16/11/14

DBMS

4th Class

Relational Algebra

Relational algebra is a procedural language which describe the step by step procedure for evaluating of a query

Relational algebra expression ~~mechan~~ represent the query evaluation

1. Selection (σ) :- Selecting "Rows"
2. Projection (Π) :- Selecting "Column"

(1) find ~~the~~ all the employees whose salary is above 5,000

Employee (Eno, Ename, Salary)

σ Employee
Salary > 5000

Employee
↖
it represent table with all rows & columns

(2) find all the employees name

Π Employee
Ename

(3) find name of the employees whose salary is above 5K

Π (σ Employee)
Ename Salary > 5000

Q1. Consider the following SQL query

Select l from R where c ;

where l and c represents the list of Columns and Condition respectively on the relation R

write the equivalent relational Algebra for this

$$\Pi_{l}(\sigma_{c} R)$$

Q Consider the following two statements

S1: $\Pi_{Ename}(\sigma_{Salary>5000}(Employee))$

S2: Select Ename
from Employee
where Salary > 5000 ;

Let the no. of tuples in the result of S_1 and S_2 are n_1 and n_2 respectively. Then which of the following relationship holds good.

Employee

Ename	salary
A	6000
A	9000

(a) $n_1 = n_2$

(b) $n_1 \leq n_2$

(c) $n_2 \subseteq n_1$

(d) $n_1 \neq n_2$

o/p

Ename
A
A

n_2

o/p

Ename
A

n_1

$n_1 \leq n_2$

Relational algebra assumes the elemination of duplicate on Implicits

Cartesian Product (X)

R

A	B
1	2
3	4

S

C	D
5	6
7	6

R×S

A	B	C	D
1	2	5	6
1	2	7	6
3	4	5	6
3	4	7	6

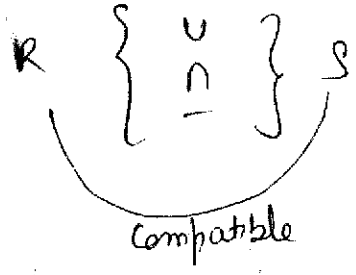
R×S returns a relation instance whose schema contains all the fields of R followed by all the fields of S and the Result of R×S contains all the possible of tuples of R

Q2. Borrower(cust_name, loan-no)
Depositor (cust_name, acc-no)
Loan (loan-no, Branch, Amount)

find name of the customer who have a loan in the branch XY2

$$\Pi_{cust\_name}\left(\sigma_{Borrower.loanno = Loan.loanno \wedge Loan.Branch = 'XY2'}(Borrower \times Loan)\right)$$

Set Manipulation operator



Relational algebra supports the use of set operation Union (∪), Intersection (∩) and difference (−) between the two compatible relations

21 find names of all customers who have an account or loan or both

(a) $\left(\Pi_{\text{cust-name}} \text{ Depositor}\right) \cup \left(\Pi_{\text{cust-name}} \text{ Borrower}\right)$

(b) ( ) ∩ ( )
← both account & loan

(c) ( ) − ( )
← account but no loan

Consider the following two statement

S1: $\left(\Pi_{\text{cust\_name}} \text{ Depositor}\right) \cup \left(\Pi_{\text{cust\_name}} \text{ Borrower}\right)$

In Relational Algebra → Eliminates the duplicates

S2: (Select cust_name from Depositor)
union
(Select cust-name from Borrower)

Eliminates the duplicates →

The no. of tuple in the result of S_1 & S_2 are n_1 & n_2 respectively Then which of the following relationship holds good.

(a) $n_1 = n_2$

(b) $n_1 \leq n_2$

(c) $n_1 \subseteq n_2$

(d) $n_1 \neq n_2$

Join

Join is defined as a Cartesian product followed by selection then projection.

Varients of Join

① Conditional Join :- In which the two relations are joined based on some conditions.

R
A B

S
C D

$$R \bowtie_{B>C} S = \sigma_{B>C}(R \times S)$$

A B C D

Find names of the customers who have a loan in branch XYZ

← projection ← selection

$$\Pi_{Cust\_name}\left[\sigma_{Loan.branch='XYZ'}\left(Borrower \bowtie_{Borrower.loan\,no = Loan.loan\,no} Loan\right)\right]$$

It is an example of Equijoin (bcz of =)

② Equi Join

Equi join in which two relation are joined only with Equality condition

R	
A	B

S	
C	D

can be → $\Pi_{A,C,D}$

$$R \underset{B=C}{\bowtie} S = \Pi_{A,B,D}\left(\sigma_{B=C}(R \times S)\right)$$

A	B	C	D
xx	10	10	xx
xx	20	20	xx

duplicate

NOTE

Every Equi join is also be a Conditional join but Every Conditional join need not be an Equi join

③ Natural join :- is an Equi join in which the equality is are specified in all field having the same name

R	
A	B

S	
B	D

$$R \bowtie S = \Pi_{A,B,D}\left(\sigma_{R.B=S.B}(R \times S)\right)$$

A	B	C	D
xx	10	10	xy
xx	20	20	xy

$$\Pi_{\text{Cust\_name}}\left[\sigma_{\text{Branch}='xy2'}(\text{Borrower} \bowtie \text{loan})\right]$$

no need to write condition

Let R and S be two relation with m and n numbers of tuples of each then find the maximum and minimum no of tuples in $R \bowtie S$

Ans	
max	mn
min	0

R	S
A B	B D

(a) m n → m×n

(b) 1, 2 3, 4 → 0

The natural join returns the result only Cartesian product when there are no ~~tuples~~ columns in common

when

Rename (ρ)

Rename operation is used to represent the relational algebr with the symbolic name

$\pi_{Cust\_name}[\sigma_{Branch='x\,y2'}(Borrower \bowtie Loan)]$

can be represent

(1) $\rho[A, (Borrower \bowtie Loan)]$

(2) $\rho[B, (\sigma_{Branch='x\,y2'} A)]$

(3) π_{Cust_name} B

Consider the following table of data

T_1

P	Q	R
11	9	6
16	b	9
26	a	7

T_2

A	B	C
11	b	7
26	c	4
11	b	6

find the number of tuples in the result of each of the following relational algebra expression.

① $T_1 \underset{P=A}{\bowtie} T_2 = 3$

② $T_1 \underset{Q=B}{\bowtie} T_2 = 2$

③ $T_1 \underset{P=A \wedge R=C}{\bowtie} T_2 = 1$

④ $T_1 \bowtie T_2 = 9$

$$\Pi_{\text{Cust\_name}}\left[\sigma_{\text{Branch}='x\times2'}\left(\text{Borrower} \overset{100}{\bowtie} \text{loan}\right)\right]$$ ← 100 testy is in 10,000 then Project

or

$$\Pi_{\text{Cust\_name}}\left[\text{Borrower} \overset{100}{\bowtie} \left(\sigma_{\text{Branch}='x\times2'} \text{loan}\right)\right]$$ ← 4 testy in 400 & then Project

→ It is optimized query of above one

Pg-61
09

(b) ✓

Q11 (a) B is FK in C is PK

$\pi_B(R_1) - \pi_C(R_2)$

tuples in B not in C
not possible

(b) ✗

(c) not always equal

(d) ✗ — the value present in FK may not present in PK (not possible)

Q1

(a) $\pi_{\text{Person\_name}}\left(\sigma_{\text{city='Miami'}}(\text{Employee})\right)$

(b) $\pi_{\text{Person\_name}}\left(\sigma_{\text{Salary} > 100{,}000}(\text{works})\right)$

(c) $\left[\pi_{\text{Person\_name}}\left(\sigma_{\text{city='Miami'}}(\text{Employee})\right)\right] \cap \left[\pi_{\text{Person\_name}}\left(\sigma_{\text{Salary}>10000}(\text{works})\right)\right]$

Q2

(a) $\pi_{\text{Person\_name}}\left[\sigma_{\text{Company\_name = First bank Corporation}}(\text{works})\right]$

(b) $\pi_{\text{person\_name, cities}}\left[\left[\sigma_{\text{Company\_name = "FBC"}}(\text{works})\right] \bowtie \text{Employee}\right]$

(c) $\pi_{\text{person\_name, cities, street}}\left[\left(\sigma_{\text{Comp\_name = "FBC"} \wedge \text{Salary} > 10000}(\text{works})\right) \bowtie \text{Employee}\right]$

10

(7)

name	Sex	Marks
A	M	600
B	M	400
C	F	900
D	F	700
E	F	500
F	F	300

Condition fails

Name	Sex	Marks
A	M	600
B	M	400
E	F	900
D	F	700
E	F	500
F	F	300

$= (C, D, E, F) - (E, F)$ — name of all females

$= (C, D)$ — name of all females whose marks are less than males marks

(a) C
(b) C, D, E
(c) C, D, E
(d) C, D

Q Consider the following table of data

A

ID	name	age
12	A	60
15	S	24
99	R	11

B

ID	Name	age
15	S	24
25	H	40
98	R	20
99	R	11

C

Id	phone
10	2200
99	2100

How many tuples does the following relational algebra expression produces, assume that the schema A ∪ B is same as

$$(A \cup B) \bowtie C$$
$$A.Id > 40 \vee C.Id < 15$$

(a) 7
(b) 4
(c) 5
(d) 9

A ∪ B

ID	name	age
12	A	60
15	S	24
99	R	11
25	H	40
98	R	20

A ∪ B

A.ID	C.Id
12	10 ✓
12	99 ✗
15	10 ✓
15	99 ✗
99	10 ✓
99	99 ✗
25	10 ✓
25	99 ✗
98	10 ✓
98	99 ✓

Division operation (÷)

Q find name of the faculty who teaches all the Course.

faculty

Fid	Fname
1	A
2	B
3	C

Teaches

Fid	Cid
1	98
1	99
2	98
3	99
1	100

Course

Cid	Cname
98	DB
99	CD
100	DS

$$\left[\Pi_{\text{Fname, Cid}}(\text{faculty} \bowtie \text{Teaches})\right] \div \left[\Pi_{\text{Cid}}\ \text{Course}\right]$$

Fname	Cid
A	98
A	99
B	98
C	99
A	100

÷

Cid
98
99
100

o/p →

Fname
A

Definition

Consider the two relation instances A & B in which A has exactly two fields X and Y and B has just one field Y with the same domain as in A with defined the division operation A ÷ B as the set of all X values such that for every Y value in B there is a tuple X, Y in A

A

X	Y
X_1	Y_1
X_2	Y_{100}
X_3	Y_1
X_1	Y_2
X_2	Y_2
X_1	Y_3

(a)

B

Y
Y_1
Y_2
Y_3

A ÷ B

X
X_1

(b)

B_2
Y
Y_1
Y_2

$A \div B_2$
X
X_1
X_2

(c)

B_3
Y
Y_1

$A \div B_3$
X
X_1
X_2
X_3

Q find the name of publisher who publishes all the Catogory of books (use division)

Book (ISBN, title, Author, Price, PID, Catogory)

Publisher (Pid, Pname, City)

$$\left[\pi_{Pname, Catogory}(Publisher \bowtie Book)\right] \div \left[\pi_{Catogory} Book\right]$$

(a) $(A,B) \div B = A$

(b) $(A,B,C) \div C = (A,B)$

(c) $(A,B,C) \div (B,C) = A$

(d) $(A,B) \div C = \phi$

(e) $(A,B) \div (B,C) = \phi$

(f) $(A,B) \div (A,B,C) = \phi$

} not possible

Q Let R_1 R_2 are two relation with n_1 & n_2 tuples respectively where $n_2 > n_1$ then find the min & max no of tuples in each of the following relational algebra expressions.

	Expression	Min	Max
(1)	$\sigma_{age>15} R_1$	0	n_1
(2)	$\Pi_{name} R_2$	1	n_2
(3)	$R_1 \cup R_2$	n_2	n_1+n_2
(4)	$R_1 \cap R_2$	0	n_1
(5)	$R_1 - R_2$	0	n_1
(6)	$R_1 \bowtie R_2$	0	$n_1 \times n_2$

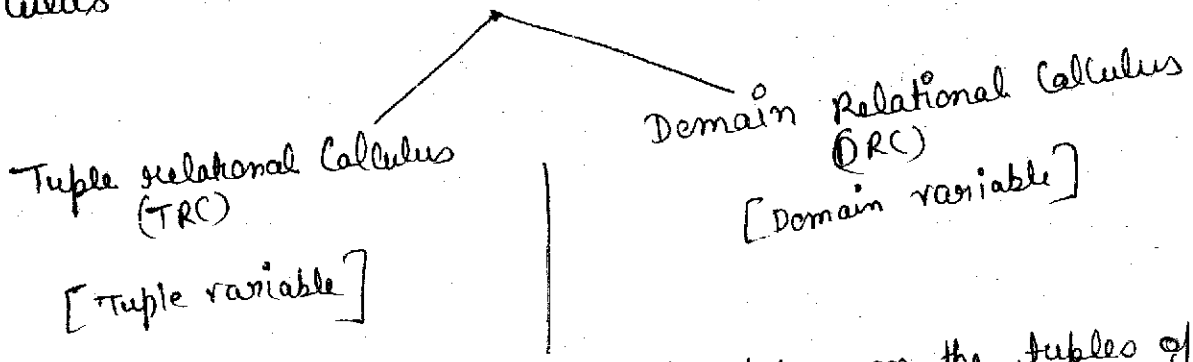
If all column contains same name then eliminate duplicate

If $n_1 C n_2$ n_2 contains all n_1

Relational Calculus

R.C. which describes the desired answer without ~~explicit~~ express it about how answer is to be computed this ~~norm~~ norm Procedural style of query is called declarative.

There are two varients of R.C and are based on predicate Calculus



(a) →

Tuple variable :- It is a variable that takes on the tuples of a relation schema as values.

Domain variable :- It is a variable that ranges over the values in the domain of some attributes

(b) → form of TRC query

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

T : Tuple variable; P(T) : formula which

form of DRC query

$\{ \langle x_1, x_2 \cdots x_n \rangle \mid P(\langle x_1, x_2 \cdots x_n \rangle) \}$

$x_1, x_2 \cdots x_n$: domain variable; $P(\langle x_1, x_2 \cdots x_n \rangle)$: formula which described

Borrower (cust_name, loan_no)

Loan (loan_no, Branch, Amount)

L B A

Q1 Find all the loans of an amount above 5000

(T represents all the columns of Loan)

TRC: $\{ T \mid T \in Loan \ (T.amount > 5000) \}$

DRC: $\{ \langle L, B, A \rangle \mid \langle L, B, A \rangle \in Loan \ (A > 5000) \}$

Q2 find the loan_no of loans of an amount above 5000

here some col required in o/p

TRC: $\{ T \mid \exists L \in Loan \ (L.amount > 5000 \wedge T.Loan\_no = L.Loan\_no) \}$

DRC: $\{ \langle L \rangle \mid \exists B, A \ (\langle L, B, A \rangle \in Loan \ (A > 5000)) \}$

(it is assignment (assignment of col loan_no to the loan_no of T))

Q Find name of the customer and his amount of loan who have a loan in Branch XY2

TRC: $\{ T \mid \exists B \in Borrower \ (T.Cust\_name = B.Cust\_name) \wedge \exists L \in Loan \ (L.Branch = 'XY2' \wedge L.Loan\_no = B.Loan\_no \wedge T.amount = L.amount) \}$

(1) Selection (2) assignment

DRC: $\{ \langle C, A \rangle \mid \exists L (\langle C, L \rangle \in Borrower) \wedge \exists B (\langle L, B, A \rangle \in Loan (B = 'XY2')) \}$

Q Interpret the following TRC Expressions

$$\{T / \exists S \in Student\ (T.Sname = S.Sname) \wedge \exists C \in Course\ (C.Sno = S.Sno \wedge C.CourseName = 'CS')\}$$

Ans finding the names of all student studying in the course CS

Pg 60

Q 3

(a) $\pi_A(R)$

$$\{T / \exists R \in R\ (T.A = R.A)\}$$

1	3
2	4

(b) $\{T / T \in R\ (T.B = 17)\}$

1	3
1	4
2	3
2	4

(c) $\{T / \exists R \in R(T.A = R.A \wedge T.B = R.B \wedge T.C = R.C) \wedge \exists S \in S\ (T.D = S.D \wedge T.E = S.E \wedge T.F = S.F)\}$

(d) $\{T / \exists R \in R\ (T.A = R.A) \wedge \exists S \in S\ (T.F = S.F \wedge R.C = S.D)\}$

Here Conversion of Algebra Exp to TRC

$\Pi_F(S)$

$= \{T / \exists T \in S\}$

Pg 60 Q4 — here Algebra Exp to DRC

(a) $\{ \langle A \rangle \mid \exists B,C (\langle A,B,C \rangle \in r_1) \}$ — here A is required

(b) $\{ \langle A,B,C \rangle \mid \langle A,B,C \rangle \in r_1 (B = 17) \}$ — here nothing required so take all with condition

(c) $\{ \langle A,B,C \rangle \mid \langle A,B,C \rangle \in r_1 \vee \langle A,B,C \rangle \in r_2 \}$ — we don't use U symbol in this so take $\vee_2$ (both)

(d) $\{ \langle A,B,C \rangle \mid \langle A,B,C \rangle \in r_1 \wedge \langle A,B,C \rangle \in r_2 \}$ — $\wedge$ $\searrow$ $\wedge$

(e) $\{ \langle A,B,C \rangle \mid \langle A,B,C \rangle \in r_1 \wedge \langle A,B,C \rangle \notin r_2 \}$

(f) $\{ \langle A,B,C \rangle / \exists P (\langle A,B,P \rangle \in r_1) \wedge \exists Z (\langle Z,B,C \rangle \in r_2) \}$

Q5

(a) $\{ \langle a \rangle \mid \exists b (\langle a,b \rangle \in r \wedge b = 7) \}$ — Table projection selection

$$\Pi_A \left(\sigma_{B=7} R \right)$$

(b) $R \bowtie S$

(c) $$\Pi_A \left[S \bowtie \left[\left(\rho_{a,b_1} R \right) \bowtie_{b_1 > b_2} \left(\rho_{c,b_2} R \right) \right] \right]$$

(b) $\left(\rho_{a,c} S \right)$

Transaction

Transaction is a collection of operation that form a single logical unit of work

eg → A —100→ B

```
Read(A)
A = A - 100
write(A)
Read(B)        ↓ fail
B = B + 100
write(B)
Commit
```

Transaction Properties

(ACID)

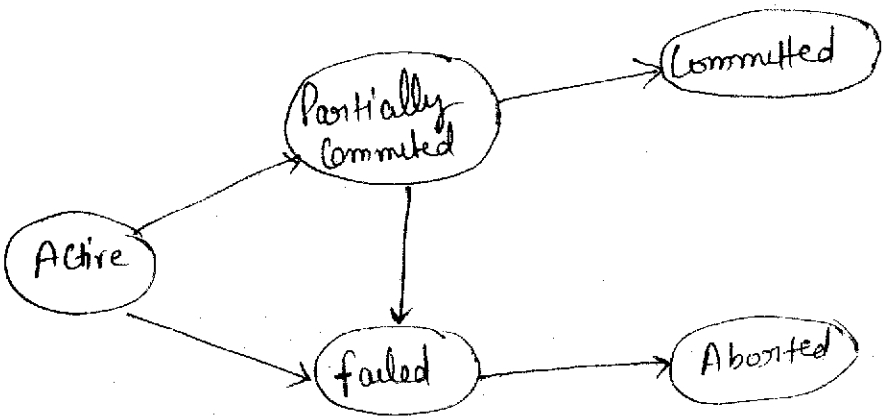
① Atomicity :- All/None
Transaction manager (always monitor each transaction)

② Consistency :- Correctness → user / Application Programmer

③ Isolation :- Each T's is unaware of other T's → Concurrency Control mgr

④ Durability :- All updations done by a T's must become permanent → Recovery mgr

F

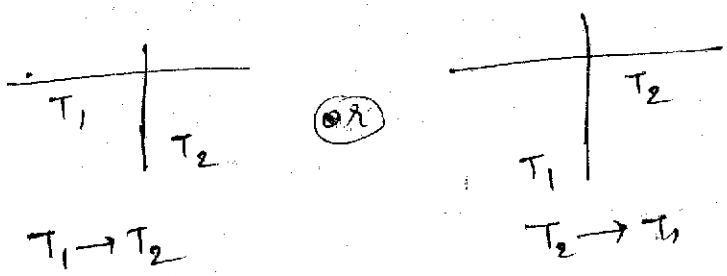


Transaction States

Schedule :- represents the order in which the operations of a transactions are executed.

① Serial Schedule :- It consists of operations belonging to various transaction where the instructions belonging to one single transaction appears together in that schedule

eg :- If there are two Transaction then two possibility of serial schedules

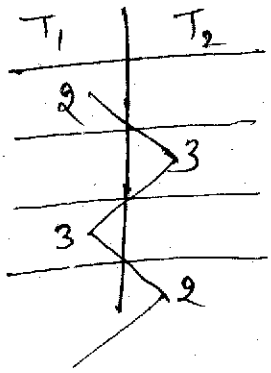


If there are n-transaction then the possible no. of serial schedules are $n!$

eg → 3 transaction / 3! Possible Serial schedules

Every serial schedule leads to consistent state in the database system

② Concurrent schedule :- If two transactions are executing Concurrently the OS may makes it one ——— one transaction little while than perform a Context switch execute the second transaction for a little while then switch back to the first transaction and so on



Why Concurrency

① To minimize the longer waiting time

T_1	T_2
1000	~~2~~

$T_1 \rightarrow T_2$
T_2 has to wait

T_1	T_2
3	
	2 — Complete
3	
⋮	

② In Computer system it is possible that when one transaction is executing IO operation then other can execute CPU operation and vice versa.

T_1	T_2
R(A)	
A=A+500	R(A)
W(A)	A=A+500
	W(A)

Effective Utilization of Resources

Q1 find the possible no. of concurrent schedules over two transactions each with two operations of ~~each~~.

T_1	T_2
$R_1(A)$	$R_2(A)$
$W_1(A)$	$W_2(A)$

$$\frac{(2+2)!}{2! * 2!} = \frac{4!}{2 \times 2} = \frac{24}{4} = 6$$

$T_1 \rightarrow T_2$:
$R_1(A)\ W_1(A)\ R_2(A)\ W_2(A)$
$R_1(A)\ R_2(A)\ W_1(A)\ W_2(A)$
$R_1(A)\ R_2(A)\ W_2(A)\ W_1(A)$

$T_2 \rightarrow T_1$:
$R_2(A)\ W_2(A)\ R_1(A)\ W_1(A)$
$R_2(A)\ R_1(A)\ W_2(A)\ W_1(A)$
$R_2(A)\ R_1(A)\ W_1(A)\ W_2(A)$

6 Concurrent schedules possible

Q2 find the no. of Concurrent schedules over three transactions with two, three, four operations respectively.

T_1	T_2	T_3
2	3	4

$$\frac{(2+3+4)!}{2! \times 3! \times 4!} = \frac{9!}{2 \times 6 \times 24} = 1260$$

generalization

$$\frac{(n_1+n_2+n_3+\cdots+n_n)!}{(n_1! \times n_2! \times n_3! \times \cdots \times n_n!)}$$

If there are n-transaction with $n_1, n_2, \ldots n_n$ operation then total no of Concurrent schedule is

$$\frac{(n_1 + n_2 + n_3 + \ldots + n_n)!}{((n_1! * n_2! * \ldots * n_n!)}$$

no of Non-Serial Schedule is

$$\frac{n_1 + n_2 + \ldots + n_n}{(n_1! * n_2! * \ldots * n_n!)} - n!$$

Serial Schedule $= n!$

Any Concurrent schedule that when executed must be equivalent to some serial schedule to ensure the Consistency of the database

Problems due to Concurrent execution of transaction

① Lost-update Problem (write write Conflict)

Suppose that the operations of T_1 & T_2 are ~~interlinked~~ Interleaved in such a way that T_2 reads the value of A before T_1 updates its value in the database.

Now when T_2 updates the value of A in the database after it if T_1 updates its value in database the value of A updated by transaction T_2 is overwritten by transaction T_1 and hence is lost this is known as lost update problem or ww Conflict.

(or)

If update of one transaction is overwritten by the update of another transaction is called lost update problem

T_1	T_2
1000 R(A)	
900 A = A − 100	
	R(A) 1000
	X = A * 0.04 40
	A = A + X 1040
	w(A) ← 1040
900 w(A) →	
R(B)	
B = B + 100	
w(B)	

A
~~1000~~
~~1040~~
900

(2) Dirty Read problem (w R conflict)

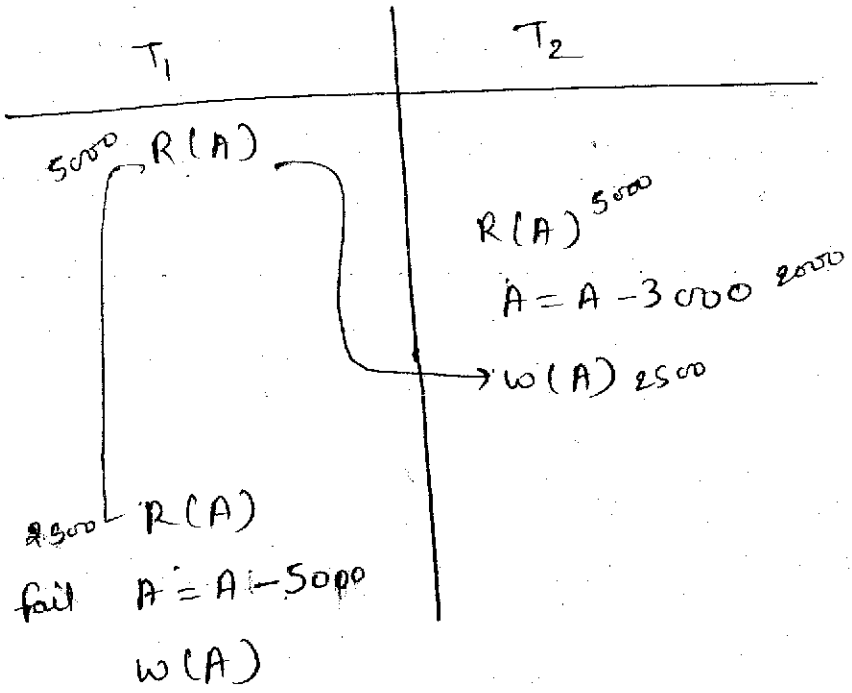
T_1	T_2
1000 R(A)	
900 A = A − 100	
900 w(A) ←	dirty read
	900 R(A)
	X = A * 0.04 36
	A = A + X 936
	w(A) 936
R(B)	
B = B + 100	
w(B)	

loss of 4 rs

Reading an uncommitted data is called dirty read

③ UnRepeatable Read Problem (R-W conflict)

when a transaction tries to read the value of a data item twice & another transaction updates the same data item in between the two read operation of the first transaction as a result the first transaction reads varied ——— values of same data item during its execution. this is known as unrepeatable read.



Consider the following schedule.

unrepeatable Read

if this write operation commit ✗ dirty Read

S1: $R_2(X)$ $R_2(Y)$ $R_1(X)$ $R_1(Y)$ $W_1(X)$ $R_2(X)$

S2: $R_2(x)$ $R_2(y)$ $W_2(X)$ $R_1(X)$ $R_2(Y)$ dirty Read

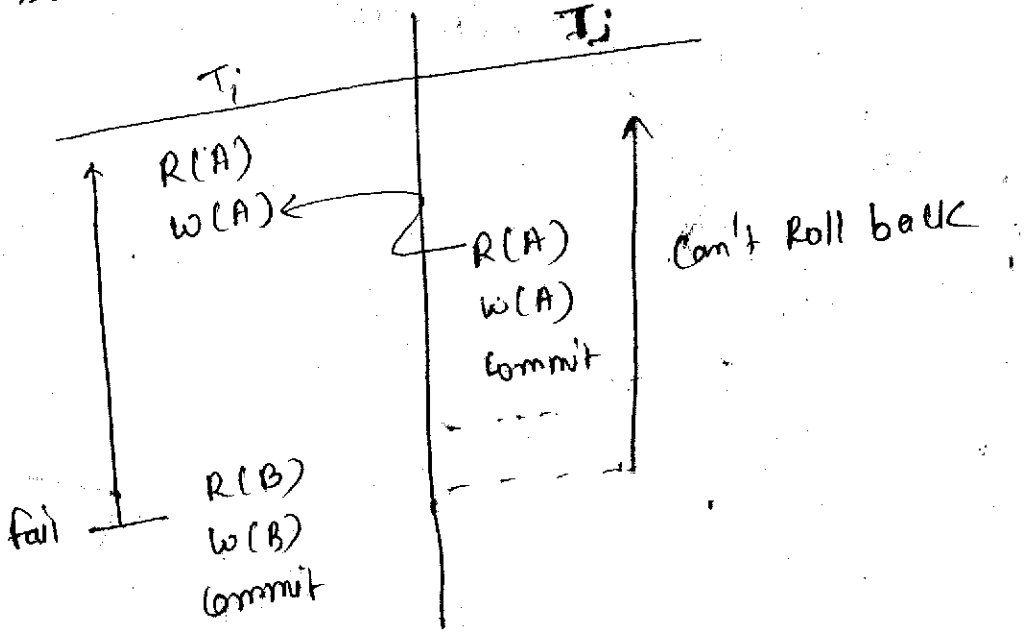
S3: $R_2(X)$ $R_2(Y)$ $R_1(X)$ $R_1(Y)$ $W_1(X)$ $W_2(X)$ lost update

which of the above schedules results in unrepeatable Read prob

Schedules based on recoverability

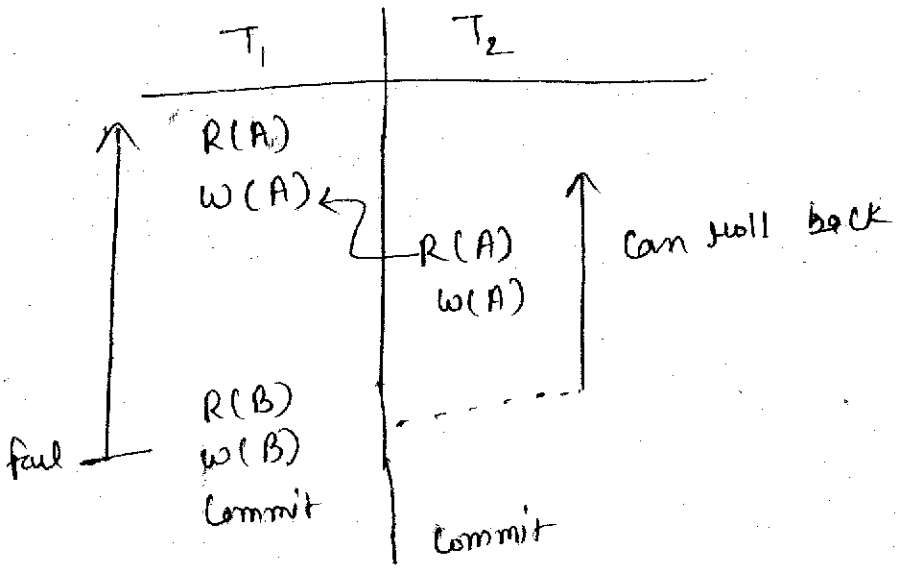
① Non-Recoverable Schedule

Let T_i and T_j be two transaction in which T_j reads the data that was return by T_i. if T_i fails after T_j commits Roll back of T_j is not possible such situation is the example of non-recoverable schedule.



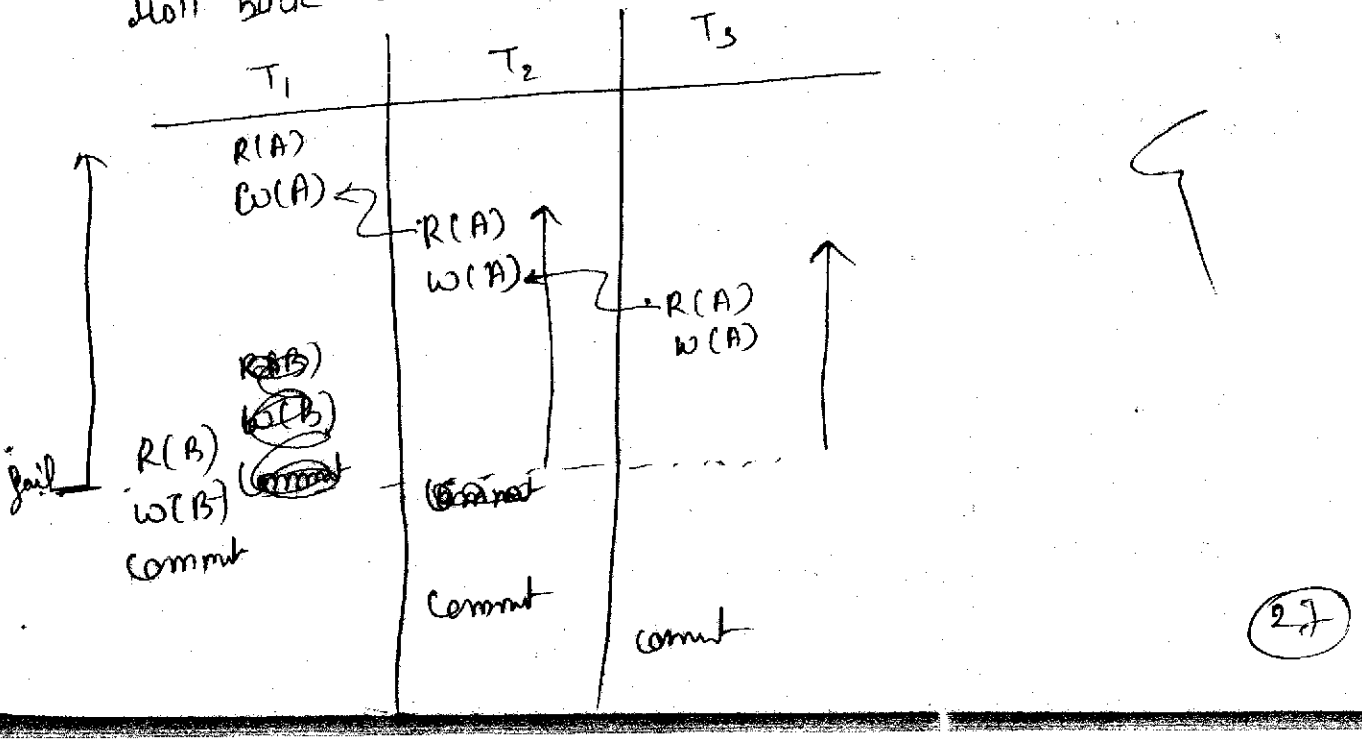
(2) Recoverable schedule

A recoverable schedule is one where for each pair of transaction T_i and T_j such that T_j reads a value that was returned by T_i. The commit operation of T_j should appear after the commit operation of T_i



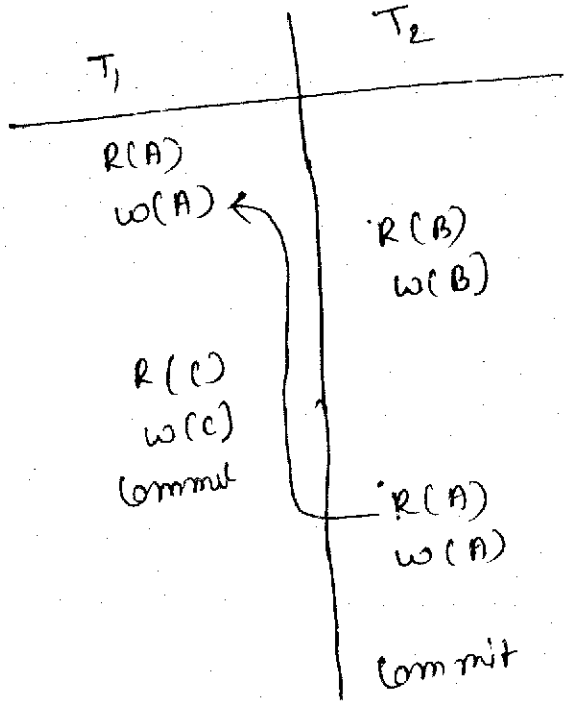
(3) Cascading Rollbacks (Cascading Aborts)

A single transaction failure leads to multiple transaction to roll back is called a schedule with cascading rollbacks or CA



(4) Cascadeless Schedule

A schedule is said to be cascadeless where for each pair of transaction T_i and T_j such that T_j reads a data item that was previously returned by T_i and the commit operation of T_i appears before the read operation of T_j



NOTE

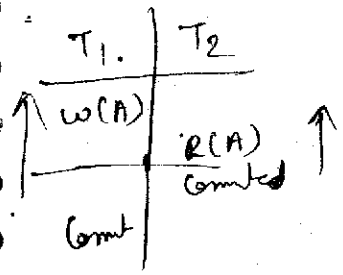
Every Cascade schedule is also be Recoverable

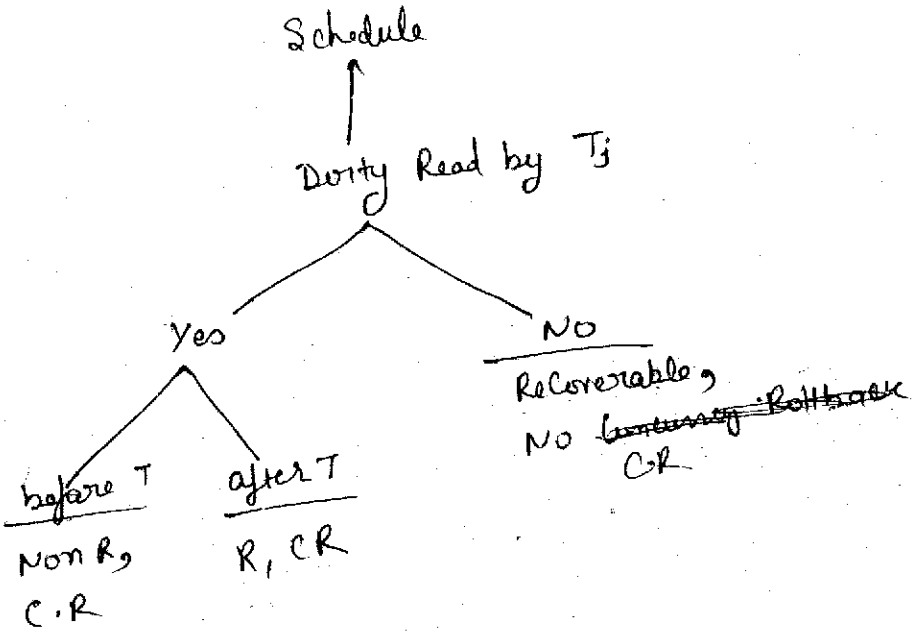
(5) Strict Schedule

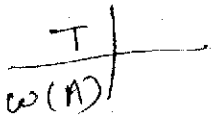
A schedule is said to be strict if a value returned by a transaction T is not read or over written by other transaction until either T aborts or commits

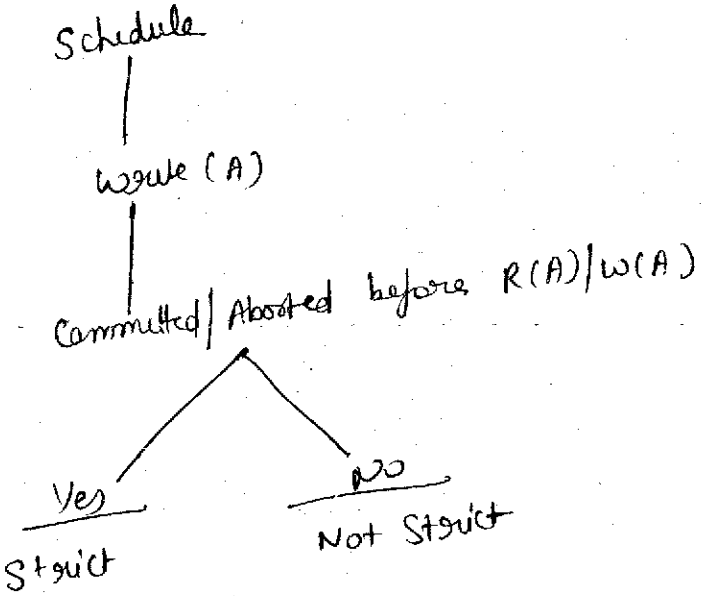
T_1	T_2
R(A)	
W(A)	
Commit / Abort	R(A) / W(A)

Every Strict schedule is both Recoverable and Cascadeless







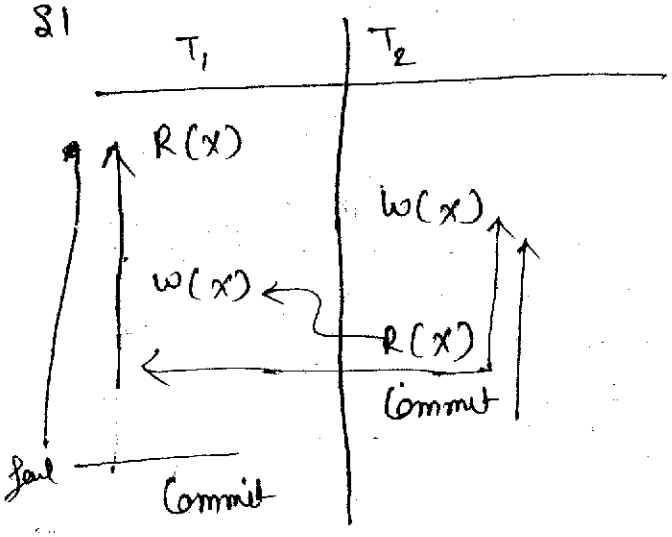


Q2

for each of the following schedule test for Recoverability.

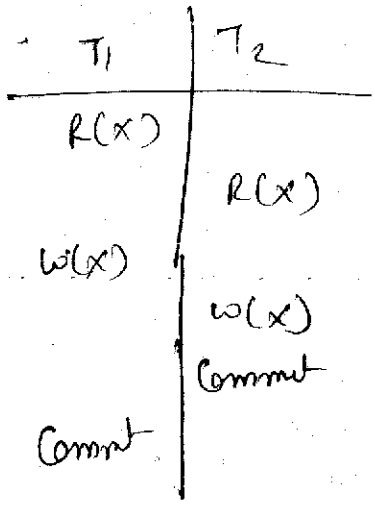
(1)

S1



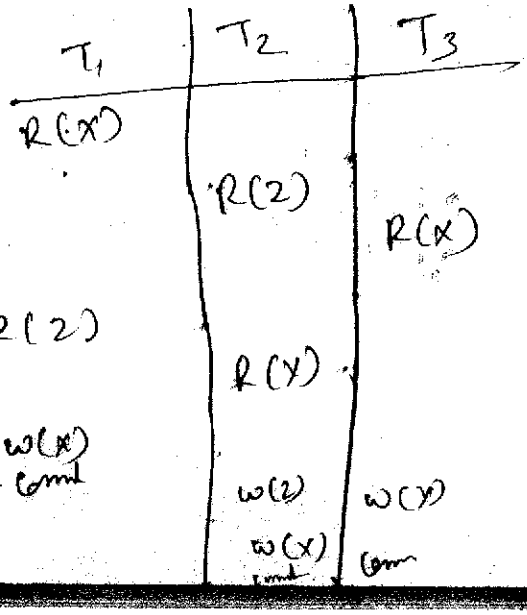
before
non R
C R
not strict

(2)



R1
no CR
not strict

(3)



Here no dirty read
so Recoverable
NO CR
Strict

(30)

Serializable Schedule

Any Concurrent schedule that when executed equals to some serial schedule is said to be serializable schedule.

Conflict Serializable schedule

Conflict operation

① $w_1(A)$ $w_2(A)$

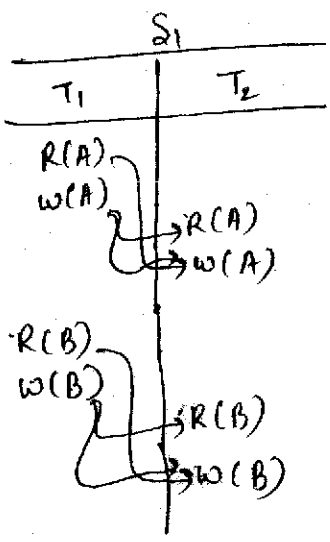
② $w_1(A)$ $R_2(A)$

③ $R_1(A)$ $w_2(A)$

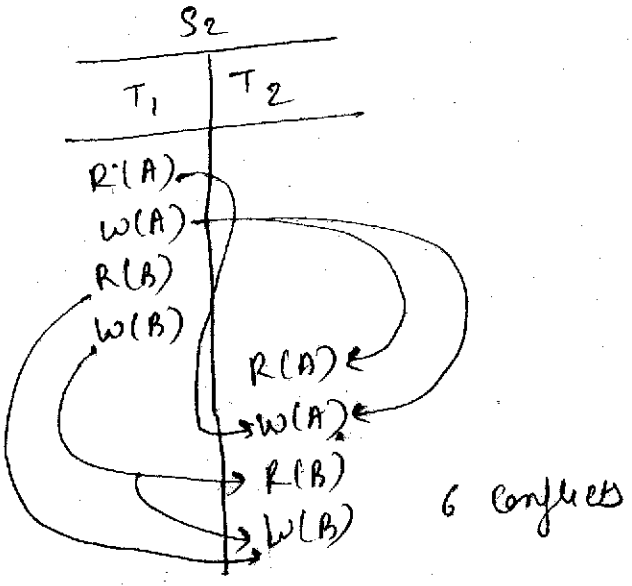
Conflict Equivalent Schedules

$S_1 \subseteq S_2$

The two schedules are said to be Conflict Equivalent iff all the Conflicting operations in both S_1 and S_2 must be executing in the same order



6 Conflicts



6 Conflicts

$S_1 \overset{c}{=} S_2$

If a Concurrent schedule Conflict equal to a serial schedule is said to be Conflict serializable schedule.

S_1 is Conflict Serializable $(T_1 \rightarrow T_2)$

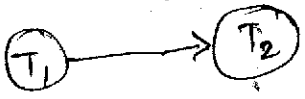
- Test for Conflict Serial schedule

① Construct a precedence graph for the schedule in which each vertex corresponds to one transaction and each edge corresponds to a operation
(let there is a conflict from T_i to T_j then draw a directed edge from T_i to T_j

② If the graph contains no cycles then schedules is said to be Conflict serializable otherwise no Conflict serializable

③ The serializebelity order is determined based on the directed edges

eg → precedence graph for S1 (Prev Page)

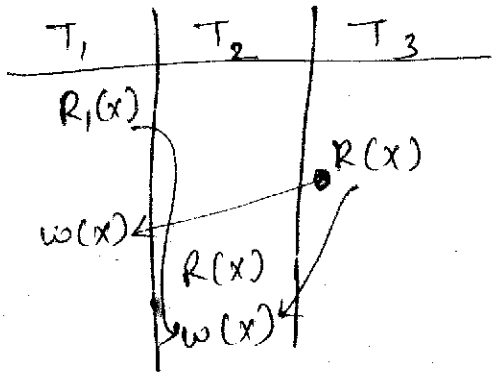


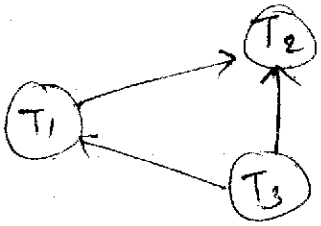
S1 is C.S $(T_1 \rightarrow T_2)$

Pg-74

Q1

(1)





remove whose outdegree is 0

T2

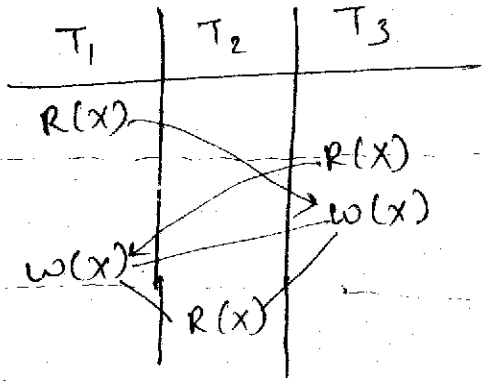
then T3

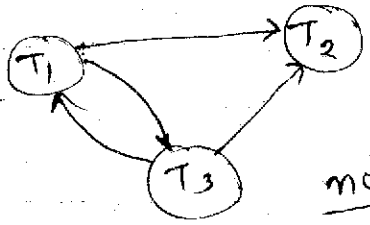
then T1

so $T_2 \rightarrow T_3 \rightarrow T_1$

no cycle so C.S

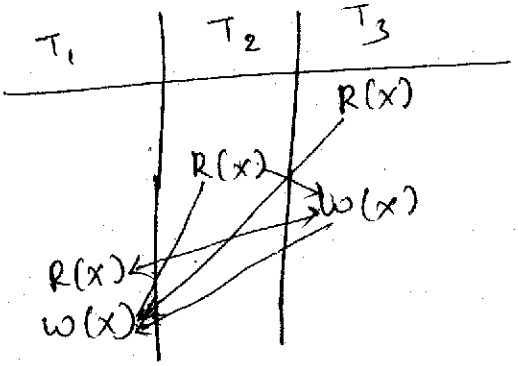
(2)

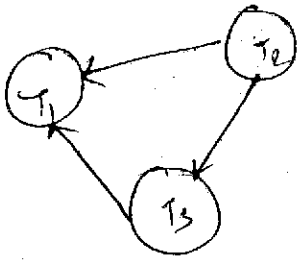




not C.S

(3)

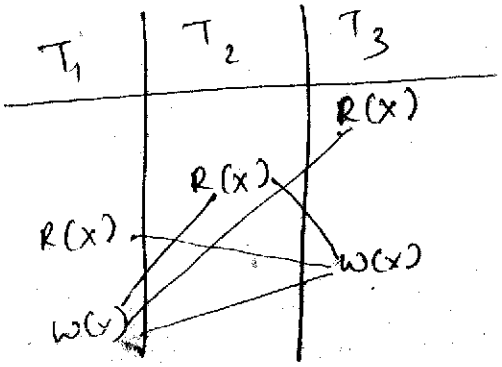




C.S

$T_2 - T_3 - T_1$

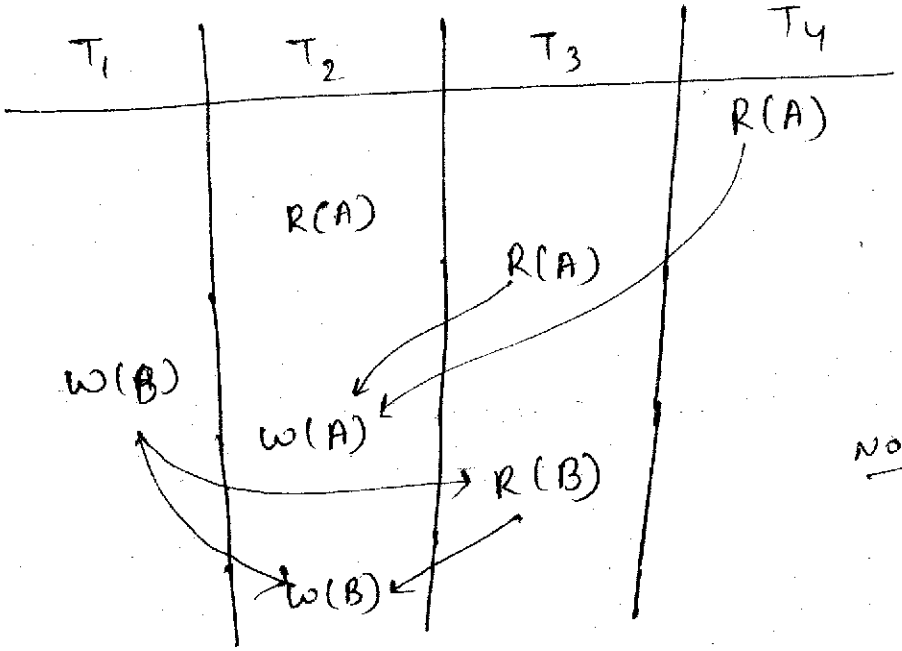
(4)



not C.S

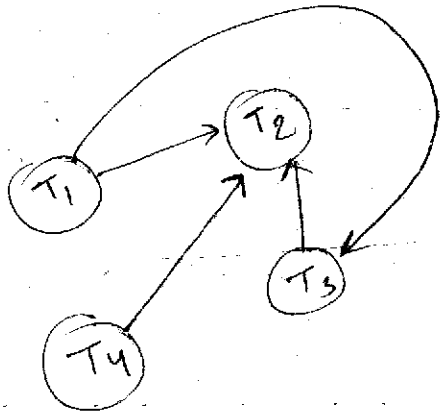
one Concurrent Schedule may be Conflict Equent to more than one schedule

Consider the following schedule



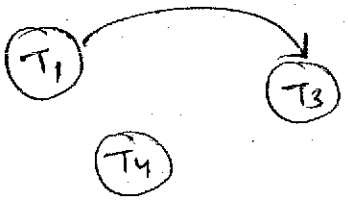
No cycle so C.S

(a) Not C.S
(b) $T_3 - T_4 - T_1 - T_2$
(c) $T_1 - T_4 - T_3 - T_2$
(d) $T_2 - T_3 - T_1 - T_4$



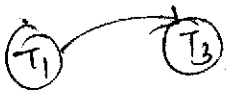
Remove T_2

$x - x - x - T_2$



Two possibility

(a) remove T_4 or T_3



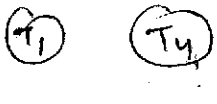
$x - x - T_4 - T_2$

remove T_3

$x - T_3 - T_4 - T_2$

remove T_2

$T_4 - T_3 - T_4 - T_2$

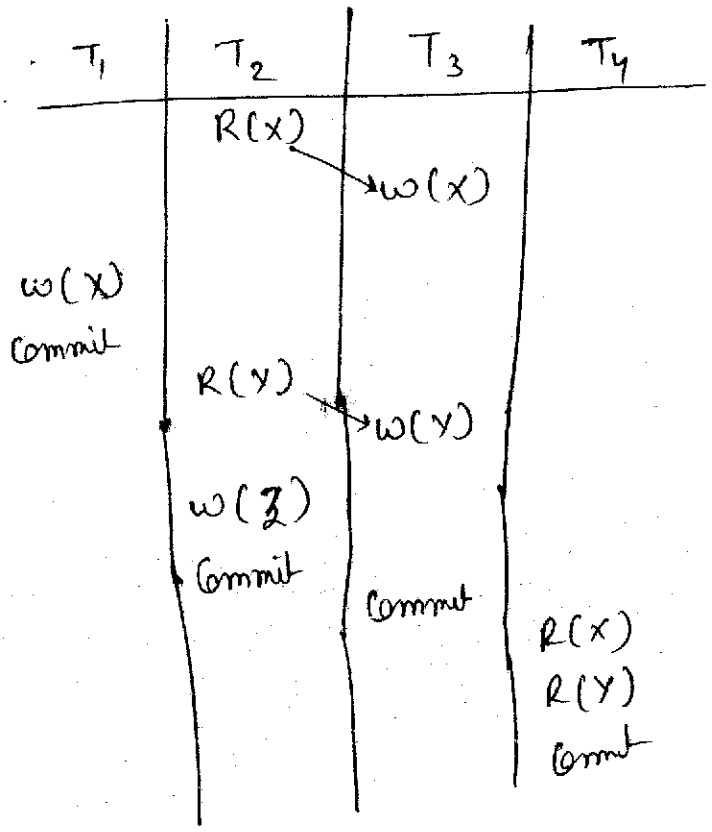


$x - x - T_3 - T_2$

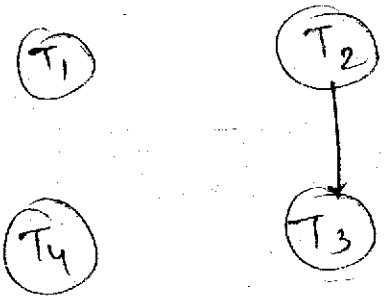
Possible

(1) $T_1 - T_4 - T_3 - T_2$
(2) $T_4 - T_1 - T_3 - T_2$
(3) $T_1 - T_3 - T_4 - T_2$

Q 2



no cycle
C.S



Case ① remove T3

T1 T2

T4

<3 = 6

$T_1 - T_2 - T_4 - T_3$
$T_1 - T_4 - T_2 - T_3$
$T_2 - T_1 - T_4 - T_3$
$T_2 - T_4 - T_1 - T_3$
$T_4 - T_1 - T_2 - T_3$
$T_4 - T_2 - T_1 - T_3$

$T_4 - T_2 - T_1 - T_3$
$T_1 - T_2 - T_4 - T_3$
$T_1 - T_4 - T_2 - T_3$
$T_2 - T_1 - T_4 - T_3$
$T_2 - T_4 - T_1 - T_3$
$T_4 - T_1 - T_2 - T_3$

Case (2) ~~Remove~~

$T_2 - T_3 - T_4 - T_1$

$T_2 - T_4 - T_3 - T_1$

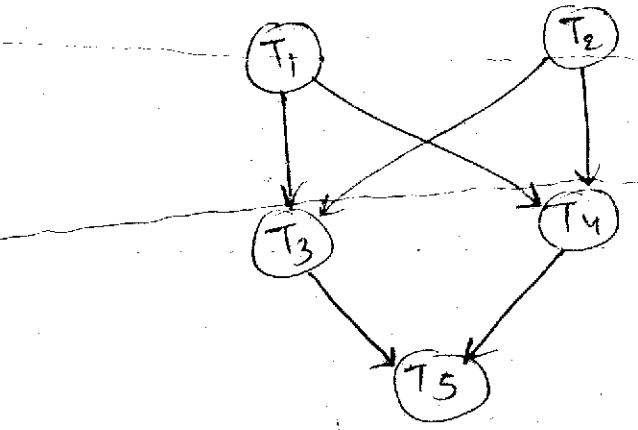
$T_4 - T_2 - T_3 - T_1$

$T_1 - T_2 - T_3 - T_4$

$T_2 - T_1 - T_3 - T_4$

$T_2 - T_3 - T_1 - T_4$

Q Consider the following graph find the no. of ~~Conflict~~ Serial schedules for the precedence graph



$T_2 - T_1 - T_4 - T_3 - T_5$

$T_1 - T_2 - T_4 - T_3 - T_5$

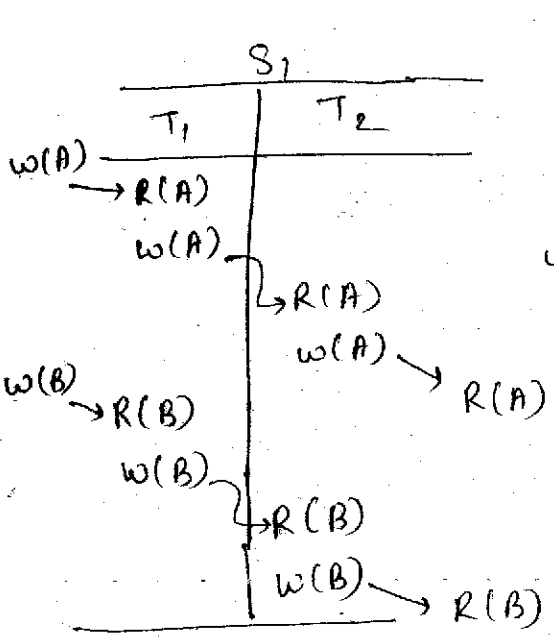
$T_1 - T_2 - T_3 - T_4 - T_5$

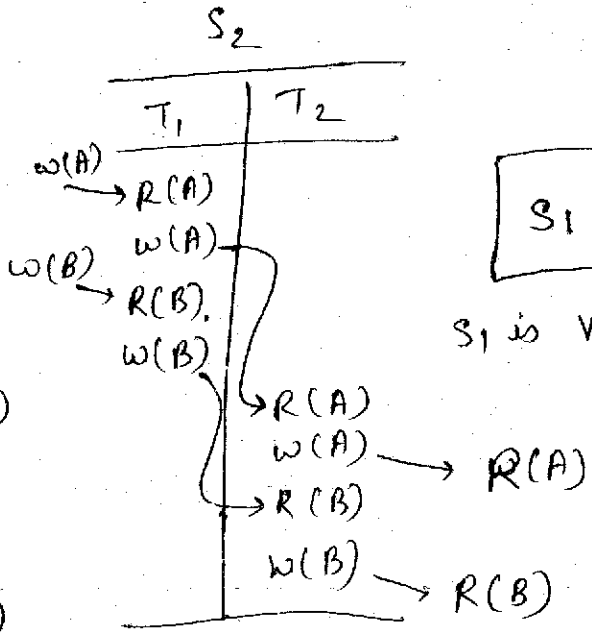
$T_2 - T_1 - T_3 - T_4 - T_5$

(2) View Serializability

View Equivalent schedules

The Two schedule S_1 and S_2 are said to be view Equivalent iff all the write-Read sequences must be executed in the same order for each data item in both the schedules





$S_1 \equiv_V S_2$

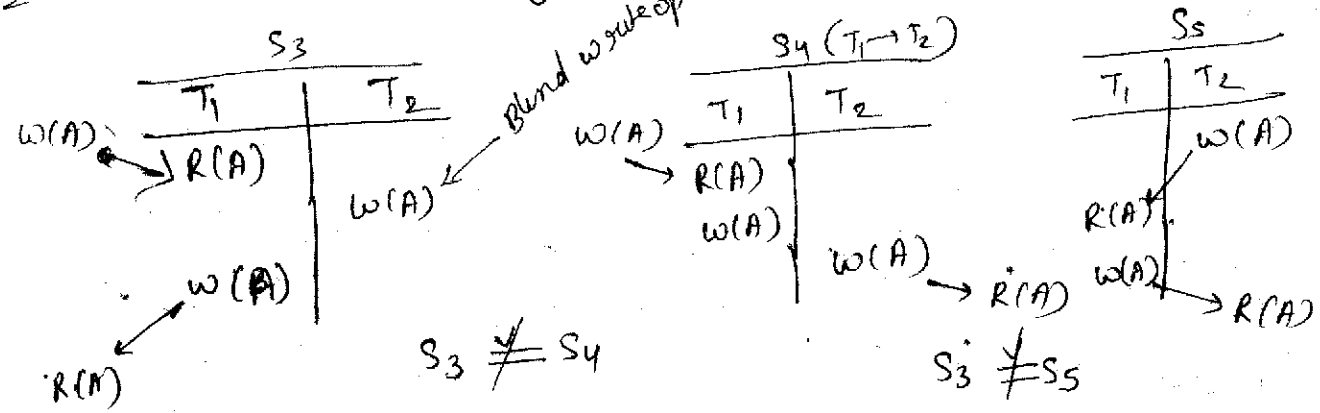
S_1 is V.S $(T_1 \rightarrow T_2)$

$T_1 \rightarrow T_2$
S_1 is CS $(T_1 - T_2)$
S_1 is V.S $(S_1 \rightarrow S_2)$

A Concurrent schedule if view equivalent to some serial schedule is said to be view serializable schedule

eg → S_1 is view Serializable $(T_1 \rightarrow T_2)$

Q Test is the following schedule is view serializable



$S_3 \neq S_4$

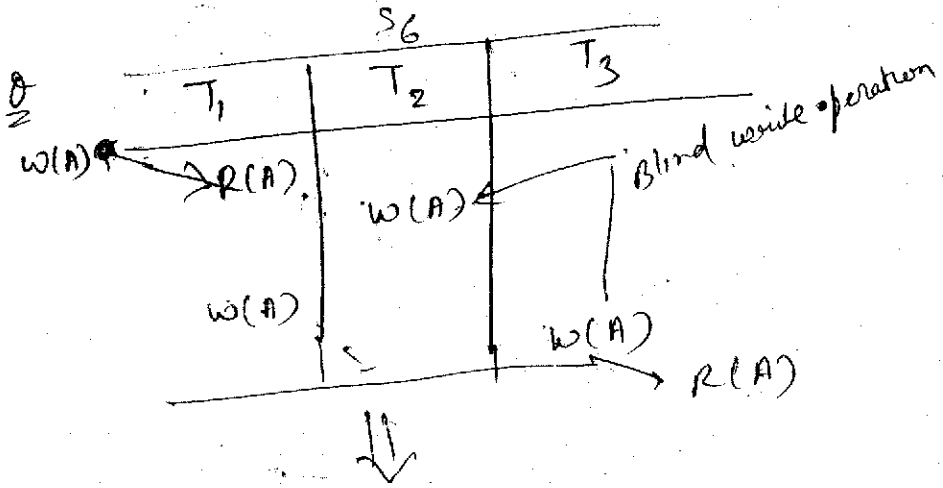
$S_3 \neq S_5$

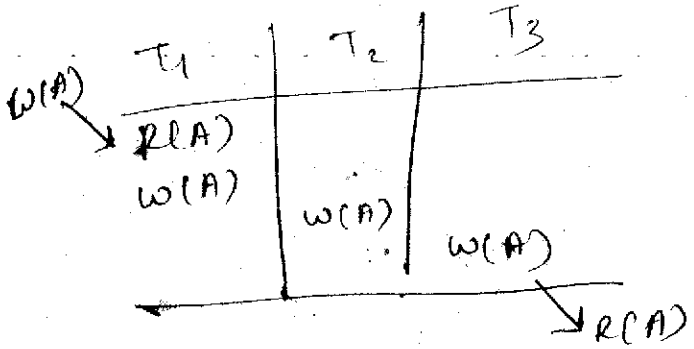
S_3 is not VS

S_3 is not C.S

→ So S_3 is not VS

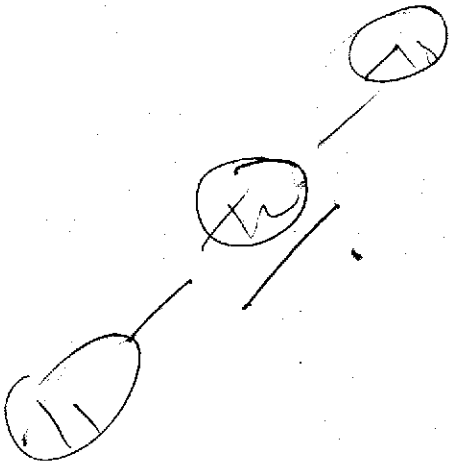
Q





T_1 execute by T_2

T_2 execute by T_3



S_6 is V.S ($T_1 \rightarrow T_2 \rightarrow T_3$)

S_6 is not C.S

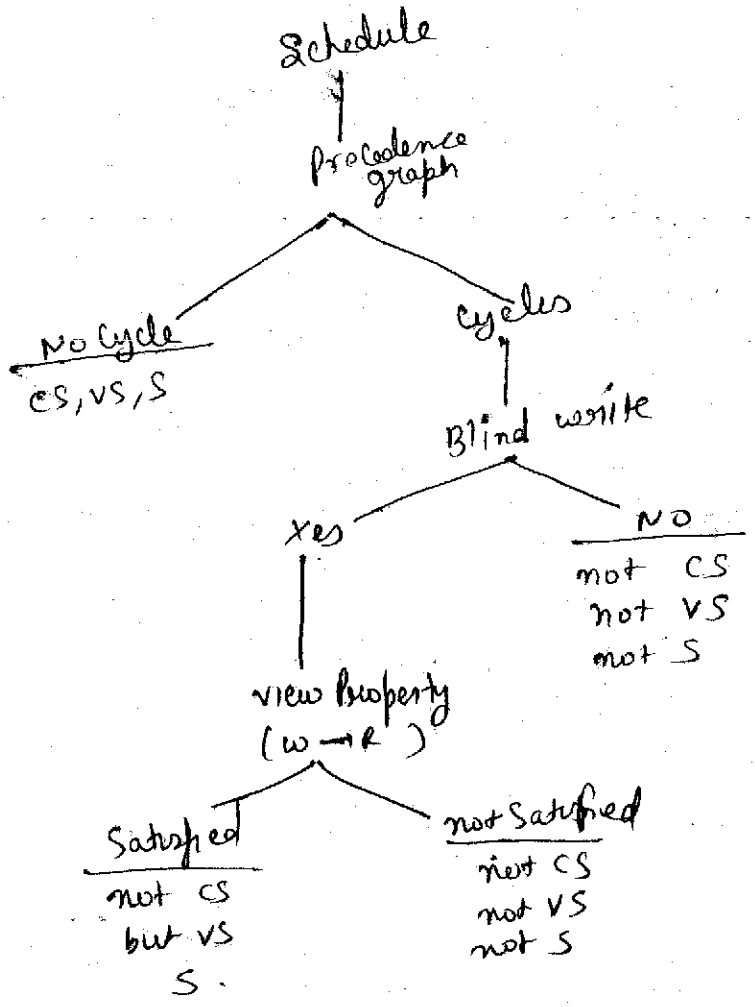
① A transaction that performing a write opr without reading it of some data is called blind write opr

② Every conflict serializable schedule is also view serializable but every view serializable need not be conflict serializable.

③ A schedule that is not C.S but V.S then there must be at least one blind write operation

④ A schedule that is C.S or V.S or both is said to be serializable schedule

⑤ A schedule that is not C.S and there is no blind write opr Then the schedule must not be V.S



Pg-74
Q-3

(a)

(b)

(c)

R(x) R(y) W(x) R(x)

Recoverable
C.R
Not strict

dirty Read
from $T_3 \to T_2$

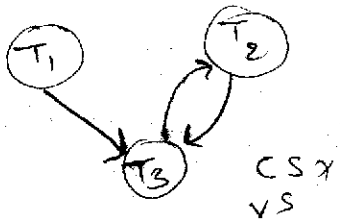
(d)

2 – 3
3 –

Cycle →

dirty read 3 – 2

Recov
C.R
Not Strict



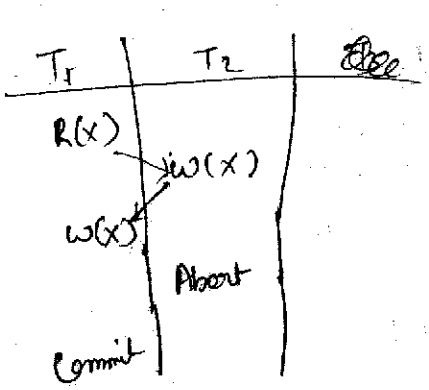
CSR
VS
S

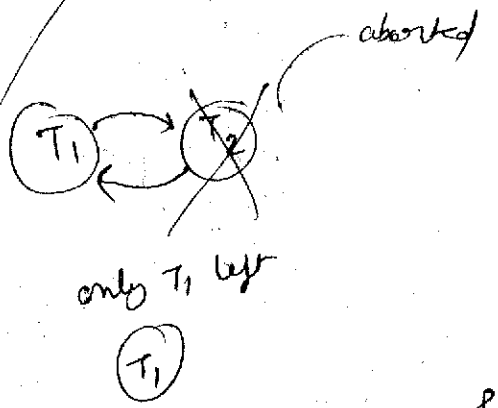
(e)

T_1	T_2	T_3
R(x)		
R(y)	R(y)	
w(x)		
	w(y)	
		R(y)
	R(y)	

not V.S

if we change
the above by T_1 • w(x)
then strict



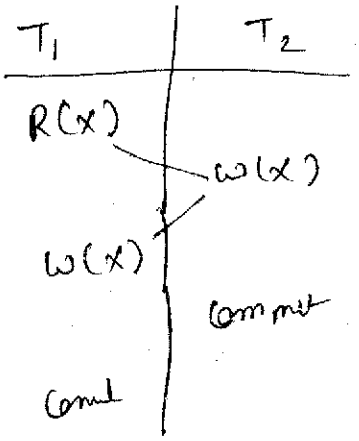


(4)

no $T_1 \to T_2$ possible in future
C.S no VS,S no dirty read
R_1 no ch

no Strict

(7)



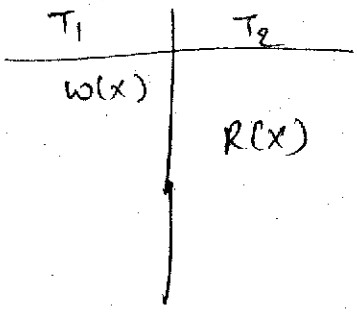
Cycle
So not CS

NOT V.S

No

R, NO C.R
Not Strict

(8)

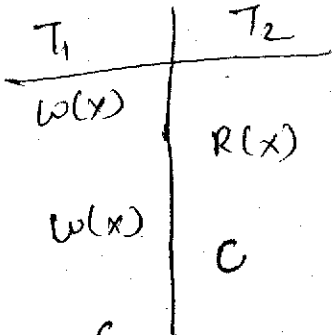


T1 abort
so CS
VS
S

~~not~~ dirty read
R, CR

Not Strict

(9)



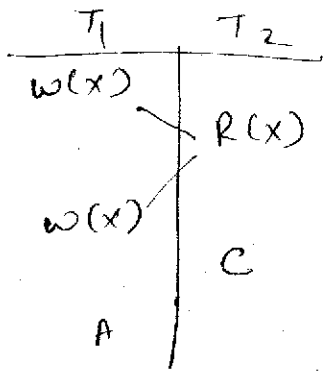
NOT CS
NOT VS
NOT S

~~dirty Read~~

NOT R
C.R is there

NOT Strict

(i)



2 transaction, if one abort then we can say it is C.S

T1 abort so C.S

VS

S

dirty read

NOT Recoverable

C.R required

NOT Strict

(J)

CS

VS

S

Strict

So R

No C.R

(K)

CS

VS

S

No dirty read

Strict

So R ← every strict is

No C.R ←

(L)

NOT CS

NOT VS

NOT S

dirty read is there (from 1 - 3)

but not committed by

so R

C.R required

NOT Strict

Concurrency Control

The data item can be shared by the data items in the mutually exclusive manner i.e only one transaction can exists the data item

Lock based Protocols

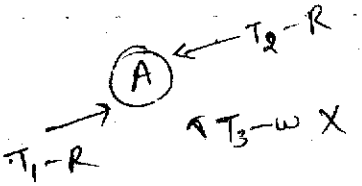
2-Lock modes

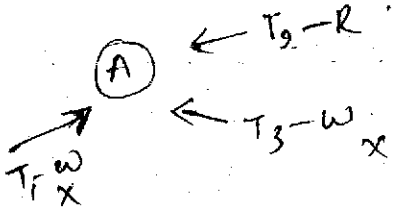
① Shared lock (Lock-S) :- Only Read data

② Exclusive lock (Lock-X) :- Both Read and write data

B	S	X
S	✓	X
X	X	X

← compatible matrix of lock modes





eg → $T_1 : B \xrightarrow{100} A$

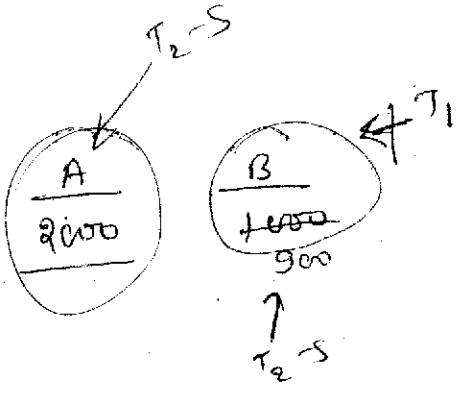
Lock-X(B)
R(B)
B = B-100
W(B)
Unlock-(B)
Lock-X(A)
R(A)
A = A+100
W(A)
Unlock(A)

T_2 : Display (A+B)

Lock-S(A)
R(A)
Unlock(A)
Lock-S(B)
R(B)
Unlock(B)
display (A+B)

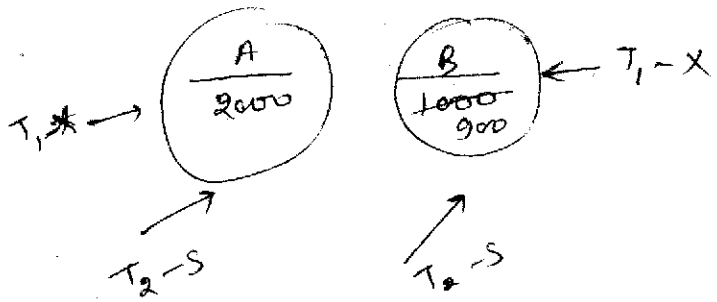
eg →

T_1	T_2
Lock-X(B)	
1000 — R(B)	
900 B = B-100	
900 W(B)	
Unlock(B)	
	Lock-S(A)
	R(A) 2000
	Unlock(A)
	Lock-S(B)
	R(B) 900
	Unlock(B)
	Display (A+B) 2900
Lock-X(A)	
R(A)	
A = A+100	
W(A)	
Unlock(A)	
	Commit
Commit	

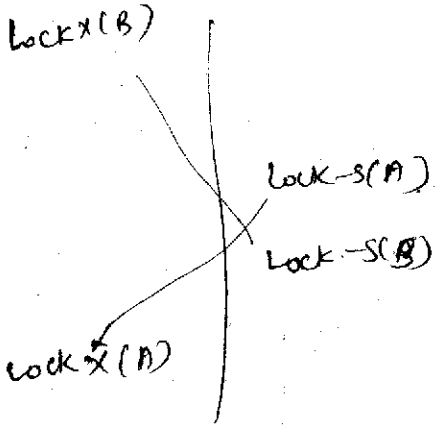


— inconsistent result due to early unlock of B

T_1	T_2
Lock-X(B)	
R(B)	
B = B − 100	
W(B)	
	Lock-S(A)
	R(A)
	Lock-S(B)
	R(B)
	Display(A+B)
Lock-X(A)	
R(A)	
A = A + 100	
W(A)	
	Commit
Commit	
Unlock(B)	
Unlock(A)	
	Unlock(A)
	Unlock(B)



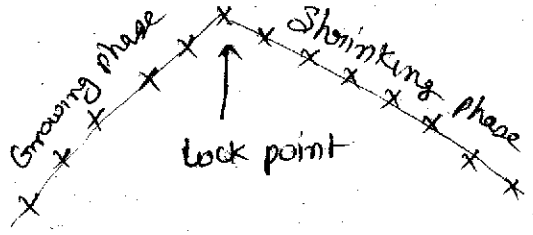
→ "Deadlock"



2- phase locking protocal (2PL)

① Growing phase ;- T's can acquire the locks but can't release

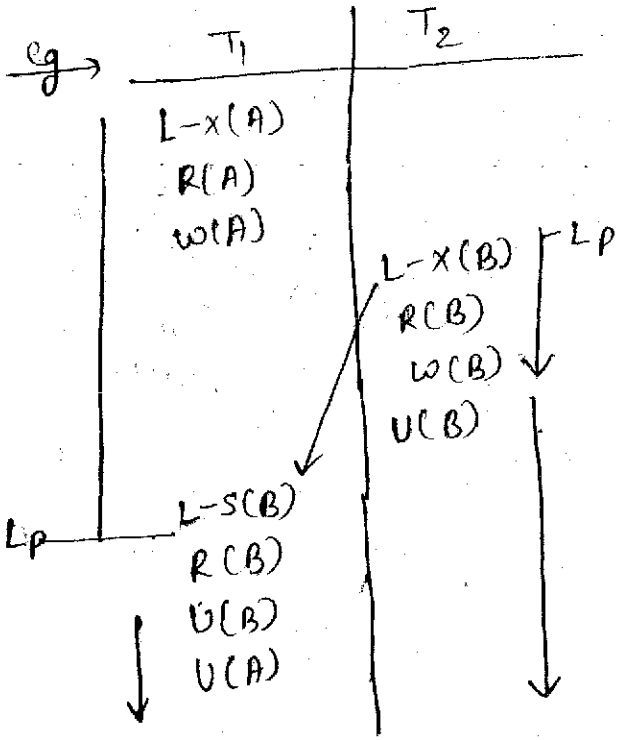
② Shrinking phase :- T's can release the lock but can't acquire

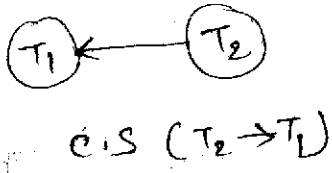


Lock Point

The point in a transaction that performed its final lock

* Any Schedule that is possible under 2PL must ensures Conflict serializability

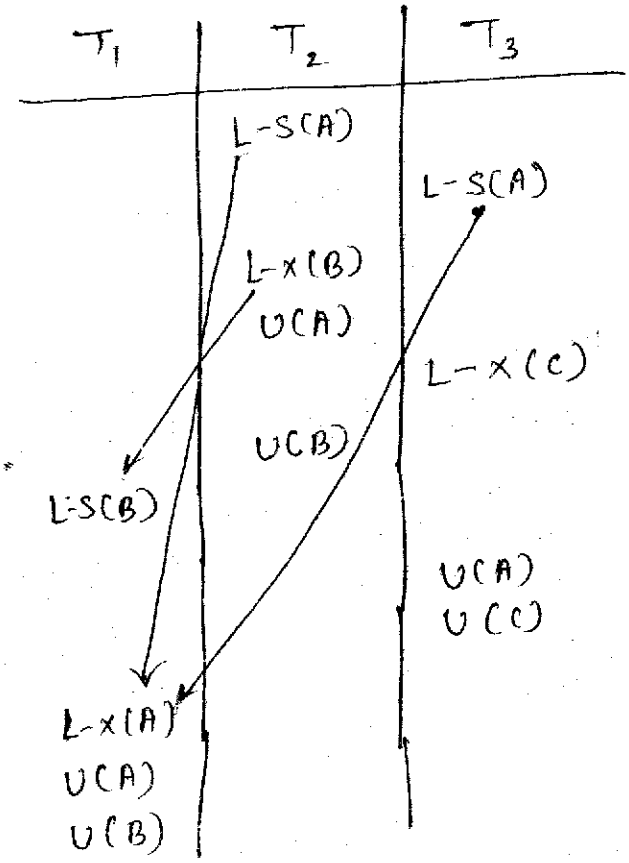


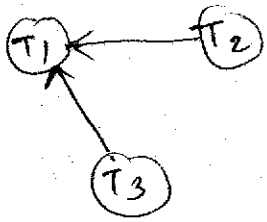


C.S ($T_2 \rightarrow T_1$)

Dead lock can occur under 2PL ← see Prev Page eg

Find the Conflict serializable order for the following schedu

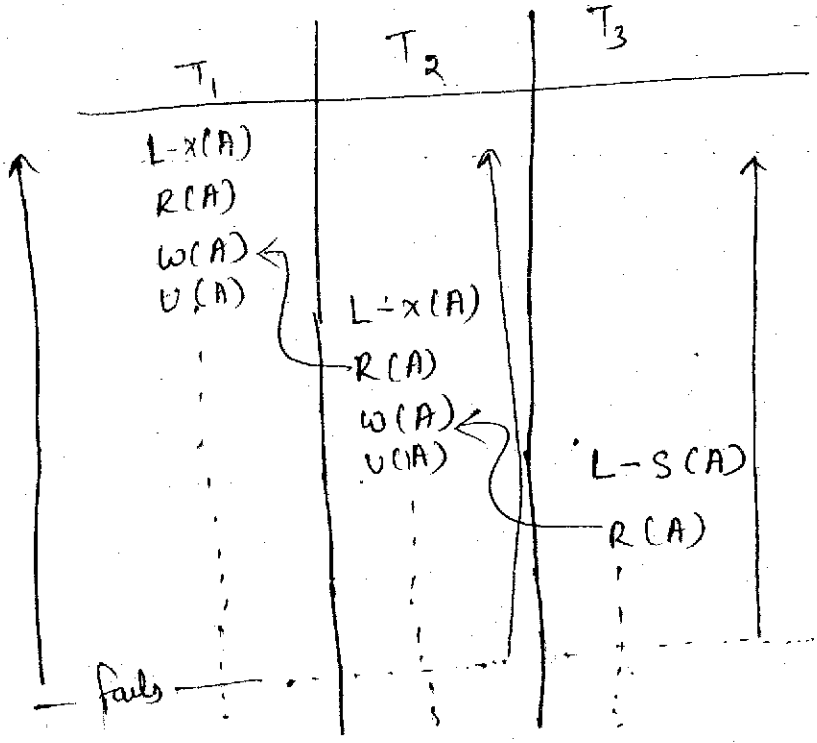




C.S

$\{T_2 - T_3 - T_1\}$

$\{T_3 - T_2 - T_1\}$



here Roll back is bcz of unlock b4 commit

if we unlock Resources after Commit then Rollback no occurs

Cascading Rollback may occur under ~~SS~~ 2PL protocols

CR can be avoided by a modification of 2PL called Strict 2-PL or Regorous 2PL

Strict 2PL

It requires that in addition to locking being 2 phase all the exclusive mode locks taken by a transaction must be held until it commits

Rigorous 2PL

It requires that all locks taken by the transactions must be held untill it commits

Q. Consider the following schedules

S_1	S_2	S_3	S_4
L-X(A)	L-X(A)	L-S(A)	L-S(A)
R(A)	R(A)	R(A)	R(A)
W(A)	W(A)	L-X(B)	L-X(B)
U(A)	L-S(B)	R(B)	R(B)
L-S(B)	R(B)	U(A)	W(B)
R(B)	U(A)	W(B)	Commit
U(B)	U(B)	Commit	U(A)
Commit	Commit	U(B)	U(B)

which of the above schedules ~~results~~ is according to Strict 2PL

not 2PL	Basic 2PL (2PL)	Strict 2PL	Strict 2PL Rigorous 2PL

4-8

Every Rigorous 2PL is Strict 2PL but Every Strict 2PL need not be Rigorous 2PL

Most database system uses either strict or rigorous 2PL

Graph based Protocol

It is a deadlock free protocol

eg →

T_1	T_2
B ←	→ A
A	B

deadlock

T_1	T_2
A	A
B	B

Deadlock not possible so order matters

(1) It requires that we need to have prior knowledge about the ordering about the data items that is how each transaction is going to access the data items to acquire such prior knowledge we

If $d_i \rightarrow d_j$ is an ordering then any transaction that requires both d_i and d_j must access d_i before accessing d_j

(2) The ordering about the data items is represented using a directed graph called the database graph

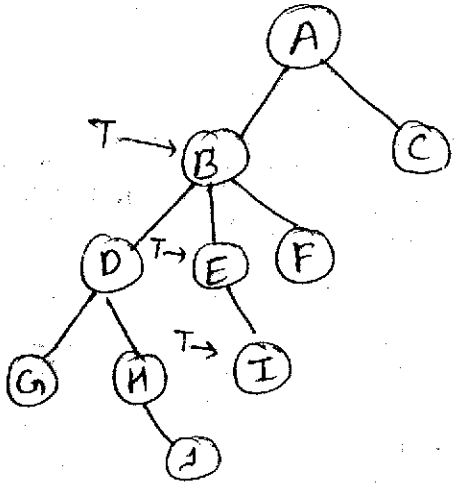
One of the simple protocol called tree protocol which is restricted to employee only exclusive mode of locks and must also observed the following rules

(a) first locks by transaction must be any data item

$$D = \{d_1, d_2, \ldots d_n\}$$

$$d_i \longrightarrow d_j$$

Directed graph



(b) Subsequently a transaction T can lock a data item only if the parent of the data item is currently locked by T

(c) Unlocking may be done ~~at by~~ any time

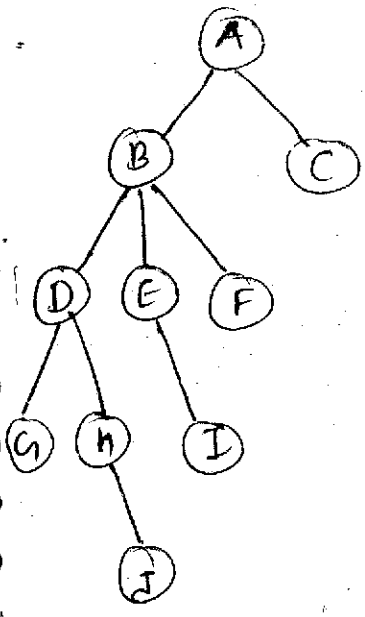
(d) A data item that has been locked and unlocked by the a transaction T that can not be relocked by the same transaction.

i.e Each data item can locked by a transaction atmost once.

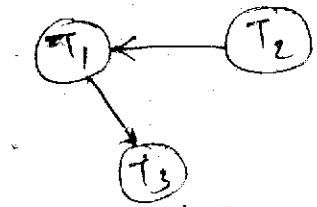
NOTE

Any Schedule that is possible under tree protocol must ensure the Conflict Serializability

Q) Find the Serializability order of the following schedule based on the database graph given below.



T_1	T_2	T_3
L-x(B)		
	L-X(D)	
	L-x(H)	
	U(D)	
L-x(E)		
L-x(D)		
U(B)		
U(E)		
		L-x(B)
		L-x(E)
	U(H)	
L-x(G)		
U(D)		
		U(E)
		U(B)
U(G)		

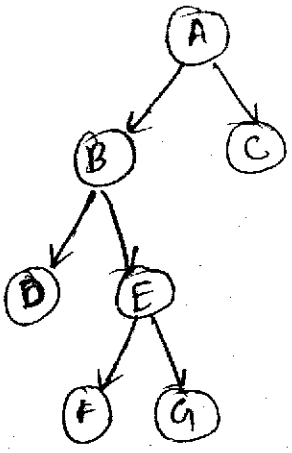


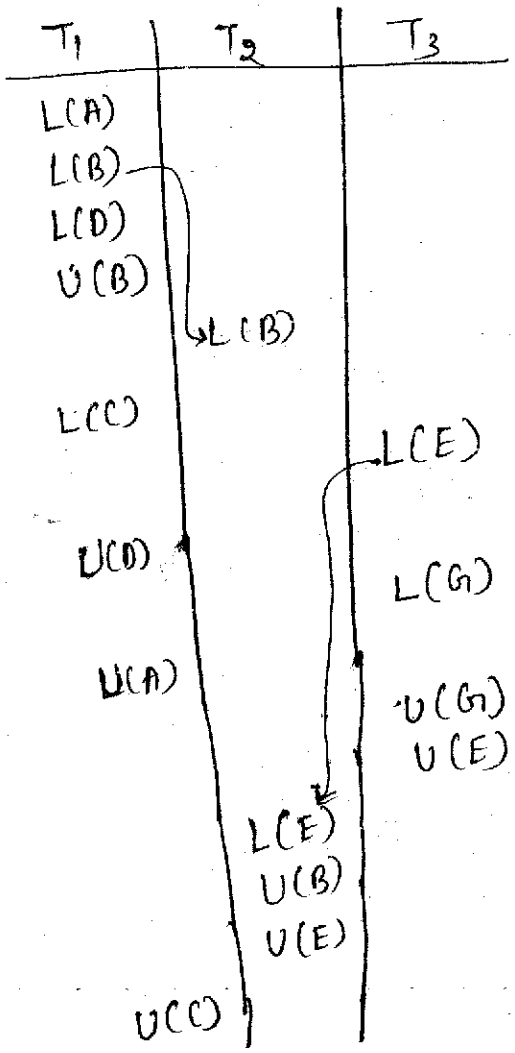
Since No cycle so

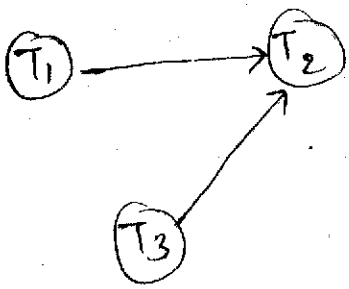
C.S

$(T_2 \rightarrow T_1 \rightarrow T_3)$

find the serializability order of the following schedule based on the database graph given below.







No cycle so

C.S

$T_1 \rightarrow T_3 \rightarrow T_2$

$T_3 \rightarrow T_1 \rightarrow T_2$

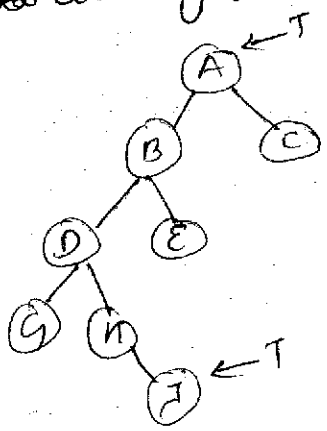
Advantage

- free from Deadlock
- Ensure Conflict Serializability

Drawback

- Need to have Prior Knowledge of data ordering of data items
- Increased locking overhead

eg → If a Trans need data items A and J then it has to lock B, D, H unnecessarily



Time Stamp based protocols

(1) It determines ordering about the transactions based on the time stamp of a transaction

Time Stamp :— The time when the transaction enters into the system for execution is known as Time stamp.

(2) Each transaction is given a fixed unique Time stamp denoted by $Ts(T_i)$

(3) If T_j enters after T_i then the relation bet^n their time stamp is $\boxed{Ts(T_i) < Ts(T_j)}$

(4) If $Ts(T_i) < Ts(T_j)$ then the system must ensure that the produced schedule is equivalent to a serial schedule $T_i \rightarrow T_j$

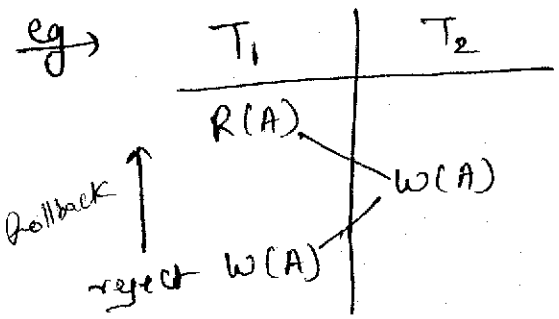
(5) If any Transaction that is not executing in the order of ~~their~~ their time stamp such operation is rejected and the transaction is rolled back

(6) The aborted Trans will restart in system with fresh time stamp

(1) Time Stamp ordering protocol

It requires to process all the Conflicting in Read & write operation in the order of ~~their~~ Their Time stamps.

eg →



~~Rule~~:

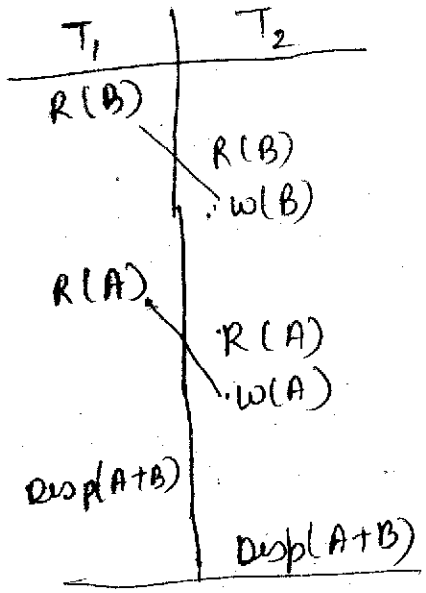
(1) $T_S(T_1) < T_S(T_2)$

(2) $T_1 \rightarrow T_2$

(3) The schedule is not possible under T.O.P

(4) The Schedule is Possible under Thomas write Rule (T.W.R)

Q Test the following schedule possible under Time Stamp O.P

①


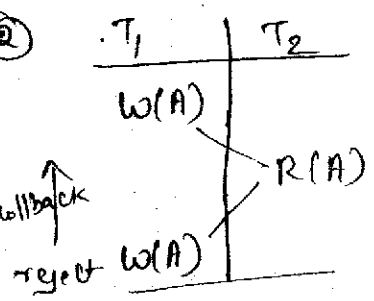
① $TS(T_1) < TS(T_2)$

② $T_1 \longrightarrow T_2$ (order of time stamp)

③ both the conflicts are from T_1 to T_2 ($T_1 \longrightarrow T_2$)

So The schedule is possible under T.O.P

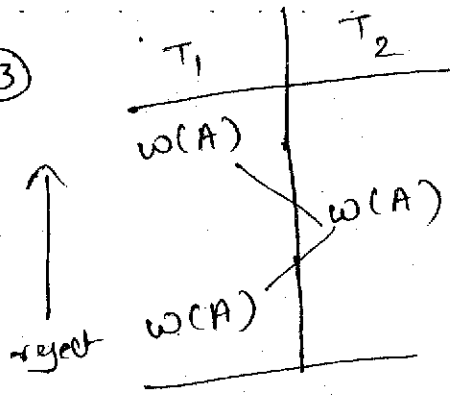
④ Also possible under TWR

②


① $TS(T_1) < TS(T_2)$

⑤ $T_1 \longrightarrow T_2$

ⓒ Not possible under T.O.P

③


① $TS(T_1) < TS(T_2)$

⑤ $T_1 \longrightarrow T_2$

ⓒ Not possible under TOP

ⓓ not possible

Advantage

- Ensures the Conflict Serializability

Disadvantage

- Starvation may occur due to continuously aborting (Rollback) & restarting the Transaction

(2) Thomas write Rule (T.B.W.R)

eg (1)

T_1	T_2
R(A)	
	W(A)
Ignore w(A)	

obsolete write or useless write

This protocol is a modification of time Stamp ordering protocol
It requires to ignore the obsolete write operations
also called useless write

eg (2)

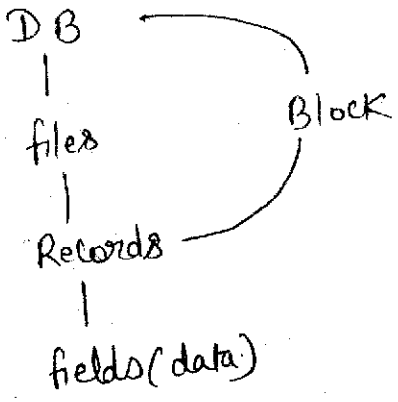
T_1	T_2
W(A)	
	W(A)
W(A)	

Blind write op

Not possible under TOP
and also
Not possible under T.W.R

$T_1 \rightarrow T_2$	Rollbacks	Allowed
T.O.P	① $R_2(A)$ $w_1(A)$ ② $w_2(A)$ $R_1(A)$ ③ $w_2(A)$ $w_1(A)$	① $R_2(A)$ $R_1(A)$
T.w.R	① $R_2(A)$ $w_1(A)$ ② $w_2(A)$ $R_1(A)$ ③ $w_2(A)$ $w_1(A)$ ↑ y not obsolete	① $R_2(A)$ $R_1(A)$ ② $w_2(A)$ $w_1(A)$ ↑ y obsolete

File Organization

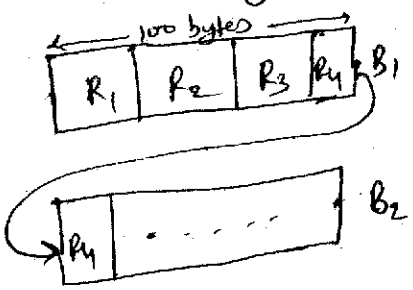


Blocking factor :- It is the avg no. of file records stored in a disc block.

for storing records into blocks we have two strategy.

① Spanned Strategy

In which partial part of a record can be stored into a block



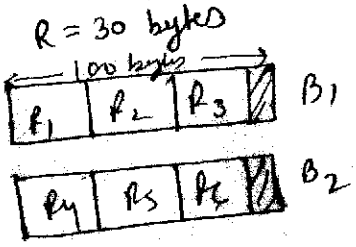
Adv :- No wastage of memory

Disad :- More no. of Block access

Suitable :- Variable length records

② Unspanned Strategy.

In which no records can be stored in more than one block



Adv :- Less no of Block access

Disadv :- wastage of memory

Suitable :- fixed length records.

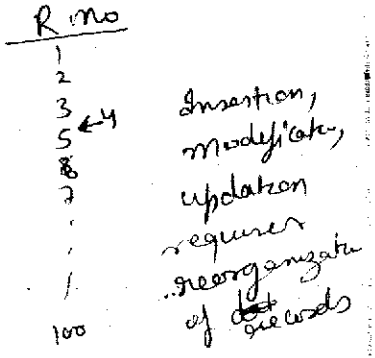
Organization of records in a file

① unordered file organization :- Any record can be placed any where in the file where there is a place for the records usually records are inserted at the end of file.
// user sequential

Advantage
- Insertion is Easy

Disadvantage
- searching is expensive.

② Ordered file Organization :- Records ~~can be~~ are ordered based on some search key value.
// uses Binary search



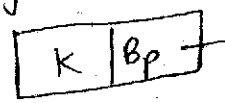
Advantage
- Searching is efficient

Disadvantage
- Insertion is expensive

Index

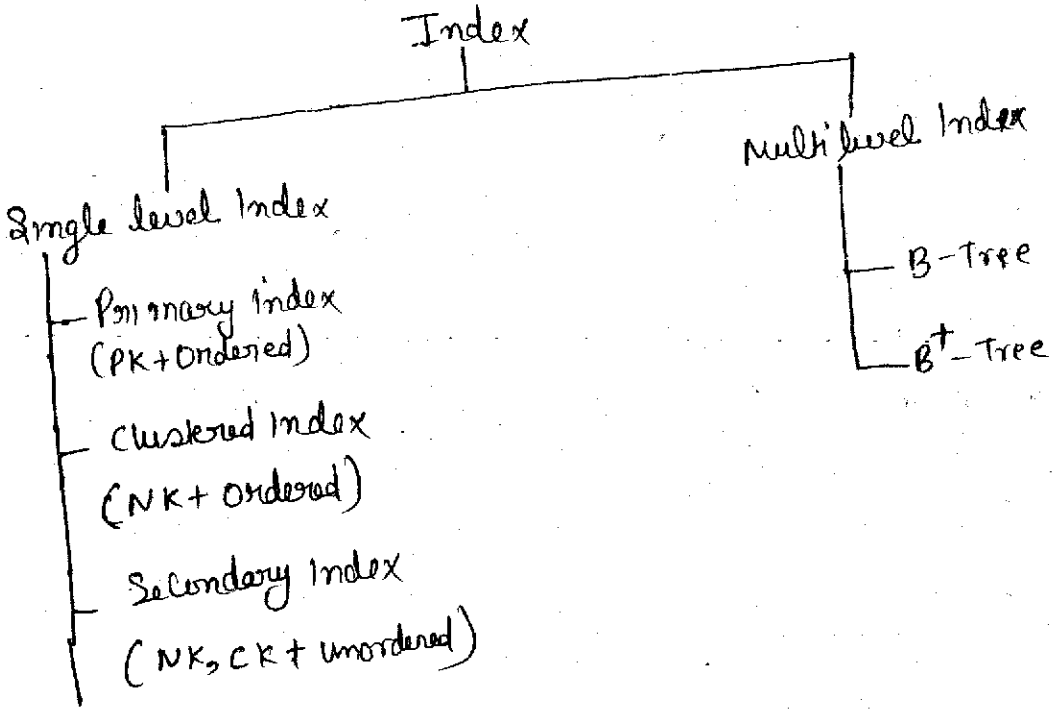
Index is used to speedup the retrieval of records in responce to certain search condition

① Index is a collection of records with two fields Key and block pointer



② Index may be created on any field of the file (Primary Key, Candidate key or non key)

③ Index is an ordered file on which we perform binary search



Other classification of Index

① Sparse index

② Dense index

① Sparse Index :- It has an Index entry only for some search key values

② Dense Index :- It has an Index entry for every search key values.

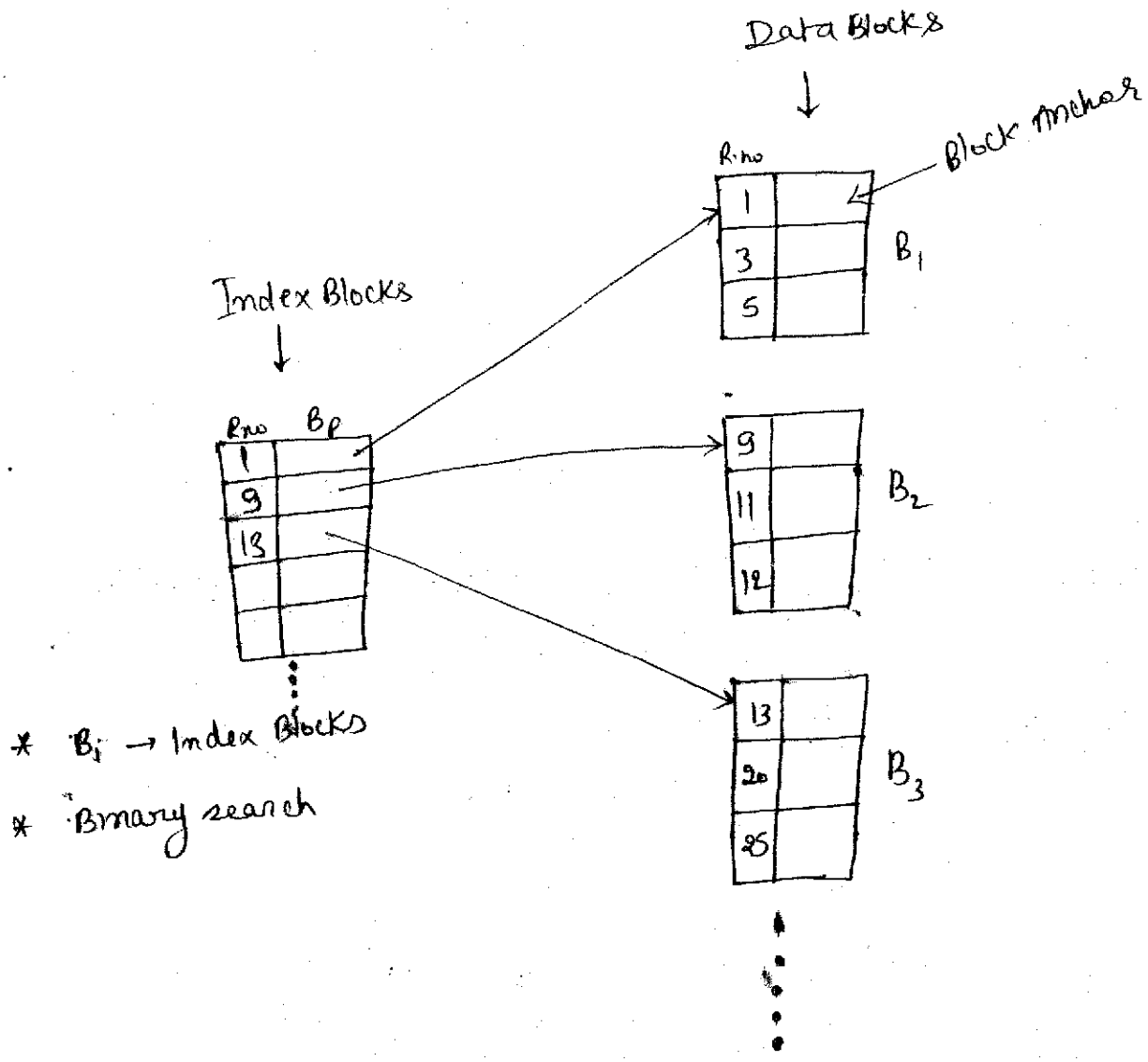
Primary Index

PI is an ordered file whose records are of fixed length with two fields the first field of the same data type as the ordering key called the primary key of the data file and the second field is pointer to disk block.

① An Index entry is created for first record of each block called "block Anchor"

② The no. of Index records are equals to no. of blocks in the data file (data blocks)

③ The type of Primary Index is called sparse Index



* B_i → Index Blocks
* Binary search

* The avg no of Block access
$= \log_2 B_i + 1$

* B-Blocks
* Binary search
* Any no of Block access

Suppose that we have an ordered file with 30,000 records stored on a disc with block size 1024 bytes. Find record are of fixed size and as are unspare record with records 100 bytes

Now suppose that if we create Primary Index on the ordering key field of the file of size 9 bytes and a Block pointer is 6 bytes.

① Then find the avg no. of block accesses without & with Primary Index.

Given, $r = 30,000$
$R = 100$ bytes — fixed, unspanned
$B = 1024$ bytes
$K = 9$ byte, $B_p = 6$ bytes

Blocking factor $= \left\lfloor \frac{1024}{100} \right\rfloor = 10$ records/Block (bc of unspanned (given))

No. of Data blocks $= \left\lceil \frac{30,000}{10} \right\rceil = 3000$

The avg no. of Block accen $= \lceil \log_2 3000 \rceil$

$= 12$ block accen

Size of index record = $K + Bp$

$= 9 + 6 = 15$ Bytes

No of index records = 3000

Blocking factor = $\left\lfloor \frac{1024}{15} \right\rfloor = 68$ index records / Blocks

No. of index Blocks = $\left\lceil \frac{3000}{68} \right\rceil = 45$

The avg no. of Block accen = $\lceil \log_2 45 \rceil + 1$

$= 6 + 1$

$= 7$ block accen

Clustered Index

If records of a file are ordered on a non key field which does not have distinct value for each record that field is called clustering field.

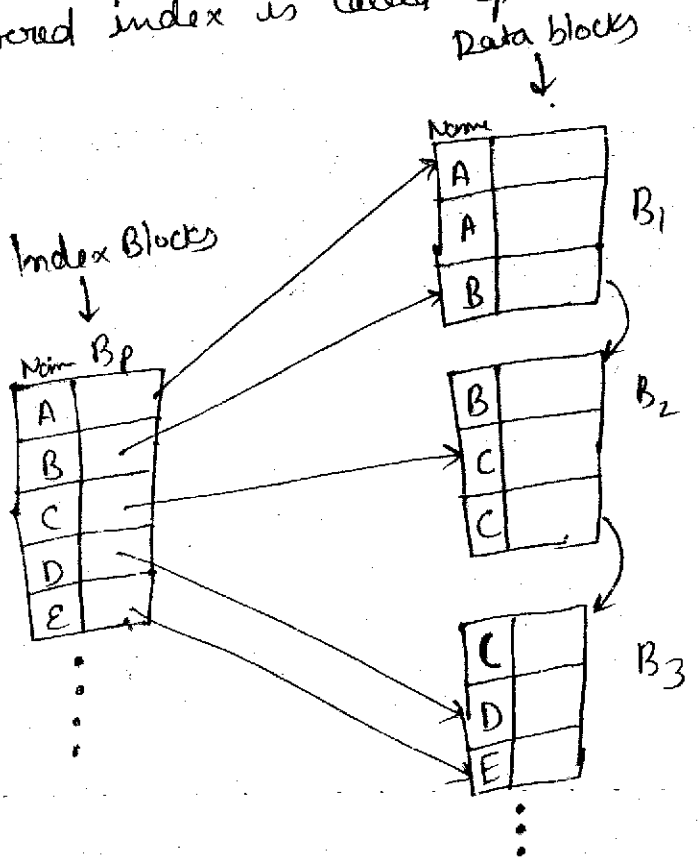
we can create different type of Index called Clustering Index to speed up the retrieval of records that have the same value for the clustering field

clustering Index is an ordered file with two field. The first field is of the same ~~field~~ type as the clustering field of the data type & the second field is a block pointer

An Index entry is created for each distinct value of a non key and

Each block pointer points to the first block in the data file that has the record with that clustering field

The type of clustered index is called sparse index



* B_i – Index Blocks
* Binary search

The avg no of Block access
$>= \log_2 B_i + 1$

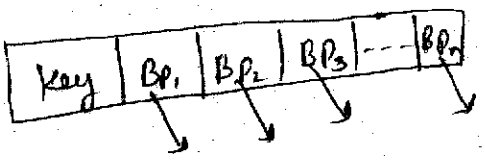
* B – Blocks
* Binary search
* avg no of Block access
$>= \log_2 B$

Secondary Index

Secondary Index Provide is

Secondary Index may be ~~called~~ or Candidate Key or a non key or unordered file

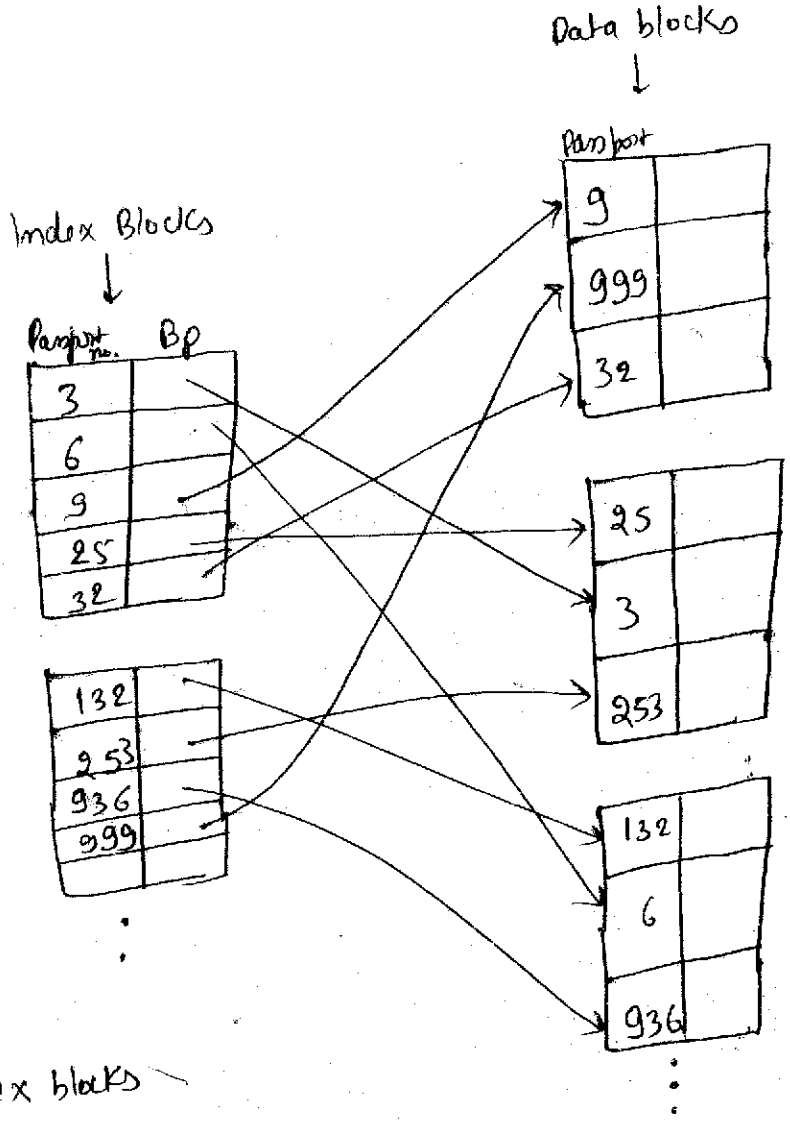
① Secondary index (NK + unorder)



② Secondary index (CK + unorder)

The no. of Index records are equals to the no. of records in a data file i.e an index entry is created for each record in a data file

The type of This Index is called dense index



* B_i – Index blocks
* Binary Search
* The avg no of Block accen
$= \log_2 B_i + 1$

* B – Blocks
* Linear Search
* Avg no of block accen $= \frac{B}{2}$

Q2 If a secondary Index is created on the ~~Primary~~ key field of the file of Q1 then find the avg no. of block accesses using with or without Index.

Given

$r = 30,000$

$R = 100$ bytes, fixed, unspanned

$B = 1024$

$K = 9$ bytes, $Bp = 6$ bytes

Blocking factor $= \lfloor \frac{1024}{100} \rfloor = 10$ records/block

No of data blocks $= \lceil \frac{30000}{10} \rceil = 3000$

The avg no. of Block accers $= \frac{3000}{2} = 1500$

Size of index record $= K + Bp$

$= 9 + 6 = 15$ bytes

no of index records $= 30,000$

Blocking factor $= \lfloor \frac{1024}{15} \rfloor = 68$ index records/blocks

No. of index Blocks $= \lceil \frac{30000}{68} \rceil = 442$

The avg no. of Block accers $= \lceil \log_2 442 \rceil + 1$

$= 9 + 1$

$= 10$ Block accers

Multilevel Index

(1) An index over an index is called multiple level index

(2) On any of the first level index we create a primary index called second level index on which we create a primary index called third level index and so on until all the index records fit in one disc block

(3) Number of index ~~block~~ records in the n^{th} level are equals to no. of blocks in the $(n-1)$ level
where $n \geq 2$

(4) The no. of blocks to access is equals to $(l+1)$ where l is the no. of levels i.e
At each level we access one block and one block in the data file.

Q3 Suppose if we create a multi level index on the key field of the file of question 2 Then find the no. of block access.

$r = 30,000$

$R = 100$ bytes

B

K =

Size of index record $= K + Bp$

$= 9 + 6 = 15$

1st No of index records $= 30000$

Blocking factor $= \left\lfloor \frac{1024}{15} \right\rfloor = 68$ index records/Blocks

No. of index Blocks $= \left\lceil \frac{30000}{68} \right\rceil = 442$

2nd no. of index records $= 442$

no. of index blocks $= \left\lceil \frac{442}{68} \right\rceil = 7$

3rd no. of index records $= 7$

no. of index blocks $= \left\lceil \frac{7}{68} \right\rceil = 1$

The no. of block acces $= B\ l + 1$

$= 3 + 1$

$= 4$ block acces

Pg-82
Q=7

Size of index record = $k + B_p$
= 6+10 = 16

No of index records = 16,384

Blocking factor = $\left\lfloor \frac{1024}{16} \right\rfloor = 64$

No of index Blocks = $\frac{16}{64}$ = 256

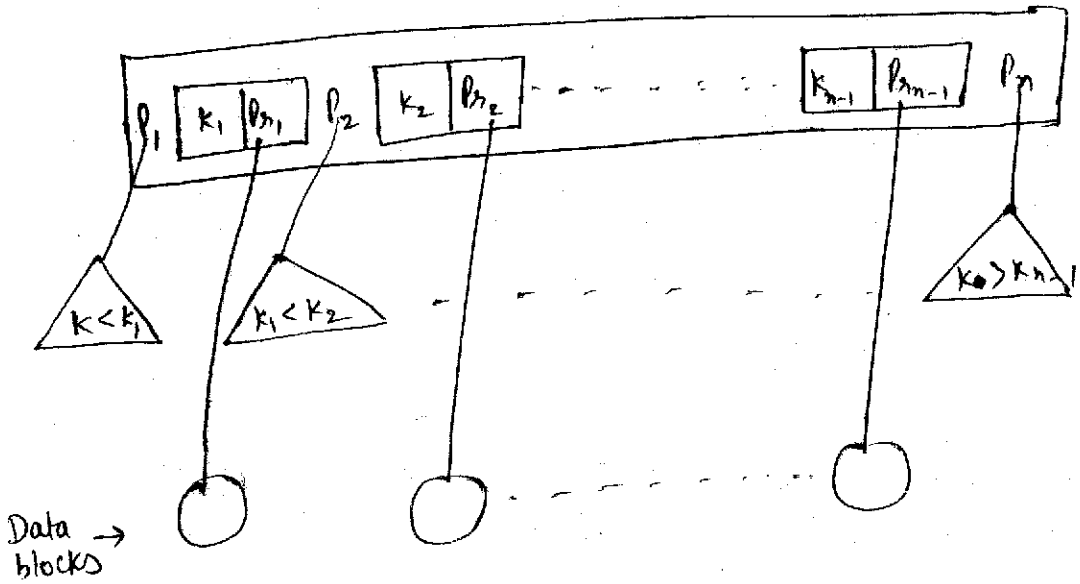
Implementation multilevel Index Using B-Tree

B-Tree

① B-Tree is a height balanced searchtree

② The node structure of a B-Tree corresponds to one block in the index file

Structure of a B-Tree node



$P_1, P_2, \ldots\ldots P_n$ ← Block Pointers / index pointer / tree pointer

$K_1 < K_2 < \ldots\ldots < K_{n-1}$ ← keys

$P_{r_1}, P_{r_2}, \ldots\ldots P_{r_{n-1}}$ ← Data pointers / Record Pointers

① B-Tree provides direct access i.e the key to search is found at some level of it bypasses all the next levels of index and access the data blocks

Order of B-Tree (Let n)

Order of B-Tree is the no. of tree pointers in a node

$n \leftarrow$ tree pointers

$(n-1) \leftarrow$ Index records

$$n*P + (n-1)*(K + Pr) \leq B$$

Q4 Calculate the order of a B-Tree stored on a disc with block size 512 bytes, key size 9 bytes and record pointers of size 7 bytes and block pointer of size 6 bytes.

$$n*6 + (n-1)*(9+7) \leq 512$$

$$6n + 16n - 16 \leq 512$$

$$22n \leq 528$$

$$n \leq 24$$

Q5 Calculate the approximate no. of entries that can be stored in a B-Tree of level-3 of Q4 [Assume that each node is 69% full]

order of 69% full

$= 24 \times 0.69$

$= 16$

Root	1 node	16 Pointers	15 keys
Level 1	16 nodes	256	240
level 2	256	4096	3840
level 3	4096	65536	61,440

Total index records $= 15 + 240 + 3840 + 61440$

$= 65,535$

Q6 find the no. of levels required to store 20,000 index records in a B-Tree of order 12.

Root	1 node	12 pointer	11 keys
Level 1	12	144	132
Level 2	144	1728	1584
Level 3	1721	20738	19008

3 level required or $\log_{12} 20000 = 4$

$\log_{\text{nodes}}^{\text{no. of records}}$ = no of levels (including root)

4

B. Tree Insertion

Order : n

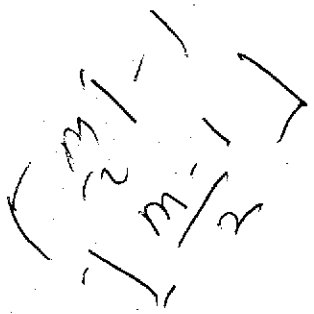
Max : n ← pointers

$(n-1)$ ← Key

Min : $\lceil \frac{n}{2} \rceil$ ← pointers

$\lceil \frac{n}{2} \rceil - 1$ ← Keys

} except for root



eg

order : 5

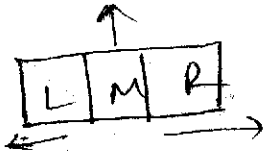
Max : 5 pointers
4 keys

Min : 3 pointers
2 keys

(1) A new element or Key is always inserted in the leaf node

(2) If the node is full we split the node into two nodes moving the middle element to one level above and insert it at its proper place

eg →

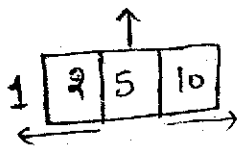


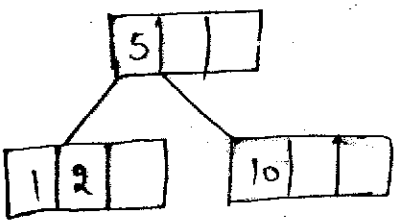
Q7 Insert the following keys in the B-Tree of order 4

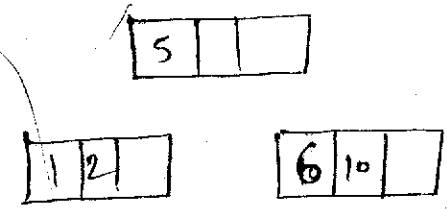
Keys: 2, 5, 10, 1, 6, 9, 4, 3, 12, 18, 20, 25

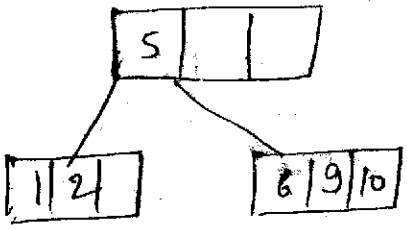
Max : 4 pointers, 3 keys

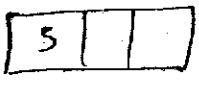
Min : 2 pointers, 1 key

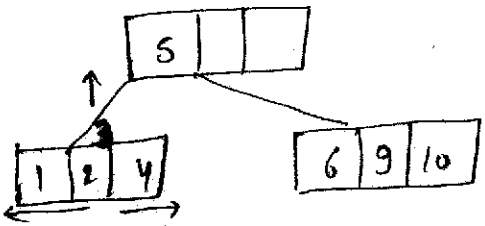


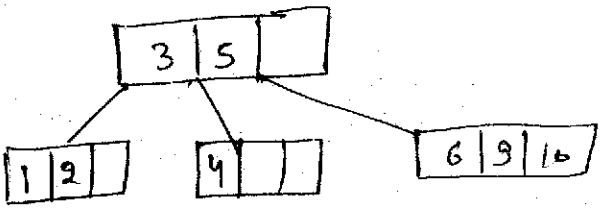


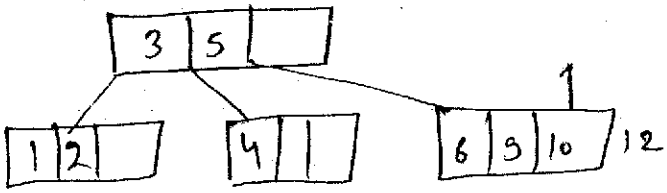


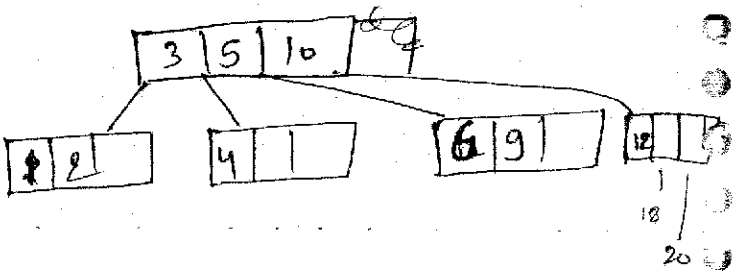


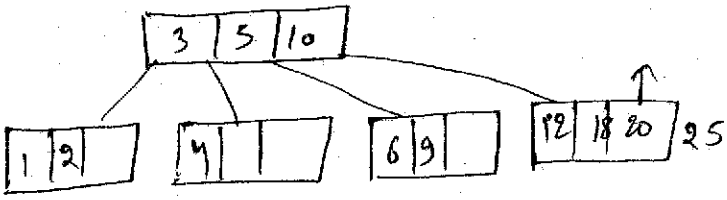


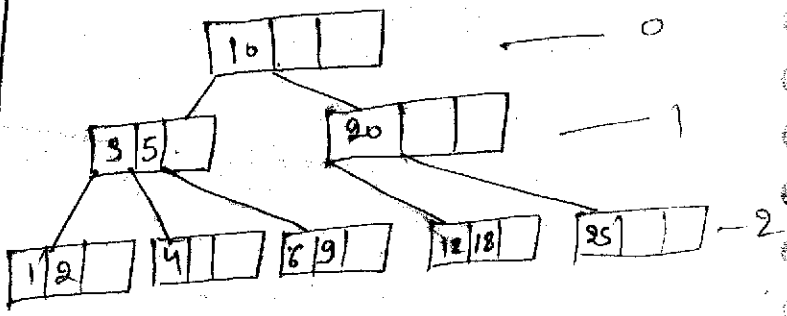












no. of splits = 5

no. of Root splits = 2

no. of leaf splits = 4

total no. of nodes = 8

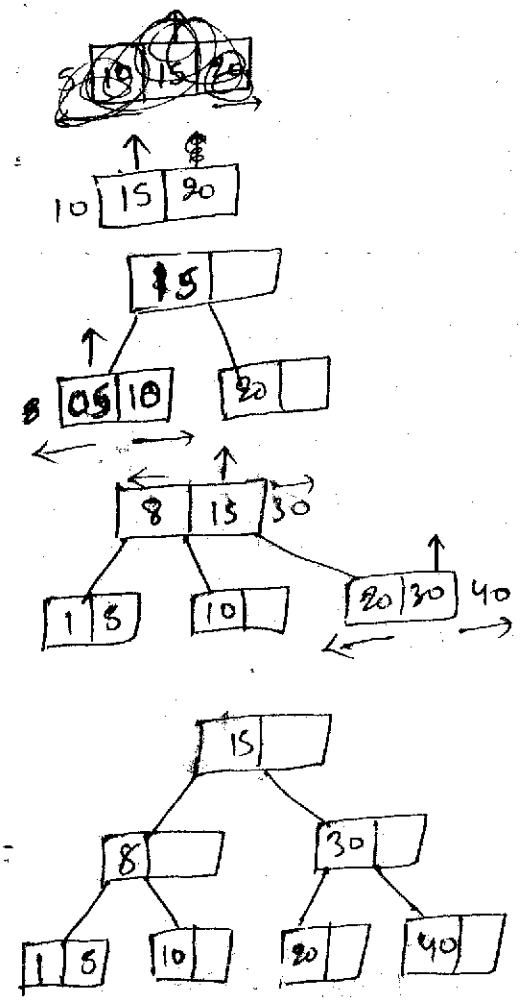
No of nodes with fill factor 100% = 0

Root = 10
no of levels = 2 levels

8 Consider a B-Tree of order 3

20, 15, 10, 5, 8, 30, 1, 40

Max: 3 pointer, 2 keys



No. of splits = 4

No. of Root splits = 2

No. of leaf splits = 3

total no. of nodes = 7

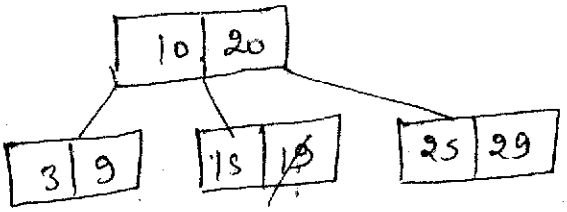
No. of nodes with full factor 100% = 1

Root = 15

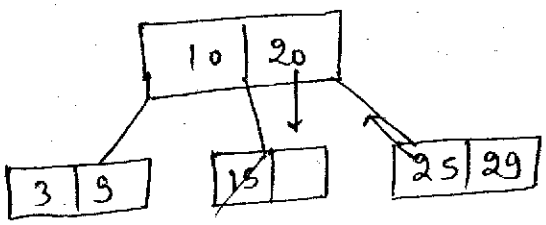
no. of levels = 2

B-Tree Deletion

Consider the following B-Tree



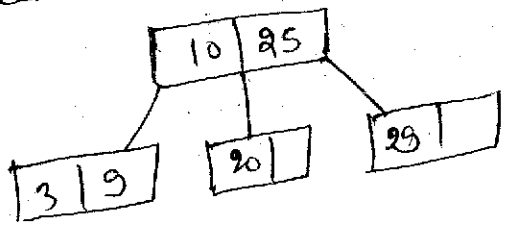
Case① Delete 19



If deleting the key results in more than or equals to min no. of keys, no pointer modifications required i.e

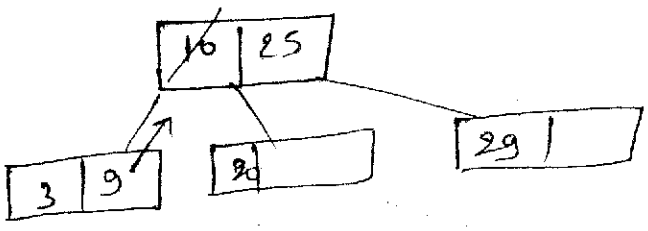
If the node contains more than min keys then delete it

Case② Delete 15

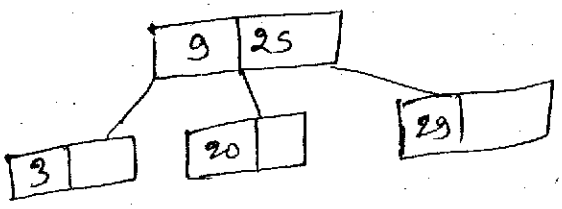


If deleting the key results in less than the min no of keys then borrow the keys either from left or right sibling

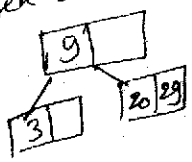
Case ③ delete 10 (deleting Internal node)



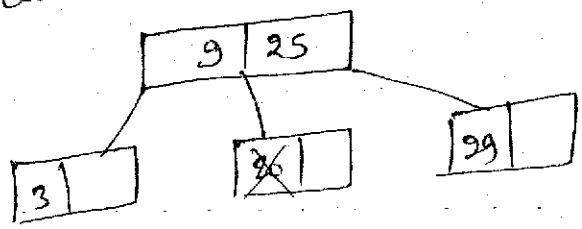
If key to be deleted i.e not a leaf node than find the smallest key greater than the key to delete or largest key less than the key to delete & replace.



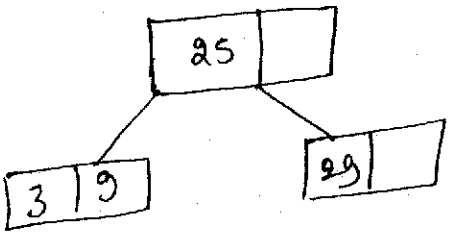
If we delete 25 then



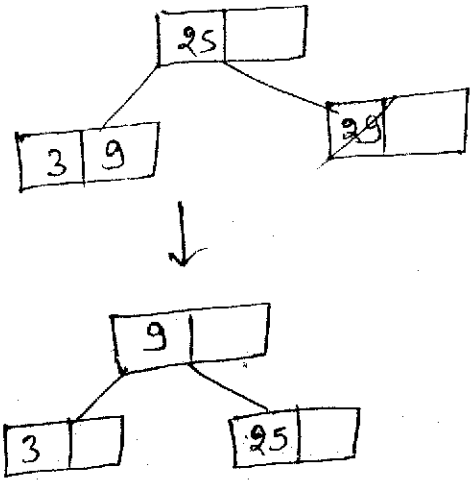
Case ④ delete 20



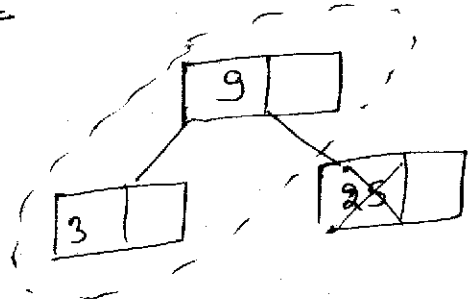
If sibling do not have spare (extra) keys then merge the node parent either with it Right siblings



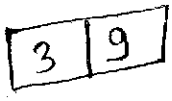
Case (5) delete 29



Case (6) delete 25



The Case in which the parent also do not have ~~so~~ suffcient keys then merge the node parent and sibling to make one node.



Implementation of Multilevel Index using B^+ tree

B^+ Tree (ordered access)

Two modifications of B-Tree node

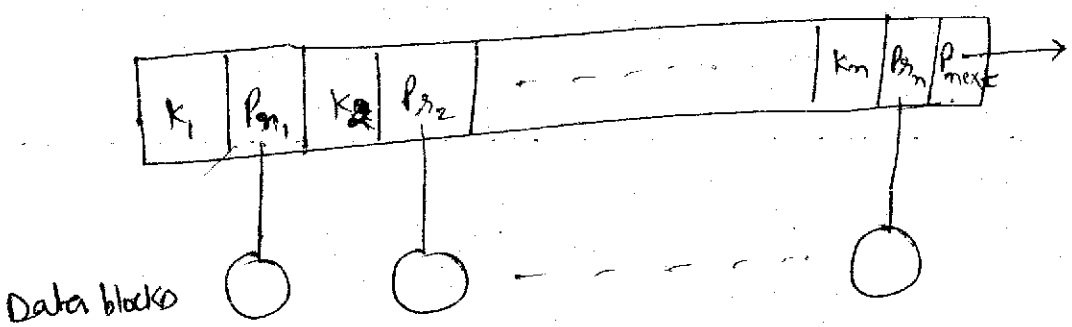
① Leaf node → Remove tree pointers; Contains only Key, Record pointers.

② Internal nodes → Remove record pointers; Contains only Key, tree pointers

NOTE

The Structure of leaf and internal node in a B^+ tree is different

① Structure of Leaf node



$K_1 < K_2 \ldots\ldots < K_n$ ← Key

$P_1, P_2, \ldots\ldots P_{rn}$ ← Record pointer / Data pointer

P_{next} ← Block pointer / tree pointer / index pointer

Order of leaf node (Let n)

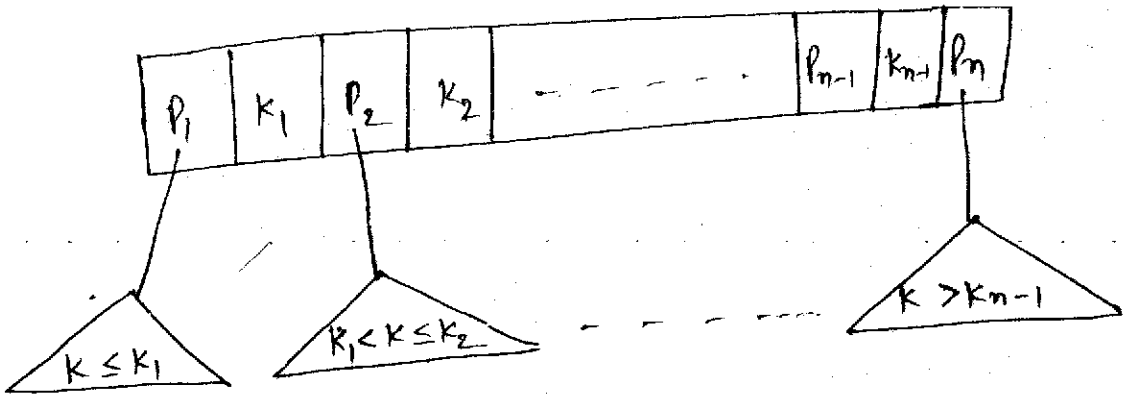
Order of leaf node is the no. of index records in a node

$n \leftarrow$ keys, Record pointer

$1 \leftarrow$ Block pointer (P_{next})

$$n * (K + P_R) + P_{next} \leq B$$

(2) Structure of Internal node



$P_1, P_2, \ldots\ldots P_n \leftarrow$ True pointer | Block pointer | Index pointer

$K_1 < K_2 < \ldots\ldots < K_{n-1} \leftarrow$ keys

~~Order of~~

Order of Internal node (Let n)

Order of internal node is the no of tree pointers in it

$n \leftarrow$ tree pointers

$(n-1) \leftarrow$ keys

$$n * p + (n-1) * k \leq B$$

NOTE Order of leaf and internal node of a B^+ Tree is different

$B = 512,\ K = 9,\ P_R = 7,\ P = 6$

Q8 Calculate the order of B^+ tree Suppose that the search Key is field is 9 bytes The block size is 512 bytes a record pointer is 7 bytes. and the block pointer is 6 bytes

order of internal node →

$$9x(n-1) + 6xn \le 512$$

$$16n - 9 \le 512$$

$$16n \le 521$$

$$\boxed{n \le 34}$$

order of leaf node

$$n(9+7) + 6 \le 512$$

$$16n \le 506$$

$$\boxed{n \le 31}$$

Q9 Calculate the approximate no. of entries that can be stored in a B^+ tree of level-3 of Q8 [Assume that each node is 69% full]

Order of leaf node

$= 31 \times .69$

$= 21$

order of internal node

$= 34 \times .69$

$= 23$

			no need this col
Root	1 node	23 pointers	—
Level 1	23 nodes	529	—
Level 2	529 "	12,167	—
Leaf Level	12,167	$12,167 \times 21 = 9,55,507$	—

Compare it with B-tree

more index stored here

aprox [4 times) and in less level

Pg-82

Q1

$K = 8$ bytes

$B = 512$

$P = 4$

here asking Int node →

$n \times 4 + 8n - 8 \leq 512$

$12n \leq 520$

$\boxed{n \leq 43}$

Q2 $P = 6, K = 14, B = 512$

$n \times 6 + (n-1)14 \leq 512$

$20n \leq 526$

$\boxed{n \leq 26}$

Q4 $B = 1024$

$P_R = 7$

$K = 9$

$P = 6$

$n(K+P) + P_{next} \leq 1024$

$n(9+7) + 6 \leq 1024$

$16n \leq 1018$

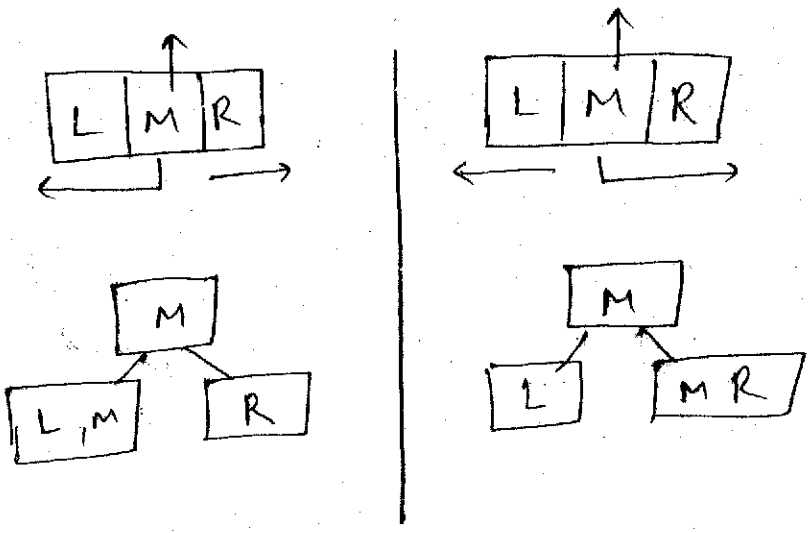
$\boxed{n \leq 63}$

Q5

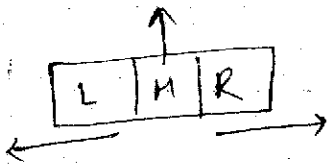
B^+ - Tree Insertion

Property :-

Leaf Split



Internal Split



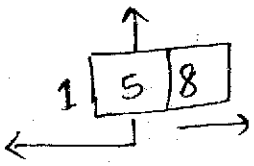
PC

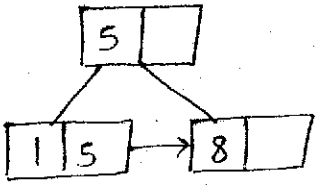
Q10 Insert the following keys in a B^+ tree of ~~order 3~~ Internal node order 3 and leaf node order - 2

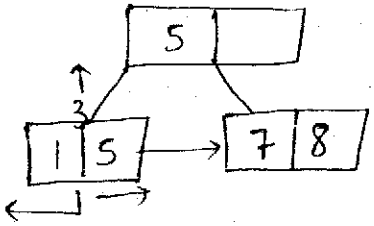
order of Internal 3 → 3 pointers, 2 keys

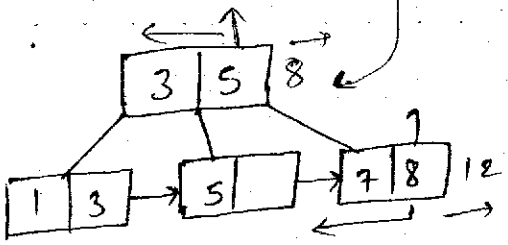
order of Leaf 2 → 2 keys

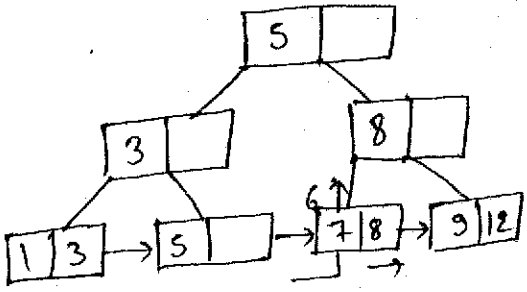
Keys: 8, 5, 1, ~~7~~, 3, 12, 9, 6

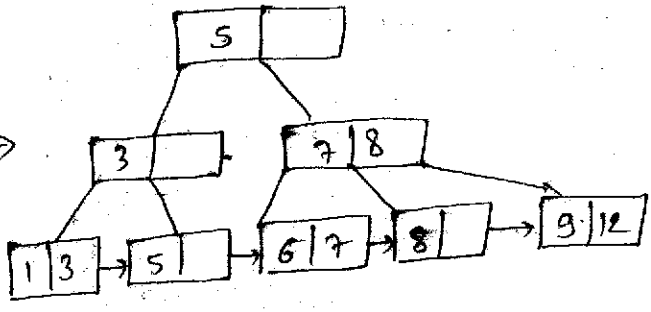












① No of splits = 5

② No of leaf node splits = 4

③ Root node splits = 2

④ Internal node = 1

⑤ no of nodes = 8

⑥ no of nodes of 100% fill factor = 4

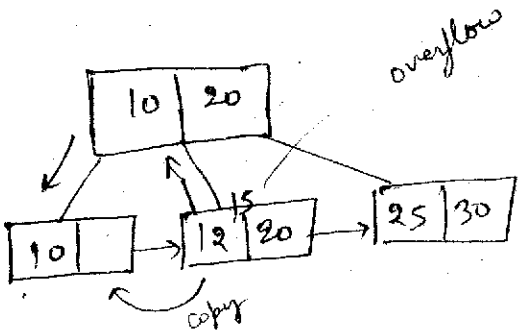
⑦ Root node = 5

⑧ Levels = 2

87

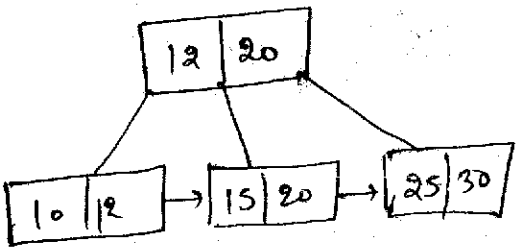
B^+ tree Rotation

B^+ tree can incorporate rotation to reduce the no. of ~~page~~ node splits. a rotation occurs when a leaf node is full but of its sibling is not full rather than splitting the leaf node we move a records to its siblings adjusting the pointers as necessary. typically the left sibling is checked first (if exist) and then the right sibling



Insert 15

← This is special case (avoid this)
If asked in exam (do it by rotation then only do this case otherwise do normally



PP

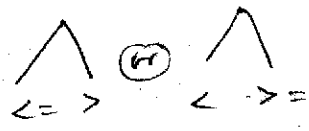
B^+ tree Deletion

Same prop of B-Tree

① ② ③ } same

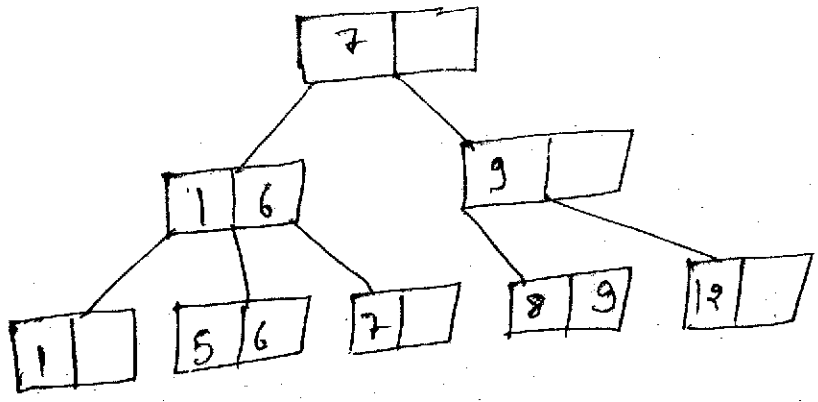
+

① Property

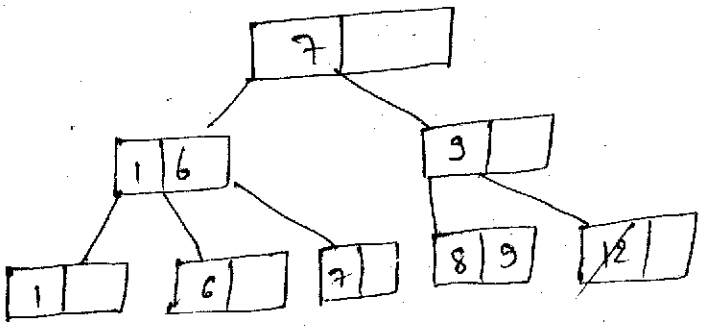


② Height balance

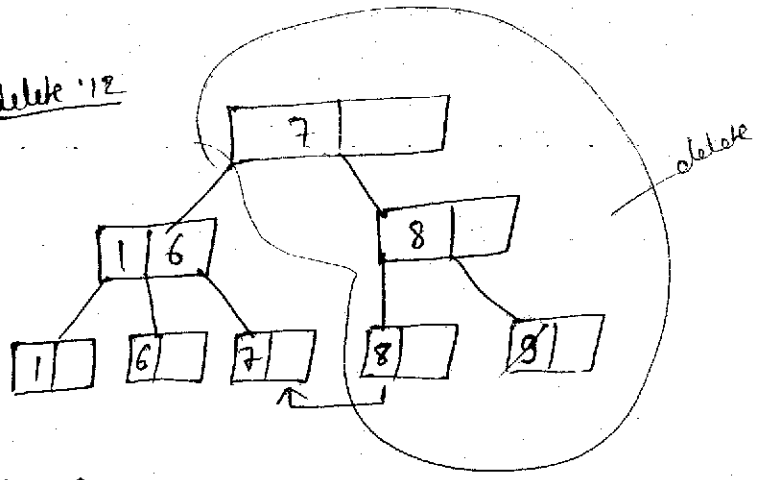
③ Min order



delete 5, 19, 9

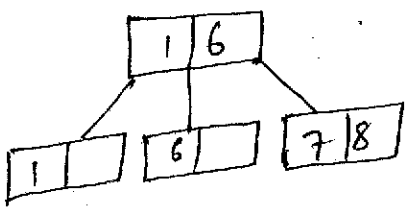


delete 12



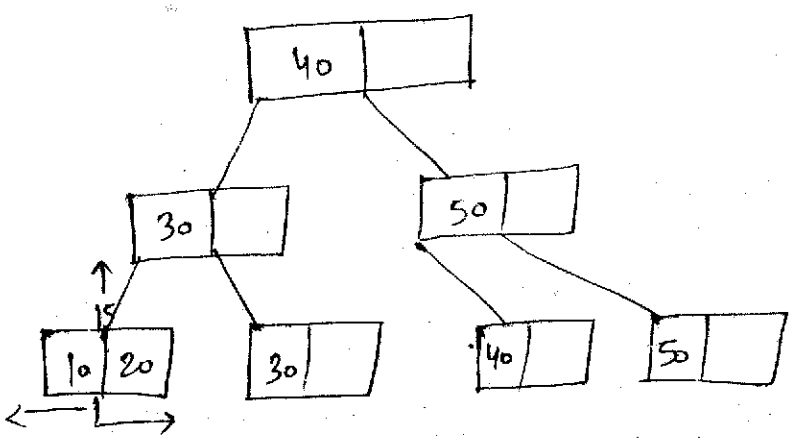
If we merge 8 to 7 then
If we delete right sub tree of ⑦
If we delete Right sub tree
then link of 7 node becomes
1 (violate prop) so
delete root ⑦ node

delete 9

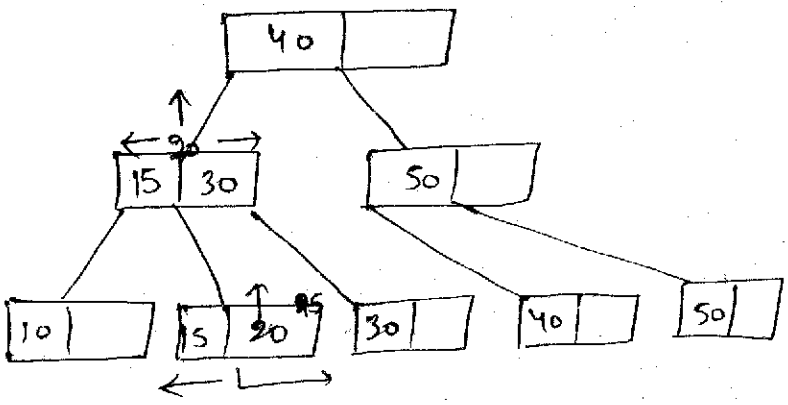


Pg-82

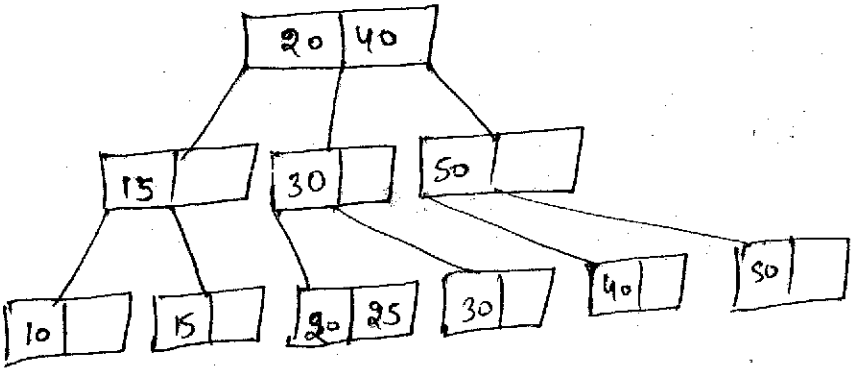
05, 6



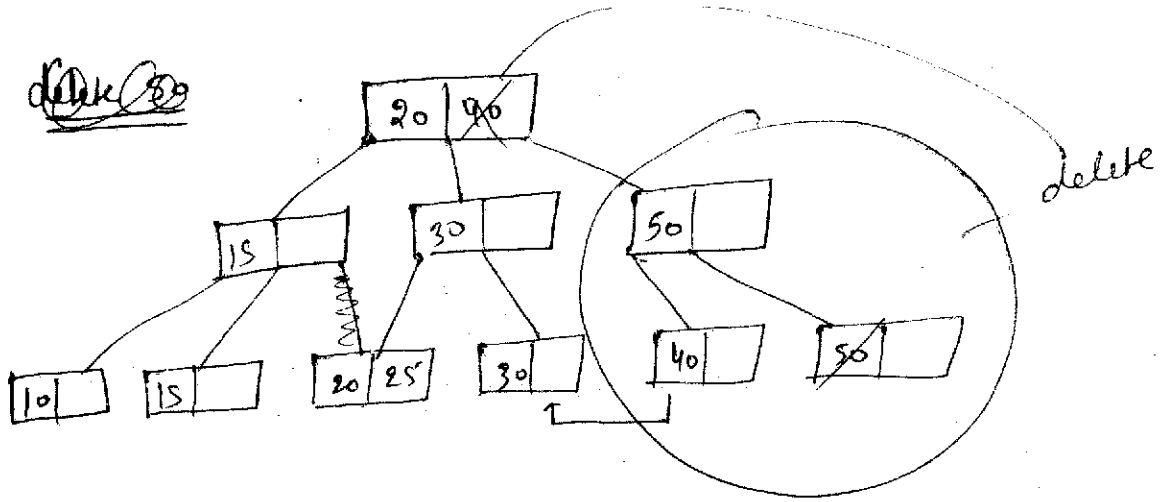
Insert 15



Insert 25



(6) delete 90



delete 50