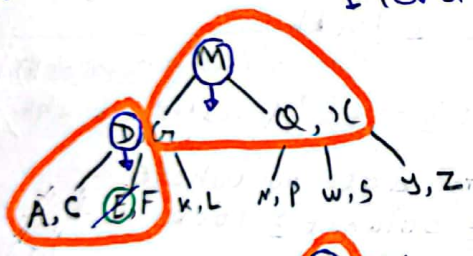
child = n = 6 } max
Data = n-1 = 5
Data = $\lceil \frac{n}{2} \rceil - 1 = 2$ } min

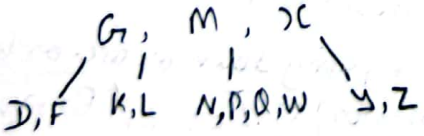
Delete : ~~H~~, ~~T~~, ~~R~~, ~~E~~, ~~A~~, ~~C~~, ~~S~~





=> Tree will collapse 1 level



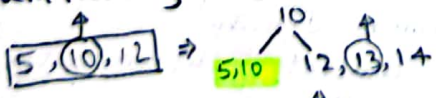


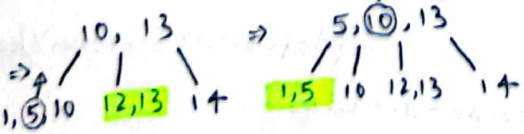
→ Insertion in B+ tree :

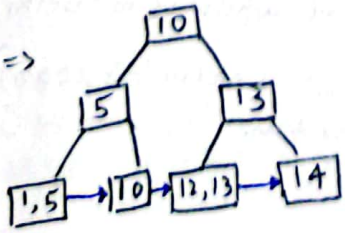
5, 10, 12, 13, 14, 1, 2, 3, 4 ; order = 3

maintaining copy in left

C = 3 } max
D = 2







Relational algebra, Relational calculus, ~~SQL~~ :-

*) Relational algebra = Procedural
Relational calculus = Non-Procedural
SQL = Both Procedural & Non-Pr.

*) Procedural = U Provide what data to be retrieved & How to retrieve that data
Non-Procedural = U just provide what data to retrieve.

*) Table(s) → RA → single table w/o name (Duplicates removed)

*) RA operators:

1) Fundamental:

Select	σ	Set diff.	$-$
Project	π	Cross Pro.	$\times$
Union	$\cup$	Rename	ρ

2) Derived:
Join $\bowtie$ → $\sigma_p(R \times S)$ Division $\div$
Intersection $\cap$ Assignment $=$

- Project π : $\pi_{c_1, c_2}(T_1)$
  min.^{m} cols to select = 1
  max.^{m} ——— = all
  Not commutative
- Select σ : $\sigma_p(T)$
  min.^{m} tuple to select = 0
  max.^{m} ——— = all
  commutative ✓
- Union $\cup$: $R \cup S$
  cond^{n}s on R & S → i) Same no. of attr.s
  ii) Same respective domains
- Set diff^n. — : $R - S$
  Same ...
- Cartesian product $\times$: $R_1 \times R_2$
  $R_1 \times R_2 \Rightarrow$ m.n tuples
- Rename ρ : $\rho_{R(P,Q,R)}$ (Project) (old) (new)
- Natural join $\bowtie$: $R_1 \bowtie R_2 \Rightarrow \sigma_\theta(R_1 \times R_2)$
  Theta join $\bowtie_\theta$: $R_1 \bowtie_\theta R_2$
  Left outer join : $R_1 ⟕ R_2$
  Right outer join : $R_1 ⟖ R_2$
  Full outer join : $R_1 ⟗ R_2$

- Division $\div$:

R

A	B
1	x
1	y
1	z
2	y
4	y
3	x
4	x
2	x
4	z

S

B
x
y

$R \div S = \frac{AB}{B} =$

A
1
2
4

- Examples:

1) Name of all customer having bank acc.
$\pi_{c\,Name}(Depositor)$

2) All details of bank branches
$\pi_{brName, city, assets}(Branch)$

3) Acc. no. where balance < 1000
$\pi_{accNo}(\sigma_{bal<1000}(Account))$

4) Branch name which is in Delhi & asset < 1 lackh
$\pi_{brName}(\sigma_{brCity='delhi' \wedge asset<1lakh}(Branch))$

5) Branch name & acc. no. where $10{,}000 \geq balance \geq 1000$
$\pi_{brName, accNo}(\sigma_{bal \geq 1000 \wedge bal \leq 10000}(Account))$

6) customers having balance > 1000
$\pi_{cName}(\sigma_{bal>1000}(Depositer))$

7) Customers having loan or acc. or both.
$\pi_{c\,Name}(Depositor) \cup \pi_{c\,Name}(Borrower)$

8) Name, accBal of customers who have account in bank.
$\pi_{cName, bal}(\sigma_{a.accNo = d.accNo}(Account \times Depositor))$

9) Name, loan amount of customers who have loan in bank.
$\pi_{cName, amount}(\sigma_{d.cName=b.cName \wedge b.loanNo=l.loanNo}(depositor \times borrower \times loan))$

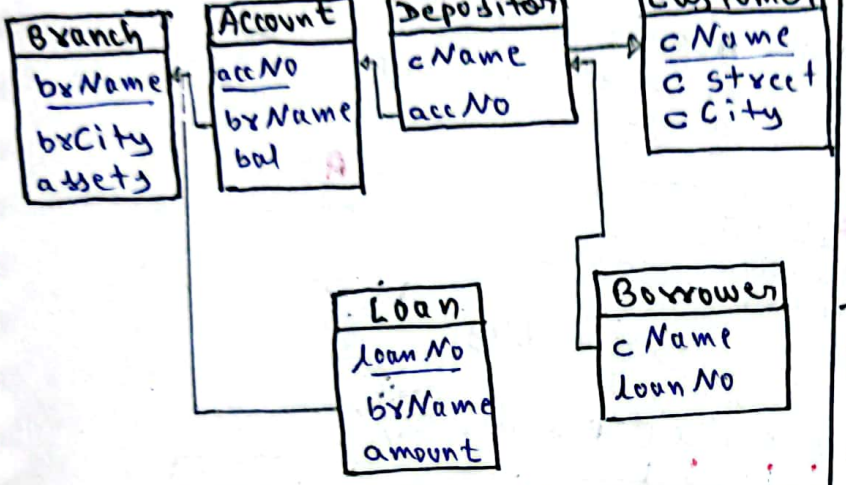
10) LoanNo, amount, brName which is in Delhi.
$\pi_{loanNo, amount, brName}(\sigma_{b.brName='Delhi' \wedge b.brName=l.br.Name}(branch \times loan))$

11) Name, acc bal of customers who've account in bank.

$\Pi_{cName, bal}(account \bowtie depositor)$

12) cName, loan amount of customers who've loan in bank.

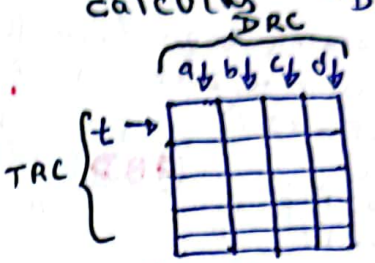
$\Pi_{cName, Amount}(loan \bowtie borrower)$



Note:
- $\Pi_{A_1}(\Pi_{A_2}(\sigma_{F_1}(\sigma_{F_2}(\gamma))))$ = $\Pi_{A_1}(\sigma_{F_1 \wedge F_2}(\gamma))$
- $\sigma(\Pi) \rightarrow \Pi(\sigma)$ ✓ Allowed
- $\Pi(\sigma) \rightarrow \sigma(\Pi)$ ✗ not allowed

*) Relational calculus < TRC, DRC

DRC



t = tuple variable to range on table

*) Free & bounded variables:-

→ Basic atomic expressions:

1) R(t) 2) t.A θ <const>

Student(t) t.marks > 50

3) $t_1.A < t_2.B$

~~Wff~~ → Basic composite expressions:

1) $R_1(t_1) \wedge R_2(t_2) \wedge t_1.A < t_2.B$

2) $F_1 \wedge F_2$, $F_1 \vee F_2$, $\neg F_2$; F_1, F_2 = atomic expr.

3) $\forall t(F)$, $\exists t(F)$

→ Types of variables:

1) Free: Variables to which quantifiers ($\exists, \forall$) are not applied.

2) Bounded: Variables to which quantifiers ($\exists, \forall$) are applied.

Note: By default every atomic expression is composite.

Note:

TRC = { _____ | _____ }

Free var.s — Bounded var.s

Reference Tables for examples:

Employee(Fn, Mn, Ln, SSn, Bdate, Addr, Sex, Super_ssn, Dno)

Department(Dname, Dno, Mgr_ssn)

Dept_Loc(Dno, DLoc)

Project(pName, pNo, pLoc, Dno)

~~Wff~~ Note: '∃' in TRC is cross product in RA

→ TRC examples:-

1) Student(Fn, Ln, Marks)

find student names whose marks > 50

$\{t.Fn, t.Ln \mid Student(t) \wedge t.marks > 50\}$

2) Find Names & addresses of all employees who work for the research dept.

Employee → Fn, Mn, Ln, Addr

Department → Dname = 'Research'

$\{t.Fn, t.Mn, t.Ln, t.addr \mid Employee(t) \wedge \exists d(Department(d) \wedge d.name = 'Research' \wedge d.Dno = t.Dno)\}$

3) Project → pLoc = 'Stafford', pNo

Department → Dno, mgr_ssn

Employee → Ln, Bdate, Addr.

$\{p.pNo, d.Dno, e.Ln, e.Bdate, e.Addr \mid Project(p) \wedge Employee(e) \wedge p.pLoc = 'Stafford' \wedge \exists d(Department(d) \wedge p.Dno = d.Dno \wedge d.mgr\_ssn = e.ssn)\}$

3) For every project located in Stafford, list out pNo, controlling Dno and department manager's (Ln, Bdate, Addr)

Note: RA, TRC restricted to safe query, DRC restricted to safe query, SQL → all these expressive power is same.

Note: By default every above method eliminates duplicates, but SQL with distinct keyword, eliminated duplicates o/w don't

*) DRC example:

List the Bdate, Addr of employee whose name is 'John B Smith'; 10

Employee → Bdate, Addr, Fn, Mn, Ln

Employee (F_n^q, M_n^r, L_n^s, ssn, B_{date}^u, $Addr^v$, sex, sal, super_ssn, Dno)

$$\{u, v \mid \exists(q)\exists(r)\exists(s)\exists(u)\exists(v)\ (Employee(q,r,s,u,v) \wedge q='John' \wedge r='B' \wedge s='Smith')\}$$

Note: Remaining topic from RDBMS: (Decomposition to normal form):-

Ex: Decompose to 2NF ③ R(A B C D E F G H I J)

R(ABCDEFGHIJ)

F: AB→C, BD→EF, AD→GH, A→I, H→J PD's

→ To make Lossless

(ABD) (AI) (ADGHIJ) (BDEF) (ABCI)

A→I; AD→GH, A→I PD, H→J; BD→EF; AB→C, A→I PD

(ADGHJ) (AI) (ABC) (AI)

AD→GH, H→J; A→I; AB→C; A→I

① CK = ABD

② AB^+ = ABCI, BD^+ = BDEF, AD^+ = ADGHIJ, A^+ = AI

⇒ 5 tables

Ex: Decompose to 3NF

R(ABCDEFGHIJ)

F: AB→C, BD→EF, AD→GH, A→I, H→J PD's

① CK = ABD

② convert to 2NF

③ convert to 3NF

AB^+ = ABCI, BD^+ = BDEF, AD^+ = ADGHIJ, A^+ = AI

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

(ABD) (AI) (ADGHIJ) (BDEF) (ABCI)

A→I; AD→GH, A→I PD, H→J; BD→EF; AB→C, A→I PD

(ADGHJ) (AI) (ABC) (AI)

AD→GH, H→J; A→I; AB→C; A→I

Convert to 3NF

(ADGH) (HJ)

AD→GH; H→J TD

⇒ 6 tables

Note: Remaining topic from RDBMS: (No. of SK's formula):

i) If we've 'n' att.s with 1 CK then #SK's = 2^{n-1}

ii) If we've 'n' att.s with 1 CK such that PA's = K then #SK's = 2^{n-K}

iii) If we've 'n' att.s with 2 CK's such that PA's are K_1 & K_2 resp.

#SK's = A ∪ B = A + B − AB = $2^{n-K_1} + 2^{n-K_2} - 2^{n-K_3}$

where, K_3 = No. of Att.s in AB combined

iv) For 3 CK we e A ∪ B ∪ C = A + B + C − AB − BC − CA + ABC

Q-1) R(a_1, a_2, a_3) CK = {a_1}

#SK's = 2^{3-1} = 4

Q-2) R($a_1, a_2, \ldots a_n$) CK = {$a_1 a_2 a_3$}

#SK's = 2^{n-3}

Q-3) R($a_1, a_2, \ldots a_n$) CK = {a_1, $a_2 a_3$}

#SK's = A ∪ B = A + B − AB = $2^{n-1} + 2^{n-2} + 2^{n-3}$; AB combined ⇒ {$a_1 a_2 a_3$}

Q-4) R($a_1, a_2, \ldots a_n$) CK = {$a_1 a_2$, $a_1 a_3$}

#SK's = A ∪ B = A + B − AB = $2^{n-2} + 2^{n-2} - 2^{n-3}$; AB combined ⇒ {$a_1 a_2 a_3$}

#) SQL (Structured Query Language)

DDL	DML	DCL	TCL
create truncate	select delete	grant	commit
alter rename	insert	revoke	rollback
drop	update		save point

Set Transaction

*) Some basic queries:

```
• create database DB1;
• create table T1 (A1 dType1(size1), A2 dType2(size2));
• alter table T1 add A3 dType3(size3);
• alter table T1 modify A1 dType4(size4);
• alter table T1 drop A3;
• alter table T1 change column A1 (old) A5 (new) dType5(size5);
• alter table T1 rename to T2;
• drop table T2;
• drop database DB1;
• truncate table T2; // removes counters, deletes data but not structure
• rename table T2 to T3;
• select * from T3;
• insert into T3 values ('Tron', 786);
• insert into T3 (name, Bike) values ('Clu', 101);
• update T3 set marks = marks+5;
• delete from T3 where roll = 104;
• delete from T3; // delete all details
• grant insert, select on T3 to Flynn;
• revoke insert, delete on T3 from Clu;
```

*) Flow during evaluation

1) from (cartesian product of tables)
2) where (select tuples acc. to cond.n)
3) group by (divide rows into groups)
4) having (Select the grp acc. to cond.n).
5) expressions (if any) (expressions in select, like count(*), (max(C1) - min(C1))
6) distinct (eliminate duplicates)
7) Set operations (union, intersect, except)
8) order by (Sorting of rows of the result)

*) Select where p (here p is:-

.. and ..	wild card :-
.. or ..	'S%'
not ..	'%S'
in ..(list)	'%S%'
not in ..	'___'
between .. and ..	'S__'
.. is null	'S__%'
.. is not null	
.. like %.__B	
<, <=, >, >=, <>, !=	

*) Order by: Sort rows

Company(name, invoice_no)

q1) All company in alphabetical order of name
Select * from company order by name asc;

q2) .. reverse alphabetical order of name AND numerical order of invoice_no
Select * from company order by name desc, invoice_no asc;

Note: Before

		marks
1	A	10
2	B	null
3	C	9

After

		marks
1	A	15
2	B	null
3	C	14

update T1 set marks = marks+5;

*) order by : Sort rows
Company (name, invoice_no)
Q1) All company in ap

*) Aggregate fx.ns :
- avg, min, max, sum, count
- except count(*), aggregate fx.ns ignore null values.
- count can be used with any kind of data
- min & max can be used with nos, strings, dates
- avg & sum can only be used with nos
- count(constant) => returns no. of tuples in that table like count(4), count('Tron')
- Aggregate fx.ns can't be used inside where clause.
- Aggr.fn(C1, C2) → error
  Aggr.fn(C1) → ✓

Q1) Select Sum(marks) as Total, avg(marks) as Average from student;

Note: Student

Rno	name	m1	m2
1	A	2	2
2	B	2	4
3	C	4	6
4	D	4	8
5	E	4	null
6	F	null	6

Q1) Select count(m1+m2) from Student; o/p: 4 (3, 6, 10, 12)

Q2) Select count(distinct m1) from student; o/p: 3

Q3) Select Sum(m1+m2) from Student; o/p: 31

Q4) Select name from Student order by max(m2);
o/p: Error (order by expects col. name)

*) Group by clause: Used to compute aggregate fx.ns of a.grp.

Student

R.no	name	Branch	Year	Gender
1	A	CS	I	M
2	B	CS	II	F
3	C	IT	I	F
4	D	IT	I	F
5	E	CS	II	M
6	F	ME	I	F

Q1) Find no. of students in each branch
Select Branch, count(*) from Student group by branch;

Q2) Find no. of students in each branch and year
Select Branch, year, count(*)
from Student group by Branch, year;

Branch	year	count(*)
CS	I	1
CS	II	2
IT	I	2
ME	I	1

Note: All column that appear in select clause must appear in the group by clause, o/w error will come.

Q3) Find no. of Female students in each branch.
Select Branch, count(*) from Student where gender='F' group by Branch;

*) Having clause :
- used to write group conditions
- Aggregate fx.ns can be used in having clause.
- Columns that appear in having clause must be an aggr. fn. (or) must be part of group by clause.
- Column appearing in having clause must be a single value per the group.

Note:
- where → used to select rows
- having → used to select the group
- In absende of group by clause, entire relation will be considered as one group.

Q) Select count(*) from Student having count(*)>2;
o/p:

count(*)
6

Q4) Find no. of female students in each branch & display the result if there are more than one student in the branch.
Select Branch, count(*)
from Student
where gender = 'F'
group by Branch
having count(*) > 1;

Branch	count(*)
IT	2

Note:

*) Set operations:
union, intersect, except
Depositor (accno, cust_name)
Borrower (loan_no, cust_name)
Q1) Find name of the customers who've an account but not loan
(Select cust_name from Depositor)
except
(Select cust_name from Borrower)

*) Join:
Employee (eno, ename, city, deptno)
Dept (dno, dname)
Q1) Find name of the employees working in research department
Select ename from Employee, Dept
where deptno = dno and dname='R';

*) Join .. on:
Q2) .. Same
Select ename from Employee join Dept
on depno=dno where dname='R';

*) Natural join:
Employee (eno, ename, city, dno)
Dept (dno, dname)
Q3) ... same
Select ename from Employee
natural join Dept where
dname='R';

*) Outer join:
Select * from T1 natural left outer join T2;
Select * from T1 right outer join T2
where A1 = B1;
Select * from T1 full outer join T2
on A1 = B1;

*) Self join: A table joined to itself.
• Alias is mandatory

Employee

eno	ename	salary	mngrEno
1	Kiran	9000	—
2	John	6000	1
3	Rajesh	6000	1
4	Mahesh	4000	2

Q1) Find ename & his manager
Select e.ename as employee, m.ename as manager
from Employee e, Employee m
where e.mngrEno = m.eno;

Employee	Manager
John	Kiran
Rajesh	Kiran
Mahesh	John

Q2) Find no of employees working under Kiran

count(*)
2

Select count(*) from Employee e, Employee m
where m.eno = e.mngrEno and m.ename='Kiran';

*) Nested Query:
Supplier (sid, sname, city, turnover)
Supply (sid, partid)
Catalog (Partid, pname, color)
Q1) Find name of supplier who is supplying partid 99.
Select sname from Supplier
where sid in (Select sid from Supply where partid = 99);
Q2) Find name of suppliers who supply blue color parts.
Select sname from Supplier
where sid in (Select sid from Supply
where partid in (Select partid
from Catalog
where color='Blue'));
Q3) Find pname supplied by supplier 'A'.
Select pname from catalog
where partid in (select partid from Supply
where sid in (Select sid from
Supplier where
sname = 'A'));
Q4) Find name of supplier who've max^m turnover
Select sname from supplier
where turnover = (Select max(turnover) from Supplier);
Q5) Find names of all suppliers who've not supplied only the 'Blue' part
Select sname from Supplier
where sid not in (select sid from supply
where partid not in (Select
partid from catalog
where color != 'Blue'));

*) Corelated Subquery: outer → inner
• Both outer & inner query are evaluated simultaneously.
• For each row of outer query, all the rows of inner query are evaluated.
• They're executed in Top-Bottom-Top manner
• when inner query uses the reference of outer query, then both the queries are said to be corelated.

• Supplier

Sno	Sname
1	A
2	B

Supply

Sno	partid
1	98
1	99
2	98

Q1) Find name of the suppliers who supply partid 99
Select S.Sname from Supplier S
where 99 in (Select Sp.partId from Supply Sp
where S.Sno = Sp.Sno);

• S1

Name	marks
A	100
B	500
C	300
D	400
E	200

S2 (Duplicate of S1)

Name	marks
A	100
B	500
C	300
D	400
E	200

Find student having 3rd highest marks

```
Select S1.name from S1
where 3 = (Select count(distinct marks)
           from S2
           where S1.marks ≤ S2.marks);
```

o/p: C

*) Exists operator:
- Testing for emptyness of a relation
- Empty → ret. false
- Non-Empty → ret. True

Q1) Name of supplier supplying partid 99

```
Select Supplier S
where exists (select * from Supply SP
where S.Sno = SP.Sno and SP.partid = 99);
```

*) Set comparison operators:
- op any (list); like x < any (5,10,15)
- op all (list); like x > all (5,10,15)

op is comparison operator

• Supplier

Sno.	Sname	City	Turnover
1	A	Hyd.	3L
2	B	Bang.	4L
3	C	Hyd.	5L
4	D	Delhi	6L

Q1) Find name of suppliers whose turnover is better than turnover of some suppliers of Hyd.

```
Select Sname from Supplier
where City != 'Hyd' and
  turnover > all (Select turnover
                  from supplier
                  where city = 'Hyd');
```

Note: Union all, Intersect all, except all } Retains all duplicates

Union, intersect, except } Removes duplicates

• T1

A1	B1
a	aa
b	bb
c	cc

T2

A2	B2	C2
a	a	a
b	a	null

Q1) Select * from T1 where A1 = all(); o/p: 3 rows

Q2) Select * from T1 where B1 in (Select A2 from T2 where C2 is null); o/p: 0 rows

Q3) Select * from T1 where A1 in (Select C2 from T2); o/p: 1 row

Q4) Select * from T1 where exists(Select count(*) from T2 where B2='x'); o/p: 3 rows

Q5) Select * from T1 where not exists(Select * from T2 where C2 ≥ 'a' and C2 != A1); o/p: 1 rows

Note: when two tables are joined (natural join) wrt primary key & foreign key, then the no. of tuples present in the resulting table is always equals to tuples in foreign key relation (min also & max also)

Note: A foreignkey cell may contain a null value.

Note: Some ways of writing DRC

① Select * from student where branch = CSE

$\{(Roll, name, branch) \mid Student(Roll, name, branch) \wedge branch = CSE\}$

② Select roll from student where branch = CSE

$\{(Roll) \mid Student(Roll, name, branch) \wedge (name, branch) \wedge branch = CSE\}$ (name, branch): Not o/p

③ Loan no. of each loan of amt over 1200

$\{\langle L \rangle \mid \exists_{b,a} \langle L, b, a \rangle \in loan \wedge a > 1200\}$ b, a: not o/p

④ All details of instructor whose salary > 80,000

$\{\langle i,n,d,s \rangle \mid \langle i,n,d,s \rangle \in instructor \wedge salary > 80000\}$

Note: Safety of expression :-

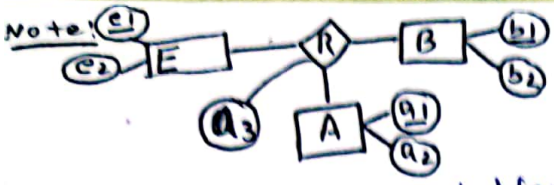
TRC: $\{t \mid \neg(t \in instructor)\}$ → infinite relation

So, use dom

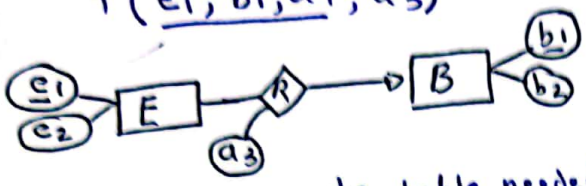
$dom(t \in instructor \wedge t.salary > 80,000)$

DRC: $\{\langle i,n,d,s \rangle \mid \neg(\langle i,n,d,s \rangle \in instructor)\}$

Note:



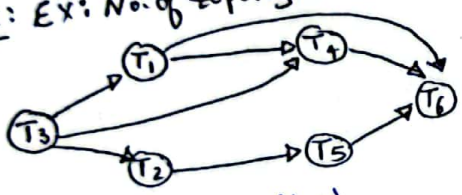
Then seperate table needed for R
T(e1, b1, a1, a3)



Then no seperate table needed for R, E(e1, e2, b1, a3)

Note: Find no. of serial schedules which are conflict equivalent to S.
1. Draw Precedence graph
2. If no cycle,
Ans = # of topological sorts

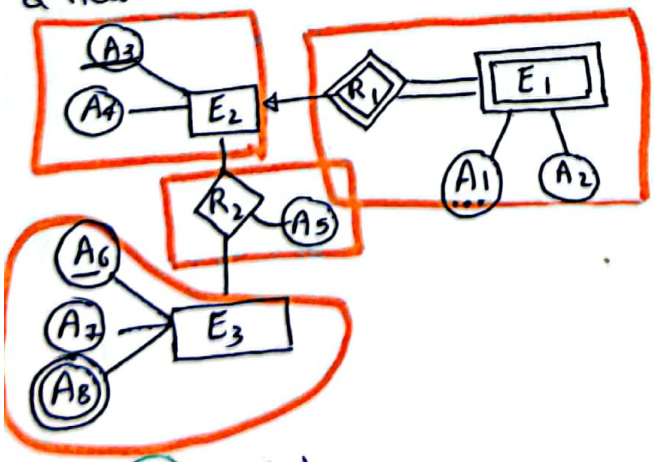
Note: Ex: No. of topological sorts



fixed fixed
T_3 - - - - T_6

4!/(2! x 2!) ; T_2 before T_5
T_1 before T_4 or Alter

Note: Bec. of many to many mapping, a new table needed for relationship R_2:-

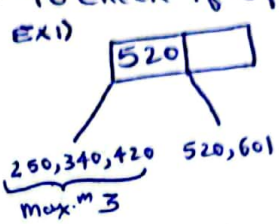


$E_1 R_1 (\underline{A_3} A_1 A_2)$
$E_2 (\underline{A_3} A_4)$
$R_2 (\underline{A_3 A_6} A_5)$
$E_{31} (\underline{A_6} A_7)$
$E_{32} (\underline{A_6} A_8)$

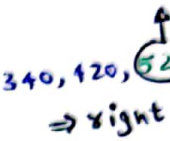
⇝ ⇒ foreign key referencing

Notes:
- Properties of B+ trees :-
  - → Height balanced (all leaves are at same level)
  - → Keys are in sorted order
  - → Leaf has ptr to next leaf
  - → non-Leaf has ptr to leaf or non-leaf but not to data record
- B+ takes more memory than BST
- we prefer B+ tree over BST bec B+ trees stores records in disk blocks
- To check if splitting done left bias or right bias:-

Ex:)

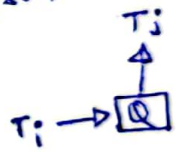


max.m 3



⇒ right biasing was used

- In B+ tree of order 'd' & leafs 'n' :-
#nodes accessed during search = $O(\log_d n)$
- B & B+ trees stores data in HDD, not in MM
- wait-die / wound-wait :-
  - → They are deadlock prevention methods used in Timestamp based protocol.
  - → wait-die : (non-premptive)
    if $TS(T_i) < TS(T_j)$
    T_i waits
    else if $TS(T_i) > TS(T_j)$
    T_i rolled back
  - → wound-wait : (premptive)
    if $TS(T_i) > TS(T_j)$
    T_i waits
    else if $TS(T_i) < TS(T_j)$
    T_j rolled back



i.e. T_i kills/wounds T_j

Alter

T_3 - - - - T_6

$T_3 <$ $T_1 <$ $T_2 <$ $T_4 - T_5 - T_6$; $T_5 - T_4 - T_6$
$T_4 - T_2 - T_5 - T_6$
$T_2 <$ $T_1 <$ $T_4 - T_5 - T_6$; $T_5 - T_4 - T_6$
$T_5 - T_1 - T_4 - T_6$

⇒ 6